**Kauno technologijos universitetas**
Matematikos ir gamtos mokslų fakultetas

# Oro taršos rodiklių tyrimas

Baigiamasis magistro studijų projektas

**Adomas Knyva**
Projekto autorius

**Prof. Dr. Tomas Ruzgas**
Vadovas

**Kaunas, 2025**

**Kauno technologijos universitetas**
Matematikos ir gamtos mokslų fakultetas

# Oro taršos rodiklių tyrimas

Baigiamasis magistro studijų projektas
Taikomoji matematika (6211AX006)

**Adomas Knyva**
Projekto autorius

**Prof. Dr. Tomas Ruzgas**
Vadovas

**Doc. Dr. Kristina Pupalaigė**
Recenzentė

**Kaunas, 2025**

**Kauno technologijos universitetas**
Matematikos ir gamtos mokslų fakultetas
Adomas Knyva

# Oro taršos rodiklių tyrimas

Akademinio sąžiningumo deklaracija

Patvirtinu, kad:

1. baigiamąjį projektą parengiau savarankiškai ir sąžiningai, nepažeisdama(s) kitų asmenų autoriaus ar kitų teisių, laikydamasi(s) Lietuvos Respublikos autorių teisių ir gretutinių teisių įstatymo nuostatų, Kauno technologijos universiteto (toliau – Universitetas) intelektinės nuosavybės valdymo ir perdavimo nuostatų bei Universiteto akademinės etikos kodekse nustatytų etikos reikalavimų;

2. baigiamajame projekte visi pateikti duomenys ir tyrimų rezultatai yra teisingi ir gauti teisėtai, nei viena šio projekto dalis nėra plagijuota nuo jokių spausdintinių ar elektroninių šaltinių, visos baigiamojo projekto tekste pateiktos citatos ir nuorodos yra nurodytos literatūros sąraše;

3. įstatymų nenumatytų piniginių sumų už baigiamąjį projektą ar jo dalis niekam nesu mokėjęs (-usi);

4. suprantu, kad išaiškėjus nesąžiningumo ar kitų asmenų teisių pažeidimo faktui, man bus taikomos akademinės nuobaudos pagal Universitete galiojančią tvarką ir būsiu pašalinta(s) iš Universiteto, o baigiamasis projektas gali būti pateiktas Akademinės etikos ir procedūrų kontrolieriaus tarnybai nagrinėjant galimą akademinės etikos pažeidimą.

Adomas Knyva

*Patvirtinta elektroniniu būdu*

Knyva, Adomas. Oro taršos rodiklių tyrimas. Magistro studijų baigiamasis projektas / vadovas dr. Tomas Ruzgas; Kauno technologijos universitetas, Matematikos ir gamtos mokslų fakultetas.

Studijų kryptis ir sritis (studijų krypčių grupė): Taikomoji matematika (Matematikos mokslai).

Reikšminiai žodžiai: oro tarša, sveikatos rodikliai, statistinė analizė, modeliavimas.

Kaunas, 2025. 79 p.

## Santrauka

Magistro baigiamajame darbe yra tiriami ir analizuojami oro taršos rodikliai Lietuvos teritorijoje.

Oro taršos tema yra aktuali visiems gyventojams, kadangi paveikia kiekvieno žmogaus sveikatą. Šis tyrimas konkrečiai yra ypač aktualus Lietuvos gyventojams, kadangi yra tiriama Lietuvos teritorija, tačiau tyrimo metodai yra universaliai pritaikomi bet kokiam pasaulio regionui.

Be oro taršos rodiklių šiame projekte yra apžvelgiami ir sveikatos duomenų rodikliai. Tyrime yra nagrinėjamos Pasaulio sveikatos organizacijos pasiūlytos oro taršos rodiklių rekomendacijos ir jų ribos, taip pat atsižvelgiama į Europos Sąjungos direktyvas oro taršos rodikliams.

Šio tyrimo tikslas yra atlikti oro taršos rodiklių statistinį tyrimą Lietuvos teritorijoje.

Tyrimo uždaviniai yra atvirų oro taršos bei sveikatos duomenų šaltinių analizė, laiko eilučių analizė oro taršos tendencijoms rasti, ryšių egzistavimo tarp oro taršos ir sveikatos rodiklių patikrinimas, ryšių tarp potencialių taršos šaltinių bei oro kokybės nustatymas.

Baigiamajame darbe yra naudojami laiko eilučių duomenų kompensavimo, išskirčių paieškos metodai. Tyrime taip pat naudojami statistiniai normalumo patikrinimo, koreliacijų radimo ir laiko eilučių dekompozicijos metodai, kuriuose ieškoma sezoniškumo, ciklišKumo ir tendencijų deterministinių rodiklių. Tyrime yra naudojami taršos koncentracijos bei vėjo krypties ir stiprumo duomenų koregavimo metodai. Tyrime pristatomi du nauji metodai, iš kurių viena yra skirtas specifinio tipo daugiamatės laiko eilutės trūkstamų verčių užpildymui (sumų-interpoliavimo išskirčių paieškos ir užpildymo metodas), o kitas (nuo meteorologinių sąlygų priklausantis oro taršos rodiklių ir šaltinių sąsajos modelis) yra skirtas stacionarių taršos šaltinių įtakai oro taršos rodikliams įvertinti.

Tyrime surastas metodas užpildyti trūkstamas laiko eilutės reikšmes, įvertintas ryšys tarp $PM_{2.5}$ oro taršos rodiklio ir padidėjusio mirtingumo, taip pat surasti koeficientai įvertinantys stacionarių taršos šaltinių įtaką oro taršos rodiklių reikšmėms.

Magistro baigiamuoju darbu išanalizuoti atvirų oro taršos bei sveikatos duomenų šaltiniai, aptartos jų problemos bei privalumai, atrinkti naudingiausi duomenų rinkiniai, atlikta vienmatė bei daugiamatė laiko eilučių analizė, atliktas trūkstamų duomenų papildymas bei panaudoti metodai išskirčių pašalinimui, surastos oro taršos rodiklių tendencijos, išnagrinėti egzistuojantys ryšiai tarp oro taršos ir sveikatos rodiklių. Rezultatai palyginti su panašiais tyrimais bei nustatyti ryšiai tarp taršos šaltinių ir oro kokybės rodiklių.

Knyva, Adomas. Air Pollution Indicators Study. Master's Final Degree Project / supervisor dr., Tomas Ruzgas; Faculty of Mathematics and Natural Sciences, Kaunas University of Technology.

Study field and area (study field group): Applied Mathematics (Mathematical Sciences).

Keywords: air pollution, health indicators, statistical analysis, modelling.

Kaunas, 2025. 79 p.

## Summary

The master's thesis investigates and analyzes air pollution indicators across the territory of Lithuania. The topic of air pollution is relevant to all residents, as it affects the health of every individual. Although this study focuses specifically on Lithuania, its methods are universally applicable to any region of the world.

In addition to air pollution metrics, this project also reviews health-related data indicators. The study examines the World Health Organization's recommended air pollution indices and their threshold values, and takes into account the European Union's directives on air quality standards.

The aim of this research is to conduct a statistical analysis of air pollution indicators within Lithuania. The objectives are to analyze open-source air quality and health data, to perform time-series analyses to identify pollution trends, to test for associations between air pollution and health outcomes, to determine relationships between potential pollution sources and air quality.

The thesis employs methods for compensating time-series data and detecting outliers. It also uses statistical tests for normality, correlation analysis, and time-series decomposition techniques to uncover seasonality, cyclicality, and deterministic trends. Additionally, methods are applied to adjust pollutant concentration data for wind direction and speed.

Two new approaches are introduced: one for filling missing values in a specific type of multivariate time series (a sum-interpolation with outlier detection and imputation method), and another model that links air pollution indicators to stationary pollution sources, accounting for meteorological conditions, to assess their impact on air quality metrics.

The study develops a method for imputing missing time-series values and evaluates the relationship between $PM_{2.5}$ concentration and increased mortality. It also derives coefficients quantifying the influence of stationary sources on air pollution levels.

As a result of this master's thesis, open air quality and health data sources have been analyzed, their strengths and limitations discussed, and the most useful datasets selected. Both univariate and multivariate time-series analyses were conducted, missing data were imputed, and outliers handled. Pollution trends were identified, and existing links between air pollution and health indicators explored. The findings are compared with similar studies, and relationships between pollution sources and air quality metrics are established.

**Turinys**

## Lentelių sąrašas

## Paveikslų sąrašas

## Santrumpų ir terminų sąrašas

**Santrumpos:**

API - aplikacijų programavimo sąsaja (angl. *application programming interface*).

ES – Europos Sąjunga.

GWR – geografiškai svorinė regresija (angl. *geographically weighted regression*)[1].

HCB – heksachlorobenzenas.

HR – pavojaus santykis (angl. *hazard ratio*).

IKP – interkvartilinis plotis (angl. *interquartile range*)[2].

IT – tarpinis tikslas (angl. *interim target*).

MAD – absoliutinis medianos nuokrypis (angl. *median absolute deviation*)[3].

PCA – pagrindinių komponenčių analizė (angl. *principal component analysis*).

pr. m. e. – prieš mūsų erą (angl. *before common era*)[4].

ppb – dalys milijarde (angl. *parts per billion*).

PM – kietosios dalelės (angl. *particulate matter*).

PSO – Pasaulio sveikatos organizacija (angl. *World Health Organization*).

RBF – radialinė (spindulinė) bazinė funkcija su plonos plokštelės splainu (angl. *radial basis function with thin plate spline*)[1].

RMSE – šaknis iš vidutinio kvadratinio nuokrypio (angl. *root-mean-square error*)[5,6].

**Terminai:**

**AQG lygis** – konkreti rekomendacijos forma, sudaryta iš tam tikro oro teršalo koncentracijos, kuri yra susieta su dydžio vidurkiu laike, skaitinės vertės išraiškos. Ilgalaikis AQG lygis apibrėžiamas kaip žemiausias oro teršalų poveikio lygis, kurį viršijus, PSO rekomendacijų rengimo grupės įsitikinimais, didėja neigiamas poveikis sveikatai. Trumpalaikis AQG lygis apibrėžiamas kaip aukštas paros verčių pasiskirstymo procentilis (pvz., 99-asis procentilis atitinka 3-4 dienas per metus, kai ši vertė viršijama)[7].

**Krigingas** – kitaip dar vadinamas Gauso procesų regresija, tai yra (originaliai geostatistinis[8,9]) interpoliavimo metodas, pagrįstas Gauso procesais ir kovariacijomis[10]. Šiame metode taip pat vyrauja autokoreliacija[11].

**Molinė masė** – tam tikros medžiagos vieno molio masė. Skaitine verte lygi medžiagos molį sudarančių atomų santykinių atominių masių sumai, išreikštai gramais[12].

# Įvadas

## Temos aktualumas.

Visuotinėje literatūroje yra teigiama[13,14,15], jog vienas iš pirmųjų oro taršos problematikos paminėjimų yra įvykdytas apie 400 m. pr. m. e., Hipokrato veikale „Airs, eaux, lieux“[16], kuriame yra minima miazmos[17] (gr. *μίασμα*) sąvoka, tiesiogiai reiškianti blogą orą. Miazmos teorija[14,18,19] aprašo idėją, jog pūvanti organinė medžiaga arba dulkių dalelės (susidarančios iš žemėje esančių metalų bei druskų), su garuojančio vandens pagalba, gali kilti nuo paviršiaus ir koncentruotis, taip sudarant oro taršą sukeliančią žmonėms įvairias ligas[15].

Nors pastaroji teorija po šimtmečių (apie XIX a.) buvo paneigta ir mokslinė visuomenė priėmė dar iki šių laikų galiojančią mikroorganizmų sukeliamų ligų teoriją (angl. *germ theory of disease*)[20], tačiau oro taršos analizė bei taršos sąsajos su ligomis bei visuotinėmis problemomis paieškos tikrai nesustojo ir ši tema yra ypač opi šiais laikais, kai oro tarša liečia 99% visų žemės gyventojų[21].

## Tyrimo tikslas.

Šio mokslinio darbo tikslas yra atlikti oro taršos rodiklių statistinį tyrimą Lietuvos teritorijoje.

## Tyrimo uždaviniai.

1. Išanalizuoti atvirų oro taršos bei sveikatos duomenų šaltinius;
2. Atlikti laiko eilučių analizę oro taršos tendencijoms rasti;
3. Patikrinti ryšių egzistavimą tarp oro taršos ir sveikatos rodiklių;
4. Nustatyti ryšius tarp potencialių taršos šaltinių bei oro kokybės.

## Tyrimo hipotezė.

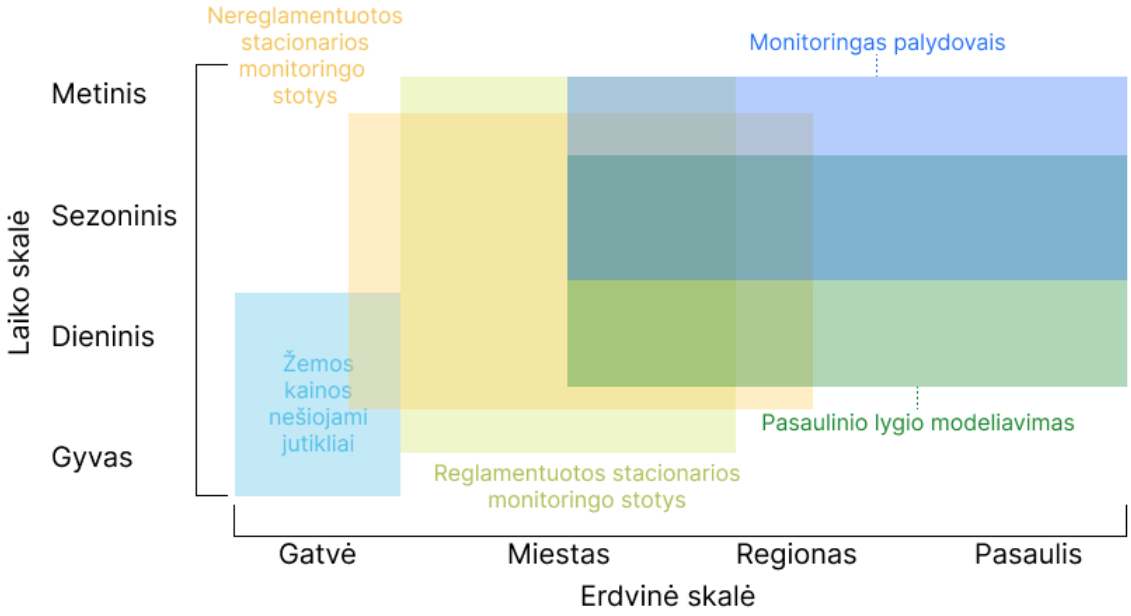
Pagal išankstines nuomones tikėtina surasti teigiamą ryšį tarp specifinių teršalų ir visuomenės sveikatos problemų.

# 1. Literatūros apžvalga

## 1.1. Oro taršos monitoringas

Skirtingų oro taršos monitoringo technologijų apžvalga Croman‘o straipsnyje[22] leidžia giliau suprasti gautos informacijos panaudojimo galimybes.

Tame pačiame straipsnyje yra pateikiama pagalbinė diagrama, leidžianti daugmaž vizualiai įsivaizduoti technologijų ir jų panaudojimo būdų santykį. Nors **1 pav.** ir yra sukurtas tik iliustravimo tikslams ir nėra paremtas konkrečių faktų pagrindu, tačiau paveiksliuke galima matyti pačią esmę – ne visi monitoringo technologijų tipai bendrąja prasme tinka bet kokiam panaudojimui.




**1 pav.** Skirtingų oro taršos monitoringo technologijų tipai bei jų panaudojimo būdai pagal erdvinę ir laiko skales[22].

**1 pav.** galime matyti, jog trumpalaikį oro taršos monitoringą labai smulkiu mastu (pvz., gatvėje) verta atlikti su nešiojamu, paprastu matuokliu. Žemos kainos oro kokybės jutiklių (angl. *low-cost sensors*) apžvalga iškelia išvadą, jog šie matuokliai nėra pakankamai patikimi juos naudoti vietoj oficialių ir standartizuotų oro taršos monitoringo stočių, tačiau geriausi pigūs oro kokybės jutikliai gali būti pakankamai geri duomenims gauti (lyginant su standartizuotais matuokliais determinacijos koeficientas[23,24] $R^2 > 0{,}75$, o krypties koeficientas arti 1,0)[25], ypač jei tai nėra duomenys, kurie privalo būti itin kokybiški (pvz., skirti valstybinės reikšmės statistikoms vesti).

Minėtam specifinių smulkaus masto zonų oro kokybės tyrimo panaudojimui dažniausiai netinka oficialūs, sertifikatais patvirtinti (angl. *field proven*) oro taršos matuokliai, kadangi jų įrengimas ir palaikymas yra labai brangus. Tam, kad oro kokybės monitoringo stotis būtų patvirtinta oficialiam meteorologinių duomenų matavimui, reikia pereiti įrenginio kokybės įvertinimus[26,27,28], kurie prideda dar papildomos kainos prie ir taip jau brangių jutiklių (vienas įrenginys gali kainuoti tūkstančius eurų[26,29]).

Taigi, pirkti iš karto sertifikuotą oro kokybės monitoringo stotį yra labai brangus reikalas, tačiau egzistuoja būdų, skirtų šiai didelei kainai sumažinti naudojant žemesnės kainos oro kokybės

matavimo įrenginius. Šis kainos mažinimo būdas yra reliatyviai pigių matuoklių kalibravimas pagal rinkoje patvirtintus įrenginius. Tam yra galimi du variantai[30]:

- Laboratoriniai testai – kontroliuojamoje laboratorijos aplinkoje žemos kainos įrenginiai yra matuojami lygiagrečiai su patvirtintais įrenginiais ir surandama kalibravimo kreivė (angl. *calibration curve*), kurią pritaikius žemos kainos jutiklio įverčiams, gaunami transformuoti duomenys, kurie teoriškai turėtų būti tikslesni;
- Aktualios aplinkos testai (angl. *field tests*) – vietoje kalibravimo laboratorijoje, šis procesas vyksta iškart aplinkoje, kurioje bus eksploatuojamas žemos kainos matuoklis. Tam yra naudojama strategija sumontuoti abudu monitoringo įrenginius vienoje vietoje ir tam tikrą laiko tarpą rinkti duomenis, pagal kuriuos galima sukurti adaptyvų modelį, leidžiantį sukalibruoti žemos kainos monitoringo stotį ir paruošti ją darbui aplinkos sąlygoms, kuriomis vyko modelio treniravimas.

Žvelgiant į Castell'o straipsnyje pateiktus rezultatus galima atrasti daug netikslumų kalibruojant laboratorijoje, o po to naudojant jutiklį realioje testavimo aplinkoje (matuojant CO koncentraciją aktyvioje gatvėje, laboratorinis kalibravimas lėmė 166 ppb skirtumą matuojant realiomis sąlygomis, o aktualios aplinkos kalibravimas rodė tik 0,7 ppb neatitikimą)[31], todėl yra rekomenduojama kalibruoti žemos kainos įrenginius iškart jų eksploatacijos vietoje.

Taigi, net ir žinant kaip tai įvykdyti, kokybiškos oro taršos monitoringo problema vis dar išlieka, kadangi stočių parengimas ir priežiūra reikalauja daug piniginių bei laiko sąnaudų.

## 1.2. Laiko eilučių analizė

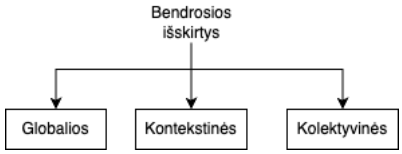
### Išskirčių radimas ir sutvarkymas

Išskirčių radimas laiko eilutėje yra gana dinamiška problema, kadangi niekas nėra radęs absoliučios tiesos šiuo klausimu, tačiau mokslininkai yra sukūrę ne vieną labai gerą metodą išskirčių paieškai, todėl tą metodą tereikia (kažkiek subjektyviai) pasirinkti pagal atliekamą tyrimą[3].

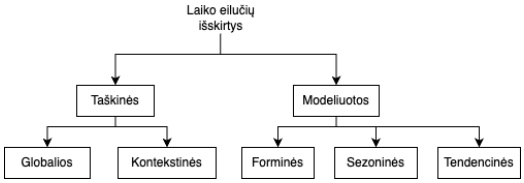
Išskirčių ieškojimas nėra pati paprasčiausia procedūra, nes procesai skiriasi keliose kategorijose:

- Vienmačių (angl. *univariate*) ir daugiamačių (angl. *multivariate*) laiko eilučių skirtumas;
- Išskirties paieškos tipo. Galima išskirčių ieškoti „gyviems" duomenims naudojant nuspėjimo modelį (angl. *prediction model-based*) arba galima ieškoti išskirčių istorinės prigimties duomenims aproksimuojant ir prieš matavimą, ir po matavimo turimais duomenimis (angl. *estimation model-based*).

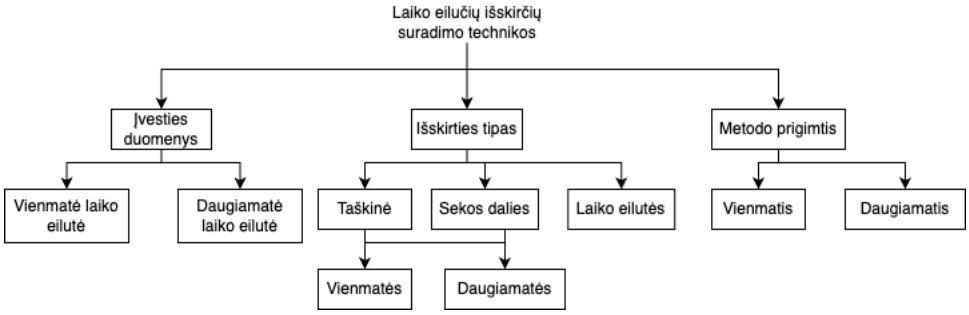
Dėl to, kad laiko eilučių išskirčių teisingas kategorizavimas yra konceptuali problema, tai yra pasiūlyta netgi ne vienas siūlomos taksonomijos variantas (žr. **2 pav.**).




**(a)** Egzistuojanti taksonomija[32].




**(b)** Elgesio vedama taksonomija[32].




**(c)** Išskirčių suradimo technikų taksonomija[3].

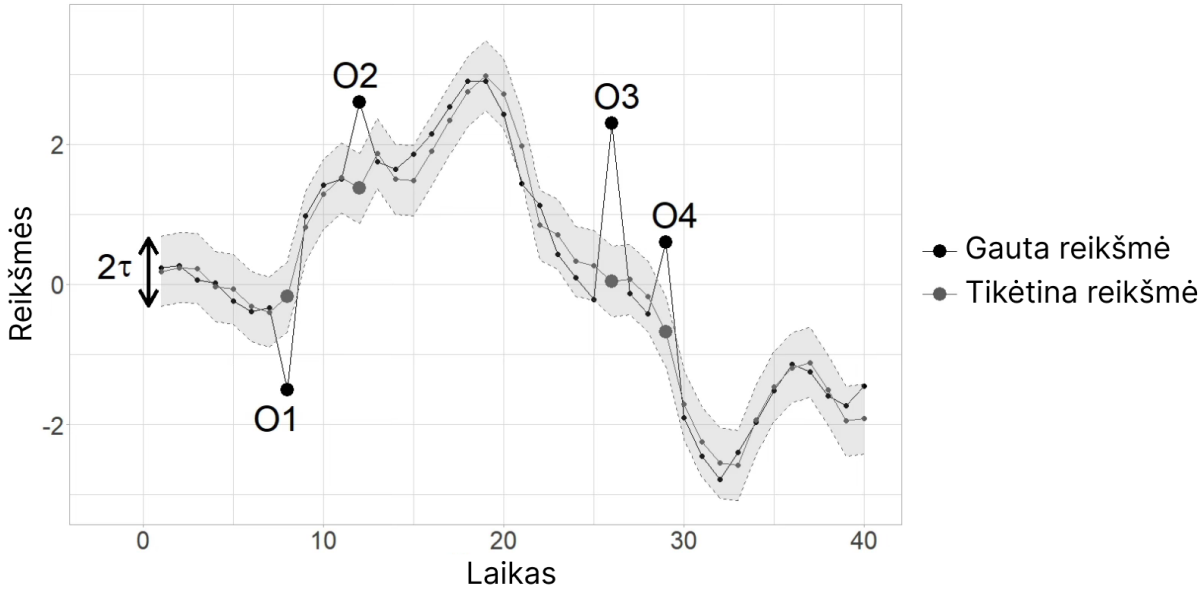
**2 pav.** Skirtingų išskirčių taksonomijų palyginimas.

### *Vienmatės taškinės išskirtys[3]*

Turint vienmatę laiko eilutę išskirčių paieškas galima atlikti gana paprastos idėjos metodu – pagal tikrų reikšmių laiko eilutę $X = (x_1, x_2, \dots, x_n)$ sukurti tikėtinų reikšmių laiko eilutę $\hat{X} = (\widehat{x_1}, \widehat{x_2}, \dots, \widehat{x_n})$, sugalvoti tam tikrą išskirties ribą $\tau$ ir pagal tai ieškoti laiko verčių $t$, su kuriomis realios laiko eilutės vertė neįeina į tikėtiną intervalą (dvipusis medianos metodas) pagal (1) formulę:

$$|x_t - \hat{x}_t| > \tau \tag{1}$$

čia $x_t$ – gauta (tikra) reikšmė laiko momentu *t*; $\hat{x}_t$ – tikėtina reikšmė laiko momentu *t*; $\tau$ – išskirties riba.




**3 pav.** Taškinių išskirčių paieška vienmatėje laiko eilutėje, pagrįsta tikėtinomis ir realiomis reikšmėmis[3].

Šio tyrimo metu vienmačių išskirčių radimui galima pasinaudoti ir nuspėjimo, ir istorinės prigimties aproksimavo modeliais (žr. **1 lentelė**).

**1 lentelė.** Vienmatės laiko eilutės išskirčių radimo technikų palyginimas, kai $k \geq 1$, ir $k_1, k_2 \geq 0$: $k_1 + k_2 > 0$ [3].

| | **Naudojami duomenys** | → | **Aproksimuojama vertė** | → | **Išskirties taškai** |
|---|---|---|---|---|---|
| Istorinės prigimties aproksimavimo modelis | $\{x_{t-k_1}, \dots, x_t, \dots, x_{t+k_2}\}$ | → | $\widehat{x_t}$ | → | $\lvert x_t - \widehat{x_t}\rvert > \tau.$ |
| Nuspėjimo modelis | $\{x_{t-k}, \dots, x_{t-1}\}$ | → | $\widehat{x_t}$ | | |

## Trūkstamų verčių užpildymas

Turint bet kokią laiko eilutę galima tikėtis trūkstamų duomenų, todėl verta pasiruošti metodą duomenų užpildymui. Metodų, panašiai kaip ir išskirčių ieškojime, yra labai daug ir jie priklauso nuo to, kokie duomenys yra turimi ir kokiu būdu norima užpildyti trūkstamas vietas[33].

Trūkstamų verčių radimas oficialiai vadinasi duomenų kompensavimu, o tam technikų yra nemažai, taigi vėl tenka pasirinkti labiausiai tyrimui tinkamą būdą. Žemiau sąraše pateikiami keli paprasčiausi metodai, skirti duomenų kompensavimui daugiamatėse laiko eilutėse[34]:

- ***Paprastasis aritmetinio vidurkio metodas.*** Apskaičiuojamas pagal (2) formulę:
$$\hat{Y}_{m_k t} = \frac{1}{N}\left(Y_{m_k 1} + Y_{m_k 2} + \cdots + Y_{m_k N}\right); \quad (2)$$
čia $\hat{Y}_{m_k t}$ – *t* (tikslo) subkategorijos aritmetinis vidurkis laiko momentu $m_k$; *N* – subkategorijų (išskyrus *t*) skaičius; $Y_{m_k 1}, Y_{m_k 2}, \dots, Y_{m_k N}$ – subkategorijų reikšmės tuo pačiu laiko momentu $m_k$;
- ***Normaliojo santykio metodas.*** Apskaičiuojama pagal (3) formulę:
$$\hat{Y}_{m_k t} = \frac{1}{N}\left[\left(\frac{T_{.t}}{T_{.1}}\right) Y_{m_k 1} + \left(\frac{T_{.t}}{T_{.2}}\right) Y_{m_k 2} + \cdots + \left(\frac{T_{.t}}{T_{.N}}\right) Y_{m_k N}\right]; \quad (3)$$
čia $\hat{Y}_{m_k t}$ – *t* (tikslo) subkategorijos vidurkis, gaunamas normaliojo santykio metodu, laiko momentu $m_k$; *N* – subkategorijų (išskyrus *t*) skaičius; $T_{.t}$ – kintamojo stebėjimų, išmatuotų tikslo subkategorijoje $t$, suma; $T_{.j}$ – kintamojo stebėjimų, išmatuotų subkategorijoje $j$ ($j = 1, \dots, N$), suma; $Y_{m_k 1}, Y_{m_k 2}, \dots, Y_{m_k N}$ – subkategorijų reikšmės tuo pačiu laiko momentu $m_k$;
- ***Normaliojo santykio ir koreliacijų svorinis metodas***. Formulė priklauso nuo svorinio koeficiento, apskaičiuojamo (4) formule:
$$w_j = \frac{(n-2) r_{jt}^2}{1 - r_{jt}^2}; \quad (4)$$
čia $w_j$ – *j*-osios subkategorijos svoris; *n* – koreliacijos skaičiavime naudojamų matavimų skaičius ($n \geq 2$); $r_{jt}$ – koreliacijos koeficientas tarp subkategorijų *t* (tikslo) ir *j* ($j \neq t$).
Galutinis vertės apskaičiavimas yra atliekamas (5) formule:
$$\hat{Y}_{m_k t} = \frac{1}{\sum_{j=1}^{N} w_j}\left[w_1 Y_{m_k 1} + w_2 Y_{m_k 2} + \cdots + w_N Y_{m_k N}\right] = \frac{\sum_{j=1}^{N} w_j Y_{m_k j}}{\sum_{j=1}^{N} w_j}; \quad (5)$$
čia $\hat{Y}_{m_k t}$ – *t* (tikslo) subkategorijos vidurkis, gaunamas normaliojo santykio ir koreliacijų svoriniu metodu, laiko momentu $m_k$; *N* – subkategorijų (išskyrus *t*) skaičius; $w_j$ – *j*-osios

subkategorijos svoris ($j \neq t$); $Y_{m_k1}, Y_{m_k2}, \dots, Y_{m_kN}$ – subkategorijų reikšmės tuo pačiu laiko momentu $m_k$.

Pastarasis metodas naudoja koreliacijos koeficientą, kuris šaltinyje nėra apibrėžtas tiksliai, tačiau turint:

1. Normaliai pasiskirsčiusių[35,36] laiko eilučių sistemą galima naudoti Pearson'o koreliacijos koeficientą imtims (angl. *sample Pearson correlation coefficient*) $r_{xy}$ [37]. Turint reikšmių poras $\{(x_1, y_1), \dots, (x_n, y_n)\}$ iš skirtingų imčių *x* ir *y*, koreliacijos koeficientas yra randamas (6) formule:
$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}; \quad (6)$$
čia $r_{xy}$ – Pearson'o koreliacijos koeficientas; $n$ – yra imties dydis[38] ($n > 3$); $x_i, y_i$ – reikšmių porų, iš *x* ir *y* imčių, taškai su indeksu *i*; $\bar{x}$ – imties *x* vidurkis (tas pats galioja ir kintamajam $\bar{y}$ su imtimi *y*), apskaičiuojamas formule $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$;
2. Nenormaliai pasiskirsčiusių laiko eilučių sistemą (bent viena eilutė nėra normaliai pasiskirsčiusi) galima naudoti Spearman'o koreliacijos koeficientą $r_s$ [39,40,41]. Jis yra apskaičiuojamas (7) formule:
$$r_s = 1 - \frac{6\sum_{i=1}^{n} d_i^2}{n(n^2 - 1)}; \quad (7)$$
čia $r_s$ – Spearman'o koreliacijos koeficientas; *n* – stebėjimų skaičius; $d_i$ – skirtumas tarp ranguotų kintamųjų reikšmių $R_x$ ir $R_y$, atitinkamai imtims *x* ir *y*, su *i*-tuoju indeksu, apskaičiuojamas formule $d_i = R_{xi} - R_{yi}$;
3. Nenormaliai pasiskirsčiusių ir ranginių laiko eilučių sistemą galima naudoti Kendall'o koreliacijos koeficientą $\tau$ [42,43,44]. Kadangi egzistuoja trys Kendall'o koreliacijos koeficientų versijos, kurių kiekviena yra šiek tiek skirtinga, reikia paminėti, jog šiame tyrime yra naudojama Tau-b (originaliai pavadinta $\tau_W$ [45]) atmaina. Ji yra apskaičiuojama (8) formule:
$$\tau = \frac{n_c - n_d}{\sqrt{(n_0 - n_1)(n_0 - n_2)}}; \quad (8)$$
čia $\tau$ – Kendall'o Tau-b koreliacijos koeficientas; $n_c$ – suderintų porų skaičius; čia suderinta pora – pora su koeficientais $0 < i < j < n$, kur $x_i < x_j$ ir $y_i < y_j$ arba $x_i > x_j$ ir $y_i > y_j$; $n_d$ – nesuderintų porų skaičius; čia nesuderinta pora – pora su koeficientais $0 < i < j < n$, kur $x_i < x_j$ ir $y_i > y_j$ arba $x_i > x_j$ ir $y_i < y_j$; $n_0 = \frac{n(n-1)}{2}$; čia *n* – stebėjimų skaičius; $n_1 = \sum_i \frac{t_i(t_i-1)}{2}$; čia $t_i$ – lygių reikšmių skaičius *i*-tojoje lygių reikšmių grupėje pagal empirinį *x* imties pasiskirstymą; $n_2 = \sum_j \frac{u_j(u_j-1)}{2}$; čia $u_j$ – lygių reikšmių skaičius *j*-tojoje lygių reikšmių grupėje pagal empirinį *y* imties pasiskirstymą.

Kadangi prieš koreliacijos koeficientus išvardinti metodai veikia praktiškai tik vienu laiko momentu[1], jų tikslumas gerokai suprastėja, jeigu tam tikru laiko momentu turima daug trūkstamų verčių.

**Normalumo tikrinimas**

[1] Nors normaliojo santykio bei normaliojo santykio ir koreliacijų svorinis metodai atsižvelgia į skirtingų laiko eilučių santykį, tai nėra dinamiškas veiksmas, kadangi jis nepriklauso nuo laiko požiūriu šalimais esančių reikšmių.

Normalumo tikrinimas atliekamas naudojantis Shapiro-Wilk‘o testu[46,47,48], kurio statistika apskaičiuojama (9) formule:

$$W = \frac{\left(\sum_{i=1}^{n} a_i x_{(i)}\right)^2}{\sum_{i=1}^{n}(x_i - \bar{x})^2}; \tag{9}$$

čia $W$ – Shapiro-Wilk‘o normalumo testo statistika; $n$ – stebėjimų skaičius; $a_i$ – koeficientas iš Shapiro-Wilk‘o svorių lentelės[49], priklausantis nuo $n$; $x_{(i)}$ – didėjimo prasme surūšiuota pradinė vertė; $x_i$ – $i$-tojo matavimo reikšmė; $\bar{x}$ – aritmetinis visų matavimų vidurkis.

Gavus $W$ statistiką, iš Shapiro-Wilk‘o $p$-reikšmių lentelės susirandama artimiausia reikšmė $p$[2], kuri ir nusako normalumą ($p < 0{,}05$ atmeta normalumo hipotezę)[3].

Normalumo tikrinimui taip pat yra pasitelkiamos asimetrijos (angl. *skewness*) ir eksceso (angl. *kurtosis*) statistikos[51,52,53,54]. Asimetrijos statistika yra skirta patikrinti pasiskirstymo simetriškumui ir yra apskaičiuojama (10) formule, o eksceso statistika nusako skirstinio verčių tankį labiausiai užpildytoje vietoje ir yra apskaičiuojama (11) formule:

$$A = \frac{1}{nD^{\frac{3}{2}}} \sum_{i=1}^{n} (x_i - \bar{x})^3, \tag{10}$$

$$E = \frac{1}{nD^2} \sum_{i=1}^{n} (x_i - \bar{x})^4 - 3\,; \tag{11}$$

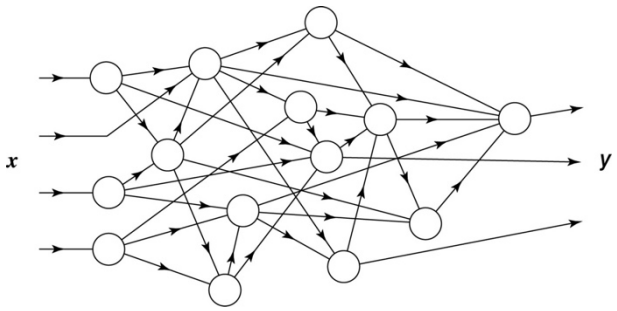
čia $A$ – asimetrijos statistika; $n$ – matavimų skaičius; $D$ – dispersija (angl. *variance*)[55,56], apskaičiuojama formule $D = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n}$; $x_i$ – $i$-tojo matavimo reikšmė; $\bar{x}$ – aritmetinis visų matavimų vidurkis; $E$ – eksceso statistika.

**Kiti duomenų kompensavimo metodai**

Beje, norint plačiau įsigilinti į duomenų kompensavimą, negalima ignoruoti neuroninių tinklų galimybių šioje temoje. Neuroniniais tinklais galima sukurti nuspėjimo modelius, naudojančius dalinai parametrinius (angl. *semi-parametric*), deterministinius bei tiesioginio sklidimo[57,58] (angl. *feed-forward*) (žr. **4 pav.**) interpoliavimo sluoksnius[59].

[2] Kai reikšmė lentelėje nerandama, reikia ieškoti artimiausių reikšmių šalimais (su tam tikru $n$) ir rasti reikiamą vertę interpoliuojant[50].

[3] Šis metodas yra populiariausias skaičiavimo metodas, kadangi nėra reikalo analitiškai skaičiuoti vertės. Originaliame straipsnyje[48] yra išrašyta skaičiavimo eiga, kurią replikuojant galima atlikti koeficientų radimus ir be lentelės.




**4 pav.** Tiesioginio sklidimo dirbtinio neuroninio tinklo schema, kur *x* - įėjimo vektorius, o *y* - išėjimo vektorius[58].

Be abejo, neuroninio tinklo architektūra gali būti labai įvairi, todėl šiame skyriuje toliau nebus gilinamąsi į šios temos šaknis, kadangi tyrimo tikslas nėra gauti pačius tiksliausius rezultatus, o verčiau apžvelgti potencialias idėjas.

Tačiau vieta, į kurią verta toliau pasigilinti, yra jau trumpai paminėtas interpoliavimas.

Norint atlikti deterministinį interpoliavimą vienmatei laiko eilutei, tai galima būtų atlikti gana nesunkiai, kadangi laiko eilutė jau savaime turi interpoliavimo mazgus intervale nuo pradžios iki pabaigos: jei reikšmių stebėjimo intervalo pradžia žymima *a*, o pabaiga – *b*, tai interpoliavimo mazgai bus $a = x_0 < x_1 < \cdots < x_k < \cdots < x_n = b$, kurie turi atitinkamas reikšmes $y_k = f(x_k)$ su $k = 0,1, \dots, n$, kai $f(x)$ nurodo žinomą funkciją, kuri ir yra pagrindas interpoliavimo funkcijos $P(x)$ paieškoms, o galutinis metodo tikslas yra rasti $P(x)$: $P(x_k) = f(x_k)\ \forall k = 1,2, \dots, n$. Determinizmas randamas toje vietoje, kur pačiam naudotojui galima apibrėžti funkcijų aibę $u_k(x)$ (kuri gali būti sudaryta iš tiesiškai nepriklausomų funkcijų kaip $1, x, x^2, \dots, x^n\ arba\ 1, \sin x, \sin 2x, \dots, \sin nx$), skirtą interpoliavimo funkcijos paieškoms, kadangi $P(x) = \sum_{k=0}^{n} c_k u_k(x)$[60,61].

Daugiamatės laiko eilutės atveju minėtas būdas, naudojamas tiesiogiai, yra neaktualus, kadangi prarandama visa papildoma informacija iš papildomų laiko eilučių.

Galų gale, pilnai peržvelgus visų duomenų kompensavimo metodų sąrašą, reiktų nepamiršti paminėti, jog egzistuoja dar ir stochastiniai metodai (neuroniniai tinklai jau paminėti praėjusiose pastraipose), kurie apima regresijos bei autoregresijos metodus[61].

**Komponenčių dekompozicijos analizė**

Norint išsiaiškinti kas slypi laiko eilutės sudėtyje yra įprasta atlikti komponenčių dekompozicijos analizę[62] skaidant laiko eilutę pagal tendencijos (angl. *trend*), sezoniškumo (angl. *seasonality*), ciklišku mo (angl. *cyclical*) ir nepastovumo (angl. *irregular*) komponentes. Apibendrintai dekompozicijos metodas yra atliekamas su (12) formule:

$$Y_t = f(T_t, S_t, E_t); \tag{12}$$

čia $Y_t$ – laiko eilutės reikšmė *t* laiko momentu; $f(\cdot)$ – funkcijos forma, priklausanti nuo pasirinkto dekompozicijos metodo; $T_t$ – tendencijos-ciklišku mo ar bendra judėjimo komponentė; $S_t$ – sezoniškumo komponentė; $E_t$ – nepastovumo (sudaryta iš likučio (angl. *remainder*) arba liekanos (angl. *residual*)) komponentė.

Dekompozicijos forma gali būti sudedamoji, nusakoma lygtimi $Y_t = T_t + S_t + E_t$ arba sudauginamoji, nusakoma lygtimi $Y_t = T_t \cdot S_t \cdot E_t$.

Reikia paminėti, jog atliekant dekompoziciją reiktų turėti pilnai užpildytą laiko eilutę be trūkstamų verčių[63]. Paprasčiausia būtų pasinaudoti jau minėtu interpoliavimu, pirmyn-užpildančiu (angl. *forward-filling*) arba atgal-užpildančiu (angl. *backward-filling*) metodais. Taip pat yra būdų duomenis užpildyti ir regresiniais metodais[64].

**Tiesinė regresija**

Tiesinės regresijos tikslas yra rasti geriausias reikšmes lygčiai, modeliuojančiai vieną kintamąjį kito kintamojo duomenimis. Paprasta tiesinės regresijos forma aprašoma (13) lygtimi[65,66]:

$$Y = \beta_0 + \beta_1 X + \varepsilon; \tag{13}$$

čia $Y$ – priklausomas kintamasis; $\beta_0$ – konstanta (angl. *intercept*), nurodanti vertikaliosios ašies kirtimo tašką, kai $X = 0$ ir $\varepsilon = 0$; $\beta_1$ – krypties koeficientas (angl. *slope*); $X$ – nepriklausomas kintamasis; $\varepsilon$ – likučiai (angl. *residuals*).

Koeficientų $\beta_1, \beta_0$ aproksimacijos $\hat{\beta}_1, \hat{\beta}_0$ yra skaitiškai randamos (14) ir (15) lygtimis atitinkamai[66]:

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sum_{i=1}^{n}(X_i - \bar{X})^2}, \tag{14}$$

$$\hat{\beta}_0 = \bar{Y} - \hat{\beta}_1 \bar{X}; \tag{15}$$

čia $\hat{\beta}_1$ – krypties koeficiento $\beta_1$ skaitinė aproksimacija; $n$ – matavimų skaičius; $X_i, Y_i$ – $i$-tosios $X$ ir $Y$ kintamųjų reikšmės; $\bar{X}, \bar{Y}$ – $X$ ir $Y$ kintamųjų aritmetiniai vidurkiai; $\hat{\beta}_0$ – konstantos $\beta_0$ skaitinė aproksimacija.

Taip pat reikia paminėti likučių skaitiniam aproksimavimui skirtą (16) formulę:

$$\hat{u}_i = Y_i - \hat{Y}_i; \tag{16}$$

čia $\hat{u}_i$ – likučių skaitinė aproksimacija $i$-tajame taške; $Y_i$ – $i$-toji priklausomo kintamojo $Y$ reikšmė; $\hat{Y}_i$ – $i$-tosios priklausomo kintamojo $Y$ reikšmės aproksimacija, apskaičiuojama kaip $\hat{Y}_i = \hat{\beta}_0 + \hat{\beta}_1 X_i$.

***$R^2$ statistika***

Determinacijos koeficientas (angl. *adjusted R-squared*), žymimas $R^2$, nustato modelio tikslumą[67] ir yra apskaičiuojamas (17) formule[68]:

$$R^2 = \frac{\sum_{i=1}^{n}\left(\hat{Y}_i - \bar{Y}\right)^2}{\sum_{i=1}^{n}(Y_i - \bar{Y})^2}; \tag{17}$$

čia $R^2$ – determinacijos koeficientas; $\hat{Y}_i$ – $i$-tosios priklausomo kintamojo $Y$ reikšmės aproksimacija; $\bar{Y}$ – priklausomo kintamojo $Y$ aritmetinis vidurkis; $Y_i$ – $i$-toji priklausomo kintamojo $Y$ reikšmė.

***Pasikliautinumo intervalas***

Norint pridėti dar daugiau naudingos informacijos į regresijos grafiką, naudinga jame pavaizduoti ir pasikliautinumo intervalą. Pasikliautinumo intervalas apskaičiuojamas (18) formule[68]:

$$PI_{1-\alpha}^{\beta_1} = \left[\hat{\beta}_1 - z_\alpha \times SP\left(\hat{\beta}_1\right), \hat{\beta}_1 + z_\alpha \times SP\left(\hat{\beta}_1\right)\right]; \tag{18}$$

čia $PI_{1-\alpha}^{\beta_1}$ – pasikliautinumo intervalas krypties koeficientui $\beta_1$, su reikšmingumo lygmeniu $\alpha$; $\hat{\beta}_1$ – krypties koeficiento $\beta_1$ skaitinė aproksimacija; $z_\alpha$ – kritinė, dvipusio (angl. *two-tailed*) Z-testo, reikšmė[69,70,71]; $SP\left(\hat{\beta}_1\right)$ – standartinė krypties koeficiento $\beta_1$ paklaida, apskaičiuojama (19) formule[72]:

$$SP\left(\hat{\beta}_1\right) = \sqrt{\frac{n\sum_{i=1}^{n}(X_i - \bar{X})^2\hat{u}_i^2}{(n-2)[\sum_{i=1}^{n}(X_i - \bar{X})^2]^2}}; \tag{19}$$

čia $n$ – matavimų skaičius; $X_i$ – $i$-toji nepriklausomo kintamojo $X$ reikšmė; $\bar{X}$ – nepriklausomo kintamojo $X$ aritmetinis vidurkis; $\hat{u}_i$ – likučių skaitinė aproksimacija $i$-tajame taške.

### 1.3. Oro masių modeliavimas

Norint nagrinėti oro taršos modeliavimo metodologiją, reikia apžvelgti nuo ko prasidėjo oro kaip dalelių sandaros detalus matematinis modeliavimas. Oro masės, kaip sūkurių visumos, sąvoką pirmasis detaliai pradėjo nagrinėti Taylor‘as[73], kuris savo straipsnyje į oro masę pradėjo žiūrėti ne kaip į individualių sūkurių mozaiką, bet kaip į dinamišką sistemą, kurios kiekvienas elementas yra tam tikra prasme susijęs su kiekvienu kitu. Šiuo požiūriu į visos oro masės galutinę išeigą yra žiūrima lyg tai būtų bendras visų ją sudarančių sūkurių poveikio vidurkis. Taylor‘as savo straipsnyje apžvelgia trimatį modelį, kurio formuluotes labai apsunkina vertikalios – aukščio – dimensijos turėjimas, tačiau šiame magistro darbe į trimatę erdvę nebus atsižvelgiama ir bus naudojamas tik gerokai paprastesnis dvimatis modelis.

Šiame inovatyviame straipsnyje taip pat apžvelgiama kaip pasikeičia vėjo kryptis ir greitis atsižvelgiant į šių parametrų matavimo aukštį. Pasirodo, jog palei žemę (iki 200 metrų aukščio) „vėjo greitis nukrinta apie 40% lyginant su gradiento greičiu ir pakinta apie 20 laipsnių lyginant su gradiento kryptimi“[73].

Gamtinės prigimties straipsniuose galima rasti metodų, skirtų geoerdvinių duomenų naudojimui norint užpildyti trūkstamas vertes, o į tai žiūrint kita prasme – sukurti verčių nuspėjimo modelį.

Vienas iš jų vadinasi RBF ir šis metodas yra skirtas kritulių kiekio apskaičiavimui naudojant deterministinį interpoliavimą stočių gardelei[1]. Metodas yra skirtas vienmačiams duomenims, tačiau nėra konkrečiai apribotas kritulių temoje. Šiame metode (20) formule yra ieškoma vertės $\hat{z}(x_0)$:

$$\hat{z}(x_0) = \sum_{i=1}^{N} \lambda_i \phi\left(\left||x_i - x_0|\right|\right) + \mu; \tag{20}$$

čia $\hat{z}(x_0)$ – aproksimuojamas kritulių kiekis stotyje $x_0$; $N$ – papildomų stočių skaičius; $\phi(\cdot)$ – parinkta radialinė (spindulinė) bazinė funkcija, kitaip dar vadinama plonos plokštelės splainu (angl. *thin plate spline*), apskaičiuojama (21) formule:

$$\phi\left(\left|\left|x_i - x_0\right|\right|\right) = \left[\rho = \left|\left|x_i - x_0\right|\right|\right] = \phi(\rho) = c^2\rho^2 \ln(c\rho); \quad (21)$$

čia $x_i$ – $i$-tosios stoties koordinatės, $x_0$ – nagrinėjamos stoties koordinatės; $c$ – glodinimo parametras[74] (angl. *smoothing parameter*), randamas minimizuojant spėjimų reikšmių RMSE; $\rho$ – norma[4] tarp taškų $x_i$ ir $x_0$; $\lambda_i$ – $i$-tosios stoties svoris; $\mu$ – šališkumo (angl. *bias*) vertė.

Stoties svoris yra gaunamas kartu su šališkumo verte išsprendžiant (22) lygčių sistemą:

$$\begin{bmatrix} \boldsymbol{\Phi} & \mathbf{1} \\ \mathbf{1}' & 0 \end{bmatrix} \times \begin{bmatrix} \boldsymbol{\Lambda} \\ \mu \end{bmatrix} = \begin{bmatrix} \boldsymbol{Z} \\ 0 \end{bmatrix}; \quad (22)$$

čia $\boldsymbol{\Phi}$ – matrica, kurios *i, j* elemento reikšmė atitinka $\phi\left(\left|\left|x_i - x_j\right|\right|\right)$; $\mathbf{1}$ – vienetinis stulpelio vektorius; $\mathbf{Z}$ – stulpelinis, visų realių stotyse gautų verčių $z$, vektorius.

Taigi, čia apibūdinamas tik vienas iš daugelio galimų metodų vienmatės eilutės verčių nuspėjimui. Šiek tiek geografiškai labiau pritaikyti modeliai gali atsižvelgti į stočių aukštį, kas prideda dar vieną modelio dimensiją (pvz., GWR[1]), tačiau šių modelių kūrimas dažnai atsiremia į neuroninių tinklų architektūrą.

Viena iš svarbiausių ir efektyviausių sprendimo technikų vienmačių duomenų uždaviniuose yra stochastiniai interpoliavimo metodai iš krigingo metodų šeimos[1]. Paprastas krigingo metodo panaudojimas yra (23) formulėje[11]:

$$\hat{Z}(s_0) = \sum_{i=1}^{N} \lambda_i Z(s_i)\,; \quad (23)$$

čia $\hat{Z}(s_0)$ – aproksimuojama reikšmė geografiniame taške $s_0$; $N$ – išmatuotų reikšmių skaičius; $\lambda_i$ – nežinomas svoris išmatuotai reikšmei $i$-tajame taške; $Z(s_i)$ – išmatuota reikšmė $i$-tajame taške.

**Teršalų judėjimo atstumai**

Nors visi nagrinėti straipsniai pritaria idėjai, jog tolstant nuo taršos šaltinio, oro tarša mažėja, tačiau konkretūs taršos mažėjimo skaičiai nėra bendrai sutarti, kadangi galima tikėtis oro taršo padarinių netgi už 10000 kilometrų nuo šaltinio[80]. Negana to, kad pats taršos nukeliavimo atstumas varijuoja labai daug, Wylie apžvalgoje galime rasti citatą, dar labiau apsunkinančią nagrinėjamas situacijas – „oro tarša priklauso ne tik nuo vėjo greičio bei krypties, bet ir nuo topografijos“[81].

Judėjimo suvokimui ypač naudingi šaltiniai, detaliai apžvelgi antys teorinę matematinę taršos judėjimo pusę[82]. Viename iš jų paminima, jog ilgalaikės taršos aproksimavimas gali būti tikrai gana tikslus, tačiau kasdienės taršos skaičiavimai randa labai nepastovius rezultatus[83].

### 1.4. Oro taršos rodiklių poveikis sveikatai

Tikriausiai pagrindinė oro taršos problema visuomenės mastu yra neigiama įtaka sveikatai. Tikrai nemažai mokslininkų nagrinėja šią opią problemą įvairiais požiūriais.

[4] Straipsnyje šis dydis apibrėžiamas kaip radialinis (spindulinis) atstumas (angl. *radial distance*), tačiau tai atitinka normos apibrėžimą Euklido erdvėje[75,76,77,78,79].

Viename nagrinėtų straipsnių tyrėjai randa statistiškai reikšmingą sąsają tarp vaikų sergamumo alerginiu rinitu (TLK-10 AM klasifikacijoje atitinkančio AR J30.1–J30.4 kodus) ir $SO_2$ koncentracijos[84]. Norėtųsi paminėti, jog Alasauskas šiame tyrime neatsižvelgia į „statistinio reikšmingumo“ sąvokos nuvalkiojimą ir dalinai remiasi hipotezės priėmimu (kai $p = 0{,}021$), kai $p < 0{,}05$, nors remiantis didžiuliu mokslininkų konsensusu, norint tvirtinti statistinio reikšmingumo atradimą, reiktų naudotis $p < 0{,}005$ reikšme[85].

Kitame šaltinyje apskaičiuota sąsaja tarp greitosios pagalbos iškvietimų dėl hipertenzinių ligų (TLK-10 AM klasifikacijoje atitinkančių I05-I09 kodus) ir CO koncentracijos padidėjimo dieną prieš[86].

Dar vienas tyrimas besikoncentruojantis į tam tikro tipo vaikų sergamumo priklausomybę nuo taršos nagrinėja ryšį tarp vaikų sergamumo vėžiu (TLK-10 AM klasifikacijoje atitinkančio C00-C96 kodus) ir $NO_2$ bei dulkių[5] koncentracijų. Šio tyrimo išvadose teigiama, jog egzistuoja stiprus teigiamas ryšys tarp abiejų minėtų rodiklių koncentracijos ir vaikų susirgimo vėžiu[87].

Brunekreef‘o ir Holgate‘o apžvalginiame tyrime yra samprotaujama ne tik apie oro taršos poveikį sveikatai, bet netgi pagrįstai abejojama ar iš viso gali egzistuoti tam tikrų taršos rodiklių (PM ir ozono) „saugi“ riba, kadangi net minimalūs kiekiai neigiamai paveikia žmonių sveikatą[88], ypač kraujotakos bei kvėpavimo ligų kategorijose (TLK-10 AM klasifikacijoje atitinkančiose I00-I99 bei J00-J99 kodus atitinkamai).

### 1.4.1. Pasaulinė sveikatos organizacija

PSO yra parengusi pasaulio išsamų oro kokybės rekomendacijų dokumentą, kuriame detaliai apžvelgiami oro taršos poveikį žmonių sveikatai lemiančių rodiklių nagrinėjimo moksliniai straipsniai ir, remiantis konkrečiais duomenimis, išvedamos rekomendacijos visiems svarbiausiems oro kokybės rodikliams: kietosioms dalelėms (angl. *particulate matter*) ($PM_{2.5}$ ir $PM_{10}$), ozonui, azoto dioksidui, sieros dioksidui bei anglies monoksidui[7,89]. Šios rekomendacijos yra sudaromos ilgalaikiui (metų arba ozono atveju – didžiausiam 6 mėnesių) arba trumpalaikiui (24 valandų) laikotarpiams koncentracijos aritmetinio vidurkio (angl. *mean*) požiūriu.

PSO rengia savo rekomendacijų dokumentą remiantis mokslinių straipsnių rezultatų meta-analize, kurios metu yra sukuriama metodika apskaičiuoti rekomenduojamą specifinių oro teršalų ilgalaikę ir(arba) trumpalaikę ribą, kuri nurodytų teršalų koncentraciją, kuri nedaro reikšmingos įtakos populiacijos mirtingumui. Ši rodiklio riba yra vadinama AQG lygiu (angl. *air quality guideline level*). Taigi, minėtas AQG lygis yra gairė, kurios turėtų siekti valstybės, viršijančios ją. Tačiau rekomendacijų ribos ties ką tik paminėta verte nesibaigia, kadangi PSO pateikia ir rodiklių vertes, kurias siekiant užfiksuojamas mirtingumo padidėjimas. Šios vertės yra vadinamos tarpiniais tikslais (toliau – IT), kurių gali būti iki 4. Jų skaitomumo ir logiškos sąsajos ryšys yra šiek tiek painus, todėl verta paminėti, jog mažesnis lygio skaičius reiškia didesnę teršalų koncentraciją (todėl IT1 lygmenį pasiekus galima tikėtis didesnio mirtingumo nei pasiekus IT2 lygmenį).

Šie IT lygmenys lemia skirtingus mirtingumo pokyčius skirtingiems teršalams, todėl verta apžvelgti detaliau kiekvienos rekomendacijos balus.

#### $PM_{2.5}$ tarša

[5] Tyrime nėra specifikuota dulkių sąvoka, tačiau naudojant dedukciją galima nuspėti, kad autorius kalba apie $PM_{2.5}$ ir $PM_{10}$ koncentracijas.

***Ilgalaikė (metų)***

Jei populiacijos mirtingumas su AQG lygiu yra prilyginamas 100, tai mirtingumas bus:

- 104 siekiant[6] IT4;
- 108 siekiant IT3;
- 116 siekiant IT2;
- 124 siekiant IT1.

Šios projekcijos remiasi tiesiniu pavojaus santykiu (toliau – HR), kurio reikšmė yra 1,08 per 10 $\mu g/m^3$ rodiklio $PM_{2.5}$ koncentracijos padidėjimo, rastame PSO tyrime, atsižvelgiant į neatsitiktinį mirtingumą. Didesnėmis koncentracijomis tiesiškumas funkcijoje galimai prarandamas, kas lemia nevienodą skaičių kitimą.

***Trumpalaikė (24 valandų)[7]***

Jei populiacijos mirtingumas su AQG lygiu yra prilyginamas 100, tai mirtingumas bus:

- 101 siekiant IT4;
- 101 siekiant IT3;
- 102 siekiant IT2;
- 104 siekiant IT1.

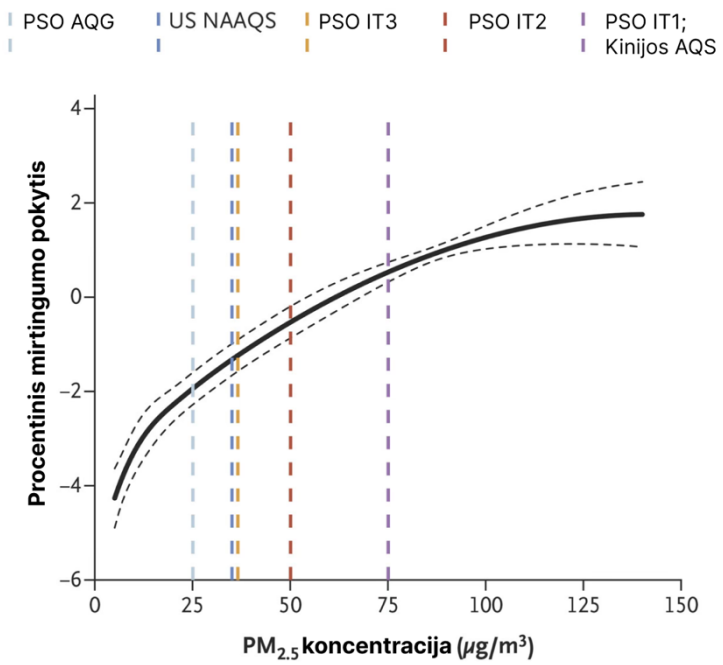
Šios projekcijos remiasi tiesiniu HR, kurio reikšmė yra 1,0065 per 10 $\mu g/m^3$ rodiklio $PM_{2.5}$ koncentracijos padidėjimo, rastame PSO tyrime, atsižvelgiant į neatsitiktinį mirtingumą.

Matoma, jog trumpalaikėje analizėje procentinis skirtumas yra labai nedidelis lyginant IT4 ir IT3, taigi apvalinant gaunama ta pati reikšmė – reiškia skirtumas mirtingumo požiūriu yra labai nedidelis.

Toliau pridedamas paveikslas, kuris leidžia pamatyti mirtingumo pokytį priklausomai nuo rodiklio koncentracijos tam tikru metu. Turint šį grafiką, galima pastebėti, kodėl yra naudojami tiesiniai HR modeliai (kadangi vaizduojama kreivė gali būti neblogai aproksimuojama tiesiniu modeliu) ir kaip šie tiesiniai modeliai gali varijuoti savo tikslumu, priklausomai nuo turimų duomenų.

[6] Pasiekti lygį reiškia neviršyti jo ribos. Tai IT4 lygis skaitytųsi pasiektas, jeigu matuojamo rodiklio vidurkio rezultatas pagal vertę būtų didesnis už AQG rekomendaciją, bet mažesnis už IT4 vertę.

[7] Skirtingai nuo ilgalaikio, kuris yra tikrinamas vidurkio prasme visiems metams, trumpalaikis AQG lygis skaitosi pasiektas, jei jo rezultatas yra pasikartoja 24 valandų vidurkio apskaičiavime bent 1 procentą viso laiko (3-4 dienas per metus).




**5 pav.** Koncentracijos ir atsako kreivė (angl. *pooled concentration-response curve*). Vaizduojama koncentracijos ir atsako kreivė, susijusi su 2 dienų slenkančio vidurkio $KD_{2.5}$ koncentracijos sąsajomis su kasdieniu mirtingumu dėl visų priežasčių. Vertikalioji ašis rodo procentinį skirtumą nuo bendro vidutinio poveikio (gauto iš viso $KD_{2.5}$ koncentracijos diapazono kiekvienoje vietovėje) mirtingumui. Nulis vertikalioje ašyje nurodo bendrą vidutinį poveikį, o kreivės dalis, esanti žemiau nulio, reiškia mažesnį įvertį nei vidutinis poveikis. Punktyrinėmis linijomis pažymėtos oro kokybės gairės arba standartai, taikomi 24 valandų vidutinei $KD_{2.5}$ koncentracijai pagal PSO AQG[8], PSO IT1, PSO IT2, PSO IT3, JAV nacionalinį aplinkos oro kokybės standartą (US NAAQS) ir Kinijos oro kokybės standartą (Kinijos AQS)[90].

Nagrinėjant **5 pav.** matoma, kaip iš tiesų gali atrodyti mirtingumo kreivė ir kokį mirtingumo pokytį nusako skirtingi AQG lygiai (šis paveikslas skirtas tik grafiniam palyginimui, kadangi dabartinės AQG rekomendacijos čia nėra vaizduojamos).

### $PM_{10}$ tarša

#### *Ilgalaikė (metų)*

Jei populiacijos mirtingumas su AQG lygiu yra prilyginamas 100, tai mirtingumas bus:

- 102 siekiant IT4;
- 106 siekiant IT3;
- 114 siekiant IT2;
- 122 siekiant IT1.

Šios projekcijos remiasi tiesiniu HR, kurio reikšmė yra 1,04 per 10 $\mu g/m^3$ rodiklio $PM_{10}$ koncentracijos padidėjimo, rastame PSO tyrime, atsižvelgiant į neatsitiktinį mirtingumą.

#### *Trumpalaikis (24 valandų)*

Jei populiacijos mirtingumas su AQG lygiu yra prilyginamas 100, tai mirtingumas bus:

[8] Šiame grafike vaizduojami PSO rekomendacijų standartai iš senesnio, nuo 2021 m. nebegaliojančio, PSO rekomendacijų dokumento.

– 100,2 siekiant IT4;
– 101 siekiant IT3;
– 102 siekiant IT2;
– 104 siekiant IT1.

Šios projekcijos remiasi tiesiniu HR, kurio reikšmė yra 1,0041 per 10 $\mu g/m^3$ rodiklio $PM_{10}$ koncentracijos padidėjimo, rastame PSO tyrime, atsižvelgiant į neatsitiktinį mirtingumą.

**$O_3$ tarša**

***Ilgalaikė (piko laikotarpio[9])***

Jei populiacijos mirtingumas su AQG lygiu yra prilyginamas 100, tai mirtingumas bus:
– 101 (102[10]) siekiant IT2;
– 104 (108) siekiant IT1.

Šios projekcijos remiasi tiesiniu HR, kurio reikšmė yra 1,01 per 10 $\mu g/m^3$ rodiklio $O_3$ koncentracijos padidėjimo, rastame PSO tyrime, atsižvelgiant į neatsitiktinį mirtingumą. Kvėpavimo takų mirtingumo projekcijos remiasi tiesiniu HR, kurio reikšmė yra 1,02 per 10 $\mu g/m^3$ rodiklio $O_3$ koncentracijos padidėjimo.

***Trumpalaikė (8 valandų)***

Jei populiacijos mirtingumas su AQG lygiu yra prilyginamas 100, tai mirtingumas bus:
– 101 siekiant IT2;
– 103 siekiant IT1.

Šios projekcijos remiasi tiesiniu HR, kurio reikšmė yra 1,0043 per 10 $\mu g/m^3$ rodiklio $O_3$ koncentracijos padidėjimo, rastame PSO tyrime, atsižvelgiant į neatsitiktinį mirtingumą.

**$NO_2$ tarša**

***Ilgalaikė (metų)***

Jei populiacijos mirtingumas su AQG lygiu yra prilyginamas 100, tai mirtingumas bus:
– 102 (103) siekiant IT3;
– 104 (106) siekiant IT2;
– 106 (109) siekiant IT1.

Šios projekcijos remiasi tiesiniu HR, kurio reikšmė yra 1,02 per 10 $\mu g/m^3$ rodiklio $NO_2$ koncentracijos padidėjimo, rastame PSO tyrime, atsižvelgiant į neatsitiktinį mirtingumą. Kvėpavimo takų mirtingumo projekcijos remiasi tiesiniu HR, kurio reikšmė yra 1,03 per 10 $\mu g/m^3$ rodiklio $NO_2$ koncentracijos padidėjimo.

***Trumpalaikė (24 valandų)***

[9] Piko laikotarpiu (angl. *peak season*) yra laikomas didžiausio koncentracijos vidurkio šešių mėnesių iš eilės laikotarpis, kai vidurkis skaičiuojamas pagal 8 valandų maksimalų vidurkį paroje.
[10] Skliaustuose yra rodomas prilyginimas mirtingumo santykiui, atsižvelgiant tik į kvėpavimo takų ligų mirtingumą (angl. *respiratory mortality*).

Jei populiacijos mirtingumas su AQG lygiu yra prilyginamas 100, tai mirtingumas bus:
- 102 siekiant IT2;
- 107 siekiant IT1.

Šios projekcijos remiasi tiesiniu HR, kurio reikšmė yra 1,0072 per 10 $\mu g/m^3$ rodiklio $NO_2$ koncentracijos padidėjimo, rastame PSO tyrime, atsižvelgiant į neatsitiktinį mirtingumą.

**$SO_2$ tarša**

***Trumpalaikė (24 valandų)***

Jei populiacijos mirtingumas su AQG lygiu yra prilyginamas 100, tai mirtingumas bus:
- 101 siekiant IT2;
- 105 siekiant IT1.

Šios projekcijos remiasi tiesiniu HR, kurio reikšmė yra 1,0059 per 10 $\mu g/m^3$ rodiklio $SO_2$ koncentracijos padidėjimo, rastame PSO tyrime, atsižvelgiant į neatsitiktinį mirtingumą.

**CO tarša**

***Trumpalaikė (24 valandų)***

Jei populiacijos mirtingumas nuo miokardo infarkto su AQG lygiu yra prilyginamas 100, tai mirtingumas nuo miokardo infarkto bus:
- 106 siekiant IT1.

Šios projekcijos remiasi tiesiniu HR, kurio reikšmė yra 1,019 per 1 $mg/m^3$ rodiklio CO koncentracijos padidėjimo, rastame PSO tyrime, atsižvelgiant į mirtingumą nuo miokardo infarkto.

**Bendra lentelė su tarpiniais tikslais ir jų vertėmis**

Pastarieji išvardinti skyreliai su kiekvieno elemento tarpinių tikslų (IT) mirtingumo reikšmėmis buvo skirti parodyti, kokį poveikį visuomenės sveikatai gali lemti tam tikro IT lygmens pasiekimas. Žemiau pavaizduojama lentelė, kurioje nurodomi skirtingų IT ribų pasiekimo skaitiniai įverčiai visiems matuojamiems elementams.

**2 lentelė.** PSO siūlomi ilgalaikiai bei trumpalaikiai teršalų AQG lygiai ir tarpiniai tikslai.

| **Teršalas** | **Laikotarpis** | **IT1** | **IT2** | **IT3** | **IT4** | **AQG lygis** |
|---|---|---|---|---|---|---|
| $PM_{2.5}$, $\mu g/m^3$ | Ilgalaikis (metų) | 35 | 25 | 15 | 10 | 5 |
| | Trumpalaikis (24 valandų) | 75 | 50 | 37,5 | 25 | 15 |
| $PM_{10}$, $\mu g/m^3$ | Ilgalaikis (metų) | 70 | 50 | 30 | 20 | 15 |
| | Trumpalaikis (24 valandų) | 150 | 100 | 75 | 50 | 45 |
| $O_3$, $\mu g/m^3$ | Ilgalaikis (piko sezono) | 100 | 70 | - | - | 60 |
| | Trumpalaikis (8 valandų) | 160 | 120 | - | - | 100 |
| $NO_2$, $\mu g/m^3$ | Ilgalaikis (metų) | 40 | 30 | 20 | - | 10 |
| | Trumpalaikis (24 valandų) | 120 | 50 | - | - | 25 |
| $SO_2$, $\mu g/m^3$ | Trumpalaikis (24 valandų) | 125 | 50 | - | - | 40 |
| CO, $mg/m^3$ | Trumpalaikis (24 valandų) | 7 | - | - | - | 4 |

### 1.4.2. Europos Sąjungos direktyva

Iš dalies panašiai kaip ir PSO, bet teisiniu požiūriu įpareigojančiai, Europos Parlamentas ir Taryba 2008 metais išleido direktyvą[91] (vėliau pakoreguotą 2015 m.[92]), įpareigojančią ES nares valstybes vykdyti įvairius, su oro kokybės vertinimu susijusius, uždavinius, tarp kurių pirmasis skamba taip: „Ši direktyva nustato priemones, kuriomis siekiama suformuluoti ir nustatyti aplinkos oro kokybės tikslus, skirtus išvengti, užkirsti kelią arba sumažinti žalingą poveikį žmonių sveikatai ir visai aplinkai“. Šioje direktyvoje yra išskirti du ribinių verčių sudarymo priežastingumai:

- Žmonių sveikatos apsauga;
- Augmenijos apsauga.

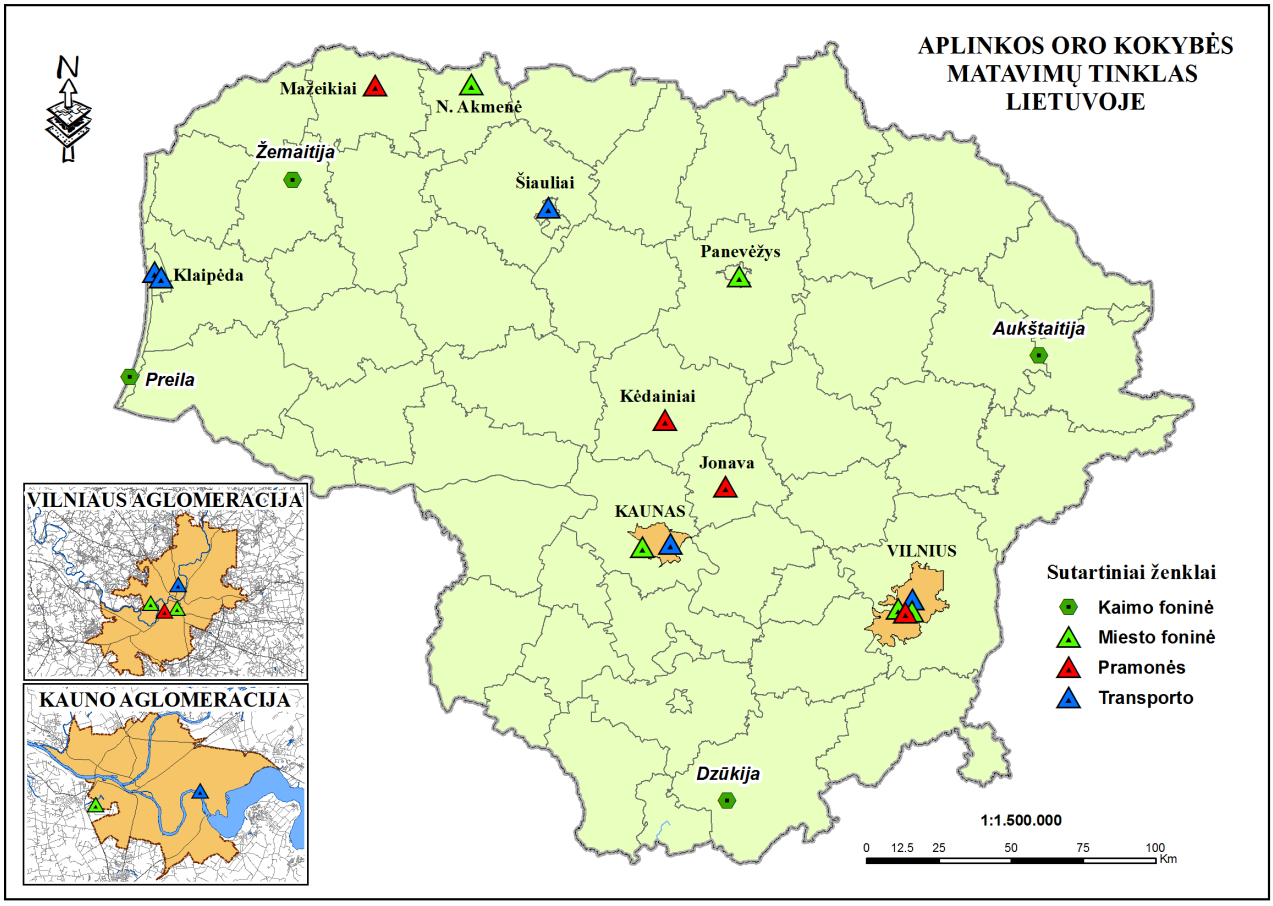
Priežastingumai tarpusavyje neprieštarauja vienas kito egzistavimui, tačiau gali skirtis teršalų kiekio ribinės vertės priklausomai nuo priežastingumo. Žinant tai, reikia paminėti, jog į ES direktyvos ribas šiame tyrime nebus atsižvelgiama, tačiau šios direktyvos dėka, į šiame tyrime nagrinėjamų rodiklių bendrą sąrašą, prie anksčiau išvardintų, PSO rekomendacijose minėtų, teršalų prisideda švinas (Pb) bei benzeno netiesioginė[11] atmaina – heksachlorobenzenas.

[11] Heksachlorobenzenas nesusidaro iš benzeno natūraliomis sąlygomis.

## 2. Duomenys ir tyrimo metodai

### Lietuvos oro monitoringo stotys

Lietuvos teritorijoje oro kokybės matavimo stočių tinklas yra išdėstytas remiantis ES direktyvomis, kas apima po kelias stotis didžiuosiuose miestuose (Vilniuje, Kaune ir Klaipėdoje), pasienyje (kaimo foninė tarša) bei pramoninėse vietovėse.




**6 pav.** Aplinkos oro kokybės matavimų tinklas Lietuvoje[93].

Kiekviena stotis turi savo aprašų puslapius, kuriuose išvardinti matuojami parametrai. Sekančioje lentelėje yra išvardinami stočių matuojami meteorologiniai bei taršos matavimo parametrai, tačiau yra apsiribojama tik aktualiais, PSO gairėse bei ES direktyvoje minimais teršalų rodikliais.

**3 lentelė.** Bendrųjų meteorologinių rodiklių[12] bei PSO rekomendacijoms (žr. skyrių 1.4.1) aktualių teršalų matavimo prieinamumas oro monitoringo stotyse.

| Oro matavimo stotis | Stoties paskirties kategorija | Meteorologiniai matavimai | | | | | Teršalų matavimai | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | T | p | ϕ | v | θ | $NO_2$ | CO | $O_3$ | $SO_2$ | $KD_{2.5}$ | $KD_{10}$ |
| Mobili laboratorija[94] | Mobili | + | + | + | + | + | + | + | + | + | + | + |
| Naujoji Akmenė[95] | Miesto foninė | + | + | + | + | + | | | | + | + | + |
| Jonava[96] | Pramonės | + | + | + | + | + | + | | + | | | + |

[12] Meteorologinių matavimų santrumpų paaiškinimas (iš kairės į dešinę): T – temperatūra, p – slėgis, ϕ – santykinis drėgnis, v – vėjo greitis, θ – vėjo kryptis.

| Oro matavimo stotis | Stoties paskirties kategorija | Meteorologiniai matavimai | | | | | Teršalų matavimai | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | T | p | ϕ | v | θ | $NO_2$ | CO | $O_3$ | $SO_2$ | $KD_{2.5}$ | $KD_{10}$ |
| Kaunas, Noreikiškės[97] | Miesto foninė | + | + | + | + | + | + | + | + | + | + | + |
| Kaunas, Petrašiūnai[97] | Transporto | + | + | + | + | + | + | + | + | + | + | + |
| Kėdainiai[98] | Pramonės | + | + | + | + | + | + | | + | + | | + |
| Klaipėda, Centras[99] | Transporto | + | + | + | + | + | + | + | | + | | + |
| Klaipėda, Šilutės plentas[99] | Transporto | + | + | + | + | + | + | + | + | | + | + |
| Mažeikiai[100] | Pramonės | + | + | + | + | + | + | | + | + | | + |
| Šiauliai[101] | Transporto | + | + | + | + | + | + | + | + | + | | + |
| Vilnius, Savanorių pr.[102] | Pramonės | + | + | + | + | + | + | + | | + | | + |
| Vilnius, Žirmūnai[102] | Transporto | + | + | + | + | + | + | + | + | | + | + |
| Vilnius, Senamiestis[102] | Miesto foninė | + | + | + | + | + | + | + | | + | | + |
| Vilnius, Lazdynai[102] | Miesto foninė | + | + | + | + | + | + | | + | + | + | + |
| Žemaitija[103] | Kaimo foninė | | | | | | | | + | | | |
| Panevėžys[104] | Miesto foninė | + | + | + | + | + | + | + | + | | | + |
| Dzūkija[105] | Kaimo foninė | + | + | + | + | + | | | + | | | |
| Aukštaitija[106] | Kaimo foninė | + | + | + | + | + | | | + | | +[13] | |
| Preila[14] | Kaimo foninė | | | | | | | | | | | |

## 2.1. Atviri Lietuvos oro taršos duomenų šaltiniai

Lietuvoje atvirai yra pateikiami kelių rūšių duomenų rinkiniai, aktualūs oro monitoringo tematikoje[15]:

- Lietuvos valstybinio oro monitoringo automatinių stočių matavimo duomenys[107];
- Integruoto monitoringo duomenys[108];
- Lietuvos valstybinio oro monitoringo matavimo duomenys (pusiau automatinių tyrimų)[109];
- Į aplinkos orą išmetamų teršalų apskaitos duomenys[110].

### Lietuvos valstybinio oro monitoringo automatinių stočių matavimo duomenys

Naudingiausias duomenų šaltinis tikriausiai būtų Lietuvos valstybinio oro monitoringo automatinių stočių matavimo duomenys (toliau – automatinis monitoringas), kadangi šio šaltinio duomenys yra sudaromi iš matavimų, atliekamų kas pusvalandį arba kas valandą, kas teikia naudingiausią informaciją nedidelio laiko tarpo skalėje. Beje, norint oro monitoringo duomenis panaudoti bet kokiam mašininio mokymosi modeliui, kurio veikimo principas yra paremtas gyvu (angl. *real-time*) duomenų skaičiavimu, šie duomenys tikriausiai gautų pirmenybę lyginant su kitais.

Deja, trumpai peržvelgus duomenis pasimato, jog duomenys toli gražu nėra puikiai užpildyti, todėl egzistuoja reali galimybė, kaip ir Yozgatligili'o atliktoje Turkijos monitoringo stočių duomenų apžvalgoje: „50-60% aktyvių stočių duomenų yra atmetama dėl nehomogeniškumo ir trūkstamų verčių skaičiaus“[34].

[13] $KD_{2.5}$ cheminė sudėtis.

[14] Tyrimo metu Preilos stotį aprašančio puslapio nėra.

[15] Šioje vietoje minimi monitoringo duomenys, kurie yra rengiami pagal ES direktyvas (žr. skyrių 1.4.2).

**Integruoto monitoringo duomenys**

Šis duomenų rinkinys yra naudingas ne tik oro monitoringo prasme, kadangi yra apimami įvairių sričių, konkrečios vietovės įvairios trukmės (trukmė turi pradinę bei galutinę datas, bet neturi valandinio tikslumo) tyrimai.

Deja, šie integruoto monitoringo duomenys nėra pilnavertiškai atverti[16], todėl nėra galimybių susieti tyrimų rezultatų kartu su jų metodais ir rodikliais, kuriuos rezultatai nusako. Taip yra todėl, kad nėra raktų, siejančių tyrimo rezultatus (stulpelis „trz_id“ iš lentelės „TyrimoRezultatas“) su tyrimo metaduomenimis (esančiais lentelėse „Tyrimas“ bei „TyrimoSudedamoji“). Pakoregavus duomenų atvėrimo procesą ir susiejus rezultatus su jų metaduomenimis, šiuos duomenis galima naudoti gana detaliai tendencijų analizei pagal matavimo vietovę.

**Lietuvos valstybinio oro monitoringo matavimo duomenys (pusiau automatinių tyrimų)**

Pažvelgus į šį duomenų rinkinį galima manyti[17], jog tai yra praėjusiame skyrelyje aptartų duomenų išviešinimas aplinkos oro tyrimų terpės specifikacijoje. Duomenų aprašui galioja tos pačios tiesos, kaip ir integruotiems monitoringo duomenims – paminėta tyrimo vieta, tyrimo atlikimo laikotarpis dienomis bei skaitinis rezultatas.

**Į aplinkos orą išmetamų teršalų apskaitos duomenys**

Į aplinkos orą išmetamų teršalų apskaitos duomenys (toliau – teršalų apskaitos duomenys) yra naudingi nagrinėjant teršalų kiekio pokyčius pagal skirtingas taršos šaltinių kategorijas. Kartu su atskiromis kategorijomis yra pateikiama ir „Iš viso“ kategorija, kurioje žymima bendra teršalo metinės sumos vertė.

Šie duomenys yra vedami metų laikotarpiui, todėl nieko lokalaus ar net sezoniško jais negalima įžvelgti, tačiau jų nauda atsiranda nagrinėjant metinio inkremento laiko eilutę.

Šiuose atviruose duomenyse galima rasti vieną problemą – „nfr_kodas“ stulpelio neužpildymas metaduomenų lentelėje verčia gilintis į oro taršos šaltinių klasifikavimo gidą[111].

## 2.2. Atviri Lietuvos sveikatos duomenys

Lietuvoje atvirai yra pateikiami kelių rūšių duomenų rinkiniai, aktualūs diagnozių bei mirtingumo tematikoje:

- Įvairių diagnozių TLK-10-AM kodų dažniai[112];
- Higienos instituto pateikiami duomenų rinkiniai[113];
- Širdies ir kraujagyslių ligomis sergančių pacientų apsilankymų statistiniai duomenys[114];
- Nuolatinių Lietuvos gyventojų, mirusių Lietuvoje ar užsienyje, ir nuolatinių Lietuvos Respublikos gyventojų negyvagimių mirties atvejų ir jų priežasčių duomenys[115];

**Įvairių diagnozių TLK-10-AM kodų dažniai**

Įvairių diagnozių TLK-10-AM kodų dažniai (toliau – TLK duomenys) yra duomenų rinkinys, kuriame galima rasti dvi duomenų lenteles:

[16] Komentaras atspindi situaciją tyrimo metu.

[17] Informacijoje apie duomenų rinkinį tai nėra parašyta, tačiau pagal sutapimus galima nuspėti.

- Diagnozių dažnis pagal E003 formos duomenis (lentelė „EspbiDiagnoze003“);
- Diagnozių dažnis pagal E025 formos duomenis (lentelė „EspbiDiagnoze025“).

Abi lentelės turi kodų sąrašus, kuriuos reiktų atsifiltruoti pagal poreikį, norint nagrinėti specifinius duomenis. Kadangi šiuos duomenis programinėje įrangoje nagrinėti yra gana nepatogu dėl jų kiekio, kategorijų atsirinkimui rankiniu būdu yra sukurta TLK-10-AM klasifikatorių knyga[116,117].

Panagrinėjus duomenų sandarą, galima pastebėti, jog duomenų rinkimas abiejose lentelėse prasideda tik nuo 2017 metų, kas gali lemti mažesnes galimybes panaudoti šiuos duomenis ilgalaikėje metinėje analizėje.

Taip pat, kadangi šie duomenys nėra sutvarkyti oficialiai statistikai vesti, juose yra daug dublikatų, kadangi kiekvienas ESPBI IS įrašas su tam tikru TLK-10-AM kodu lemia šio įrašo užfiksavimą duomenyse.

**Higienos instituto pateikiami duomenų rinkiniai**

Higienos instituto pateikiami duomenų rinkiniai yra sudaryti iš oficialiųjų sveikatos statistikų, kurias galima atsifiltruoti pačiam naudotojui.

**Širdies ir kraujagyslių ligomis sergančių pacientų apsilankymų statistiniai duomenys**

Šiame šaltinyje pateikiami duomenis ne iš visų Lietuvos sveikatos įstaigų, tačiau jie galėtų tikti nagrinėjant taršos įtaką ne tik mirtingumui, bet ir širdies ir kraujagyslių ligomis sergančių žmonių diagnozių priklausomybei.

**Nuolatinių Lietuvos gyventojų, mirusių Lietuvoje ar užsienyje, ir nuolatinių Lietuvos Respublikos gyventojų negyvagimių mirties atvejų ir jų priežasčių duomenys**

Nuolatinių Lietuvos gyventojų, mirusių Lietuvoje ar užsienyje, ir nuolatinių Lietuvos Respublikos gyventojų negyvagimių mirties atvejų ir jų priežasčių duomenys turi vieną, šiam tyrimui naudingą, duomenų lentelę „NuolatinisGyventojas“, kurioje yra naudingos informacijos apie kasmėnesinius mirčių skaičius. Pažiūrėjus į lentelės metaduomenis galėtų atrodyti, jog šioje lentelėje galima rasti ir mirties priežastis stulpelyje „gnmpd“ (Gydytojo nuomone, mirties rūšis), tačiau šis stulpelis yra nelabai vertingas, kadangi duomenys yra pseudonimizuoti, todėl mirties priežastis tampa skaičiumi nuo 1 iki 10, tačiau jokio paaiškinimo reikšmėms nėra[18].

### 2.3. Papildomi meteorologiniai duomenys iš Open-Meteo

Kadangi oro monitoringo stotys gali neturėti visų meteorologinių duomenų, verta turėti papildomą duomenų generavimo šaltinį, kuris būtų taip pat atvirai prieinamas. Tam galima pasinaudoti atvira API jungtimi[118], kuri pateikia istorinius valandinius matavimus tokiems rodikliams kaip temperatūra, slėgis, santykinis drėgnis, vėjo greitis bei vėjo kryptis.

### 2.4. Duomenų standartizavimas

Prieš atliekant skaitinę analizę, privalu standartizuoti turimus monitoringo duomenis. Kadangi monitoringo duomenys yra surenkami skirtingiems rodikliams, skirtingose stotyse, tai reiškia, jog

[18] Nors mirčių priežasčių aprašų lentelėje nėra, tačiau panaudojus atgalinę duomenų inžineriją galima palyginti gautas statistikas su oficialiosiomis, tokiu būdu randant klasifikacijų atitikmenis.

neapdirbti (angl. *raw*) duomenys gali būti įvairių formatų. Turėti skirtingų formatų duomenis skirtingiems rodikliams savaime nėra neteisinga, kadangi bet kokiu atveju matavimai yra atliekami visai skirtingose srityse (pvz., temperatūros, slėgio ir elemento koncentracijos matavimai niekaip negali būti vienodų matavimo vienetų), tačiau problema iškyla turint vienodo rodiklio matavimus skirtingais matavimo vienetais. Šios problemos pavyzdžiai turimuose monitoringo duomenyse yra tokie:

- Temperatūros matavimai Celsijaus laipsniais (℃) ir Kelvinais (*K*);
- Elemento koncentracijos matavimai dalimis milijone (*ppm*), dalimis milijarde (*ppb*) ir mikrogramais kubiniame metre ($\mu g/m^3$).

**Temperatūros standartizavimas**

Temperatūros konversijos formulė yra labai paprasta[119], kadangi norint paversti Celsijaus laipsnius Kelvinais tereikia pridėti konstantą: $T_K = t_{℃} + 273.15$.

**Koncentracijos standartizavimas**

Skirtumas tarp koncentracijos matavimo dalimis milijone (*ppm*) ir dalimis milijarde (*ppb*) yra akivaizdus, nes $C_{ppm} = C_{ppb} \cdot 1000$.

Norint paversti koncentracijos matavimą dalimis milijarde (*ppb*) į standartinį matavimo vienetą, reikia atkreipti dėmesį į PSO nustatytas gaires[7], kuriose visur yra naudojamas mikrogramų kubiniame metre mato vienetas ($\mu g/m^3$) (kaip matome ir ribas **2 lentelė**)[19].

Konvertavimo formuluotė yra gana nestandartiška, kadangi skirtingi šaltiniai, priklausomai nuo jų pagrindinės paskirties, gali naudoti nevienodą aproksimavimo lygmenį. Tarkim, vienas šaltinis[121] gali nurodyti koncentracijos konvertavimą apskaičiuoti (24) formule:

$$C_{\mu g/m^3} = \frac{C_{ppb} \cdot 12.187 \cdot M}{T}; \tag{24}$$

čia $C_{\mu g/m^3}$ – elemento masės koncentracija ($\mu g/m^3$); $C_{ppb}$ – elemento koncentracija dalelėmis milijarde (*ppb*); *M* – elemento molinė masė (*g/mol*); *T* – temperatūra (*K*).

Šio skaičiavimo būdo problema yra ta, kad remiamasi statine atmosferinio slėgio verte (*1 atm.*) bei neturima naudojamos konstantos paaiškinimo. Taigi, norint turėti šiek tiek tikslesnį skaičiavimą, reiktų atsižvelgti ir į dinamiško atmosferos slėgio naudojimą ir turėti galimybę atsekti visas naudojamas konstantas iki jų šaknų.

Internete apstu svetainių[122,123,124], kurių tikslas yra padėti konvertuoti šias koncentracijas tarpusavyje, o šalia konvertavimo langelių yra paminėtos ir pilnos formulės, iš kurių, naudojant standartinius[20] temperatūros, slėgio bei dujų molinio tūrio dydžius[125,126,127], galima išvesti ir galutinį, dinaminį koncentracijų konvertavimo apskaičiavimo būdą su (25) formule:

[19] Koncentracijos sąvoka šiame konvertavimo procese apibūdina kelis iš esmės skirtingus matavimus[120]. Galimų koncentracijų pavyzdžiai: masės koncentracija, tūrio koncentracija, dalys milijarde ir t.t.

[20] Standartiniai slėgis bei temperatūra[125] yra pavaizduoti su STP prierašu, o standartinis molinis tūris yra pavaizduotas kaip $V_m$.

$$C_{\mu g/m^3} = C_{ppb} \cdot \frac{M}{V} = C_{ppb} \frac{M}{V_m \cdot \frac{T}{T_{STP}} \cdot \frac{p_{STP}}{p}} = \frac{C_{ppb} \cdot M \cdot T_{STP} \cdot p}{V_m \cdot T \cdot p_{STP}} = \tag{25}$$

$$= \frac{C_{ppb} \cdot M \cdot p}{T} \cdot \frac{273.15\ \mathrm{K}}{22.711 \frac{l}{mol} \cdot 10^3 \frac{m^3}{mol} \cdot 10^2\ Pa};$$

čia $C_{\mu g/m^3}$ – elemento masės koncentracija ($^{\mu g}/_{m^3}$); $C_{ppb}$ – elemento koncentracija dalelėmis milijarde (ppb); M – elemento molinė masė ($g/mol$); V – tūris (l); $V_m$ – standartinis molinis tūris ($l/mol$); $T$ – temperatūra (K); $T_{STP}$ – standartinė temperatūra (K); $p_{STP}$ – standartinis slėgis ($Pa$); p – slėgis ($Pa$).

## 2.5. Vėjo vektorių vidurkiai

Norint išmatuoti vėjo vektorių vidurkį tose vietose, kur trūksta tarpinės vertės laiko požiūriu (pvz., yra 13:00 val. ir 14:00 val. matavimai, tačiau reiktų turėti vertę 13:30 val.), krypčiai apskaičiuoti galime naudotis vienareikšme formule, o vėjo greičio nuspėjimui galime naudoti vieną iš dviejų galimybių: vektorinį arba skaliarinį vidurkį[128]. Kaip minima šaltinyje, vektorinis vidurkis sąlygomis, kai vėjo kryptis smarkiai pasikeičia, gali rodyti nelabai logišką vertę (jei kryptis keičiasi 180 laipsnių, o stiprumas išlieka tas pats, tada vektorinis vidurkis bus lygus nuliui), todėl šio tyrimo metu bus naudojama skaliarinė vidurkio išraiška.

Skaliarinio vėjo greičio vidurkio skaičiavime naudojama paprasta aritmetinio vidurkio (26) formulė:

$$\bar{u} = \frac{1}{n} \sum_{i=1}^{n} u_i; \tag{26}$$

čia $n$ – matavimų skaičius; $u_i$ – vėjo greitis ($m/s$) $i$-tuoju laiko momentu.

Kadangi matuojamas tik dviejų artimiausių narių vidurkis, gauname (27) formulę:

$$\bar{u}_t = \frac{u_{t-1} + u_{t+1}}{2}; \tag{27}$$

čia $\bar{u}_t$ – ieškoma vėjo greičio vertė laiko momentu $t$, $u_{t-1}, u_{t+1}$ – kaimyninės (prieš ir po) vėjo greičio vertės.

Vėjo krypties vidurkiui $\overline{\Theta_{RV}}$ nustatyti reikia šiek tiek labiau komplikuotų skaičiavimų, kadangi iš pradžių reikia susirasti vektoriaus komponentes $\vec{u}$ ir $\vec{v}$, o kartu dar naudoti ir krypties laipsninę išraišką[21] $i$-tuoju laiko momentu – $\theta_i$. Taigi, vėjo krypties vidurkis apskaičiuojamas (28) formule:

$$\overline{\Theta_{RV}} = \arctan\left(\frac{\vec{u}}{\vec{v}}\right) + \mathrm{flow}; \tag{28}$$

čia $\overline{\Theta_{RV}}$ – vėjo krypties vidurkis; $\vec{u}, \vec{v}$ – vėjo krypties komponentės, apskaičiuojamos (29) ir (30) formulėmis:

[21] Laipsniai yra skaičiuojami nuo šiaurės krypties ir didėja palei laikrodžio rodyklę.

$$\vec{u} = -u_i \times \sin\left[2\pi \times \frac{\theta_i}{360}\right]; \tag{29}$$

$$\vec{v} = -u_i \times \cos\left[2\pi \times \frac{\theta_i}{360}\right]; \tag{30}$$

čia $u_i$ – vėjo greitis (*m/s*) *i*-tuoju laiko momentu; $\theta_i$ – vėjo krypties laipsninė išraiška *i*-tuoju laiko momentu; flow – krypties patikslinimo faktorius, apskaičiuojamas (31) formule:

$$\text{flow} = \begin{matrix} +180, \text{ kai } \arctan\left(\frac{\vec{u}}{\vec{v}}\right) < 180, \\ -180, \text{ kai } \arctan\left(\frac{\vec{u}}{\vec{v}}\right) > 180. \end{matrix} \tag{31}$$

## 2.6. Oro taršos modeliavimo ypatumai

Šio tyrimo keblumas išryškėja nagrinėjant turimus monitoringo duomenis. Nors meteorologinių duomenų papildymas nėra praktinė problema dėl tokių įrankių kaip „Open-Meteo“ (žr. skyrių 2.3), tačiau teršalų papildomų (lokalios erdvės požiūriu) duomenų integracija tampa nauju iššūkiu. Kadangi oro taršos monitoringo stočių atverti duomenys apima tik 19 stočių[107] (žr. skyrių 2), tai kyla klausimas – kaip susirasti daugiau naudingų oro taršos duomenų?

### Palydovinių duomenų naudojimas modeliavimui

Vienas iš šaltinių, matuojančių oro taršą globaliu mastu yra palydovai. Kadangi šio tyrimo metu nagrinėjamos oro masės yra antžeminės (oro taršos monitoringo stotys yra nutolusios vos per kelis metrus nuo žemės). Oro masių judėjimo greitis ir krypties kampas keičiasi priklausomai nuo atstumo nuo žemės paviršiaus[73], todėl palydovų fiksuojamą oro taršą nebūtų tikslinga naudoti kuriant dvimačius paviršinių oro masių modelius.

### Nuo meteorologinių sąlygų priklausantis oro taršos rodiklių ir šaltinių sąsajos modelis

Vienas iš šio tyrimo uždavinių yra paieškoti ryšių tarp oro taršos šaltinių ir oro taršos rodiklių. Kadangi turimi duomenys, fiksuojantys oro taršos koncentraciją kas pusvalandį ir su tais duomenimis susieti meteorologiniai rodikliai, jais galima pasinaudoti sudarant gana ilgą ir dinamišką[22] laiko eilutę.

Deja, nėra pakankamai tikslių (mažos laiko skalės prasme) oro taršos šaltinių duomenų, tačiau egzistuoja atviras duomenų šaltinis, suteikiantis prieigą prie stacionarių taršos židinių duomenų[129]. Šie duomenys aprėpia beveik 14000 Lietuvos taškų, kurie yra kategorizuoti pagal objekto būklę, tipą ir pavojingumą aplinkai (išvestinis rodiklis).

Turint šiuos duomenis, galima planuoti paprasto, taršos priklausomybės nuo šaltinio, modelį. Šį modelį bandoma sukurti originaliai, pasiremiant literatūros apžvalgoje surastais modeliavimo būdais.

#### *Išankstiniai reikalavimai modeliui*

Tarkim, kad turimas taršos šaltinių tinklas $T$, kuriame taršos šaltinis, $t \in T$:

[22] Dinamiškumas grindžiamas tuo, jog laiko tarpai nėra vienodi ir matuojamos įvairiausios rodiklių kombinacijos, taigi susidaro nehomogeniška daugiamatė laiko eilutė.

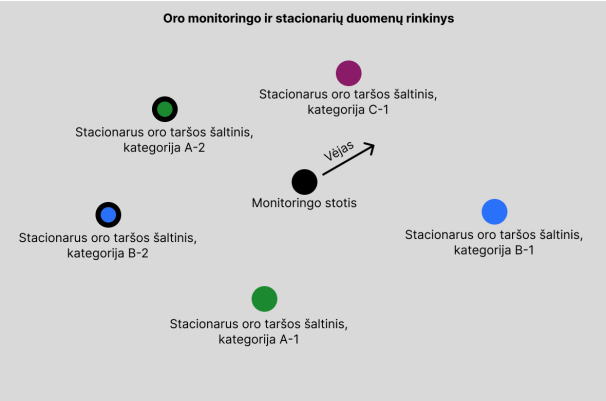
– Turi koordinates $(x_t, y_t)$;
– Yra priskirtas tam tikroms kategorijoms, su koeficientais $k_1^t, k_2^t, \dots, k_n^t$, kurių kiekviena turi savo reikšmių aibę su koeficientais $k_1^t \in \{k_1^1, k_1^2, \dots, k_1^{m_1}\}, k_2^t \in \{k_2^1, k_2^2, \dots, k_2^{m_2}\}, \dots, k_n^t \in \{k_n^1, k_n^2, \dots, k_n^{m_n}\}$; čia $n$ – kategorijų skaičius; $m_i$ – galimų reikšmių skaičius $i$-tojoje kategorijoje.

Taip pat turimas matavimo taškų tinklas $Z$, kuriame matavimo taškas $z \in Z$, $i$-tuoju matavimo momentu:
– Turi pastovias koordinates $(x_z, y_z)$;
– Turi teršalo išmatavimo reikšmę $Y_i^z$;
– Turi vėjo greitį $G_i^z$ ir vėjo kryptį $K_i^z$ (kas susideda į universalų vėjo vektorių $\vec{V}_i$);
– Turi papildomus išmatavimus, $p_{1,i}^z, p_{2,i}^z, \dots, p_{r,i}^z$, kurie turi savo koeficientus $P_1, P_2, \dots, P_r$; čia $r$ – papildomų išmatavimų skaičius.

Teršalas turi savo sklaidos greitį $S$.

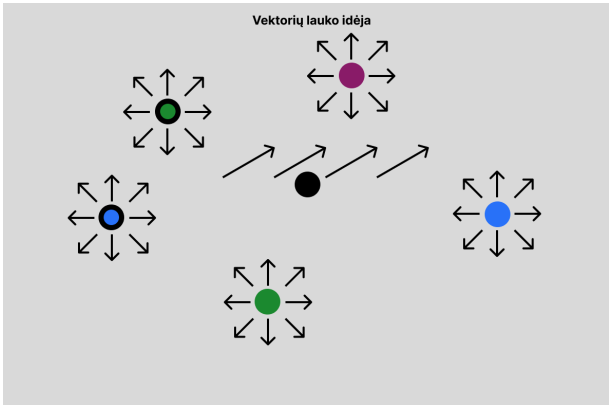
### *Modelio idėjos pagrindimas*

Kadangi modelis be konkretaus pagrindimo yra nieko nevertas, todėl šis poskyris yra skirtas paaiškinti minčių eigą. Visų pirma galima pažvelgti į turimų duomenų struktūrą.



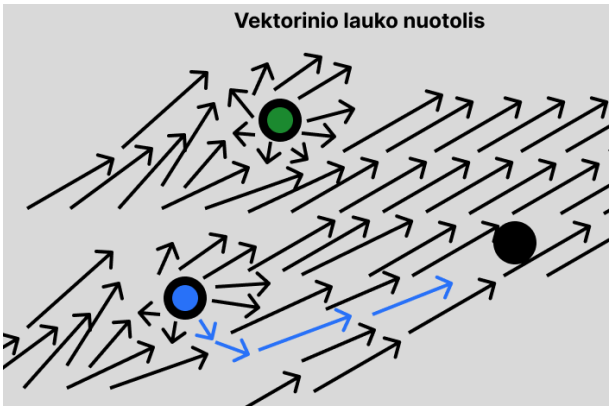

**7 pav.** Oro monitoringo ir stacionarių duomenų rinkinys.

**7 pav.** matome, kaip galėtų būti išsidėstę objektai iš monitoringo stočių ir stacionarių taršos šaltinių aibių. Čia juodai pavaizduota monitoringo stotis, kuri matuoja taršos rodiklius ir meteorologinius rodiklius, kaip vėjo stiprumą ir kryptį, todėl prie jos yra pavaizduotas vėjo vektorius. Toliau verta pažvelgti į labiausiai įprastą tokio tipo uždavinių apibrėžimo faktorių – vektorių lauką.



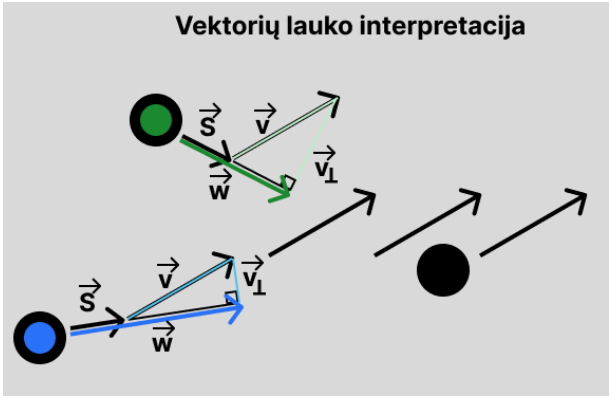

**8 pav.** Vektorių lauko idėja.

Kadangi vektorių laukas įprastai yra įvairių charakteristikų (pvz., divergencijos ar rotoriaus)[130], jo pavaizdavimas, turint tik vieną vektorių viename taške, yra keblus klausimas. Viena idėja būtų įsivaizduoti, jog visas vektorių laukas yra homogeniškas – kolinearus, vienos krypties ir vienodo stiprumo (fiksuoto vėjo parametrų). Taip pat galima įsivaizduoti, jog stacionarūs taršos šaltiniai skleidžia tam tikro stiprumo taršą visomis kryptimis vienodu stiprumu. Prieš pastraipą esančioje **8 pav.** iliustracijoje yra pavaizduota, kaip tai galėtų atrodyti, statiškai. Žemiau, **11 pav.**, galima matyti pavyzdį, kaip galėtų atrodyti taršos keliavimas iš taršos šaltinio į monitoringo stotį.




**9 pav.** Vektorinio lauko nuotolis nuo taršos šaltinio iki monitoringo stoties.

Tokio tipo vektorių lauko formalus aprašymas reikalauja netiesinių diferencialinių lygčių, tačiau šios programos atveju tai nėra optimalus sprendimas, kadangi bus atliekama daug skaičiavimų, todėl pasirinktas kitas, daug paprastesnis variantas, kurio grafinį pavaizdavimą galima matyti diagramoje.




**10 pav.** Vektorių lauko interpretacija ir pritaikymas šiam metodui.

Šiame grafike jau galima pastebėti vektorių pavadinimus, tačiau juos reikia paaiškinti. Vektorius $\vec{v}$ savyje laiko monitoringo stotyje fiksuotą vėjo greitį ir kryptį. Vektorius $\vec{s}$ vaizduoja oro teršalo sklaidos vektorių ir yra sudarytas iš sklaidos greičio $S$, nukreipto nuo taršos šaltinio vektoriaus į monitoringo vietą. **10 pav.** grafike pavaizduota, kad metode yra sudedami šie vektoriai ir iš gautos sudėties yra randamas $\vec{w}$ vektorius, kolinearus vektoriui $\vec{s}$. Taip pat yra randamas ortogonalus[131] vektorius $\hat{v}_{\perp}$, skirtas silpnėjimo koeficiento $\sigma$ apskaičiavimui (tiksliau aprašoma sekančiame skyrelyje). Taigi, gaunamas modelis, priklausantis nuo vėjo greičio ir krypties bei taršos šaltinių vietos žemėlapyje.

***Modelio lygtis***

Atitinkant išankstinius modelio reikalavimus, galima suformuluoti taršos nuspėjimui apskaičiuoti skirtą (32) lygybę:

$$\hat{Y}_i^z = \sum_{t\in T}\prod_{j=1}^{n} k_j^t \sum_{q=1}^{r} p_{q,i}^z P_q \, e^{\frac{|\vec{w}|}{d_t^z}} \sigma\left(\vec{V}_i, S, \vec{w}\right); \tag{32}$$

čia $\hat{Y}_i^z$ – $z$ monitoringo stoties, $i$-tojo momento turimos taršos vertės $Y_i^z$ aproksimacija; $k_j^t$ – $t$ taršos šaltinio $j$ kategorijos koeficientas; $p_{q,i}^z$ – $z$-tosios monitoringo stoties $i$-tojo momento $q$-tojo papildomo rodiklio reikšmė; $P_q$ – $q$-tojo papildomo rodiklio koeficientas; $\vec{w}$ – galutinis vektorius, gautas iš vektorių $\vec{s}$ ir $\vec{V}_i$ vektorinės sumos vektoriaus sudedamosios, kolinearios $\vec{s}$ vektoriui; $d_t^z$ – atstumas tarp matavimo taško $z$ ir taršos šaltinio $t$, apskaičiuojamas kaip $d_t^z = \sqrt{(x_t - x_z)^2 + (y_t - y_z)^2}$; $\sigma(\cdot)$ – silpnėjimo koeficientas, turintis atstoti vektoriaus lauko nepaisymą, apskaičiuojamas (33) formule:

$$\sigma\left(\vec{V}_i, S, \vec{w}\right) = \begin{cases} \max\{0; \dfrac{|\vec{w}|\,\mathrm{sgn}(\cos\varphi)}{S + \left|\vec{V}_i\right| + |\vec{v}_{\perp}|}\}, & kai\ \left|\vec{V}_i\right| \neq 0, \\ 1, & kai\ \left|\vec{V}_i\right| \neq 0; \end{cases} \tag{33}$$

čia $\sigma$ – silpnėjimo koeficientas; $\vec{V}_i$ – vėjo vektorius $i$-tuoju momentu; $S$ – teršalo sklaidos greitis; $\vec{v}_{\perp}$ - $\vec{w}$ vektoriui ortogonalus vektorius, vektoriaus $\vec{V}_i$ projekcija į teršalo $t$ – monitoringo stoties $z$ ašį; $\varphi$ – kampas tarp vektorių $\vec{w}$ ir $\vec{s}$.

***Minimizavimo funkcija***

Modelio tikslas yra rasti koeficientų $k_i^1, k_i^2, \ldots, k_i^{m_i}\ \forall i$ ir $P_1, P_2, \ldots, P_r\ \forall r$ reikšmes, su kuriomis būtų minimizuota reikšmė funkcijos, užrašomos (34) formule:

$$c = \min_{k_i^1,\ldots,k_i^{m_i},\ldots,k_n^{m_n},P_1,P_2,\ldots,P_r} \sum_i \left|Y_i^z - \hat{Y}_i^z\right|; \tag{34}$$

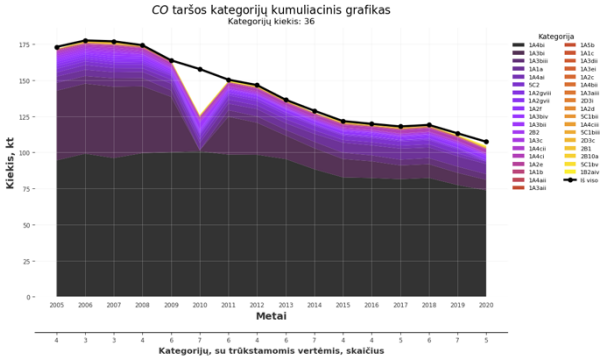
čia $k_i^1, \ldots, k_i^{m_i}, \ldots, k_n^{m_n}$ – teršalų kategorijų reikšmes atitinkantys koeficientai; $P_1, P_2, \ldots, P_r$ – papildomų rodiklių koeficientai; $Y_i^z$ – reali reikšmė stotyje $z$ $i$-tuoju momentu; $\hat{Y}_i^z$ – nuspėjama reikšmė.

## 3. Tyrimų rezultatai ir jų aptarimas

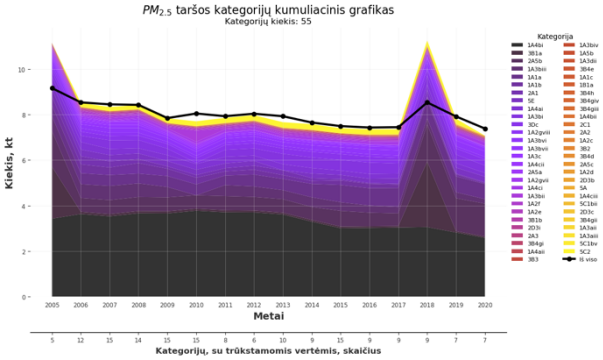
Visų pirma verta paminėti, jog šios analizės apimtyje bus nagrinėjami rodikliai, minimi PSO rekomendacijose (žr. skyrių 1.4.1) arba ES direktyvoje (žr. skyrių 1.4.2) kartu su kitais, pasirinktinais rodikliais.

### 3.1. Metinės laiko eilutės analizė visoje Lietuvos teritorijoje

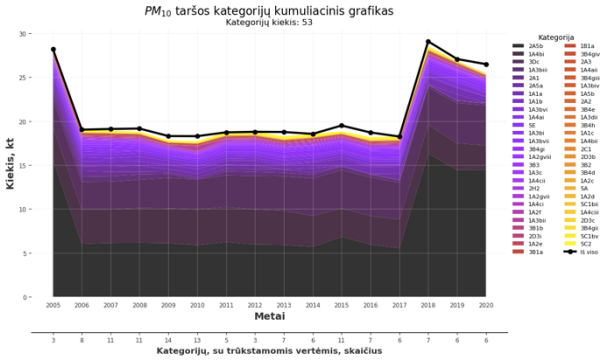
Analizuojant laiko eilutę visada verta iš pradžių pažvelgti į duomenis vizualiai. Tam skirti sekantys linijiniai grafikai, kuriuose galime matyti išskaidytus nagrinėjamus elementus pagal juos sudarančias kategorijas. Pradinei analizei bus naudojami teršalų apskaitos duomenys (žr. skyrių 2.1), tačiau juose nėra vienos iš pagrindinių teršalų kategorijų – ozono. Taigi, šioje metinės teršalų apskaitos duomenų analizėje ozonas ignoruojamas, o teršalų sričių, kurios yra tiriamos, sąrašas atrodo taip[23]: CO, $PM_{2.5}$, $PM_{10}$, $NO_X$, $SO_X$, Pb, HCB.



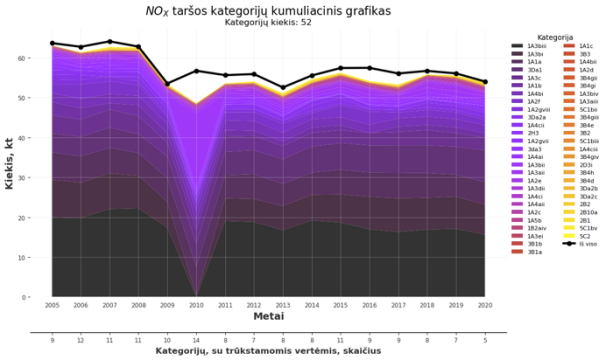

**(a)** CO teršalo apskaitos metinių duomenų grafikas.




**(b)** $PM_{2.5}$ teršalo apskaitos metinių duomenų grafikas.

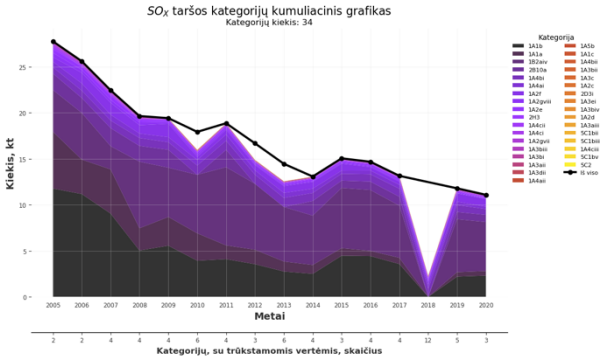



**(c)** $PM_{10}$ teršalo apskaitos metinių duomenų grafikas.

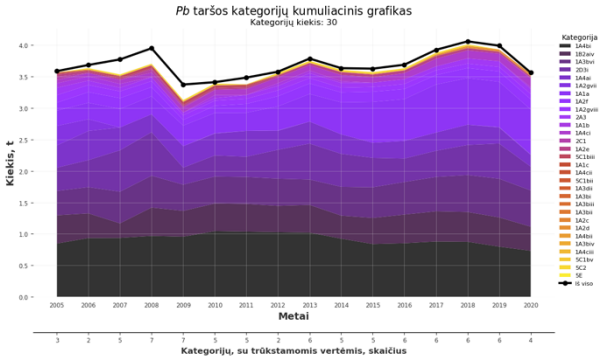



**(d)** $NO_X$ teršalo apskaitos metinių duomenų grafikas.

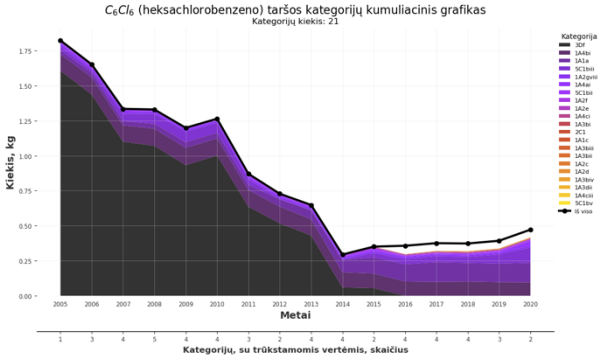
[23] Šiame sąraše esantys šie rodikliai yra minimi ir PSO rekomendacijose, ir ES direktyvoje: CO, $PM_{2.5}$, $PM_{10}$. Kadangi naginėjamuose duomenyse nėra išskirtų atskirų sieros bei azoto oksidų reikšmių, tai jų oksidų bendra suma irgi bus įtraukiama į analizę. Švinas yra įtraukiamas dėl egzistavimo ES direktyvoje. Nors heksachlorobenzenas[132] nėra benzeno[133,134] (kuris yra ES direktyvoje) šalutinis produktas ir juos koreliuoti nėra korektiška, tačiau šis teršalas yra įtraukiamas kaip papildomas sąrašo narys, dėl savo toksiškumo potencialo[135,136,137].




**(e)** $SO_X$ teršalo apskaitos metinių duomenų grafikas.




**(f)** Pb teršalo apskaitos metinių duomenų grafikas.




**(f)** HCB teršalo apskaitos metinių duomenų grafikas.

**11 pav.** Teršalų apskaitos metinių duomenų grafikai, pagal pasirinktų teršalų sąrašą. Grafikuose juoda linija yra vaizduojamas visas teršalų kasmetinis kiekis, o kartu yra sudaromas kumuliacinis taršos kategorijų grafikas. Po grafiku vaizduojama ašis su trūkstamų verčių tarp turimų kategorijų kiekiu.

Iš karto galima pastebėti dvi problemas:

1. Trūkstamų verčių vietose kumuliacinis grafikas gali būti gerokai per mažos galutinės sumos;
2. Kumuliacinis grafikas viršija visų teršalų sumą (pvz., $PM_{2.5}$ grafike).

Pirmoji problema atsiranda dėl neužpildytų duomenų vienoje iš didelę įtaką darančių kategorijų. Duomenų neužpildymas ne tokiose svarbiose kategorijose dažniausiai nedaro didelės galutinės įtakos (pvz., $PM_{2.5}$ grafike matome, jog 2007, 2008 ir 2015 metais turime net po 15 neužpildytų kategorijų, tačiau vizualus skirtumas yra minimalus), tačiau, praradus istoriškai pagrindines sudedamąsias (pvz., CO 2010 metais, $SO_X$ 2018 metais), bendra suma negali būti panaši į „Iš viso“ kategorijos rezultatą.

Antrosios problemos logiškai išspręsti nelabai galima, kadangi tai yra akivaizdi duomenų klaida, prie kurios galima bus sugrįžti išskirčių radimo skyrelyje.

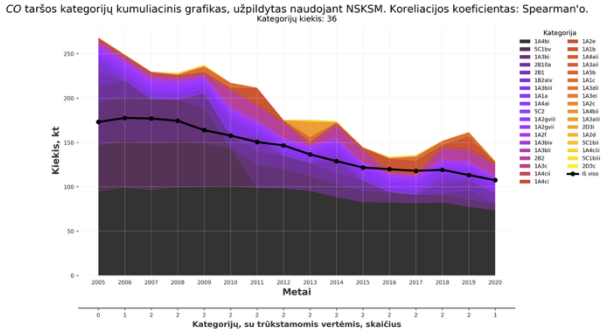
### Trūkstamų verčių užpildymas

Metinius, visos Lietuvos, tam tikro teršalo kategorijos duomenis galima įsivaizduoti kaip matricą $N \times M$, kur $N$ – maksimalus matavimų skaičius kiekvienoje subkategorijoje (metai), $M$ – subkategorijų skaičius. Taigi, užpildyti tokios matricos nežinomoms reikšmėms galima pasitelkti literatūros apžvalgoje minėtą normaliojo santykio ir koreliacijų svorinį metodą (žr. skyrių 1.2), kuris

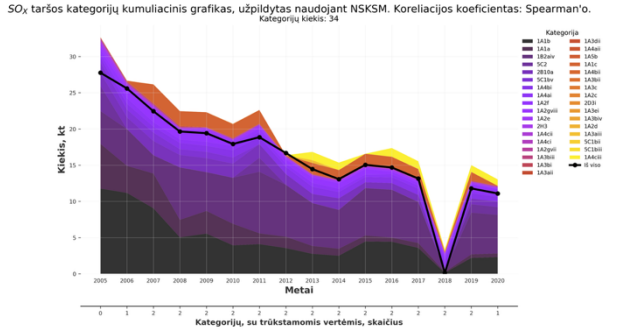
yra skirtas daugiamatėms laiko eilutėms, nors turi neigiamą bruožą neatsižvelgti į aplinkines tos pačios laiko eilutės reikšmes.

Patikrinti šio metodo naudingumui pasirinktos dvi pavyzdinės teršalų kategorijos:

1. CO – pasirinkta dėl gana gražiai užpildyto grafiko, turinčio tik vieną akivaizdžią klaidą 2010 metais (jau vien grafike matoma, kad trūksta „1A3bi" kategorijos, kuri klasifikuoja išmetamąsias dujas iš transporto priemonių[111]). Trūkstamų kategorijų kasmet yra gana nedaug (nuo 3 iki 7), todėl tam tikra prasme ši lentelė yra viena tvarkingiausių;
2. $SO_X$ – vienintelio teršalo duomenų lentelė, neturinti „Iš viso" kategorijos užpildymo (2018 metais).




**(a)** CO teršalo apskaitos metinių duomenų grafikas su užpildytomis trūkstamomis reikšmėmis (normaliojo santykio ir koreliacijų svoriniu metodu).




**(b)** $SO_X$ teršalo apskaitos metinių duomenų grafikas su užpildytomis trūkstamomis reikšmėmis (normaliojo santykio ir koreliacijų svoriniu metodu).

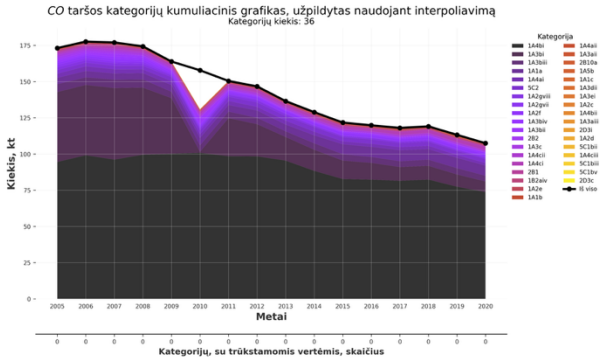
**12 pav.** Teršalų apskaitos metinių duomenų grafikai su užpildytomis trūkstamomis reikšmėmis (normaliojo santykio ir koreliacijų svoriniu metodu).

Pažvelgus į grafikus iš karto matosi trys problemos:

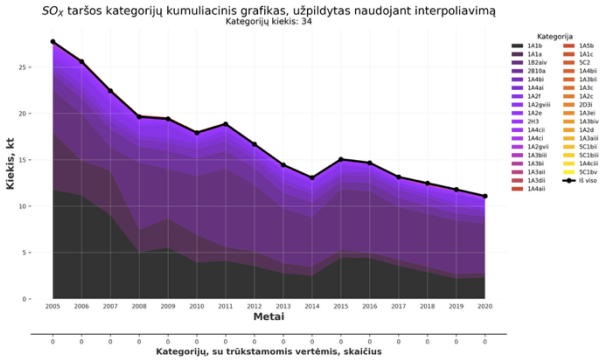
1. Vis dar yra trūkstamų verčių. Taip yra todėl, kad kai kurios kategorijos turėjo 3 ar mažiau verčių, kas neleidžia su šiomis kategorijomis atlikti koreliacijos imtims suskaičiavimo, todėl vertės lieka neužpildytos. Jeigu visas tyrimas būtų remiamas tik koreliacine analize, tokių kategorijų duomenys galėtų būti atmesti, tačiau tai gali panaikinti svarbias įžvalgas kokiam nors kitam metodui[138,139]. Dėl šios problemos reiktų galbūt kaltinti ne duomenis, o metodą, kuris yra gan išrankus duomenims;
2. Užpildytų verčių kumuliacinis grafikas gerokai viršija „Iš viso" kategorijos realias vertes, kas iš karto sugriauna logiką. Šiuo atveju vėlgi galima būtų kaltinti metodą, kuris nepakankamai korektiškai atsižvelgia į daugiamatės laiko eilutės visumą;
3. $SO_X$ 2018 metų „Iš viso" kategorijos reikšmė užpildoma gana nelogiškai, o taip atsitinka dėl didelio kiekio neegzistuojančių reikšmių kitose kategorijose. Be abejo, negalima absoliučiai užtikrintai teigti, jog šis užpildymas yra tikrai neteisingas, tačiau žiūrint bendrą vaizdą, sunku neprieiti prie šios išvados.

Galbūt yra suprantama matyti prastus užpildymo rezultatus, kadangi metodas yra gana paprastas ir nėra pritaikytas šiam uždaviniui. Taigi, remiantis geriausiomis vienmatės eilutės užpildymo strategijomis[140,141], bus naudojamas paprastas vienmatis tiesinis interpoliavimas (žr. skyrių 1.2), naudojantis integruotais programinės kalbos paketais[142,143].

Atlikus skaičiavimus verta pažvelgti į rezultatus (žr. **13 pav.**), kurie yra žymiai geresni nei su prieš tai bandytu metodu.




**(a)** CO apskaitos metinių duomenų grafikai su užpildytomis trūkstamomis reikšmėmis (tiesinio interpoliavimo metodu).




**(b)** $SO_X$ apskaitos metinių duomenų grafikai su užpildytomis trūkstamomis reikšmėmis (tiesinio interpoliavimo metodu).

**13 pav.** Teršalų apskaitos metinių duomenų grafikai su užpildytomis trūkstamomis reikšmėmis (tiesinio interpoliavimo metodu).

Duomenų užpildymas akivaizdžiai daug tikslesnis, todėl daugiau metodų metinių duomenų užpildymui bandoma nebus. Vienintelė likusi problema matoma CO grafiko 2010 metų kumuliaciniame grafike, kur panašu, kad įsiterpusi neteisinga reikšmė. Tokių reikšmių atpažinimui skirta išskirčių analizė, kuri nagrinėjama sekančiame skyriuje.

### Išskirčių radimas

Kaip jau minėta literatūros analizėje, laiko eilučių išskirtis galima nagrinėti įvairiais būdais, o šiuo atveju paieškoms ir sutvarkymui bus naudojamas MAD metodas[144]. Šis metodas randa kiekvieno (pasirinkto intervalo ilgio $k$) judančio lango (angl. *moving window*) medianą, jos vertę priskiria $X_i$ aproksimacijai, o pagal naudotą langą apskaičiuoja nuokrypio dydį naudojant (35) formulę:

$$MAD_i = mediana_i\left(\left|X_i - mediana_j(X_j)\right|\right); \tag{35}$$

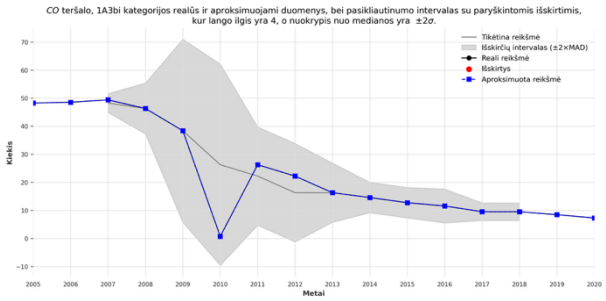
čia $MAD_i$ – nuokrypio dydis $i$-tuoju laiko momentu; $mediana_i(X)$ – mediana $i$-tuoju laiko momentu, erdvės lange $X$; $X_i$ – reikšmė $i$-tuoju laiko momentu.

Šį skaičiavimą atlikti programiškai padeda „hampel“ funkcija[145], kurioje tarp parametrų yra ne tik jau minėtas intervalo ilgis $k$, bet ir $t_0$ – nuokrypių kiekis (pasikliautinumo intervalo nustatymas)[146,147]. „Hampel“ funkcija yra įprastai naudojama programiniuose paketuose ir ji leidžia nustatyti nuokrypių kiekį, lango plotį bei šiek tiek padidina MAD vertę: $MAD^{k,t_0}_{i,hampel} = 1{,}4826 \cdot t_0 \cdot MAD_i$; čia $MAD^{k,t_0}_{i,hampel}$ – $i$-tosios reikšmės „hampel“ nuokrypis su lango ilgiu $k$ ir nuokrypių kiekiu $t_0$.

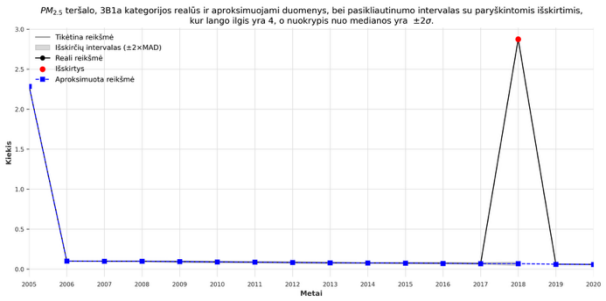
Verta pasižiūrėti į kelis metodu gautus grafikus su tikslu patikrinti išskirčių radimo tikslumą logiškai. Tam, kad patikrinti metodus, verta pažvelgti į specifinių teršalų konkrečias kategorijas, kurios

kumuliaciniuose grafikuose akivaizdžiausiai lemia neteisingus rezultatus[24]. Tam galima pasirinkti CO teršalo kategorija „1A3bi“ (atitinkanti jau minėtą, išmetamųjų dujų iš transporto priemonių, klasifikaciją[111]), kuri visais metais, išskyrus 2010-uosius, yra antra didžiausia CO sudedamoji, o minėtais metais yra labai mažos reikšmės.

Kitas pavyzdys būtų $PM_{2.5}$ teršalo kategorija „3B1a“ (atitinkanti mėšlo tvarkymo klasifikaciją[111]), kurios didžiuliai pakilimai lemia 2005 ir 2018 kumuliacinio grafiko pakilimą virš teorinės ribos – „Iš viso“ kategorijos. Taigi, logiškai mąstant, metodas turėtų rasti išskirtis abiejuose pavyzdžiuose.




**(a)** CO, kategorijos „1A3bi“, apskaitos metinių duomenų išskirčių grafikai, kai išskirtys yra ieškomos naudojant MAD metodą.




**(b)** $PM_{2.5}$, kategorijos „3B1a“, apskaitos metinių duomenų išskirčių grafikai, kai išskirtys yra ieškomos naudojant MAD metodą.

**14 pav.** Teršalų apskaitos metinių duomenų išskirčių grafikai, kai išskirtys yra ieškomos naudojant MAD metodą su lango ilgiu[25] $k = 4$ ir nuokrypiu nuo medianos lygiu $\pm 2\sigma$.

Prieš nagrinėjant rastas bei nerastas išskirtis, verta paminėti, jog algoritme nurodyta $\pm 2\sigma$ vertė atitinka normaliojo skirstinio pasikliautinumo intervalą, kuriame turėtų tūnoti ~95% visų skirstinio verčių[148,149], taigi išskirtis turi būti tikrai gana akivaizdi. Be abejo skirstinys nėra būtinai normalusis, tačiau išskirčių radimo metode tai nėra algoritmą stabdanti vieta.

Galima iš karto pastebėti, jog **(b)** grafike minėta išskirtis akivaizdžiai išsišoka ir yra užfiksuojama. Analizuojant konkrečiai šią išskirtį galima bandyti nuspėti priežastį, priklausančią nuo duomenų pateikimo ir užpildymo metodologijos – jeigu duomenys yra pildomi ranka bent kažkurioje duomenų paruošimo stadijoje (net jei tai yra tik kopijavimas iš programos į duomenų lentelę), galėjo įsivelti viena iš žmogiškiausių klaidų – taško padėjimas neteisingoje vietoje. Tokią hipotezę galima iškelti todėl, nes ir 2005, ir 2018 metų vertės, sumažintos 10 kartų, vizualia prasme žymiai logiškiau tiktų į grafiką.

Žvelgiant į **(a)** grafiką, galima matyti, jog algoritmas neranda jokios išskirties 2010 metų reikšmėje, kadangi pasikliautinumo intervalas toje vietoje iš tiesų labai platus. Tokia reikšmė turėtų būti užfiksuota kaip išskirtis, kadangi taršos verčių šuoliavimas reikšmėmis $38{,}37 \rightarrow 0{,}69 \rightarrow 26{,}27$, neskaitant išskirties gana tolydžiame grafike, yra ypač nenatūralus.

[24] Dideli tam tikro teršalo pokyčiai tarp kategorijų yra galimi, jeigu pasikeičia tam tikro teršalų šaltinio klasifikacija (teršalų šaltinis priskiriamas kitai kategorijai). Akivaizdžiai neteisingais rezultatais laikomos tos sudėtinių kategorijų laiko eilutės vietos, dėl kurių kumuliacinis grafikas yra gerokai mažesnis arba gerokai didesnis nei „Iš viso“ kategorijos kreivė.
[25] Lango ilgis yra matuojamas atimant pradinį laiko tašką iš galutinio laiko taško.

Negana to, kad yra matomas gana užtikrintos išskirties neklasifikavimas, daugelis kitų teršalų subkategorijų turi savyje išskirčių, į kurias šio tyrimo apimtyje gilintis neverta.

Taigi, norint atlikti ypač nuodugnų statistinį tyrimą būtų verta paieškoti kito, išskirčių ieškojimo bei neteisingų verčių atnaujinimo, metodo. Tam galima būtų pasitelkti metodus besiremiančius PCA[3,150].

**Sumų-interpoliavimo išskirčių paieškos ir užpildymo metodas**

Kadangi šiame duomenų rinkinyje yra pateikiami atskirų kategorijų ir galutinės visų kategorijų sumos duomenys, išskirčių paieškos gali būti atliekamos naudojantis šia duomenų kombinacija.

Tarkime, kad kategorija „Iš viso“ (žymima *m*) turi mažiausią tikimybę būti klaidinga (nors 2018 metų $SO_X$ duomenyse ši vertė nebuvo užpildyta), todėl kategorijų klaidingumas yra tikrinamas pagal ją. Taško *Y* priskyrimas prie tinkamų reikšmių yra randamas pagal (36) nelygybę:

$$1 - \varepsilon \leq \frac{\sum_{j=1}^{m-1} Y_{i,j}}{Y_{i,m}} \leq 1 + \varepsilon; \tag{36}$$

čia $\varepsilon$ – nustatytas paklaidos dydis; $Y_{i,j}$ – *i*-tojo laiko taško, *j*-osios kategorijos įvertis daugiamatėje $(N \times M)$ laiko eilutėje *Y*, $i = 1, \dots, n$.

Šis sumos palyginimas randa visus laiko taškus kuriuose kategorijų suma yra arba pakankamai per maža, arba pakankamai per didelė lyginant su bazine sumos reikšme.

Jeigu taškas $Y_{i,m}$ neatitinka šios nelygybės, tada pasitelkiamas užpildymo metodas, kurio metu:

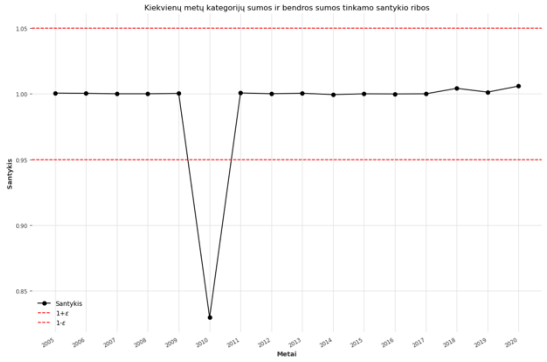
1. Kiekvienos kategorijos vienmatei laiko eilutei yra randama:
   - (jei laiko taškas nėra kraštinis[26]) Tiesinio interpoliavimo reikšmė tiriamu laiko momentu specifinės kategorijos laiko eilutėje, atmetus pradinę tame taške buvusią vertę (lyg būtų interpoliuojama tiesiškai, bet be tos vertės);
   - (jei laiko taškas yra kraštinis) Naiviosios[151] ekstrapoliacijos reikšmė yra apskaičiuojama (37) formule:

     $$\hat{Y}_t = \begin{cases} Y_{t-1} \Leftrightarrow \exists Y_{t-1}, \\ Y_{t+1} \Leftrightarrow \exists Y_{t+1}; \end{cases} \tag{37}$$

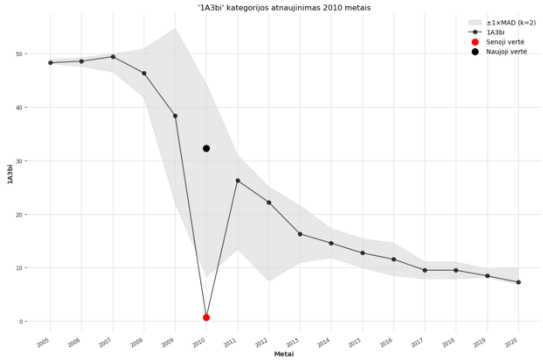
     čia $\hat{Y}_t$ – ekstrapoliuojama vertė *t* laiko momentu; $Y_t$ – reali reikšmė laiko momentu *t*.
2. Ši nauja reikšmė yra įstatoma į laiko eilutę ir iš naujo yra pritaikomas MAD nuokrypio dydis. Gavus nuokrypio reikšmę, grąžinama senoji reikšmė ir su ja vėl atliekama „hampel“ funkcijos išskirčių paieška. Jei senoji reikšmė užfiksuojama kaip išskirtis, ji pakeičiama į naująją ir patikrinama, ar pradinė nelygybė galioja. Jei taip, tada ciklas stabdomas, o jei ne – tęsiamas kitoms kategorijoms.

Atlikus šio metodo pritaikymą gauti grafikai, kuriuose matoma interpoliavimo įtaka – užpildomos visos likusios, prieš tai akivaizdžiai neteisingos, reikšmės.

[26] Kraštiniu laiko eilutės tašku vadinamas pradinis arba paskutinis taškas.

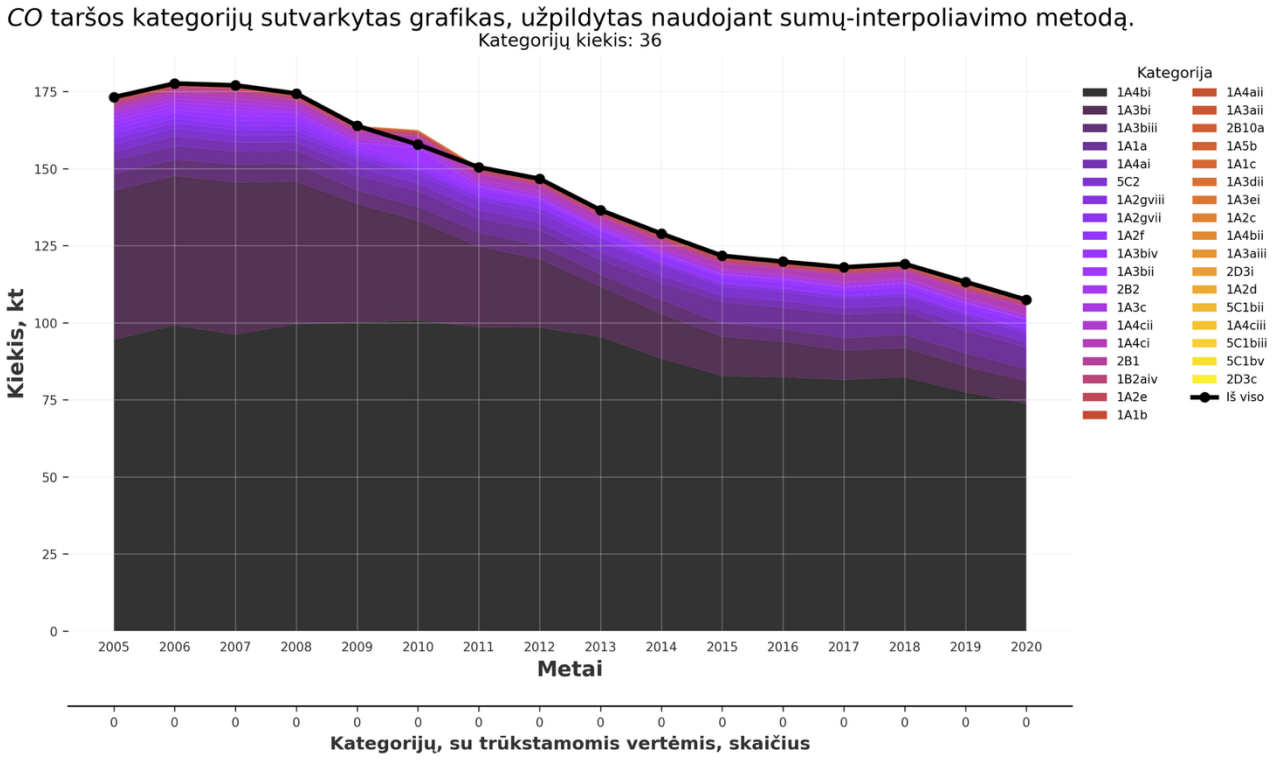



**(a)** Diagrama vaizduojanti kategorijų sumos ir bendros sumos santykį matuotais metais.



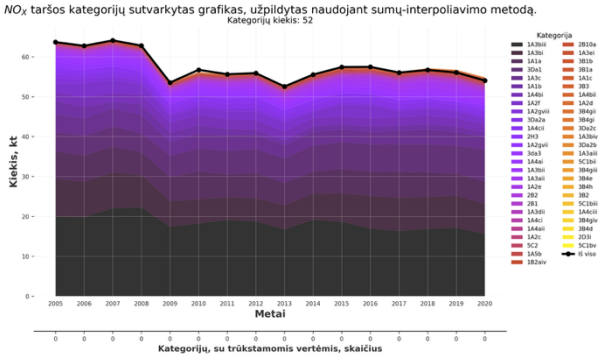

**(b)** Išskirčių paieška ir užpildymas naujomis vertėmis.

**15 pav.** Sumų-interpoliavimo metodo vidinių procesų (santykių palyginimo, išskirčių ieškojimo ir neteisingų verčių užpildymo) atvaizdavimas CO teršalo atveju.

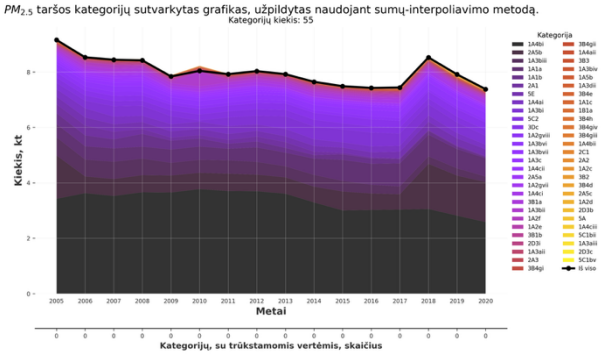



**16 pav.** CO teršalo apskaitos metinių duomenų grafikas su pakeistomis neteisingomis reikšmėmis (sumų-interpoliavimo metodu), kur $\varepsilon = 0{,}05, k = 2, t_0 = 1$.

Nors ir pavyko užpildyti neteisingus duomenis, tačiau prieš tęsiant analizę verta atkreipti dėmesį į tai, jog tik su tikrai gan nedideliu užtikrintumu galima atmesti matytą išskirtį, kadangi MAD metodo pasikliautinumo intervalas priklauso tik nuo $\pm 1\sigma$ nuokrypio nuo interpoliuotai gautos medianos.




**(a)** $NO_X$ teršalo apskaitos metinių duomenų grafikas su pakeistomis neteisingomis reikšmėmis (sumų-interpoliavimo metodu).




**(b)** $PM_{2.5}$ teršalo apskaitos metinių duomenų grafikas su pakeistomis neteisingomis reikšmėmis (sumų-interpoliavimo metodu).

**17 pav.** Teršalų apskaitos metinių duomenų grafikai su pakeistomis neteisingomis reikšmėmis (sumų-interpoliavimo metodu), kur $\varepsilon = 0{,}05, k = 2, t_0 = 3$.

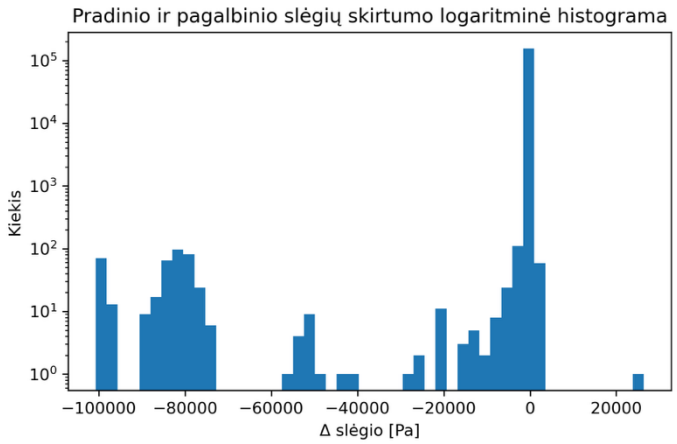
Iš **17 pav.** esančių grafikų galima matyti, jog užpildymas gerai pavyko ir su kitais, prieš tai labai netiksliais, rodikliais.

### 3.2. Automatinio monitoringo duomenys

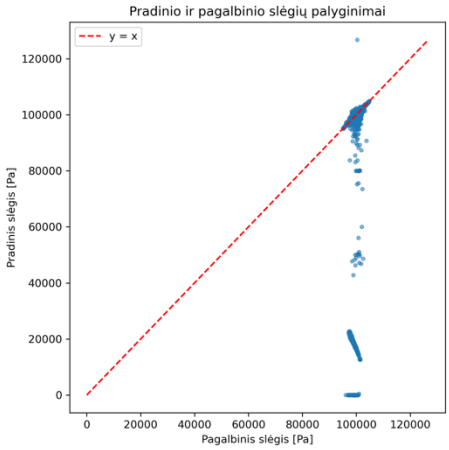
Pasiruošus ir sutvarkius metinius duomenis verta panašias operacijas atlikti ir su daug didesnio dažnio automatinio monitoringo duomenimis. Kadangi juose nėra atskiros kategorijos, pagal kurią galima būtų patikrinti duomenų teisingumą, šį veiksmą galima kitu būdu.

Kadangi beveik visos monitoringo stotys turėtų matuoti meteorologinius rodiklius (žr. skyrių 2), o tuos pačius matavimus taip pat galima gauti iš išorinio duomenų teikėjo „Open-Meteo“ (žr. skyrių 2.3), taigi sutikrinimui yra palyginami šie duomenų rinkiniai. Reikia paminėti, jog papildomas duomenų šaltinis teikia tik valandinius duomenis, o monitoringo stotys matuoja rodiklius kas pusvalandį. Būtent dėl šios priežasties papildomi meteorologiniai duomenys yra apdirbami šiais metodais, kad būtų galima gauti pusvalandžių vertes:

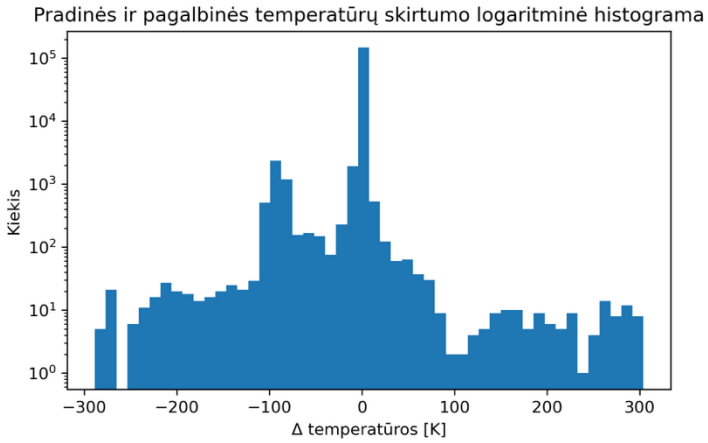
- Slėgis, temperatūra ir santykinis drėgnis yra gaunami apskaičiuojant aritmetinį vidurkį tarp artimiausių valandinių verčių;
- Vėjo greitis ir kryptis yra gaunami atliekant vėjo vektorių vidurkių apskaičiavimą (žr. skyrių 2.5).



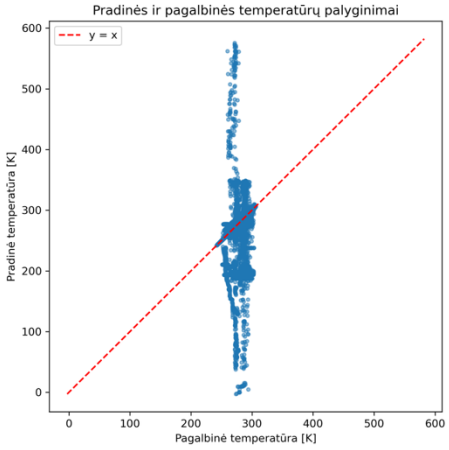

**(a)** Logaritminė histograma vaizduojanti pradinio ir pagalbinio slėgių skirtumą.




**(b)** Pradinio ir pagalbinio slėgių taškinė palyginimo diagrama.




**(c)** Logaritminė histograma vaizduojanti pradinės ir pagalbinės temperatūrų skirtumą.




**(d)** Pradinės ir pagalbinės temperatūrų taškinė palyginimo diagrama.

**18 pav.** Pradinių (monitoringo stočių) ir pagalbinių („Open-Meteo“) duomenų palyginimas diagramomis.

**4 lentelė.** Pradinių (monitoringo stočių) ir pagalbinių („Open-Meteo“) duomenų pagrindinių aprašomųjų statistikų palyginimas.

| **Matavimas** | **Slėgis, Pa**[27] | | **Temperatūra, K** | |
|---|---|---|---|---|
| | **Pradiniai duomenys** | **Pagalbiniai duomenys** | **Pradiniai duomenys** | **Pagalbiniai duomenys** |
| Aritmetinis vidurkis ($\mu$) | 98433,12 | 100272,83 | 255,828 | 280,714 |
| Standartinis nuokrypis[152] ($\sigma$) | 12597,17 | 1169,66 | 43,028 | 9,503 |
| Mažiausia vertė (min) | 0,00 | 95130,00 | -2,790[28] | 241,650 |
| 10-asis procentilis[152] | 98619,00 | 98785,00 | 193,770 | 268,550 |

[27] Asimetrijos bei ekscesо koeficientai neturi matavimo vienetų, tačiau dėl patogumo yra įdėti į bendrą lentelę. Matavimo vienetai nurodyti prie slėgio ir temperatūros stulpelių pavadinimų šioms eilutėm negalioja.

[28] Tokia temperatūra šioje visatoje nėra įmanoma[153].

| Matavimas | Slėgis, Pa[27] | | Temperatūra, K | |
|---|---|---|---|---|
| | Pradiniai duomenys | Pagalbiniai duomenys | Pradiniai duomenys | Pagalbiniai duomenys |
| Mediana | 100082,00 | 100280,00 | 273,430 | 280,850 |
| 90-asis procentilis | 101510,00 | 101740,00 | 290,490 | 292,650 |
| Didžiausia vertė (max) | 126728,00 | 104845,00 | 582,240 | 305,950 |
| Asimetrijos koeficientas | -7,326 | -0,057 | -0,741 | -0,254 |
| Eksceso koeficientas | 52,450 | 0,242 | 3,858 | -0,335 |

Taigi, matant šių meteorologinių duomenų palyginimo taškines diagramas bei statistinių rodiklių lentelę galima daryti išvadą, jog oro monitoringo stočių duomenys, nors, remiantis direktyvomis, turėtų grąžinti vertes iš kone idealiai sukalibruotų duomenų šaltinių, deja, rodo tikrai nepatikimus skaičius. Vien tai, jog oro monitoringo stotyje yra fiksuojama (arba neteisingai pateikiama) temperatūros reikšmė žemiau absoliutaus nulio arba siekianti beveik 600K jau suteikia tvirtą pagrindą netikėti šiais duomenimis. Tačiau jei tai būtų vienetinės išskirtys, galbūt tokios didelės problemos nebūtų, bet šių, akivaizdžiai klaidingų, duomenų yra pakankamai nemažai. Dėl šios priežasties meteorologiniai duomenys iš oro monitoringo stočių yra atmetami ir vietoje jų tolimesnėje analizėje yra naudojami „Open-Meteo“ meteorologiniai duomenys.

Meteorologinių duomenų netvarkingumas sukelia papildomą dilemą – ar galima pasitikėti teršalų monitoringo duomenimis? Kadangi taršos duomenys yra žymiai lokalesni, juos aproksimuojant globaliais modeliais tikėtinos didelės paklaidos. Būtent todėl tyrime šie duomenys nebus keičiami kitų šaltinių duomenimis, tačiau išskirčių analizė vis vien bus atlikta.

Duomenų dažnio patikrinimui verta pasidaryti lentelę su absoliučiu matavimų ir su dienų, kuriose buvo atliktas bent vienas matavimas, kiekiu. Patogumo dėlei verta lentelę nusibraižyti išlaikant duomenų aprašyme (žr. skyrių 2) naudotos lentelės formatą. Lentelėje žaliai pažymimi langeliai, kuriuose yra duomenys, nors pagal aprašą jų būti neturėtų, o raudonai – langeliai, kuriuose duomenys turėtų būti, bet jų nėra.

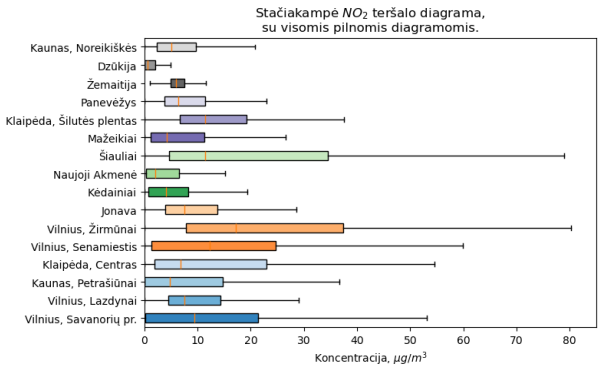
**5 lentelė.** Oro monitoringo stotyse užfiksuotų matavimų bei unikalių dienų su bent vienu matavimu kiekiai.

| Oro monitoringo stotis | Teršalų matavimai | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $NO_2$ | | CO | | $O_3$ | | $SO_2$ | | $KD_{2.5}$ | | $KD_{10}$ | |
| | Matavimų kiekis | Dienų su matavimais kiekis | Matavimų kiekis | Dienų su matavimais kiekis | Matavimų kiekis | Dienų su matavimais kiekis | Matavimų kiekis | Dienų su matavimais kiekis | Matavimų kiekis | Dienų su matavimais kiekis | Matavimų kiekis | Dienų su matavimais kiekis |
| Naujoji Akmenė | 110 | 97 | | | | | 1415 | 624 | 788 | 412 | 770 | 391 |
| Jonava | 852 | 728 | | | 1541 | 694 | 389 | 325 | 28 | 19 | 1014 | 573 |
| Kaunas, Noreikiškės | 2857 | 1256 | 269 | 45 | 2822 | 984 | 933 | 721 | 2837 | 512 | 557 | 328 |

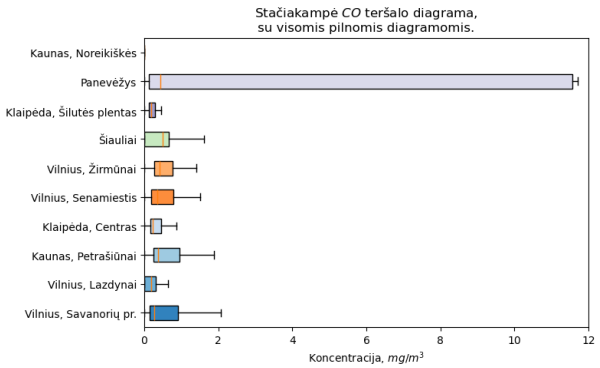
| Oro monitoringo stotis | Teršalų matavimai | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $NO_2$ | | CO | | $O_3$ | | $SO_2$ | | $KD_{2.5}$ | | $KD_{10}$ | |
| | Matavimų kiekis | Dienų su matavimais kiekis | Matavimų kiekis | Dienų su matavimais kiekis | Matavimų kiekis | Dienų su matavimais kiekis | Matavimų kiekis | Dienų su matavimais kiekis | Matavimų kiekis | Dienų su matavimais kiekis | Matavimų kiekis | Dienų su matavimais kiekis |
| Kaunas, Petrašiūnai | 2360 | 1063 | 135 | 81 | 2722 | 1153 | 1309 | 420 | 3275 | 836 | 2476 | 532 |
| Kėdainiai | 1094 | 717 | | | 1937 | 1011 | 748 | 544 | 31 | 30 | 894 | 468 |
| Klaipėda, Centras | 2520 | 1059 | 390 | 93 | 932 | 165 | 2234 | 991 | | | 1202 | 364 |
| Klaipėda, Šilutės plentas | 747 | 641 | 51 | 42 | 1756 | 1070 | 6 | 6 | 2267 | 793 | 1091 | 455 |
| Mažeikiai | 2596 | 1053 | | | 3202 | 1065 | 2694 | 869 | 23 | 23 | 1651 | 470 |
| Šiauliai | 1437 | 887 | 338 | 94 | 1516 | 953 | 890 | 433 | 42 | 31 | 1676 | 519 |
| Vilnius, Savanorių pr. | 1223 | 654 | 202 | 103 | 14 | 11 | 1183 | 777 | | | 1912 | 459 |
| Vilnius, Žirmūnai | 977 | 782 | 162 | 104 | 2135 | 1122 | 132 | 69 | 2307 | 827 | 1034 | 603 |
| Vilnius, Senamiestis | 2249 | 1314 | 372 | 136 | 865 | 21 | 1199 | 856 | | | 2226 | 721 |
| Vilnius, Lazdynai | 1087 | 856 | 50 | 26 | 2320 | 996 | 758 | 585 | | | 1516 | 552 |
| Žemaitija | 1969 | 906 | | | 1570 | 1172 | 1189 | 1078 | 585 | 378 | 559 | 343 |
| Panevėžys | 867 | 735 | 67 | 25 | 1764 | 1046 | | | 25 | 25 | 5248 | 573 |
| Dzūkija | 198 | 156 | | | 2643 | 830 | 1216 | 967 | | | | |
| Aukštaitija | | | | | 1239 | 800 | 1 | 1 | 691 | 388 | 67 | 30 |

Matome, jog yra tikrai nemažai stočių su papildomais duomenimis (žali langeliai), tačiau dauguma šių duomenų yra labai nedidelio kiekio. Taip pat matoma viena stotis (Vilnius, Lazdynai), kurioje nėra matuojamas $KD_{2.5}$ rodiklis. Tai iš tiesų yra nemaža problema, kadangi tik Žirmūnų stočiai tenka matuoti viso Vilniaus $KD_{2.5}$ koncentraciją.

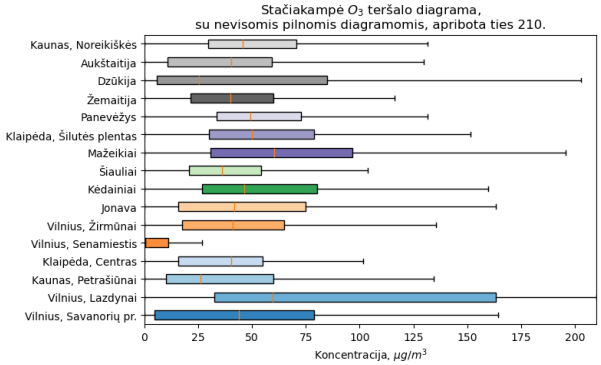
Toliau pateikiamos stačiakampės diagramos[154] (angl. *box-and-whisker plot*) kiekvienam iš tiriamų teršalų su kiekviena stotimi. Stotys, neturinčios bent 10 specifinio teršalo matavimų, tam teršalui yra atmetamos iš diagramos.



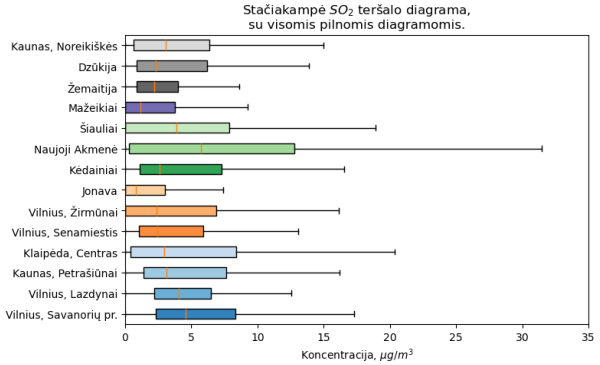

**(a)** Stačiakampė $NO_2$ teršalo diagrama, su visomis pilnomis diagramomis.



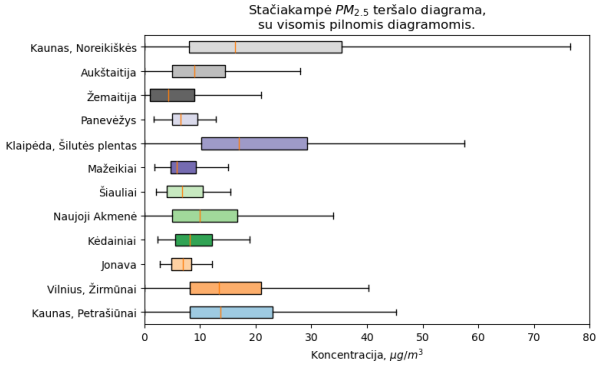

**(b)** Stačiakampė CO teršalo diagrama, su visomis pilnomis diagramomis.



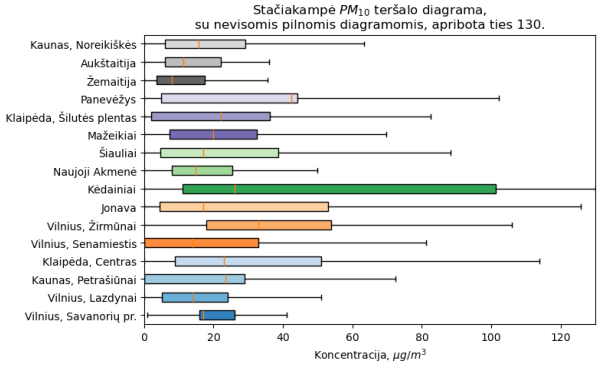

**(c)** Stačiakampė $O_3$ teršalo diagrama, su nevisomis pilnomis diagramomis, apribota ties 210 $\mu g/m^3$.




**(d)** Stačiakampė $SO_2$ teršalo diagrama, su visomis pilnomis diagramomis.




**(e)** Stačiakampė $PM_{2.5}$ teršalo diagrama, su visomis pilnomis diagramomis.



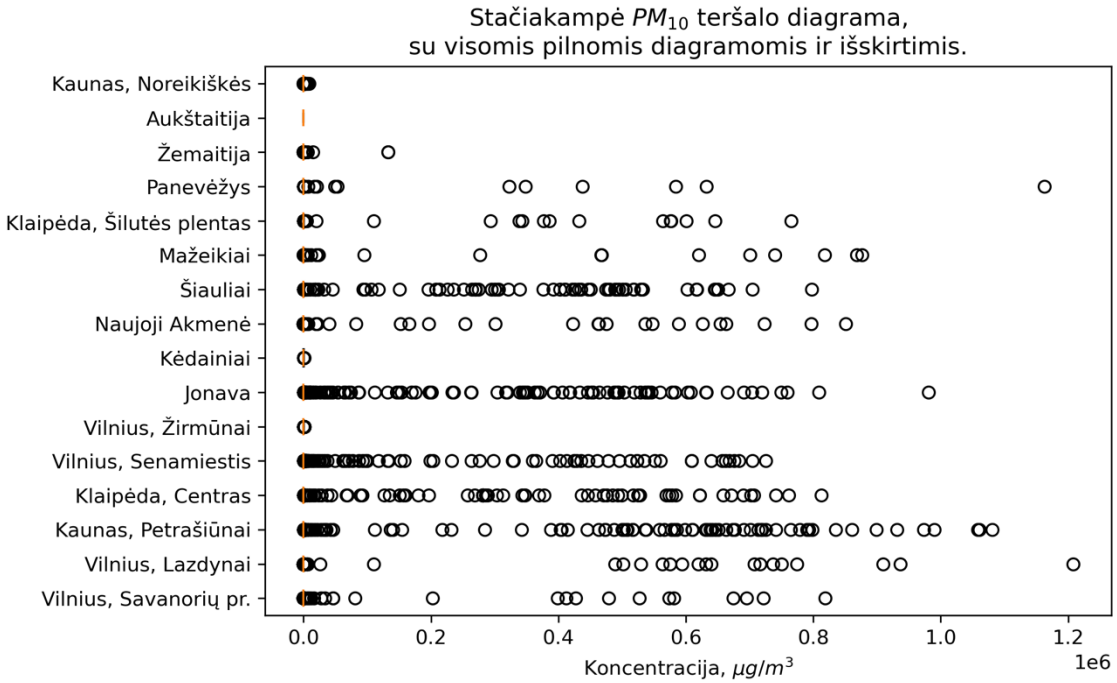

**(f)** Stačiakampė $PM_{10}$ teršalo diagrama, su nevisomis pilnomis diagramomis, apribota ties 130 $\mu g/m^3$.

**19 pav.** Stačiakampės teršalų diagramos visoms stotims, turinčioms virš 10 specifinio teršalo matavimų. Diagramos yra arba pilnai užpildytos, arba nukirptos ties tam tikra verte.

Taigi, stačiakampės diagramos vaizduoja tris tipines aprašomąsias statistikas (pirmą kvartilį (K1), medianą (K2) ir trečią kvartilį (K3)), o kartu su jomis taip pat vaizduoja ūsus, priklausomus nuo IKP, apskaičiuojamo kaip $IKP = K3 - K1$, ir nurodančius:

– Minimalaus taško vertę arba K1 - 1,5 IKP reikšmę;
– Maksimalaus taško vertę arba K3 + 1,5 IKP.

Taigi, galima atkreipti dėmesį į tai, kad stočių vidutiniai matavimai gali skirtis net per kelis kartus. Į šį pastebėjimą atsižvelgiama tolimesnėje analizėje. Beje reikia paminėti, jog šios stačiakampės diagramos nepavaizduoja išskirčių, tačiau taip buvo pasirinkta dėl ypač didelių išskirčių reikšmių, kurios pakeičia grafikus taip, kad tampa nebeįmanoma vaizdiškai analizuoti pasiskirstymo, rodančio didžiąją duomenų dalį. Tačiau galima pasižiūrėti ir į kurio nors teršalo (pvz., $PM_{10}$) stačiakampę diagramą su išskirtimis.




**20 pav.** Stačiakampė $PM_{10}$ teršalo diagrama, su visomis pilnomis diagramomis su išskirtimis.

Matome, jog iš tiesų egzistuoja daug labai didelės reikšmės išskirčių, tačiau jas atmesti būtų nekorektiška, kaip jau minėta anksčiau.

Kadangi automatinio monitoringo duomenys yra labai didelio dažnio (teoriškai gaunami kas pusvalandį), iš jų tolimesnei analizei verta susikurti laiko prasme skirtingus duomenų rinkinius, pagal kuriuos būtų galima atlikti sekančius analizės veiksmus. Taigi, sugalvota duomenis sugrupuoti pagal įvairius laikotarpius ir sukurti tokius rodiklius:

– Dienos aritmetinis vidurkis (toliau – dienos duomenys) – apskaičiuojamas iš žalių duomenų kiekvienai dienai;
– Savaitės dienos aritmetinis vidurkis – apskaičiuojamas iš dienos duomenų, sugrupuotų pagal savaitės dieną (pvz., pirmadienį, antradienį ir t.t.);
– Mėnesio aritmetinis vidurkis – apskaičiuojamas iš dienos duomenų, sugrupuotų pagal mėnesį;
– Metų aritmetinis vidurkis – apskaičiuojamas iš mėnesio aritmetinio vidurkio duomenų, sugrupuotų pagal metus;
– Pusvalandžio aritmetinis vidurkis – apskaičiuojamas iš žalių duomenų, sugrupavus juos pagal matavimo laiką (pusvalandį).

Tyrime pasirinkta būtent tokia dienos → mėnesio → metų duomenų grupavimo hierarchija tam, kad būtų išvengta per didelės įtakos sugrupuotų laiko momentų, turinčių per didelę įtaką. Konkretus argumentavimo pavyzdys būtų viena diena turinti 9 matavimus, o sekanti diena turinti 1 matavimą. Tokiu atveju šių dienų aritmetinio vidurkio svoris būtų gerokai didesnis pirmoje dienoje, kas tikriausiai lemtų neteisingą vidutinę taršos koncentraciją šioms dviem dienoms.

**Komponenčių dekompozicijos analizė**

Vienas iš daugiausiai informacijos suteikiančių laiko eilutės analizės metodų yra komponenčių dekompozicijos analizė (žr. skyrių 1.2). Tačiau norint ją atlikti, visų pirma reikia pasiruošti duomenis. Kadangi oro taršos koncentracijos logiško patikrinimo nelabai yra, kadangi per pusvalandį tikrai gali pasikeisti ypač lokalus rodiklio rezultatas, tai vienintelis duomenų sutvarkymas šioje vietoje bus neigiamų duomenų, kurie gali atsirasti monitoringo stoties perduodamuose duomenyse dėl ypač mažos koncentracijos, pavertimas nuliais.

Taip pat reikia atsižvelgti į tai, jog dekompozicijos analizei atlikti reikia turėti švarią laiko eilutę be jokių tarpų. Prieš tai darant verta pažvelgti į stotyse atliktų matavimų pasiskirstymą laiko eilutėje. Žemiau esančiame **21 pav.** taškais yra pavaizduojami matavimai (nesvarbu kokio rodiklio) visų stočių laiko eilutėse. Papildomai yra pateikiamos vertikalios linijos, vaizduojančios šiuos įvykius:

- 2008-05-21 – ES direktyvos „dėl aplinkos oro kokybės ir švaresnio oro Europoje“ įsigaliojimas[91];
- 2015-09-18 – ES direktyvos „dėl aplinkos oro kokybės ir švaresnio oro Europoje“ atnaujinimas[92];
- 2020-02-26 – paskelbta valstybės lygio ekstremalioji situacija dėl COVID-19 pandemijos[155];


**21 pav.** Visų monitoringo stočių matavimų pasiskirstymas laiko eilutėje su papildomomis vertikalėmis, vaizduojančiomis įvykius (vardinant iš kairės į dešinę): ES direktyvos „dėl aplinkos oro kokybės ir švaresnio oro Europoje“ įsigaliojimas; ES direktyvos „dėl aplinkos oro kokybės ir švaresnio oro Europoje“ atnaujinimas; paskelbta valstybės lygio ekstremalioji situacija dėl COVID-19 pandemijos.

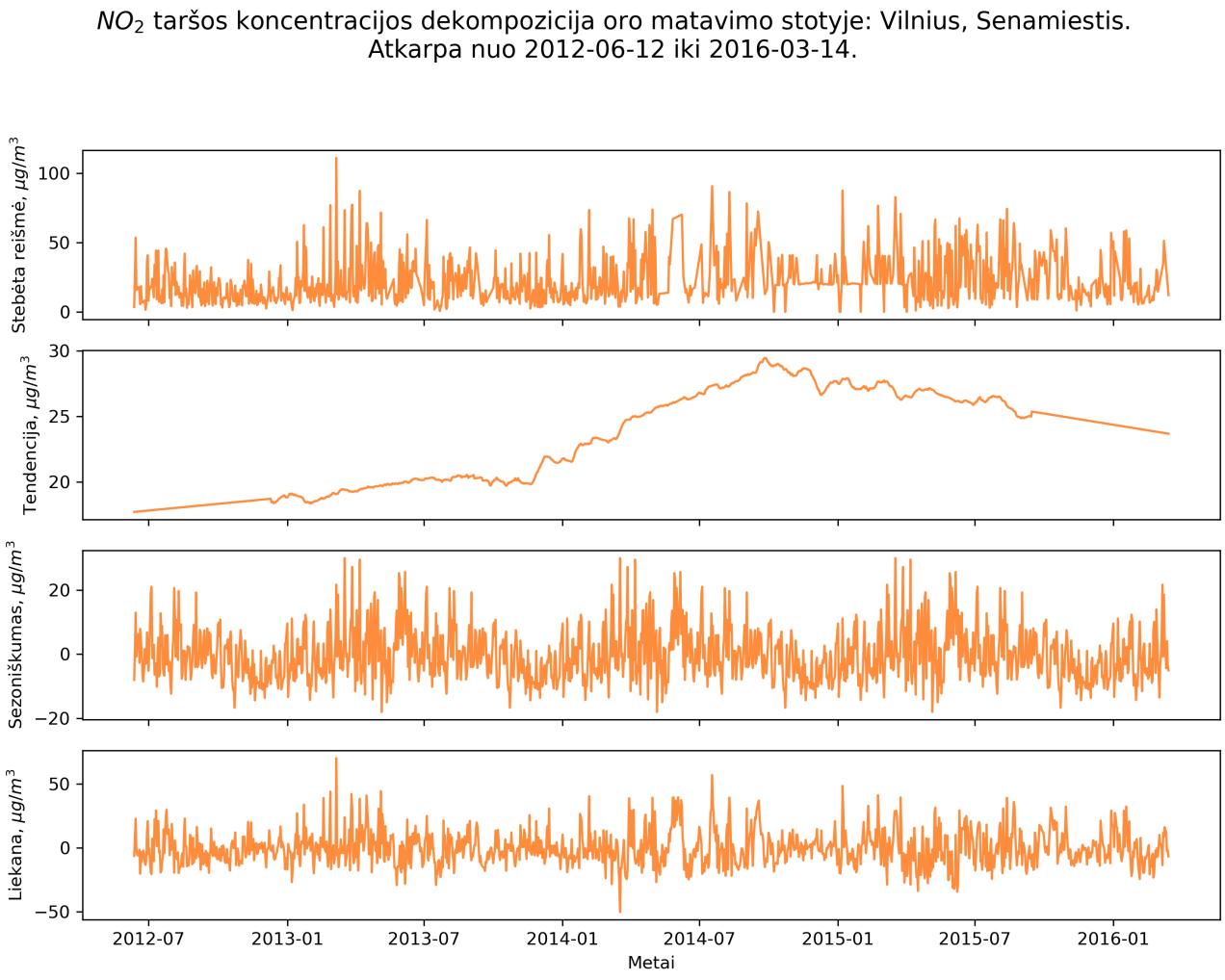
Pagal vertikales galima pastebėti, jog tiesiogiai rodiklių matavimą turinčios paveikti ES direktyvos neparodo kažkokio konkretaus padažnėjimo (bent jau lyginant su laikotarpiu prieš jas), tačiau panašu, kad matavimų pasiskirstymą labiausiai paveikė pasaulinio lygio pandemija.

Turint galvoje faktą, jog laiko eilutės analizė turėtų būti atliekama laiko eilutėje neturinčioje tarpų, oro monitoringo duomenų analizė su milžiniško pločio užpildytais tarpais (turėtų būti užpildyti duomenys nuo maždaug 2020 iki 2022 metų) bus nelabai vertinga. Taigi, dėl šios priežasties dekompozicinėje komponenčių analizėje yra atsižvelgiama į trūkstamų duomenų kiekį.

Atliekant sudedamosios formos komponenčių dekompoziciją su dienos duomenimis, galima atlikti metinio sezoniškumo analizę. Kadangi didelė dalis duomenų yra interpoliuojama (tiesiniu, į laiką atsižvelgiančiu interpoliavimo būdu[156,157]), tai rasti ilgus periodus be trūkstamų verčių yra praktiškai neįmanoma. Dėl to pasitelkiamas filtras, kuris panaikina tokius periodus, tarp kurių taškų yra didesnis atstumas nei nurodyta ir interpoliuotų taškų skaičius viršija nurodytą.

### *$NO_2$*

Kadangi $NO_2$ duomenys yra išties dažnai matuojami kai kuriose stotyse, tai jiems galima nustatyti ganėtinai žemą didžiausio interpoliavimo procento ribą (tarkim ties 30%), o didžiausią interpoliuojamą atstumą galima nustatyti kaip 2 savaites (14 dienų). Taigi, turint šiuos duomenis ir atlikus metinio sezoniškumo (365 dienų) sudėtinės dekompozicijos analizę[158,159], gaunami **22 pav.** matomi grafikai.

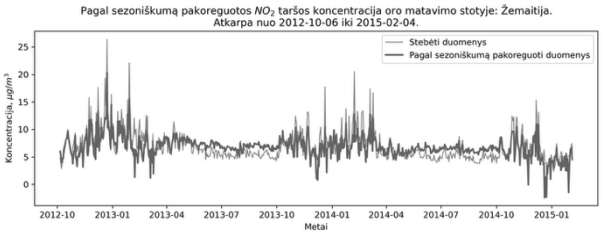



**22 pav.** $NO_2$ taršos koncentracijos dekompozicija oro matavimo stotyje: Vilnius, Senamiestis. Atkarpa nuo 2012-06-12 iki 2016-03-14.

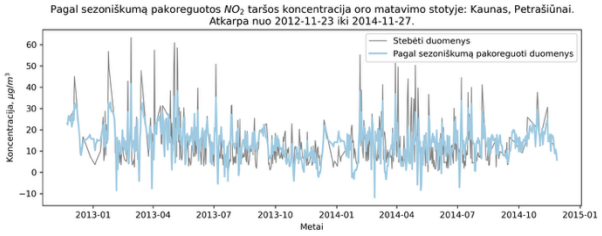
Nustačius didžiausią interpoliavimo ribą ties 30% ir didžiausią interpoliuojamą atstumą ties 14 dienų, Vilniaus, Senamiesčio, oro monitoringo stotyje gaunamas intervalas nuo 2012-06-12 iki 2016-03-14. Tai parodo, kad duomenys pastoviai matuojami buvo tik tam tikrais laikotarpiais, kurių dauguma išsidėstę panašiame laikotarpyje. Toliau esančiame sąraše išvardinti visi, nustatytas užpildymo taisykles atitinkantys, laikotarpiai su stočių pavadinimais:

- Vilniaus, Senamiesčio, oro matavimo stotyje intervalas nuo 2012-06-12 iki 2016-03-14;
- Kauno, Petrašiūnų, oro matavimo stotyje intervalas nuo 2012-11-23 iki 2014-11-27;
- Žemaitijos oro matavimo stotyje intervalas nuo 2012-10-06 iki 2015-02-04;
- Kauno, Noreikiškių, oro matavimo stotyje intervalas nuo 2012-04-12 iki 2015-01-22.

Kartu su visų komponenčių dekompozicijos grafikais galima atvaizduoti dar ir papildomus, šiuo atveju jau kitų stočių[29], grafikus, kuriuose yra panaikintas sezoniškumas ir pavaizduota tendencija[158].




**(a)** Pagal sezoniškumą pakoreguota diagrama. Žemaitijos stotis. Atkarpa nuo 2012-10-06 iki 2015-02-04.




**(b)** Pagal sezoniškumą pakoreguota diagrama. Kauno, Petrašiūnų, stotis. Atkarpa nuo 2012-11-23 iki 2014-11-27.


**(c)** Tendencinė diagrama. Žemaitijos stotis. Atkarpa nuo 2012-10-06 iki 2015-02-04.


**(d)** Tendencinė diagrama. Kauno, Petrašiūnų, stotis. Atkarpa nuo 2012-11-23 iki 2014-11-27.

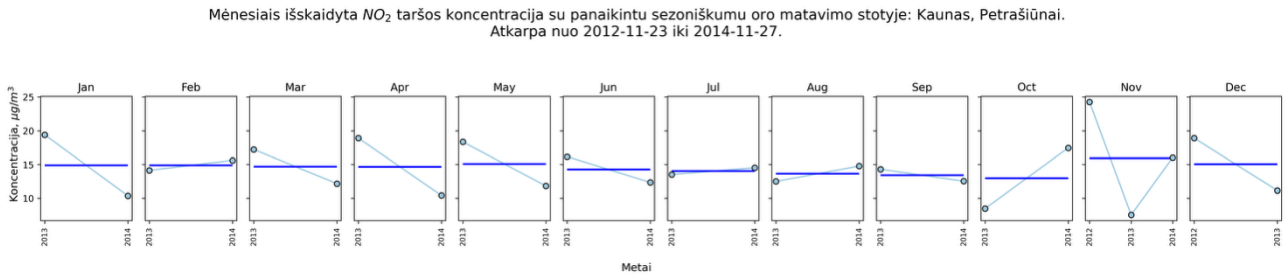
**23 pav.** Pagal sezoniškumą pakoreguotos bei tendencinė $NO_2$ taršos koncentracija oro matavimo stotyse, specifinėse atkarpose.

Lyginant **23 pav.** esančius grafikus būtų galima matyti:

- Egzistuojantį skirtumą tarp to, kad Žemaitijos stotyje, panaikinus sezoniškumą, matomi dideli $NO_2$ teršalo pokyčiai žiemos laikotarpiu, o Kauno, Petrašiūnų, stotyje rodiklio nepastovumas matomas visų metų laikotarpiu;
- Panašią, žemyn judančią, tendenciją.

Kartu su prieš tai paminėtais grafikais, galima panagrinėti ir rodiklio kasmėnesinius skirtumus[140].

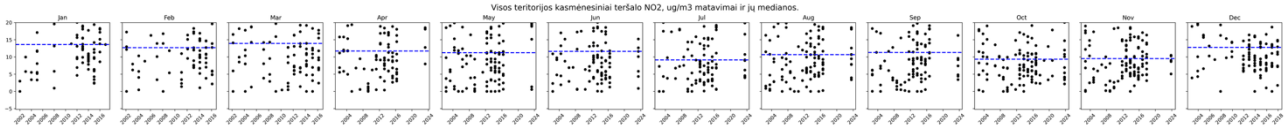
[29] Pasirinkta pavaizduoti kitų stočių duomenis, kad būtų galima pamatyti daugiau skirtingų duomenų pavyzdžių.




**24 pav.** Mėnesiais išskaidyta $NO_2$ taršos koncentracija su panaikintu sezoniškumu Kauno, Petrašiūnų, matavimo stotyje. Atkarpa nuo 2012-11-23 iki 2014-11-27.

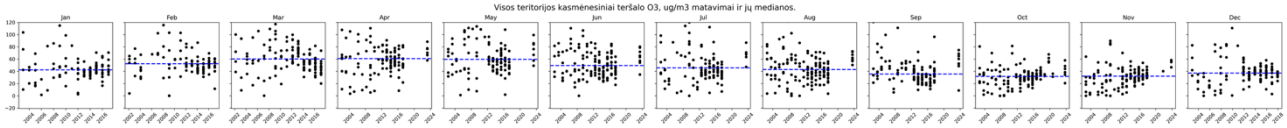
Žvelgiant į rezultatus **24 pav.**, vienintelė stotis, turinti bent kažkuo netolygų mėnesių medianų pasiskirstymą yra Kaune, Petrašiūnuose. Jos grafike matome koncentracijos padidėjimą metų gale (lapkritį ir gruodį), tačiau tai nėra pakankamai žymus pokytis.

Kitų stočių, tokiu pačiu būdu sugeneruotuose grafikuose jokių nevienodumų nesimato. Todėl verta pažiūrėti, kaip atrodytų visų stočių bendras vaizdas, jeigu būtų atrinktos tik stotys, turinčios bent po vieną kiekvieno mėnesio vidurkį.

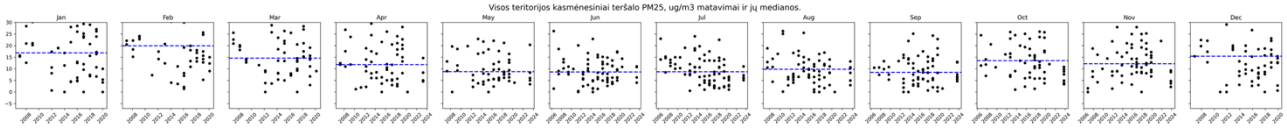



**25 pav.** Visos teritorijos kasmėnesiniai teršalo $NO_2$ matavimai ir jų medianos.

Matome, jog visos valstybės, pilnos laiko eilutės, mėnesių medianos yra šiek tiek žemesnio įverčio nei laiko atkarpos Petrašiūnų stotyje, tačiau šiame grafike galima matyti panašų taršos padidėjimą metų gale, tik jau ne lapkritį, o gruodį. Beje, grafikas yra apribotas vertikalioje ašyje tam, kad geriau matytųsi medianos linija.




**26 pav.** Visos teritorijos kasmėnesiniai teršalo $O_3$ matavimai ir jų medianos.




**27 pav.** Visos teritorijos kasmėnesiniai teršalo $PM_{2.5}$ matavimai ir jų medianos.

Viršuje esantys grafikai (žr. **26 pav.** ir **27 pav.**) rodo, jog vidutinės mėnesio taršos tendencijos gali gerokai skirtis tarp taršos rodiklių. Paviršinio ozono koncentracija panašu, jog nepriklauso nuo metų laiko. Kietųjų dalelių koncentracija gana žymiai padidėja žiemos pabaigoje – pavasario pradžioje, o tai gali būti nulemta tokių pasaulinio lygio geologinių reiškinių kaip Sacharos dulkių pernešimas iš Afrikos[160]. Tokie reiškiniai dažniausiai yra pastebimi ir visuomenės[161,162,163], todėl nedidelė tikimybė atrasti kažkokių neįtikėtinų ir negirdėtų tendencijų.

Kiti teršalai šiame skyriuje nebus nagrinėjami dekompozicijos metodu, kadangi nematoma daug naudos iš tokių išsibarsčiusių duomenų dekompozicinio nagrinėjimo.

### 3.3. Teršalų tarpusavio sąsaja

Praėjusio skyriaus (žr. skyrių 3.2) pradžioje paminėta, kokiu būdu sukuriami metiniai, mėnesio, savaitės dienos, dienos ir pusvalandžio duomenys. Dabar verta patikrinti ar tikrinant šiuos duomenis randami kokie nors ryšiai tarp taršos rodiklių. Tikrinimas būtų atliekamas toje pačioje stotyje apskaičiuojant Spearman‘o koreliacijos koeficientą tarp skirtingų teršalų ir palyginant skirtingomis duomenų agregacijomis gautus duomenis.

Įprasta kartu su koreliacijos koeficientu apskaičiuoti ir statistinį reikšmingumą, tačiau šiame tyrime tai nebus pagrindinis faktorius ryšio reikšmingumo nustatymui, pritariant, daugiau nei 800 mokslininkų sudarytai, nuomonei dėl statistinio reikšmingumo negebėjimo visada korektiškai išreikšti asociacijų tarp kintamųjų[164].

Patikrinus visus agreguotus duomenų rinkinius ir apibendrinus jų koreliacijų reikšmes, galima teigti radus ryšius tarp:

- $PM_{2.5}$ ir $PM_{10}$, dėl koreliacijos koeficientų 0,5312 (kiekvienos stoties mėnesio vidurkio), 0,6598 (kiekvienos stoties dienos vidurkio), 0,6129 (kiekvienos stoties metų vidurkio), 0,4412 (visų stočių metinio vidurkio), tačiau jiems šiek tiek prieštarauja koeficientai 0,1000 (kiekvienos stoties bendro savaitės dienos vidurkio), 0,1506 (kiekvienos stoties bendro pusvalandžio vidurkio);
- $PM_{2.5}$ ir $NO_2$, dėl koreliacijos koeficientų 0,2905 (kiekvienos stoties mėnesio vidurkio), 0,3665 (kiekvienos stoties dienos vidurkio), 0,4011 (kiekvienos stoties metų vidurkio), 0,4529 (visų stočių metinių vidurkio), 0,2683 (kiekvienos stoties bendro savaitės dienos vidurkio), tačiau jiems šiek tiek prieštarauja koeficientas 0,0839 (kiekvienos stoties bendro pusvalandžio vidurkio);
- $PM_{10}$ ir $NO_2$, dėl koreliacijos koeficientų 0,2141 (kiekvienos stoties mėnesio vidurkio), 0,3440 (kiekvienos stoties dienos vidurkio), 0,2467 (kiekvienos stoties metų vidurkio), 0,2812 (visų stočių metinių vidurkio), 0,1711 (kiekvienos stoties bendro savaitės dienos vidurkio), tačiau jiems prieštarauja koeficientas -0,0418 (kiekvienos stoties bendro pusvalandžio vidurkio);

Taigi, remiantis šiais duomenimis būtų galima teigti, jog tikriausiai egzistuoja trys tarpusavyje susiję teršalai.

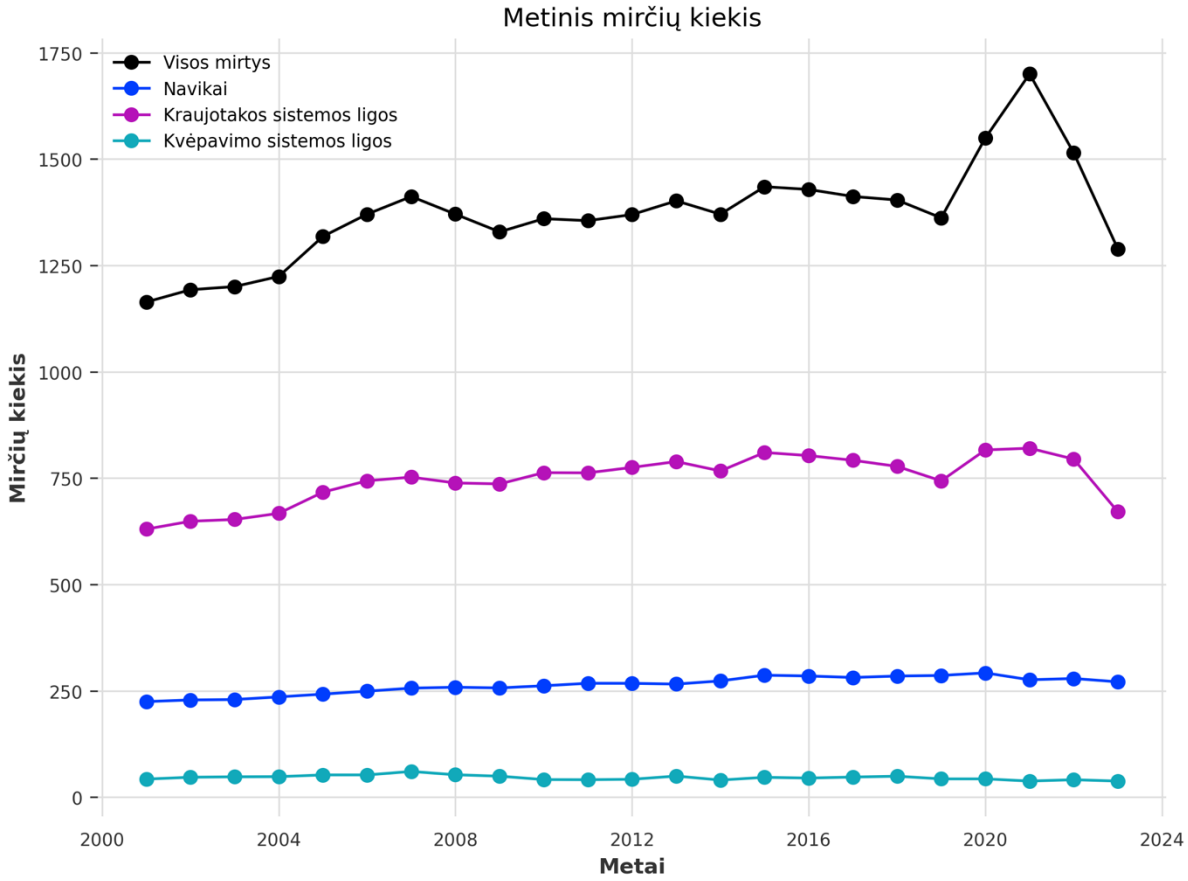
### 3.4. Sveikatos duomenų analizė

Tikslas yra ieškoti ryšių tarp metinių teršalų duomenų ir sveikatos problemų. Prieš darant analizę verta atsirinkti kategorijas, kurios bus tiriamos tolimesnėje analizėje. Pagal literatūros sąraše minėtus atskirų straipsnių radinius bei PSO rekomendacijas, nagrinėjimui galima atsirinkti šias sveikatos problemų kategorijas:

- Navikai (TLK-10-AM atitinkančios kategorijos: C00-C96);
- Kraujotakos sistemos ligos (TLK-10-AM atitinkančios kategorijos: I00-I99);
- Kvėpavimo sistemos ligos (TLK-10-AM atitinkančios kategorijos: J00-J99).

**Metinių duomenų analizė**

Minėtų kategorijų metinius duomenis verta išsitraukti iš higienos instituto pateikiamų duomenų (žr. skyrių 2.2). Tada, atsirinkus reikiamas kategorijas verta į jas pažvelgti ir panagrinėti lyginant su taršos metiniais duomenimis. Tolimesnėje analizėje nagrinėjamas mirtingumas 100 000 gyventojų, kadangi tai yra kiek labiau standartizuotas rodiklis nei absoliutus mirčių skaičius, kadangi gyventojų skaičius teritorijoje dažniausiai kinta dinamiškai (per 20 metų Lietuvos gyventojų skaičius ryškiai pakito dėl išorinių veiksnių).




**28 pav.** Metinis mirčių kiekis su pasirinktomis kategorijomis.

Verta panagrinėti, kokie metiniai duomenys galėtų būti naudojami lyginimui su šiais sveikatos duomenimis. Tam tikriausiai būtų naudinga naudoti:

- Bendrus metinių teršalų duomenis, apimančius visos Lietuvos teritoriją 2005-2020 metais (žr. skyrių 3.1);
- Automatinio monitoringo metinius duomenis, su išvestu aritmetiniu vidurkiu per visas stotis 2002-2023 metus (žr. skyrių 3.2).

Taigi, pasitelkus šių duomenų visumą verta atlikti visapusišką statistinio priklausomumo analizę[165]. Visų pirma, atliekama koreliacinė analizė tarp sveikatos kintamųjų (visos mirtys, navikai, kraujotakos sistemos ligos, kvėpavimo sistemos ligos) ir jungtinio metinių teršalų sąrašo (bendrų metinių teršalų $CO, SO_X, NO_X, PM_{2.5}, PM_{10}, Pb, HCB$ ir automatinio monitoringo vidutiniškų metinių teršalų $CO, SO_2, NO_2, PM_{2.5}, PM_{10}, O_3$).

Analizėje naudojami visi trys tipiniai koreliacijos koeficientai (Spearman‘o, Pearson‘o ir Kendall‘o) su tikslu rasti bendrai vyraujančius ryšius. Gavus koreliacijos koeficientus yra patikrinamas jų

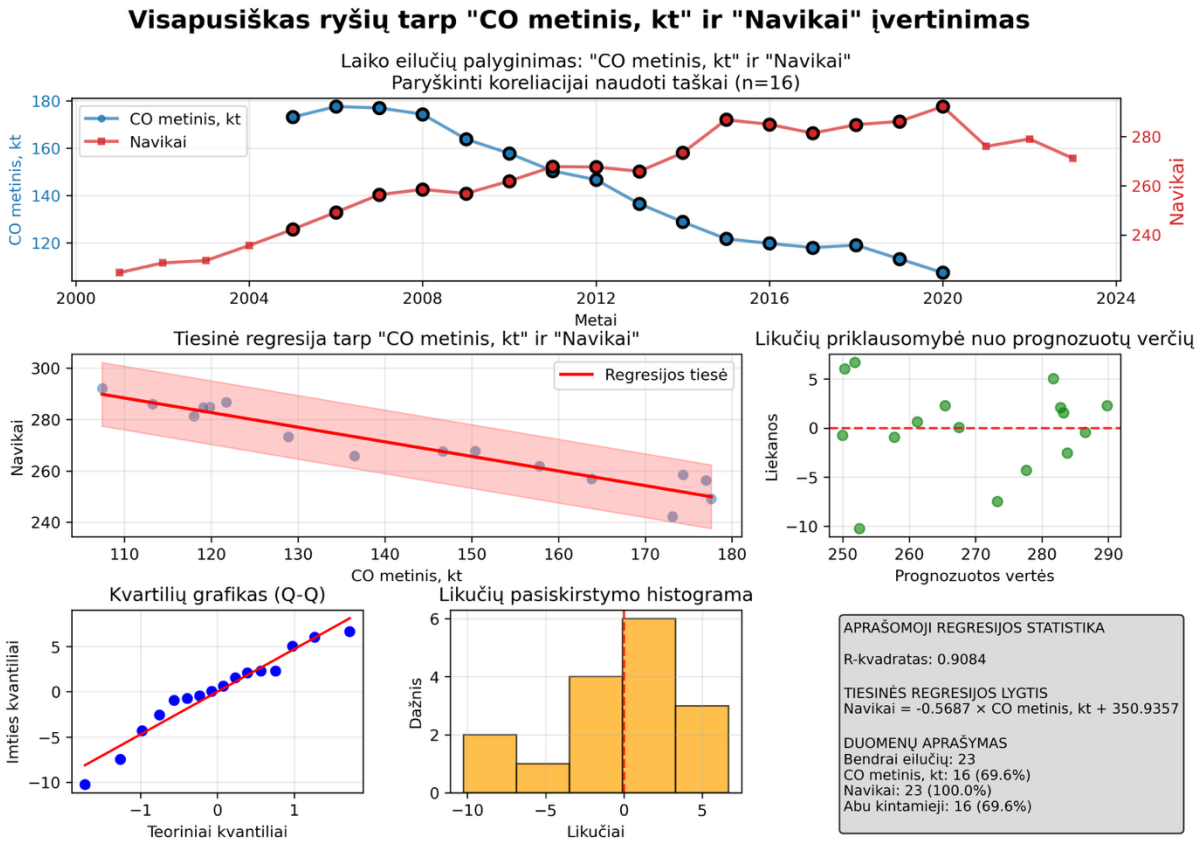
statistinis reikšmingumas pagal $p < 0{,}005$ (netipinio hipotezės atmetimo slenksčio nustatymo priežastingumas minėtas literatūros analizėje (žr. skyrių 1.2)).

Taigi, atlikus šiuos skaičiavimus ir pritaikius hipotezių atmetimą, gaunami šie ryšiai. Sąraše minimi ryšiai, kurie kartojasi dviejose iš trijų koreliacijų:

- Stiprus neigiamas ryšys tarp bendrų CO metinių teršalų ir navikų mirtingumo su statistiniais rodikliais $r_{xy} = -0{,}9596$ $(p = 0{,}0000)$, $r_s = -0{,}9545$ $(p = 0{,}0000)$, $\tau = -0{,}8545$ $(p = 0{,}0000)$;
- Stiprus neigiamas ryšys tarp bendrų $SO_X$ metinių teršalų ir navikų mirtingumo su statistiniais rodikliais $r_{xy} = -0{,}8974$ $(p = 0{,}0002)$, $r_s = -0{,}8818$ $(p = 0{,}0003)$, $\tau = -0{,}7818$ $(p = 0{,}0003)$;
- Stiprus neigiamas ryšys tarp bendrų Pb metinių teršalų ir navikų mirtingumo su statistiniais rodikliais $r_{xy} = -0{,}8939$ $(p = 0{,}0002)$, $r_s = -0{,}8273$ $(p = 0{,}0017)$;
- Stiprus teigiamas ryšys tarp $PM_{10}$ automatinio monitoringo teršalų vidurkio ir kvėpavimo sistemos ligų mirtingumo su statistiniais rodikliais $r_{xy} = 0{,}7819$ $(p = 0{,}0045)$, $r_s = 0{,}8455$ $(p = 0{,}0010)$.

Toliau verta atlikti kuo labiau visapusišką ryšių įvertinimą. Tam pasitelkiami grafikai, kuriuose pavaizduojama:

- Abiejų nagrinėjamų elementų kreivės bendroje laiko eilutėje su paryškintais elementais, įtrauktais į koreliacijos skaičiavimą;
- Tiesinės regresijos grafikas su 0,995 pasikliautinumo intervalu ($\alpha = 0{,}005$; $z_\alpha = 2{,}81$)[166];
- Taškinė diagrama pagal prognozuotas vertes ir likučius;
- Imties kvartilių ir teorinių kvartilių palyginimo grafikas (angl. *Q-Q plot*) normalumui patikrinti[161];
- Likučių pasiskirstymo histograma;
- Aprašomoji regresijos statistika.



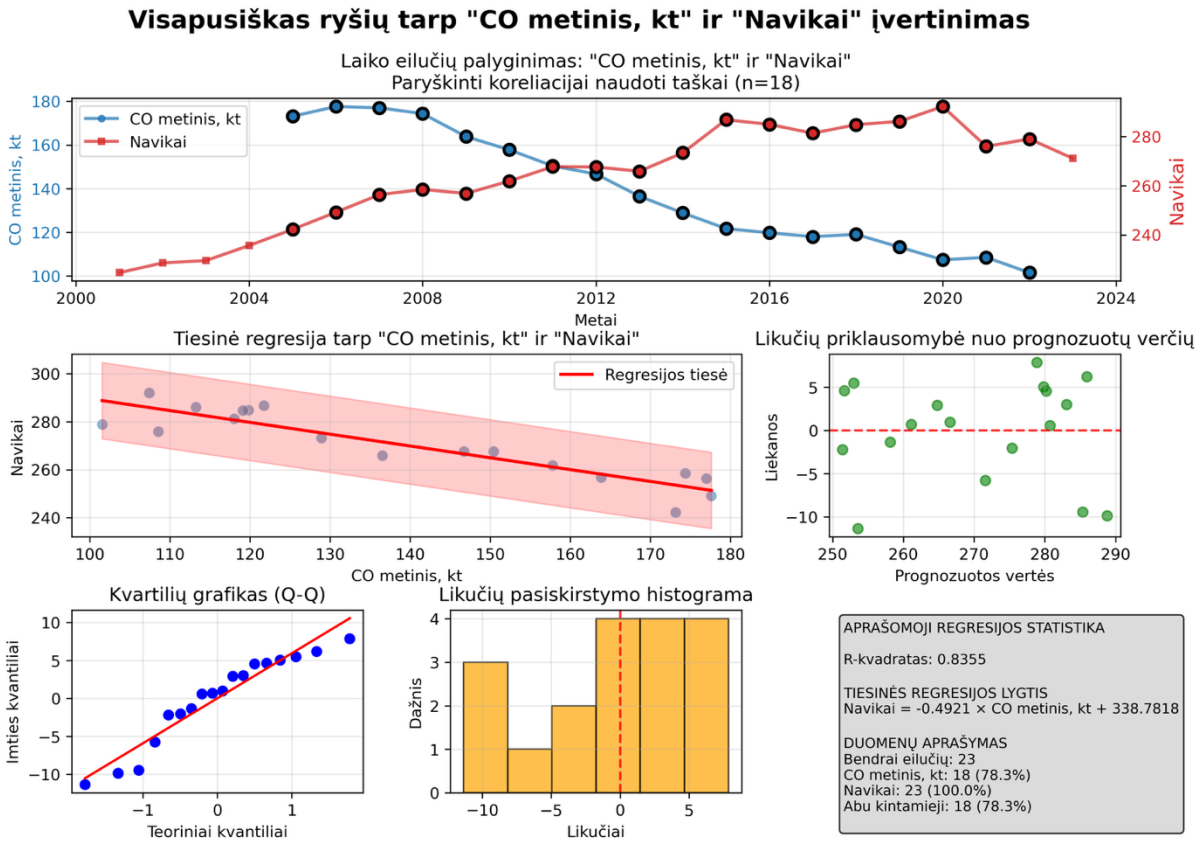

**29 pav.** Visapusiškas ryšių tarp metinių CO duomenų ir navikų mirtingumo įvertinimas.

Matant šiuos rezultatus **29 pav.** ir galima būtų teigti, jog rastas reikšmingas ryšys tarp CO metinės taršos ir mirtingumo dėl navikų priežasčių. Tačiau, remiantis bet kokia sveiko proto logika, šis rezultatas turėtų būti iš karto atmestas. Ką tokioje situacijoje daryti tyrėjui? Statistiškai reikšmingų rezultatų išmetimas iš tyrimo labai prieštarautų moksliniam metodui (angl. *scientific method*). Taigi, norint paneigti šį rezultatą, tai reikia daryti pagrįstai.

Pažvelgus į laiko eilučių palyginimą, galima matyti, jog radus teisingą šaltinį duomenų papildymui, galbūt galima būtų prailginti CO metinės taršos laiko eilutę ir iš naujo patikrinti išvadą.

Sėkmė šiuo analizės atveju šypsosi ir randamas oficialios statistikos šaltinis[168], pateikiantis duomenis kaip tik 2020-2022 metams (taigi, validumą patikrinti galima su persidengiančia reikšme).

2020 metų reikšmė originaliuose duomenyse yra lygi 107,421392, o naujame šaltinyje ši reikšmė lygi 106,33408. Šis skirtumas yra pakankamai nedidelis (mažesnis 1,01%), taigi tarkime, jog šios reikšmės yra pakankamai panašios, jog būtų galima pridėti papildomų dviejų metų duomenis (iš šio šaltinio taip pat pasiimami dar ir $SO_X$ dviejų metų duomenys). Pridėjus šiuos duomenis vėl atliekama koreliacijų ir regresijų analizė.

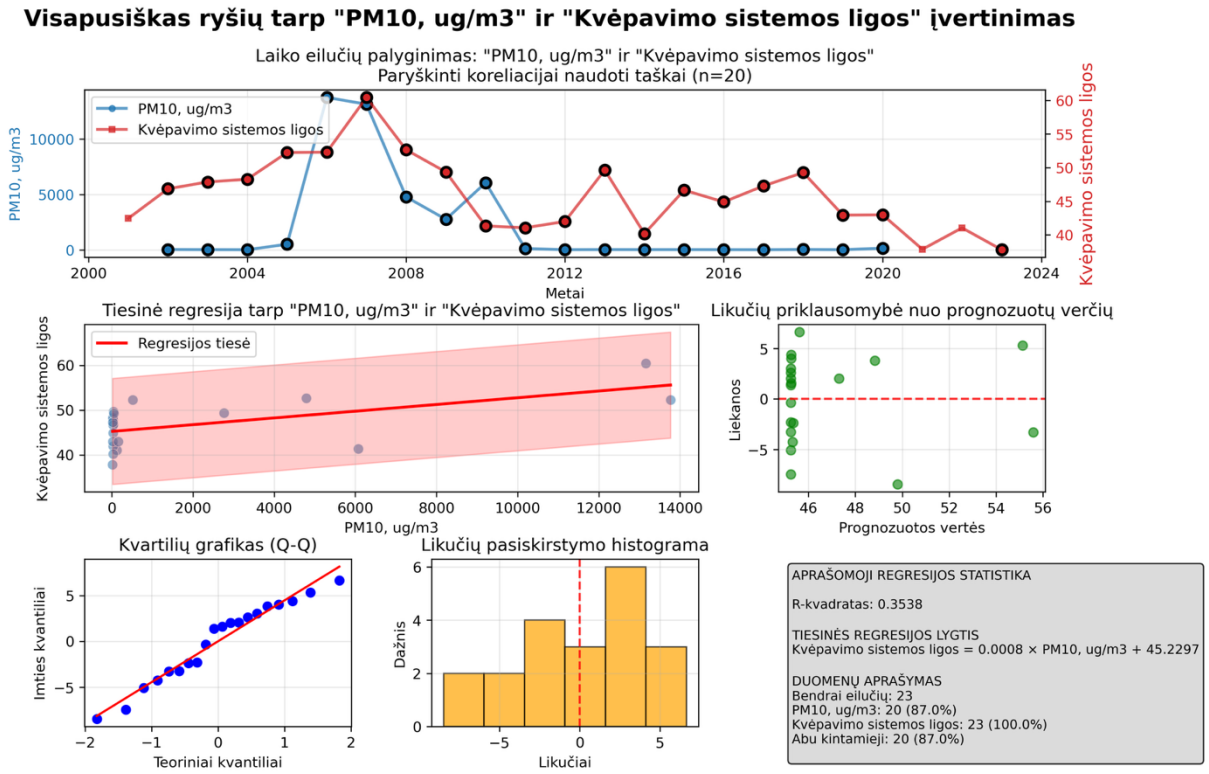



**30 pav.** Visapusiškas ryšių tarp metinių CO duomenų (su papildytais 2021 ir 2022 metais) ir navikų mirtingumo įvertinimas.

Panašu, kad papildomi duomenys rezultatų drastiškai nepakeitė, nors truputį sumažėjo $R^2$ statistika.

Regresijos analizė tarp $SO_X$ bei Pb metinių teršalų ir navikų mirtingumo atrodo praktiškai identiškai, kadangi visų šių teršalų matavimai tiesiog krinta kasmet, todėl analizės išvada gaunasi tokia pati. Ar radus tokį ryšį galima sakyti, jog navikų mirtingumas didėja dėl šių rodiklių mažėjimo? Tikriausiai, jog ne, tačiau ta pagrįsti pagrindo kol kas irgi nerasta.

Tačiau toliau verta panagrinėti vienintelį teigiamą reikšmingą ryšį – $PM_{10}$ automatinio monitoringo teršalų vidurkio ir kvėpavimo sistemos ligų mirtingumo.

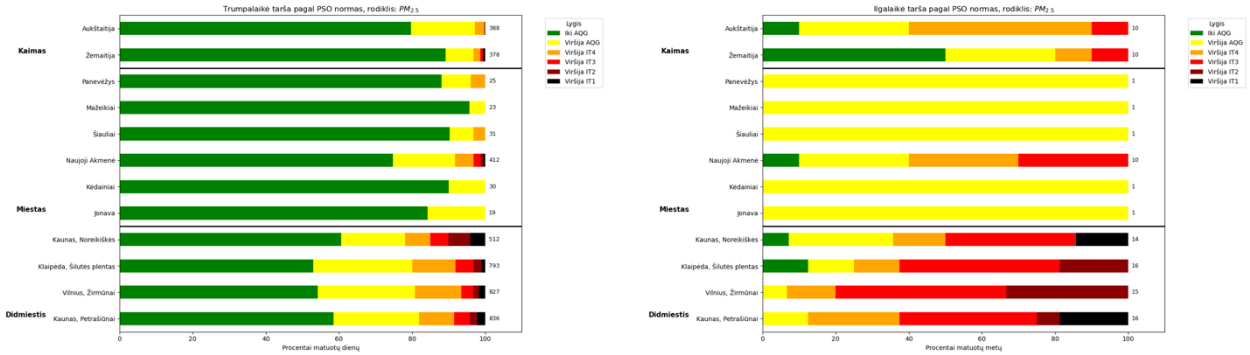



**31 pav.** Visapusiškas ryšių tarp metinio automatinio monitoringo $PM_{10}$ vidurkio duomenų ir kvėpavimo sistemos ligų mirtingumo įvertinimas.

Šiuo atveju matome, jog egzistuoja keletas neįtikėtinai aukštų metinių vidurkių, kurie mažina regresijos tikslumą. Taigi, remiantis šiais duomenimis nebūtų korektiška užtikrintai teigti, jog tikrai egzistuoja ryšys tarp mirtingumo nuo kvėpavimo sistemos ligų ir kietųjų dalelių koncentracijos.

### PSO normų atitikimas

Kaip buvo minėta literatūros apžvalgoje (žr. skyrių 1.4.1), PSO yra nustačiusi mirtingumui įtaką darančias ribas, pagal kurias yra sukurti skirtingi pavojingumo lygiai. Taigi, verta apžvelgti šių PSO rekomendacijų atitikimą pagal automatinio monitoringo duomenis.




**(a)** Trumpalaikė tarša.

**(b)** Ilgalaikė tarša.

**32 pav.** Trumpalaikės ir ilgalaikės $PM_{2.5}$ taršos PSO normų atitikimas oro monitoringo stotyse.

Matome, jog PM2.5 tarša:

– Didmiesčiuose apie pusę visų dienų viršija trumpalaikę AQG lygį;

– Didmiesčiuose apie penktadalį dienų yra viršijama IT4 lygio riba, taip pat kiekviena stotis turi išmatuotų dienų, viršijančių netgi IT1 lygio ribą (kas yra labai didelė tarša);
– Miestuose bei kaimuose trumpalaikis AQG lygio viršijimas yra mažesnis, arčiau vieno ketvirčio;
– Ilgalaikė tarša praktiškai visose stotyse (išskyrus Žemaitiją) viršija AQG lygį didžiąją metų dalį;
– Didmiesčiuose ilgalaikės taršos lygis daugiau nei pusę laiko yra didesnis už IT3 lygį.

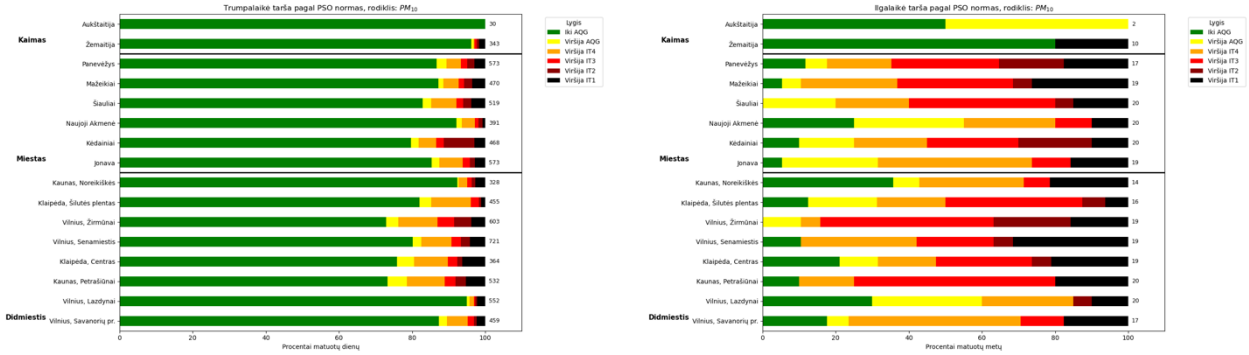
Pažvelgus į šiuos grafikus, rezultatai tikrai neramina. Tačiau kokį tai gali lemti mirtingumo pokytį? Grįžus prie lygių aprašo kiekvienam rodikliui, galima rasti neatsitiktinio mirtingumo padidėjimo koeficientus su kiekvienu lygiu. Turint jų vertes, galima būtų atlikti koeficientų sandaugą su kiekvieno lygmens procentais, tokiu būdu randant bendrą mirtingumo padidėjimo koeficientą, kas būtų užrašoma (38) formule:

$$k_{bendras} = \sum_{i \in lygiai} p_i k_i \,; \tag{38}$$

čia $k_{bendras}$ – bendras mirtingumo padidėjimo koeficientas; $p_i$ – $i$-tojo lygio procentinė matavimų dalis, $\sum_{i \in lygiai} p_i = 1$; $k_i$ – $i$-tojo lygio mirtingumo koeficientas; $lygiai$ – PSO rekomendacijas viršijantys lygiai[30].

Šiuo atveju apibendrinamas visų didmiesčių bendras ilgalaikis koeficientas ir apskaičiuojama jo vertė $\left(p_{<AQG} + p_{AQG<IT1}\right)k_{AQG} + p_{IT4}k_{IT4} + p_{IT3}k_{IT3} + p_{IT2}k_{IT2} + p_{IT1}k_{IT1} = (0{,}17 + 0{,}1) \cdot 1 + 0{,}23 \cdot 1{,}04 + 0{,}26 \cdot 1{,}08 + 0{,}05 \cdot 1{,}16 + 0{,}18 \cdot 1{,}24 = 1{,}0712$. Pilnavertiškai pasitikint PSO rekomendacijomis, šis rezultatas reikštų, jog $PM_{2.5}$ taršos normų nesilaikymas didmiesčiuose lėmė bent[31] 7,12% padidėjusį mirtingumą nuo visų priežasčių.

Toliau išvardinti kiti teršalai iš PSO rekomendacijų sąrašo.

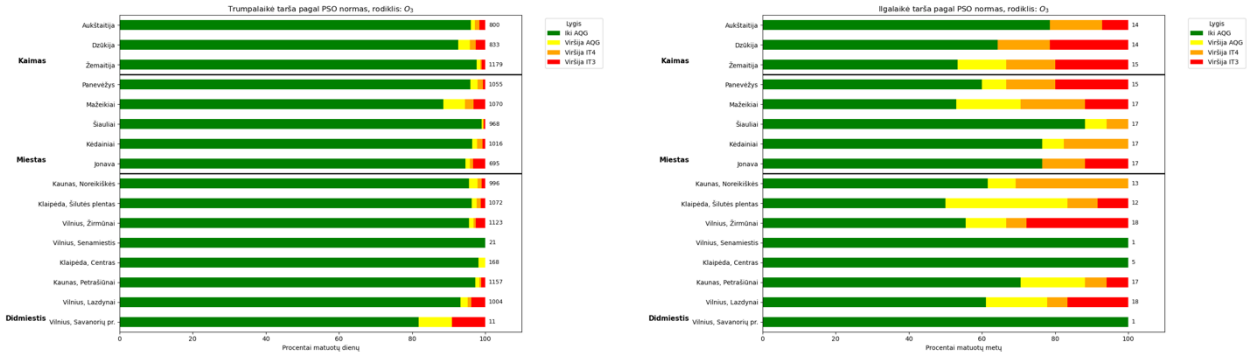



**(a)** Trumpalaikė tarša. **(b)** Ilgalaikė tarša.

**33 pav.** Trumpalaikės ir ilgalaikės $PM_{10}$ taršos PSO normų atitikimas oro monitoringo stotyse.

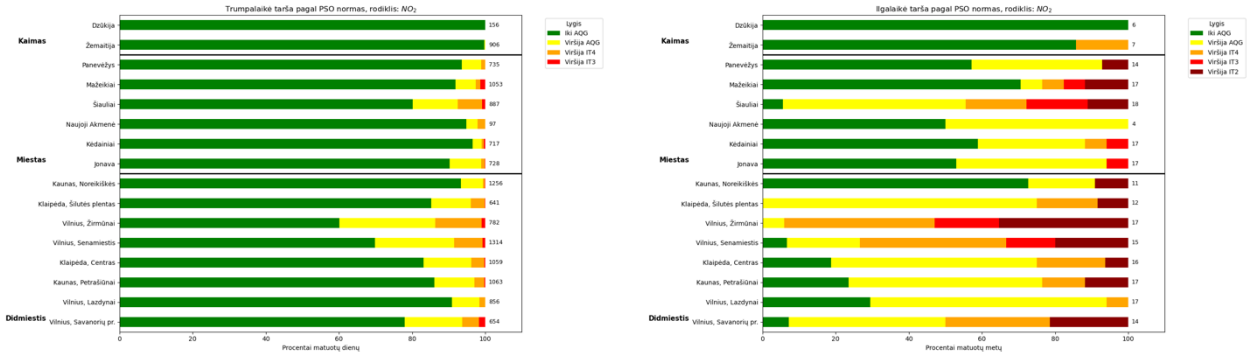
[30] Nors PSO yra pateikę dinamiškas vertes koncentracijos ir mirtingumo sąsajoms, tačiau šiuose skaičiavimuose bus pasirinkta tik ribos reikšmė.

[31] Kadangi pasirinktos minimalios lygių ribų vertės, šis skaičius, apskaičiuojant jo reikšmę dinamiškais koeficientais, būtų didesnis. Reikia nepamiršti, jog AQG ribos nepasiekimas mažina mirtingumą, todėl dinamiškas išmatavimas turėtų atsižvelgti ir į tai.



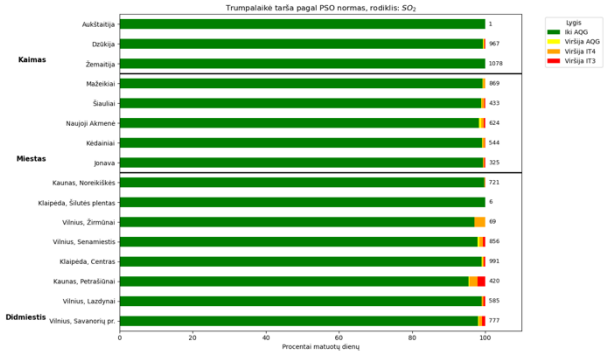

**(a)** Trumpalaikė tarša. **(b)** Ilgalaikė tarša.

**34 pav.** Trumpalaikės ir ilgalaikės $O_3$ taršos PSO normų atitikimas oro monitoringo stotyse.

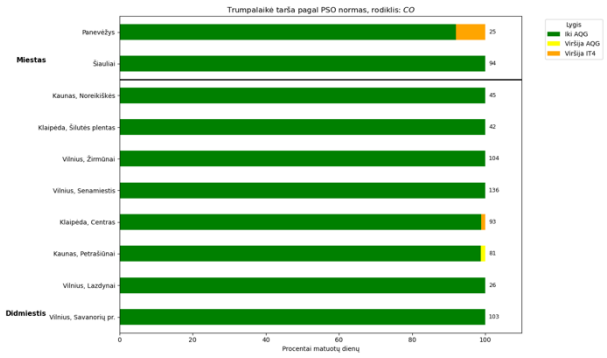



**(a)** Trumpalaikė tarša. **(b)** Ilgalaikė tarša.

**35 pav.** Trumpalaikės ir ilgalaikės $NO_2$ taršos PSO normų atitikimas oro monitoringo stotyse.




**36 pav.** Trumpalaikės $SO_2$ taršos PSO normų atitikimas oro monitoringo stotyse.




**37 pav.** Trumpalaikės CO taršos PSO normų atitikimas oro monitoringo stotyse.

Galima matyti, jog CO bei $SO_2$ kategorijos, daugmaž atitinkančios CO ir $SO_X$ kategorijas iš prieš tai aprašytos regresijos analizės, PSO rekomendacijų ribų beveik neviršija, todėl tikriausiai apčiuopiamos įtakos mirtingumui neturi.

Tačiau matoma, jog dažnai gerokai viršijamos $PM_{2.5}$ ir $NO_2$ ribos, o $O_3$ taršos AQG lygis taip pat įprastai viršijamas kiek mažiau nei pusę visų metų. Norint ištirti šias sąsajas detaliau, reiktų gilintis į lokalias vietoves ir jų rodiklius, o ne į valstybinio masto duomenis, tačiau šiame tyrime į tai nėra gilinamąsi.

## 3.5. Ryšiai tarp oro taršos šaltinių ir oro taršos rodiklių

Ryšių tarp oro taršos šaltinių ir oro taršos rodiklių nustatymui yra sukuriama minimizavimo programa, kuri, pagal teorinėje dalyje aprašytą modelį (žr. skyrių 2.6), ieško skirtingų taršos šaltinių kategorijų įtakos koeficientų. Minimizavimo funkcija neturi apribojimų[169], dėl to optimizavimui naudojamas vienas iš galimų kvazi-Niutono metodų – L-BFGS (angl. *Limited-memory Broyden, Fletcher, Goldfarb, and Shanno*)[170,171]. Visi skaičiavimai atliekami kompiuteriu, kurio parametrai nurodomi lentelėje **6 lentelė**.

**6 lentelė.** Skaičiavimams naudoto kompiuterio parametrai.

| **Komponentas** | **Specifikacija** |
|---|---|
| Modelis | Mac16,7 |
| Procesorius | 14 branduolių |
| RAM | 24 GB |
| Vaizdo plokštė | Apple M4 Pro |
| Atmintis | 494.38 GB (APPLE SSD AP0512Z) |
| OS | macOS 15.4.1 (Build 24E263) |
| Kompiliatorius | Apple clang 17.0.0 (clang-1700.0.13.3) |

Optimizacija pradedama nustatant visų kategorijų koeficientams vienodą, vienetinę, reikšmę. Tam, kad metodas būtų įvykdomas per žmogiškai suvokiamą laiką, yra sumažinamas įvesčių kiekis. Tai padaroma aplink oro monitoringo stotį apibrėžiant skirtulį su 10 kilometrų spinduliu ir į optimizaciją įtraukiant tik tas stotis, kurios yra šiame plote. Šiuose skaičiavimuose sklaidos greičio vertė yra paimama kaip 1 ir nėra keičiama. Parametrai, kurie yra optimizuojami yra:

– Objekto būklės koeficientai (stulpelis „ptz_objekto_bukle“[32]);

[32] Šie stulpeliai yra surenkami iš stacionarių taršos šaltinių duomenų rinkinio (žr. skyrių 2.6).

– Objekto tipo koeficientai (stulpelis „ptz_objekto_tipas“);
– Objekto pavojingumo aplinkai koeficientai (stulpelis „ptz_pavojingumas_aplinkai“);
– Papildomų reikšmių koeficientai:
    o Santykinė drėgmė (žymima „relative_humidity, %“);
    o Temperatūra (žymima „TEMP (temperature), K“);
    o Slėgis (žymimas „PRES (atm. pressure), Pa“);
    o Dienos valanda (žymima „hour_of_day“);
    o Šaltinio amžius dienomis (žymimas „age_days“), gaunamas iš *i*-tojo matavimo laiko vertės (angl. *timestamp*) atėmus stulpelio „ptz_anketos_data“ vertę. Šiame rodiklyje neigiama reikšmė gali reikšti matavimą prieš taršos šaltinio pastatymą.

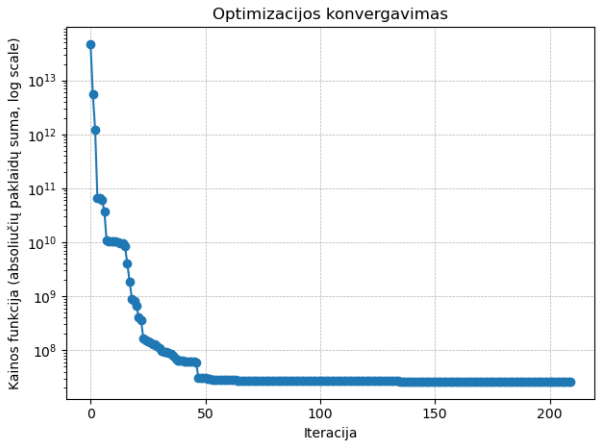
**Individualių rodiklių nagrinėjimas**

Šio skyrelio poskyriuose pavaizduojami gauti optimizacijos rodikliai su tam tikrais teršalais, o jų apžvelgimas atliekamas po to.

***$SO_2$***

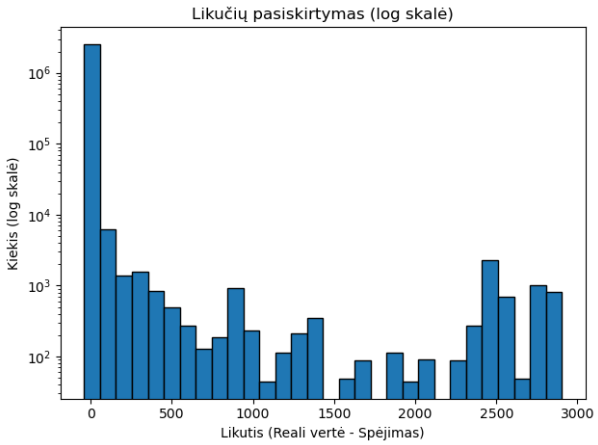
Visų pirma pabandyta metodą pritaikyti dar šiame darbe nelabai nagrinėtam teršalui – $SO_2$.

Optimizavimas užtruko 68 min 50 s, pereitos 209 iteracijos ir prieitas konvergavimas, todėl metodas sustojo (originaliai apribotas ties 500 iteracijų). Žemiau pavaizduotas iteracijų konvergavimo grafikas (žr. **38 pav.**).



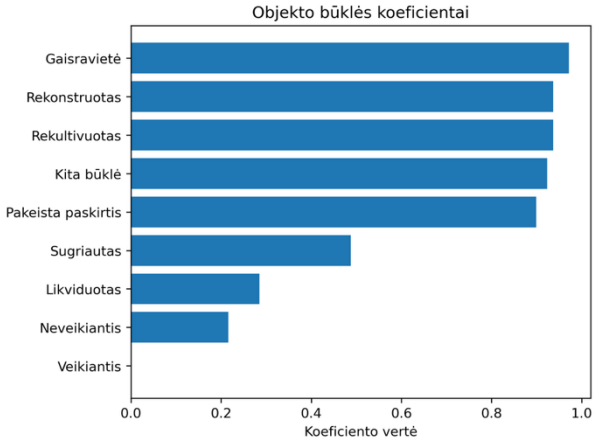

**38 pav.** Optimizacijos konvergavimas $SO_2$ teršalo priklausomybės nuo taršos šaltinių funkcijoje.

Pasibaigus optimizacijai galima pažvelgti į jos tikslumą pagal likučius, kadangi optimizacijos konvergavimo grafikas nėra labai informatyvus spėjimų tikslumo požiūriu.

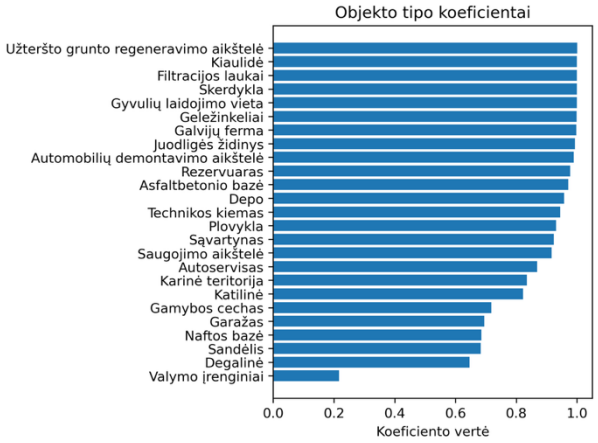



**39 pav.** Likučių pasiskirstymas optimizavus $SO_2$ teršalo priklausomybę nuo taršos šaltinių.

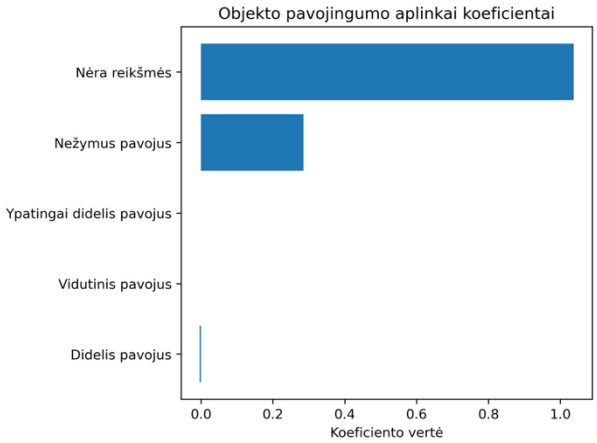
Matome, jog didžioji dalis likučių yra išties maži ir koncentruojasi apie 0 vertę, tačiau akivaizdu, jog metodas nesugebėjo visko apibendrinti, kadangi vis dar yra didelės koncentracijos nesutapimų. Beje, likučių didelė dalis gali būti maža, nes patys matavimai $SO_2$ laiko eilutėje yra tikrai ganėtinai nedideli (žr. **36 pav.**). Verta pažvelgti, kaip atrodo skirtingų kategorijų koeficientai **40 pav.** grafikuose.



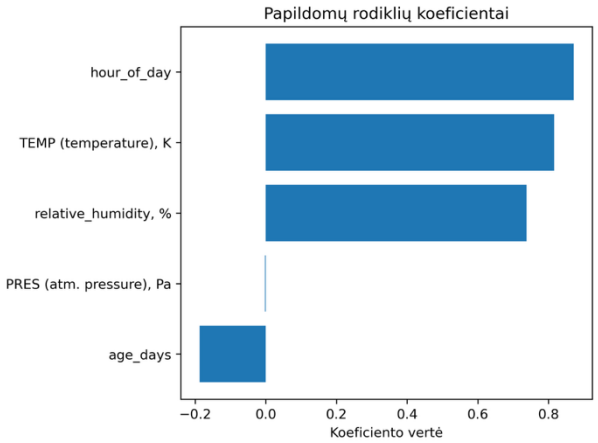

**(a)** Objekto būklės koeficientai.




**(b)** Objekto tipo koeficientai.




**(c)** Objekto pavojingumo aplinkai koeficientai.



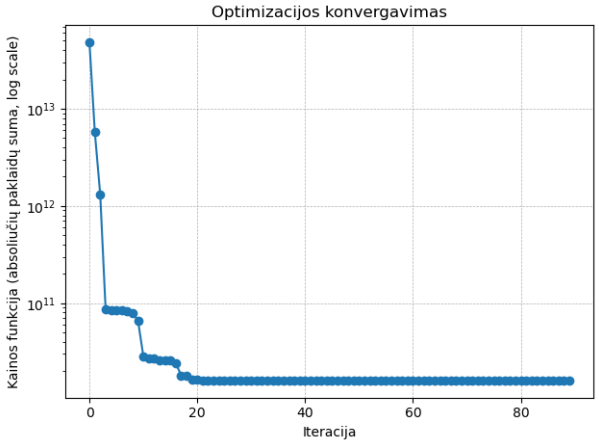

**(d)** Papildomų rodiklių koeficientai.

**40 pav.** $SO_2$ taršos rodiklių ir šaltinių sąsajos optimizacija gautos kategorijų koeficientų vertės.

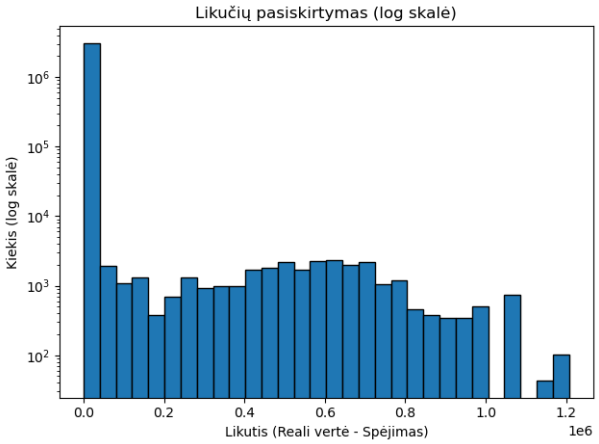
### *$PM_{10}$*

Toliau verta pažvelgti į vieną iš labiausiai ribas viršijančių rodiklių, kuris turi daug matavimų ir didelę įtaką žmonių sveikatai – $PM_{10}$.

Optimizavimas užtruko 34 min 59 s, pereitos 89 iteracijos ir prieitas konvergavimas, todėl metodas sustojo.

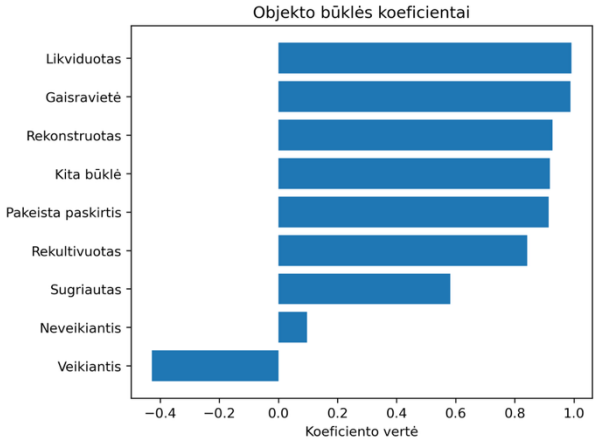



**(a)** Optimizacijos konvergavimas $PM_{10}$ teršalo priklausomybės nuo taršos šaltinių funkcijoje.

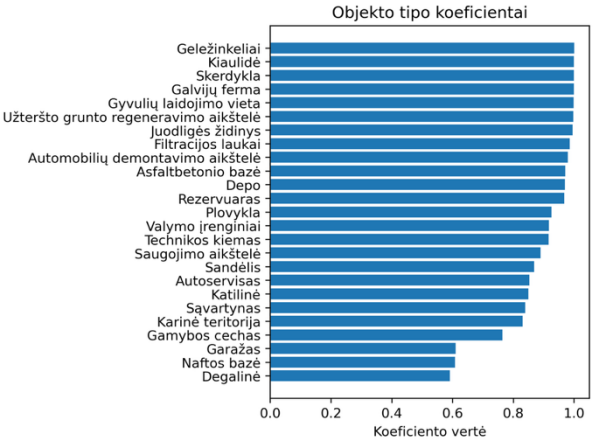



**(b)** Likučių pasiskirstymas optimizavus $PM_{10}$ teršalo priklausomybę nuo taršos šaltinių.

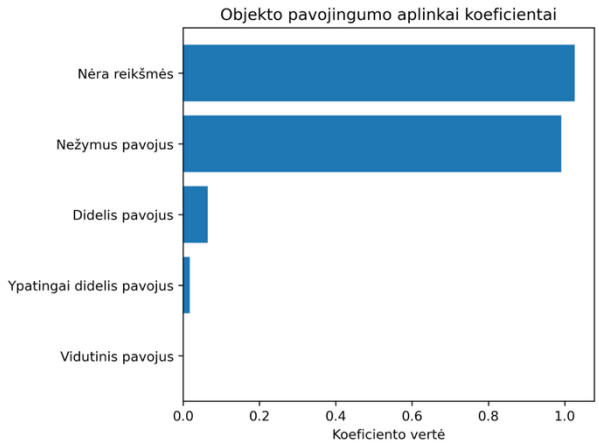
**41 pav.** PM10 teršalo optimizacijos tikslumo rodikliai.



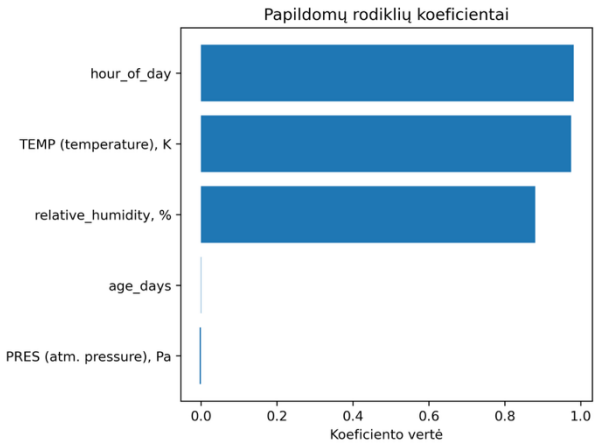

**(a)** Objekto būklės koeficientai.




**(b)** Objekto tipo koeficientai.




**(c)** Objekto pavojingumo aplinkai koeficientai.



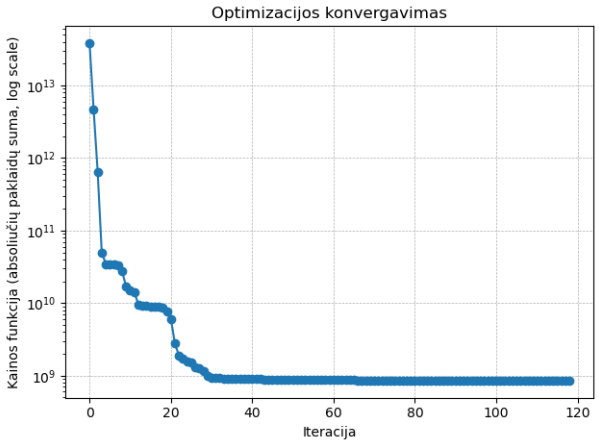

**(d)** Papildomų rodiklių koeficientai.

**42 pav.** $PM_{10}$ taršos rodiklių ir šaltinių sąsajos optimizacija gautos kategorijų koeficientų vertės.

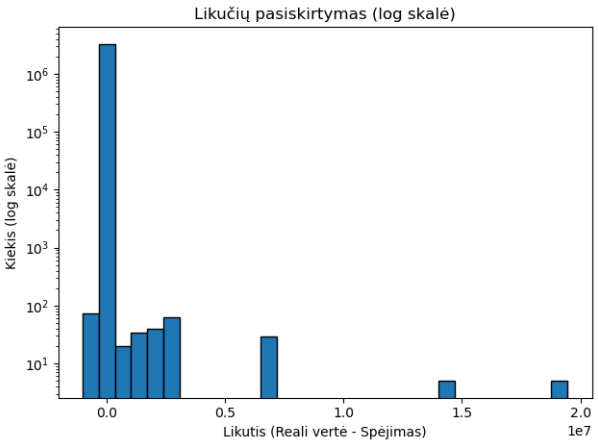
### *$O_3$*

Galiausiai verta pažvelgti į dideliais kiekiais turbūt pavojingiausią sveikatai rodiklį– $O_3$.

Optimizavimas užtruko 45 min 46 s, pereita 118 iteracijų ir prieitas konvergavimas, todėl metodas sustojo.

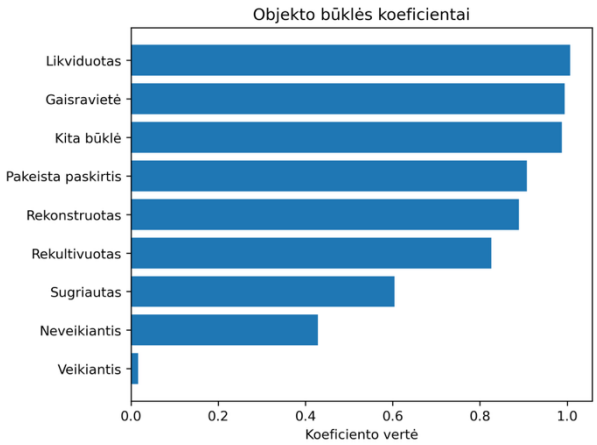



**(a)** Optimizacijos konvergavimas $O_3$ teršalo priklausomybės nuo taršos šaltinių funkcijoje.

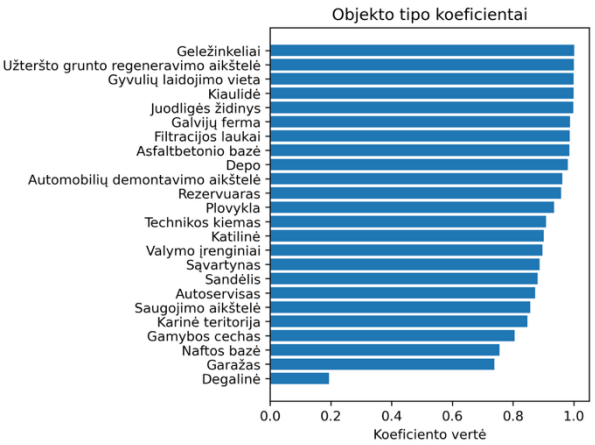



**(b)** Likučių pasiskirstymas optimizavus $O_3$ teršalo priklausomybę nuo taršos šaltinių.

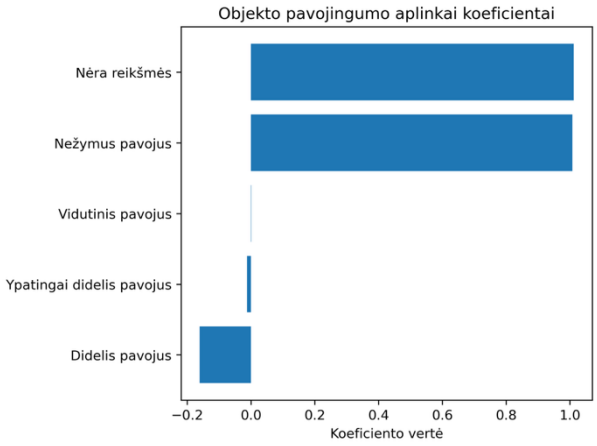
**43 pav.** $O_3$ teršalo optimizacijos tikslumo rodikliai.



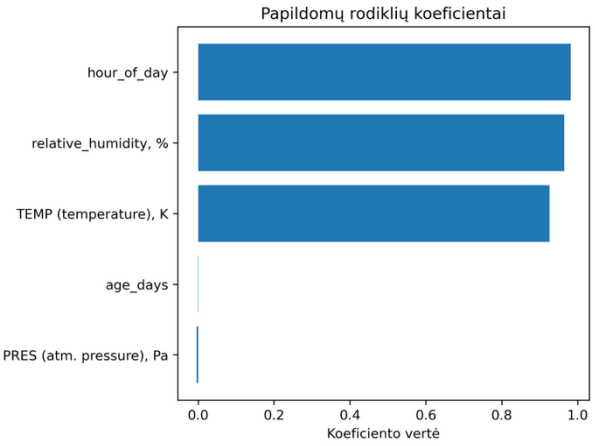

**(a)** Objekto būklės koeficientai.




**(b)** Objekto tipo koeficientai.




**(c)** Objekto pavojingumo aplinkai koeficientai.

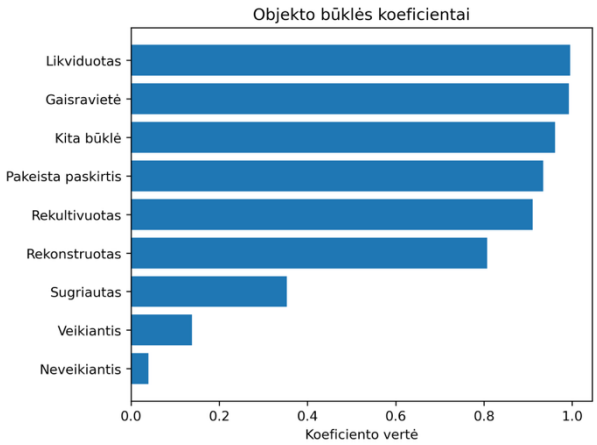



**(d)** Papildomų rodiklių koeficientai.

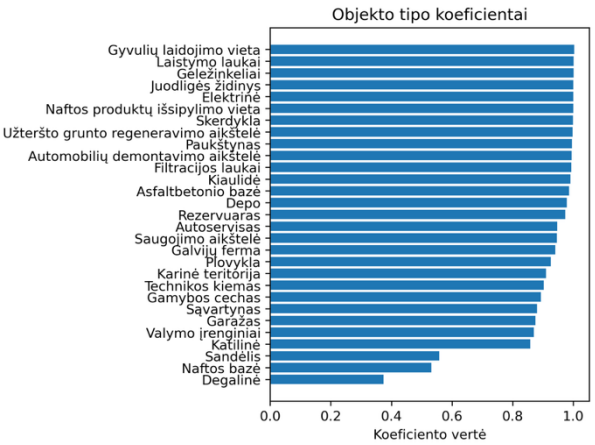
**44 pav.** $O_3$ taršos rodiklių ir šaltinių sąsajos optimizacija gautos kategorijų koeficientų vertės.

Tačiau kyla klausimas, ar pasikeistų kategorijų koeficientai, modeliuojant ne 10, bet 20 kilometrų spinduliu. Šis klausimas patikrintas $O_3$ teršalo kategorijoje, tačiau po 90 iteracijų konvergavimas pasiektas ir galutinis funkcijos rezultatas gautas $4{,}7 \cdot 10^9$, kai mažesnio spindulio optimizacijoje ši vertė buvo $8{,}5 \cdot 10^8$, kas reiškia, jog didesnis spindulys nebūtinai patobulina metodo rezultatą.

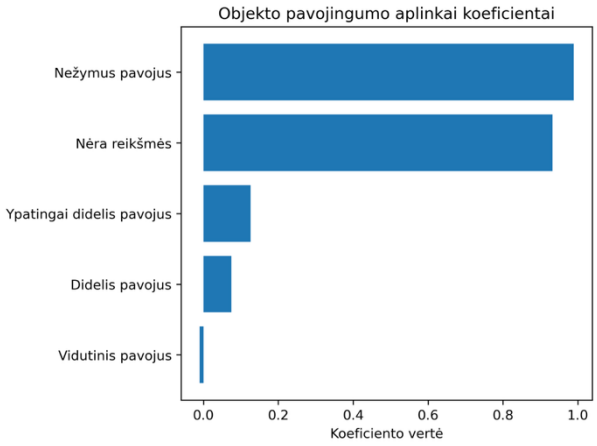
Tačiau verta pažiūrėti, ar pasikeitė kategorijų koeficientai.



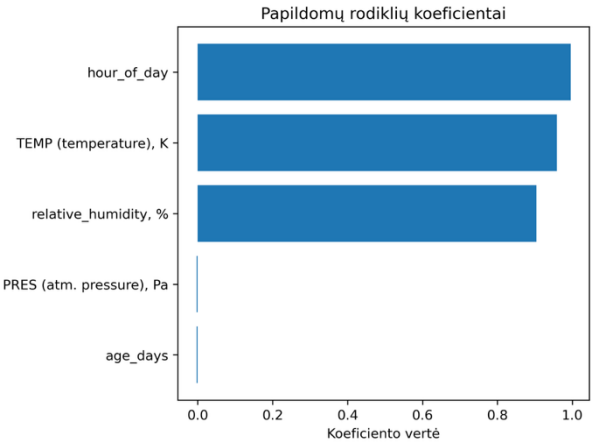

**(a)** Objekto būklės koeficientai.




**(b)** Objekto tipo koeficientai.




**(c)** Objekto pavojingumo aplinkai koeficientai.




**(d)** Papildomų rodiklių koeficientai.

**45 pav.** $O_3$ taršos rodiklių ir šaltinių sąsajos optimizacija gautos kategorijų koeficientų vertės, kai spindulys yra 20 km.

### Stacionarių taršos šaltinių įtakos apžvalga

Prieš apžvelgiant kitas kategorijas verta paminėti, jog šios optimizacijos apribojimai buvo 10 km (su ozonu dar ir 20 km) spindulio apribojimas, kas gali lemti didelę įtaką kai kurių kategorijų nepakankamam įtraukimui.

Matome, kad SO2 turėjo neigiamą koeficientą objekto amžiaus kategorijoje, kuris galėtų reikšti didelę taršą statybų metu. Visų teršalų rodikliams didelę įtaką daro temperatūros, dienos meto ir santykinio drėgnio papildomų rodiklių koeficientai. Veikiantys objektai skleidžia mažesnę taršą nei kitų būklės kategorijų. Pavojingiausiai įvertintų kategorijų objektai taip pat nerasta, jog darytų didelę įtaką taršai.

Šis optimizacijos uždavinys rėmėsi daugelio realaus pasaulio efektų supaprastinimu, todėl norint ypač tiksliai gauti taršos šaltinių įtaką taršai, tikriausiai tektų naudotis kompleksiškais neuroninio tinklo modeliais, besiremiančiais vektorinio lauko modeliavimu.

## Išvados

1. Išanalizuoti atvirų oro taršos bei sveikatos duomenų šaltiniai, aptartos jų problemos bei privalumai, atrinkti naudingiausi duomenų rinkiniai;
2. Atlikta vienmatė bei daugiamatė laiko eilučių analizė. Atliktas trūkstamų duomenų papildymas bei panaudoti metodai išskirčių pašalinimui. Surastos oro taršos rodiklių tendencijos. Patikrinti ryšiai tarp oro taršos rodiklių;
3. Išnagrinėti egzistuojantys ryšiai tarp oro taršos ir sveikatos rodiklių. Rezultatai palyginti su panašiais tyrimais;
4. Sukurtas modelis skirtas taršos šaltinių įtakai nuspėti, juo rasti ryšiai tarp specifinių oro taršos rodiklių ir taršos šaltinių kategorijų.

Išankstinės hipotezės užtikrintai patvirtinti pagrindo nerasta, tačiau tyrime yra apskaičiuotas taršos ($PM_{2.5}$) galimas poveikis visuomenės mirtingumui.

**Rekomendacijos tolimesnei veiklai.**

1. Toliau nagrinėti sveikatos ir taršos rodiklių sąsajas, tačiau verta specifiškai gilintis į ribas viršijančius rodiklius;
2. Sukurti regioninį oro taršos modeliavimo metodą naudojant vektorių lauką ir platesnį duomenų šaltinių spektrą;
3. Išnagrinėti sąsajas tarp palydovinių oro taršos rodiklių ir antžeminių stočių rodiklių duomenų.

## Literatūros sąrašas


1. DI PIAZZA, A. F. Lo CONTI. L.V. NOTO. ir kt. Comparative analysis of different techniques for spatial interpolation of rainfall data to create a serially complete monthly time series of precipitation for Sicily, Italy. *International Journal of Applied Earth Observation and Geoinformation* [interaktyvus]. 2011, Vol. 13, no. 3, p. 396–408. [žiūrėta 2025-05-14]. Prieiga per: https://linkinghub.elsevier.com/retrieve/pii/S0303243411000225.
2. BAGDONAVIČIUS, Vilijandas. Julius Jonas KRUOPIS. ir Rūta LEVULIENĖ. Parametrinė statistika. *Matematinės statistikos uždavinynas su sprendimais* [interaktyvus]. Vilnius: Vilniaus Universiteto leidykla, 2019, p. 14. [žiūrėta 2025-05-21]. ISBN 978-609-07-0247-5Prieiga per: http://www.statistika.mif.vu.lt/wp-content/uploads/2019/09/Matematines-statistikos-uzdavinynas.pdf.
3. BLÁZQUEZ-GARCÍA, Ane. Angel CONDE. Usue MORI. ir kt. A Review on Outlier/Anomaly Detection in Time Series Data. *ACM Computing Surveys* [interaktyvus]. 2022, Vol. 54, no. 3, p. 1–33. [žiūrėta 2025-05-10]. Prieiga per: https://dl.acm.org/doi/10.1145/3444690.
4. Kas yra mūsų era ir prieš Kristų? *Maldos.lt* [interaktyvus]. [žiūrėta 2025-05-01]. Prieiga per: https://www.maldos.lt/kas-yra-musu-era/.
5. RADAVIČIUS, Marijus. vidutinis kvadratinis nuokrypis. *Visuotinė lietuvių enciklopedija* [interaktyvus]. [žiūrėta 2025-05-14]. Prieiga per: https://www.vle.lt/straipsnis/vidutinis-kvadratinis-nuokrypis/.
6. Mean squared error. *Wikipedia* [interaktyvus]. 2025, [žiūrėta 2025-05-14]. Prieiga per: https://en.wikipedia.org/w/index.php?title=Mean_squared_error&oldid=1289884489.
7. *WHO Global Air Quality Guidelines: Particulate Matter (PM2. 5 and PM10), Ozone, Nitrogen Dioxide, Sulfur Dioxide and Carbon Monoxide*. 1st ed. Ed. Geneva: World Health Organization, 2021, 1 p. ISBN 978-92-4-003422-8.
8. CLARK, Isobel. What is kriging anyway? *kriging.com* [interaktyvus]. [žiūrėta 2025-05-14]. Prieiga per: https://www.kriging.com/whatiskriging.html.
9. CLARK, Isobel. Geostatistics. *The Royal School of Mines Journal* [interaktyvus]. 1978, [žiūrėta 2025-05-14]. Prieiga per: https://www.kriging.com/RSMA1978/.
10. Kriging. *Wikipedia* [interaktyvus]. 2025, [žiūrėta 2025-05-14]. Prieiga per: https://en.wikipedia.org/w/index.php?title=Kriging&oldid=1277903866.
11. How Kriging works. *esri* [interaktyvus]. [žiūrėta 2025-05-14]. Prieiga per: https://pro.arcgis.com/en/pro-app/latest/tool-reference/3d-analyst/how-kriging-works.htm.
12. molinė masė. *Visuotinė lietuvių enciklopedija* [interaktyvus]. [žiūrėta 2025-04-29]. Prieiga per: https://www.vle.lt/straipsnis/moline-mase/.
13. FOWLER, David. Peter BRIMBLECOMBE. John BURROWS. ir kt. A chronology of global air quality. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* [interaktyvus]. 2020, Vol. 378, no. 2183, p. 20190314. [žiūrėta 2025-05-01]. Prieiga per: https://royalsocietypublishing.org/doi/10.1098/rsta.2019.0314.
14. History of Air Quality. *Evotech Air Quality* [interaktyvus]. 2022, [žiūrėta 2025-05-01]. Prieiga per: https://www.evotechairquality.co.uk/articles/history-of-air-quality.
15. Hippocrates: Epidemiology, Environmental Health, & Occupational Health. *Online Safety Trainer* [interaktyvus]. 2023, [žiūrėta 2025-05-01]. Prieiga per: https://www.onlinesafetytrainer.com/hippocrates-epidemiology-environmental-health-occupational-health/.
16. ADAMS, Francis. On Airs, Waters, and Places by Hippocrates. *The Internet Classics Archive* [interaktyvus]. [žiūrėta 2025-05-01]. Prieiga per: https://classics.mit.edu/Hippocrates/airwatpl.html.
17. Definition of MIASMA. *Merriam-Webster* [interaktyvus]. 2025, [žiūrėta 2025-05-01]. Prieiga per: https://www.merriam-webster.com/dictionary/miasma.

18. LAST, John M. *A dictionary of public health*. New York Oxford: Oxford University Press, 2007, ISBN 978-0-19-989131-3.

19. Miasma theory. *Wikipedia* [interaktyvus]. 2025, [žiūrėta 2025-05-01]. Prieiga per: https://en.wikipedia.org/w/index.php?title=Miasma_theory&oldid=1287009325.

20. Germ theory of disease. *Wikipedia* [interaktyvus]. 2025, [žiūrėta 2025-05-01]. Prieiga per: https://en.wikipedia.org/w/index.php?title=Germ_theory_of_disease&oldid=1287217465.

21. Ambient (outdoor) air pollution. *World Health Organization* [interaktyvus]. 2024, [žiūrėta 2025-05-25]. Prieiga per: https://www.who.int/news-room/fact-sheets/detail/ambient-(outdoor)-air-quality-and-health.

22. CROMAN, Kevin R. Bryan N DUNCAN. Alena BARTONOVA. ir kt. Air pollution monitoring for health research and patient care. An official American Thoracic Society workshop report. *Annals of the American Thoracic Society*. 2019, Vol. 16, no. 10, p. 1207–1214.

23. NAGELKERKE, Nico JD. A note on a general definition of the coefficient of determination. *biometrika*. 1991, Vol. 78, no. 3, p. 691–692.

24. OZER, Daniel J. Correlation and the coefficient of determination. *Psychological bulletin*. 1985, Vol. 97, no. 2, p. 307.

25. KARAGULIAN, Federico. Maurizio BARBIERE. Alexander KOTSEV. ir kt. Review of the Performance of Low-Cost Sensors for Air Quality Monitoring. *Atmosphere* [interaktyvus]. 2019, Vol. 10, no. 9, p. 506. [žiūrėta 2025-04-17]. Prieiga per: https://www.mdpi.com/2073-4433/10/9/506.

26. Airly Air Quality Monitors Comply with MCERTS Performance Standards for Indicative Ambient Particulate Monitoring. *airly* [interaktyvus]. [žiūrėta 2025-04-17]. Prieiga per: https://airly.org/en/airly-air-quality-monitors-comply-with-mcerts-performance-standards-for-indicative-ambient-particulate-monitoring/.

27. Certification of sensors system for air quality monitoring. *INERIS* [interaktyvus]. [žiūrėta 2025-04-17]. Prieiga per: https://prestations.ineris.fr/en/certification/certification-sensors-system-air-quality-monitoring.

28. EVALUATION PROTOCOL FOR SENSOR SYSTEMS FOR AMBIENT AIR QUALITY MONITORING AT FIXED SITE. *INERIS* [interaktyvus]. [žiūrėta 2025-04-17]. Prieiga per: https://prestations.ineris.fr/sites/prestation.ineris.fr/files/PrestaWeb/Pages-Solution/DSC/Certification%20syst%C3%A8mes%20capteurs%20surveillance%20qt%C3%A9%20air/en_gb_NEW%20MO1347AAapplicable.pdf.

29. Air Quality Monitoring Stations - Product Overview. *aqicn.org* [interaktyvus]. [žiūrėta 2025-04-17]. Prieiga per: https://aqicn.org/products/monitoring-stations/.

30. MAAG, Balz. Zimu ZHOU. ir Lothar THIELE. A survey on sensor calibration in air pollution monitoring deployments. *IEEE Internet of Things Journal*. 2018, Vol. 5, no. 6, p. 4857–4870.

31. CASTELL, Nuria. Franck R. DAUGE. Philipp SCHNEIDER. ir kt. Can commercial low-cost sensor platforms contribute to air quality monitoring and exposure estimates? *Environment International* [interaktyvus]. 2017, Vol. 99, p. 293–302. [žiūrėta 2025-04-18]. Prieiga per: https://linkinghub.elsevier.com/retrieve/pii/S0160412016309989.

32. LAI, Kwei-Herng. Daochen ZHA. Junjie XU. ir kt. Revisiting Time Series Outlier Detection: Definitions and Benchmarks. *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)* [interaktyvus]. 2021, [žiūrėta 2025-05-10]. Prieiga per: https://openreview.net/forum?id=r8IvOsnHchr.

33. PRATAMA, Irfan. Adhistya Erna PERMANASARI. Igi ARDIYANTO. ir kt. A review of missing values handling methods on time-series data. *2016 International Conference on Information Technology Systems and Innovation (ICITSI)* [interaktyvus]. Bandung - Bali, Indonesia: IEEE, 2016, p. 1–6. [žiūrėta 2025-05-11]. Prieiga per: http://ieeexplore.ieee.org/document/7858189/.

34. YOZGATLIGIL, Ceylan. Sipan ASLAN. Cem IYIGUN. ir kt. Comparison of missing value imputation methods in time series: the case of Turkish meteorological data. *Theoretical and Applied*

*Climatology* [interaktyvus]. 2013, Vol. 112, no. 1–2, p. 143–167. [žiūrėta 2025-05-11]. Prieiga per: http://link.springer.com/10.1007/s00704-012-0723-x.

35. BOBBITT, Zach. The Five Assumptions for Pearson Correlation. *Statology* [interaktyvus]. 2021, [žiūrėta 2025-05-18]. Prieiga per: https://www.statology.org/pearson-correlation-assumptions/.

36. STATISTICS SOLUTIONS. Correlation (Pearson, Kendall, Spearman). *IntellectusConsulting by Intellectus360* [interaktyvus]. [žiūrėta 2025-05-18]. Prieiga per: https://www.statisticssolutions.com/free-resources/directory-of-statistical-analyses/correlation-pearson-kendall-spearman/.

37. Pearson correlation coefficient. *Wikipedia* [interaktyvus]. 2025, [žiūrėta 2025-05-13]. Prieiga per: https://en.wikipedia.org/w/index.php?title=Pearson_correlation_coefficient&oldid=1286945322.

38. MDEWEY (HTTPS://STATS.STACKEXCHANGE.COM/USERS/101426/MDEWEY). [interaktyvus]. [s.l.]: Cross Validated, 2017 [žiūrėta 2025-05-13]. Prieiga per: https://stats.stackexchange.com/q/268608.

39. HAUKE, Jan. ir Tomasz KOSSOWSKI. Comparison of Values of Pearson's and Spearman's Correlation Coefficients on the Same Sets of Data. *QUAGEO* [interaktyvus]. 2011, Vol. 30, no. 2, p. 87–93. [žiūrėta 2025-05-18]. Prieiga per: https://www.sciendo.com/article/10.2478/v10117-011-0021-1.

40. ALI ABD AL-HAMEED, Khawla. Spearmans correlation coefficient in statistical analysis. *International Journal of Nonlinear Analysis and Applications* [interaktyvus]. 2022, Vol. 13, no. 1. [žiūrėta 2025-05-18]. Prieiga per: https://doi.org/10.22075/ijnaa.2022.6079.

41. J.J. STROSSMAYER UNIVERSITY OF OSIJEK, FACULTY OF AGRICULTURE IN OSIJEK, CROATIA Andrijana REBEKIĆ. Zdenko LONČARIĆ. ir kt. PEARSON'S OR SPEARMAN'S CORRELATION COEFFICIENT - WHICH ONE TO USE? *Poljoprivreda* [interaktyvus]. 2015, Vol. 21, no. 2, p. 47–54. [žiūrėta 2025-05-18]. Prieiga per: https://hrcak.srce.hr/index.php?show=clanak&id_clanak_jezik=221611.

42. ZINDA, Zac. Data Science Stats Review: Pearson's, Kendall's, and Spearman's Correlation for Feature Selection. *phData* [interaktyvus]. 2021, [žiūrėta 2025-05-22]. Prieiga per: https://www.phdata.io/blog/data-science-stats-review/.

43. TTNPHNS. [interaktyvus]. 2011, [žiūrėta 2025-05-22]. Prieiga per: https://stats.stackexchange.com/a/18136.

44. Kendall rank correlation coefficient. *Wikipedia* [interaktyvus]. 2025, [žiūrėta 2025-05-23]. Prieiga per: https://en.wikipedia.org/w/index.php?title=Kendall_rank_correlation_coefficient&oldid=1283653842.

45. KENDALL, M. G. The Treatment of Ties in Ranking Problems. *Biometrika* [interaktyvus]. 1945, Vol. 33, no. 3, p. 239. [žiūrėta 2025-05-23]. Prieiga per: https://www.jstor.org/stable/2332303?origin=crossref.

46. RANI DAS, Keya. A Brief Review of Tests for Normality. *American Journal of Theoretical and Applied Statistics* [interaktyvus]. 2016, Vol. 5, no. 1, p. 5. [žiūrėta 2025-05-16]. Prieiga per: http://www.sciencepublishinggroup.com/journal/paperinfo?journalid=146&doi=10.11648/j.ajtas.20160501.12.

47. Shapiro Wilk Test. *Statistics Kingdom* [interaktyvus]. [žiūrėta 2025-05-18]. Prieiga per: https://www.statskingdom.com/doc_shapiro_wilk.html.

48. SHAPIRO, S. S. ir M. B. WILK. An analysis of variance test for normality (complete samples). *Biometrika* [interaktyvus]. 1965, Vol. 52, no. 3–4, p. 591–611. [žiūrėta 2025-05-18]. Prieiga per: https://academic.oup.com/biomet/article-lookup/doi/10.1093/biomet/52.3-4.591.

49. ZAIONTZ, Charles. Shapiro-Wilk Tables. *Real Statistics Using Excel* [interaktyvus]. [žiūrėta 2025-05-18]. Prieiga per: https://real-statistics.com/statistics-tables/shapiro-wilk-table/.

50. ZAIONTZ, Charles. Shapiro-Wilk Test. *Real Statistics Using Excel* [interaktyvus]. [žiūrėta 2025-05-18]. Prieiga per: https://real-statistics.com/tests-normality-and-symmetry/statistical-tests-normality-symmetry/shapiro-wilk-test/.

51. HATEM, Georges. Joe ZEIDAN. Mathijs GOOSSENS. ir kt. NORMALITY TESTING METHODS AND THE IMPORTANCE OF SKEWNESS AND KURTOSIS IN STATISTICAL ANALYSIS. *BAU Journal - Science and Technology* [interaktyvus]. 2022, Vol. 3, no. 2. [žiūrėta 2025-05-19]. Prieiga per: https://digitalcommons.bau.edu.lb/stjournal/vol3/iss2/7.

52. Ekscesas (statistika). *Vikipedija* [interaktyvus]. 2023, [žiūrėta 2025-05-19]. Prieiga per: https://lt.wikipedia.org/w/index.php?title=Ekscesas_(statistika)&oldid=7007067.

53. MARKELEVIČIUS, Jonas. asimetrija. *Visuotinė lietuvių enciklopedija* [interaktyvus]. [žiūrėta 2025-05-19]. Prieiga per: https://www.vle.lt/straipsnis/asimetrija/.

54. Asimetrijos koeficientas. *Vikipedija* [interaktyvus]. 2023, [žiūrėta 2025-05-19]. Prieiga per: https://lt.wikipedia.org/w/index.php?title=Asimetrijos_koeficientas&oldid=7006773.

55. Dispersija. *Vikipedija* [interaktyvus]. [žiūrėta 2025-05-19]. Prieiga per: https://lt.wikipedia.org/wiki/Dispersija.

56. HAYES, Adam. Michael BOYLE. ir Suzanne KVILHAUG. What Is Variance in Statistics? Definition, Formula, and Example. *Investopedia* [interaktyvus]. 2024, [žiūrėta 2025-05-19]. Prieiga per: https://www.investopedia.com/terms/v/variance.asp.

57. Feedforward Neural Network. *GeeksforGeeks* [interaktyvus]. 18:44:54+00:00, [žiūrėta 2025-05-13]. Prieiga per: https://www.geeksforgeeks.org/feedforward-neural-network/.

58. VERIKAS, Antanas. neuroninis tinklas. *Visuotinė lietuvių enciklopedija* [interaktyvus]. [žiūrėta 2025-05-13]. Prieiga per: https://www.vle.lt/straipsnis/neuroninis-tinklas/.

59. SHUKLA, Satya Narayan. ir Benjamin M. MARLIN. [interaktyvus]. [s.l.]: arXiv, 2019 [žiūrėta 2025-05-13]. arXiv:1909.07782 [cs]. Prieiga per: http://arxiv.org/abs/1909.07782.

60. ŠTIKONAS, Artūras. interpoliavimas. [interaktyvus]. [žiūrėta 2025-05-13]. Prieiga per: https://www.vle.lt/straipsnis/interpoliavimas/.

61. LEPOT, Mathieu. Jean-Baptiste AUBIN. ir François CLEMENS. Interpolation in Time Series: An Introductive Overview of Existing Methods, Their Performance Criteria and Uncertainty Assessment. *Water* [interaktyvus]. 2017, Vol. 9, no. 10, p. 796. [žiūrėta 2025-05-13]. Prieiga per: https://www.mdpi.com/2073-4441/9/10/796.

62. BUTEIKIS, Andrius. [interaktyvus]. [s.l.]: Vilniaus Universitetas, 2019 [žiūrėta 2025-05-21]. Prieiga per: https://web.vu.lt/mif/a.buteikis/wp-content/uploads/2019/02/Lecture_03.pdf.

63. DATA SCIENCE WIZARDS. Preprocessing and Data Exploration for Time Series — Handling Missing Values. *Medium* [interaktyvus]. 2023, [žiūrėta 2025-05-22]. Prieiga per: https://medium.com/@datasciencewizards/preprocessing-and-data-exploration-for-time-series-handling-missing-values-e5c507f6c71c.

64. DATA SCIENCE WIZARDS. A Quick Guide to Deal with Missing Data. *Medium* [interaktyvus]. 2022, [žiūrėta 2025-05-22]. Prieiga per: https://medium.com/@datasciencewizards/a-quick-guide-to-deal-with-missing-data-29f00d0ab57b.

65. GĖGŽNA, Vilmantas. 21. Tiesinė regresija. *Biostatistinės analizės pagrindai* [interaktyvus]. [žiūrėta 2025-05-23]. Prieiga per: https://mokymai.github.io/biostatistika/tiesine-regresija.html.

66. HANCK, Christoph. Martin ARNOLD. Alexander GERBER. ir kt. 4.2 Estimating the Coefficients of the Linear Regression Model. *Introduction to Econometrics with R* [interaktyvus]. Essen, Germany, 2024, [žiūrėta 2025-05-23]. Prieiga per: https://www.econometrics-with-r.org/4.2-estimating-the-coefficients-of-the-linear-regression-model.html#the-ordinary-least-squares-estimator.

67. LAPINSKAS, Remigijus. 5. Regresinės analizės elementai. *Pats trumpiausias taikomosios statistikos kursas* [interaktyvus]. [s.l.]: Vilniaus Universitetas, 2010, p. 47–48. [žiūrėta 2025-05-23]. Prieiga per: https://web.vu.lt/mif/a.reklaite/files/2013/02/2010.01-Pats-trumpiausias-su-R.pdf.

68. HANCK, Christoph. Martin ARNOLD. Alexander GERBER. ir kt. 4.3 Measures of Fit. *Introduction to Econometrics with R* [interaktyvus]. Essen, Germany, 2024, [žiūrėta 2025-05-23]. Prieiga per: https://www.econometrics-with-r.org/4.3-measures-of-fit.html.

69. Tailed Test. *ScienceDirect* [interaktyvus]. [žiūrėta 2025-05-23]. Prieiga per: https://www.sciencedirect.com/topics/mathematics/tailed-test.

70. z.test: Z-test. *RDocumentation* [interaktyvus]. [žiūrėta 2025-05-23]. Prieiga per: https://www.rdocumentation.org/packages/BSDA/versions/1.2.2/topics/z.test.

71. Z-test : Formula, Types, Examples. *GeeksforGeeks* [interaktyvus]. 00:54:08+00:00, [žiūrėta 2025-05-23]. Prieiga per: https://www.geeksforgeeks.org/z-test/.

72. HANCK, Christoph. Martin ARNOLD. Alexander GERBER. ir kt. 5.1 Testing Two-Sided Hypotheses concerning the Slope Coefficient. *Introduction to Econometrics with R* [interaktyvus]. Essen, Germany, 2024, [žiūrėta 2025-05-23]. Prieiga per: https://www.econometrics-with-r.org/5.1-testing-two-sided-hypotheses-concerning-the-slope-coefficient.html.

73. TAYLOR, Geoffrey Ingram. Eddy Motion in the Atmosphere. *Philosophical Transactions of the Royal Society of London* [interaktyvus]. 1915, Vol. 215, no. 523–537, p. 1–26. [žiūrėta 2025-04-15]. Prieiga per: https://royalsocietypublishing.org/doi/10.1098/rsta.1915.0001.

74. Smoothing Parameter - an overview. *ScienceDirect* [interaktyvus]. [žiūrėta 2025-05-14]. Prieiga per: https://www.sciencedirect.com/topics/computer-science/smoothing-parameter.

75. Euklidinis atstumas. *Vikipedija* [interaktyvus]. 2024, [žiūrėta 2025-05-14]. Prieiga per: https://lt.wikipedia.org/w/index.php?title=Euklidinis_atstumas&oldid=7181302.

76. norma. *Visuotinė lietuvių enciklopedija* [interaktyvus]. [žiūrėta 2025-05-14]. Prieiga per: https://www.vle.lt/straipsnis/norma-3/.

77. Vektoriaus norma. *Vikipedija* [interaktyvus]. 2024, [žiūrėta 2025-05-14]. Prieiga per: https://lt.wikipedia.org/w/index.php?title=Vektoriaus_norma&oldid=7173745.

78. ČIOČYS, Vaclovas. Euklido erdvė. *Visuotinė lietuvių enciklopedija* [interaktyvus]. [žiūrėta 2025-05-14]. Prieiga per: https://www.vle.lt/straipsnis/euklido-erdve/.

79. ZINEVIČIUS, Albertas. [interaktyvus]. [s.l.]: Vilniaus Universitetas, 2014 [žiūrėta 2025-05-14]. Prieiga per: https://web.vu.lt/mif/a.zinevicius/FAm2014ruduo-08.pdf.

80. How far does air pollution travel? *NAFAS Indonesia* [interaktyvus]. 2020, [žiūrėta 2025-05-24]. Prieiga per: https://nafas.co.id/article/How-far-does-air-pollution-travel.

81. WYLIE, Heather. Atmospheric Dispersion and Pollution Transport. *Air Quality Portal* [interaktyvus]. 2021, [žiūrėta 2025-05-24]. Prieiga per: https://airquality.climate.ncsu.edu/2021/06/06/atmospheric-dispersion-and-pollution-transport/.

82. WATSON, Ann Y. Richard R. BATES. ir Donald KENNEDY. Atmospheric Transport and Dispersion of Air Pollutants Associated with Vehicular Emissions. *Air Pollution, the Automobile, and Public Health* [interaktyvus]. [s.l.]: National Academies Press (US), 1988, [žiūrėta 2025-05-24]. Prieiga per: https://www.ncbi.nlm.nih.gov/books/NBK218142/.

83. ELIASSEN, Anton. A Review of Long-Range Transport Modeling. *Journal of Applied Meteorology* [interaktyvus]. 1980, Vol. 19, no. 3. [žiūrėta 2025-05-24]. Prieiga per: https://journals.ametsoc.org/view/journals/apme/19/3/1520-0450_1980_019_0231_arolrt_2_0_co_2.xml.

84. ALASAUSKAS, Šarūnas. Rūta USTINAVIČIENĖ. ir Mindaugas KAVALIAUSKAS. Vilniaus miesto mokinių rizikos sirgti alerginiu rinitu sąsajos su oro tarša nustatymas panaudojant GIS. *Visuomenės sveikata* [interaktyvus]. 2020, Vol. 2, no. 89, p. 70–76. [žiūrėta 2025-05-15]. Prieiga per: https://www.hi.lt/uploads/Institutas/zurnalo_vs%20info/2020%202/VS%202020%202(89)%20ORIG%20Alerginis%20rinitas.pdf.

85. BENJAMIN, Daniel J. James O. BERGER. Magnus JOHANNESSON. ir kt. Redefine statistical significance. *Nature Human Behaviour* [interaktyvus]. 2018, Vol. 2, no. 1, p. 6–10. Prieiga per: https://doi.org/10.1038/s41562-017-0189-z.

86. BRAZIENĖ, Agnė. Jonė VENCLOVIENĖ. Rūta BABARSKIENĖ. ir kt. INFLUENCE OF SHORT-TERM AIR POLLUTION WITH CARBON MONOXIDE ON THE RISK OF AMBULANCE CALL-OUTS RELATED TO ARTERIAL HYPERTENSION / TRUMPALAIKIS ORO TARŠOS ANGLIES MONOKSIDU POVEIKIS GREITOSIOS MEDICINOS PAGALBOS IŠKVIETIMŲ DĖL ARTERINĖS HIPERTENZIJOS RIZIKA. *Mokslas – Lietuvos ateitis* [interaktyvus]. 2015, Vol. 7, no. 4, p. 385–391. [žiūrėta 2025-05-15]. Prieiga per: https://journals.vilniustech.lt/index.php/MLA/article/view/2724.

87. ŠUKYS, Aurimas. [interaktyvus]. Akademija, 2012 [žiūrėta 2025-05-15]. Prieiga per: https://portalcris.vdu.lt/server/api/core/bitstreams/40621ba9-db89-409b-8996-08f38bc4f3a9/content.

88. BRUNEKREEF, Bert. ir Stephen T HOLGATE. Air pollution and health. *The Lancet* [interaktyvus]. 2002, Vol. 360, p. 1233–42. [žiūrėta 2025-05-15]. Prieiga per: https://vula.uct.ac.za/access/content/group/c7716642-ada7-42a0-acee-0434c085df90/Electronic%20Resources/CD3/env/sourcedocs/J%20Articles/Brunekreef%20and%20Holgate%20AP%20and%20health%20Review.pdf.

89. What are the WHO Air quality guidelines? *World Health Organization* [interaktyvus]. 2021, [žiūrėta 2025-05-01]. Prieiga per: https://www.who.int/news-room/feature-stories/detail/what-are-the-who-air-quality-guidelines.

90. LIU, Cong. Renjie CHEN. Francesco SERA. ir kt. Ambient Particulate Air Pollution and Daily Mortality in 652 Cities. *New England Journal of Medicine* [interaktyvus]. 2019, Vol. 381, no. 8, p. 705–715. [žiūrėta 2025-05-02]. Prieiga per: http://www.nejm.org/doi/10.1056/NEJMoa1817364.

91. EUROPOS PARLAMENTAS. ir EUROPOS SĄJUNGOS TARYBA. [interaktyvus]. 2008 [žiūrėta 2025-05-01]. COD 2005/0183. Prieiga per: http://data.europa.eu/eli/dir/2008/50/oj.

92. [Interaktyvus]. 2015 [žiūrėta 2025-05-01]. LASTMODIN 32015L1480R(01). Prieiga per: http://data.europa.eu/eli/dir/2008/50/2015-09-18.

93. APLINKOS APSAUGOS AGENTŪRA. Stotelių tinklas Lietuvoje. *lrv.lt* [interaktyvus]. 2025, [žiūrėta 2025-05-01]. Prieiga per: https://aaa.lrv.lt/lt/veiklos-sritys/oras/oro-monitoringo-vietos/stoteliu-tinklas-lietuvoje/.

94. APLINKOS APSAUGOS AGENTŪRA. Mobili laboratorija. *lrv.lt* [interaktyvus]. 2025, [žiūrėta 2025-05-07]. Prieiga per: https://aaa.lrv.lt/lt/veiklos-sritys/oras/oro-monitoringo-vietos/mobili-laboratorija/.

95. APLINKOS APSAUGOS AGENTŪRA. Naujoji Akmenė. *lrv.lt* [interaktyvus]. 2025, [žiūrėta 2025-05-07]. Prieiga per: https://aaa.lrv.lt/lt/veiklos-sritys/oras/oro-monitoringo-vietos/naujoji-akmene/.

96. APLINKOS APSAUGOS AGENTŪRA. Jonava. *lrv.lt* [interaktyvus]. 2025, [žiūrėta 2025-05-07]. Prieiga per: https://aaa.lrv.lt/lt/veiklos-sritys/oras/oro-monitoringo-vietos/jonava/.

97. APLINKOS APSAUGOS AGENTŪRA. Kaunas. *lrv.lt* [interaktyvus]. 2025, [žiūrėta 2025-05-07]. Prieiga per: https://aaa.lrv.lt/lt/veiklos-sritys/oras/oro-monitoringo-vietos/kaunas/.

98. APLINKOS APSAUGOS AGENTŪRA. Kėdainiai. *lrv.lt* [interaktyvus]. 2025, [žiūrėta 2025-05-07]. Prieiga per: https://aaa.lrv.lt/lt/veiklos-sritys/oras/oro-monitoringo-vietos/kedainiai/.

99. APLINKOS APSAUGOS AGENTŪRA. Klaipėda. *lrv.lt* [interaktyvus]. 2025, [žiūrėta 2025-05-07]. Prieiga per: https://aaa.lrv.lt/lt/veiklos-sritys/oras/oro-monitoringo-vietos/klaipeda/.

100. APLINKOS APSAUGOS AGENTŪRA. Mažeikiai. *lrv.lt* [interaktyvus]. 2025, [žiūrėta 2025-05-07]. Prieiga per: https://aaa.lrv.lt/lt/veiklos-sritys/oras/oro-monitoringo-vietos/mazeikiai/.

101. APLINKOS APSAUGOS AGENTŪRA. Šiauliai. *lrv.lt* [interaktyvus]. 2025, [žiūrėta 2025-05-07]. Prieiga per: https://aaa.lrv.lt/lt/veiklos-sritys/oras/oro-monitoringo-vietos/siauliai/.

102. APLINKOS APSAUGOS AGENTŪRA. Vilnius. *lrv.lt* [interaktyvus]. 2025, [žiūrėta 2025-05-07]. Prieiga per: https://aaa.lrv.lt/lt/veiklos-sritys/oras/oro-monitoringo-vietos/vilnius/.

103. APLINKOS APSAUGOS AGENTŪRA. Žemaitija. *lrv.lt* [interaktyvus]. 2025, [žiūrėta 2025-05-07]. Prieiga per: https://aaa.lrv.lt/lt/veiklos-sritys/oras/oro-monitoringo-vietos/zemaitija/.

104. APLINKOS APSAUGOS AGENTŪRA. Panevėžys. *lrv.lt* [interaktyvus]. 2025, [žiūrėta 2025-05-07]. Prieiga per: https://aaa.lrv.lt/lt/veiklos-sritys/oras/oro-monitoringo-vietos/panevezys/.

105. APLINKOS APSAUGOS AGENTŪRA. Dzūkija. *lrv.lt* [interaktyvus]. 2025, [žiūrėta 2025-05-07]. Prieiga per: https://aaa.lrv.lt/lt/veiklos-sritys/oras/oro-monitoringo-vietos/dzukija/.

106. APLINKOS APSAUGOS AGENTŪRA. Aukštaitija. *lrv.lt* [interaktyvus]. 2025, [žiūrėta 2025-05-07]. Prieiga per: https://aaa.lrv.lt/lt/veiklos-sritys/oras/oro-monitoringo-vietos/aukstaitija/.

107. APLINKOS APSAUGOS AGENTŪRA. Lietuvos valstybinio oro monitoringo automatinių stočių matavimo duomenys. *Lietuvos atvirų duomenų portalas* [interaktyvus]. 2024, [žiūrėta 2025-02-15]. Prieiga per: https://data.gov.lt/datasets/500/.

108. APLINKOS APSAUGOS AGENTŪRA. Integruoto monitoringo duomenys. *Lietuvos atvirų duomenų portalas* [interaktyvus]. 2023, [žiūrėta 2025-05-07]. Prieiga per: https://data.gov.lt/datasets/1792/.

109. APLINKOS APSAUGOS AGENTŪRA. Lietuvos valstybinio oro monitoringo matavimo duomenys (pusiau automatinių tyrimų). *Lietuvos atvirų duomenų portalas* [interaktyvus]. 2023, [žiūrėta 2025-05-07]. Prieiga per: https://data.gov.lt/datasets/1788/.

110. APLINKOS APSAUGOS AGENTŪRA. Į aplinkos orą išmetamų teršalų apskaitos duomenys. *Lietuvos atvirų duomenų portalas* [interaktyvus]. 2025, [žiūrėta 2025-05-07]. Prieiga per: https://data.gov.lt/datasets/884/.

111. EUROPEAN ENVIRONMENT AGENCY. Sud. *EMEP/EEA air pollutant emission inventory guidebook 2023: technical guidance to prepare national emission inventories*. Luxembourg: Publications Office of the European Union, 2023, 30 p. ISBN 978-92-9480-598-0.

112. VALSTYBĖS DUOMENŲ AGENTŪRA. Įvairių diagnozių TLK-10-AM kodų dažniai. *Lietuvos atvirų duomenų portalas* [interaktyvus]. 2025, [žiūrėta 2025-05-14]. Prieiga per: https://data.gov.lt/datasets/2843/.

113. HIGIENOS INSTITUTAS. SVEIKATOS STATISTIKA Traumų ir nelaimingų atsitikimų stebėsenos sistemos duomenys. *stat.hi.lt* [interaktyvus]. [žiūrėta 2025-05-14]. Prieiga per: https://stat.hi.lt/.

114. VŠĮ VILNIAUS UNIVERSITETO LIGONINĖ SANTAROS KLINIKOS. Širdies ir kraujagyslių ligomis sergančių pacientų apsilankymų statistiniai duomenys. *Lietuvos atvirų duomenų portalas* [interaktyvus]. 2023, [žiūrėta 2025-05-14]. Prieiga per: https://data.gov.lt/datasets/1893/#info.

115. HIGIENOS INSTITUTAS. Nuolatinių Lietuvos gyventojų, mirusių Lietuvoje ar užsienyje, ir nuolatinių Lietuvos Respublikos gyventojų negyvagimių mirties atvejų ir jų priežasčių duomenys. *Lietuvos atvirų duomenų portalas* [interaktyvus]. 2025, [žiūrėta 2025-05-14]. Prieiga per: https://data.gov.lt/datasets/998/.

116. *The International Statistical Classification of Diseases and Related Health Problems, tenth revision, Australian modification (ICD-10-AM/ACHI/ACS)*. Ninth edition. Ed. Darlinghurst, NSW: Independent Hospital Pricing Authority, 2015, ISBN 978-1-76007-020-5.

117. AUSTRALIJOS KLASIFIKACIJOS KŪRIMO KONSORCIUMAS (ACCD). *Tarptautinė statistinė ligų ir sveikatos sutrikimų klasifikacija, dešimtasis pataisytas ir papildytas leidimas, Australijos modifikacija (TLK-10-AM) – Sisteminis ligų sąrašas.* [interaktyvus]. 9. Ed. Australija: Nepriklausoma ligoninėms kainas nustatanti institucija, 2015, .

118. Free Weather API. *Open-Meteo.com* [interaktyvus]. [žiūrėta 2025-05-07]. Prieiga per: https://open-meteo.com.

119. HALL, B. D. P. SAUNDERS. ir D. R. WHITE. Digital representation of scales and units for temperature and related quantities. *TEMPERATURE: ITS MEASUREMENT AND CONTROL IN SCIENCE AND INDUSTRY, VOLUME 9: Proceedings of the Tenth International Temperature Symposium* [interaktyvus]. Anaheim, USA, 2024, [žiūrėta 2025-04-28]. Prieiga per: https://pubs.aip.org/aip/acp/article-lookup/doi/10.1063/5.0235453.

120. HELMENSTINE, Anne Marie. Concentration Definition (Chemistry). *ThoughtCo* [interaktyvus]. 2024, [žiūrėta 2025-04-29]. Prieiga per: https://www.thoughtco.com/definition-of-concentration-605844.

121. DEPARTMENT OF ENVIRONMENTAL SCIENCE. [interaktyvus]. [s.l.]: Aarhus University [žiūrėta 2025-04-29]. Prieiga per: https://www2.dmu.dk/atmosphericenvironment/expost/database/docs/ppm_conversion.pdf.

122. Molecular Weight Calculator (Molar Mass). *Lenntech* [interaktyvus]. [žiūrėta 2025-04-29]. Prieiga per: https://www.lenntech.com/calculators/molecular/molecular-weight-calculator.htm.

123. Unit Conversion. *Air Pollution Information System* [interaktyvus]. [žiūrėta 2025-04-29]. Prieiga per: https://www.apis.ac.uk/unit-conversion?ppb_ug=ppb&value=100&pollutant=SO2&m_weight=&select_temp=0&input_temp=&Submit=Calculate+#jump.

124. Conversion Calculator. *Markes International* [interaktyvus]. [žiūrėta 2025-04-29]. Prieiga per: https://markes.com/calculator.

125. NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. CODATA Value: molar volume of ideal gas (273.15 K, 100 kPa). *The NIST Reference on Constants, Units, and Uncertainty* [interaktyvus]. [žiūrėta 2025-04-30]. Prieiga per: https://physics.nist.gov/cgi-bin/cuu/Value?mvol.

126. Standard temperature and pressure. *Wikipedia* [interaktyvus]. 2024, [žiūrėta 2025-04-30]. Prieiga per: https://en.wikipedia.org/w/index.php?title=Standard_temperature_and_pressure&oldid=1264580099#Current_use.

127. MILLS, Ian. Tomislav CVITAŠ. Klaus HOMANN. ir kt. *Quantities, units and symbols in physical chemistry*. 2. ed., reprinted. Ed. Oxford: Blackwell Science, 1998, 166 p. ISBN 978-0-632-03583-0.

128. GRANGE, Stuart K. [interaktyvus]. 2014 [žiūrėta 2025-05-06]. Prieiga per: http://rgdoi.net/10.13140/RG.2.1.3349.2006.

129. LIETUVOS GEOLOGIJOS TARNYBA PRIE APLINKOS MINISTERIJOS. Potencialių taršos židinių duomenys. *Lietuvos atvirų duomenų portalas* [interaktyvus]. 2025, [žiūrėta 2025-05-23]. Prieiga per: https://data.gov.lt/datasets/1330/#info.

130. VAŠKAS, Petras. vektorinis laukas. *Visuotinė lietuvių enciklopedija* [interaktyvus]. [žiūrėta 2025-05-25]. Prieiga per: https://www.vle.lt/straipsnis/vektorinis-laukas/.

131. MARGALIT, Dan. ir Joseph RABINOFF. Dot Products and Orthogonality. *Interactive Linear Algebra* [interaktyvus]. [žiūrėta 2025-05-25]. Prieiga per: https://textbooks.math.gatech.edu/ila/dot-product.html.

132. KAKAREKA, Sergey. ir Tamara KUKHARCHYK. [interaktyvus]. Belarus: European Environment Agency, 2005 [žiūrėta 2025-05-13]. Prieiga per: https://www.eea.europa.eu/publications/EMEPCORINAIR5/Sources_of_HCB_emissions.pdf.

133. Benzene. *National Cancer Institute* [interaktyvus]. 2015, [žiūrėta 2025-05-13]. Prieiga per: https://www.cancer.gov/about-cancer/causes-prevention/risk/substances/benzene.

134. CDC. Benzene. *Chemical Emergencies* [interaktyvus]. 2024, [žiūrėta 2025-05-13]. Prieiga per: https://www.cdc.gov/chemical-emergencies/chemical-fact-sheets/benzene.html.

135. Hexachlorobenzene. *OEHHA* [interaktyvus]. [žiūrėta 2025-05-13]. Prieiga per: https://oehha.ca.gov/proposition-65/chemicals/hexachlorobenzene.

136. Hexachlorobenzene. *PubChem* [interaktyvus]. [žiūrėta 2025-05-13]. Prieiga per: https://pubchem.ncbi.nlm.nih.gov/compound/8370.

137. Hexachlorobenzene. *ScienceDirect* [interaktyvus]. [žiūrėta 2025-05-13]. Prieiga per: https://www.sciencedirect.com/topics/earth-and-planetary-sciences/hexachlorobenzene.

138. STATISTICS SOLUTIONS. What to do When Your Sample Size is Not Big Enough. *IntellectusConsulting by Intellectus360* [interaktyvus]. [žiūrėta 2025-05-13]. Prieiga per: https://www.statisticssolutions.com/what-to-do-when-your-sample-size-is-not-big-enough/.

139. POULIN, Bethany. When a sample size is small (n<30) in regression model diagnostics, should I remove outliers in my data? *Quora* [interaktyvus]. 2017, [žiūrėta 2025-05-13]. Prieiga per: https://www.quora.com/When-a-sample-size-is-small-n-30-in-regression-model-diagnostics-should-I-remove-outliers-in-my-data.

140. MORITZ, Steffen. Alexis SARDÁ. Thomas BARTZ-BEIELSTEIN. ir kt. [interaktyvus]. [s.l.]: arXiv, 2015 [žiūrėta 2025-05-14]. arXiv:1510.03924 [stat]. Prieiga per: http://arxiv.org/abs/1510.03924.

141. MORITZ, Steffen. ir Thomas BARTZ-BEIELSTEIN. imputeTS: Time Series Missing Value Imputation in R. BIVAND, R.Sud. *The R Journal* [interaktyvus]. 2017, [žiūrėta 2025-05-14]. Prieiga per: https://digitalcommons.unl.edu/r-journal/453/.

142. Pandas DataFrame interpolate() Method | Pandas Method. *GeeksforGeeks* [interaktyvus]. 2024, [žiūrėta 2025-05-14]. Prieiga per: https://www.geeksforgeeks.org/pandas-dataframe-interpolate/.

143. pandas.DataFrame.interpolate. *pandas* [interaktyvus]. [žiūrėta 2025-05-14]. Prieiga per: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.interpolate.html.

144. MEHRANG, Saeed. Elina HELANDER. Misha PAVEL. ir kt. Outlier detection in weight time series of connected scales. *2015 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)* [interaktyvus]. Washington, DC, USA: IEEE, 2015, p. 1489–1496. [žiūrėta 2025-05-11]. Prieiga per: http://ieeexplore.ieee.org/document/7359896/.

145. R/hampel.R. *rdrr.io* [interaktyvus]. 2024, [žiūrėta 2025-05-11]. Prieiga per: https://rdrr.io/rforge/pracma/src/R/hampel.R.

146. LEE, Dong Kyu. Junyong IN. ir Sangseok LEE. Standard deviation and standard error of the mean. *Korean Journal of Anesthesiology* [interaktyvus]. 2015, Vol. 68, no. 3, p. 220. [žiūrėta 2025-05-11]. Prieiga per: http://ekja.org/journal/view.php?doi=10.4097/kjae.2015.68.3.220.

147. LIVINGSTON, Edward H. The mean and standard deviation: what does it all mean? *Journal of Surgical Research* [interaktyvus]. 2004, Vol. 119, no. 2, p. 117–123. [žiūrėta 2025-05-11]. Prieiga per: https://linkinghub.elsevier.com/retrieve/pii/S0022480404000551.

148. Confidence interval. *Wikipedia* [interaktyvus]. 2025, [žiūrėta 2025-05-14]. Prieiga per: https://en.wikipedia.org/w/index.php?title=Confidence_interval&oldid=1288961296.

149. 68–95–99.7 rule. *Wikipedia* [interaktyvus]. 2025, [žiūrėta 2025-05-14]. Prieiga per: https://en.wikipedia.org/w/index.php?title=68%E2%80%9395%E2%80%9399.7_rule&oldid=1278465668.

150. PAPADIMITRIOU, Spiros. JIMENG SUN ir Christos FALOUTSOS. Streaming Pattern Discovery in Multiple Time-Series. [interaktyvus]. 2008, p. 4–11. [žiūrėta 2025-05-14]. Prieiga per: https://kilthub.cmu.edu/articles/Streaming_Pattern_Discovery_in_Multiple_Time-Series/6609941/1.

151. DECHOW, Patricia M. ir Richard G. SLOAN. Returns to contrarian investment strategies: Tests of naive expectations hypotheses. *Journal of Financial Economics* [interaktyvus]. 1997, Vol. 43, no. 1, p. 3–27. [žiūrėta 2025-05-18]. Prieiga per: https://linkinghub.elsevier.com/retrieve/pii/S0304405X96008872.

152. BALČIŪNAS, Aidas. [interaktyvus]. [s.l.]: Vilniaus Universitetas, 2019 [žiūrėta 2025-05-19]. Prieiga per: https://web.vu.lt/mif/a.balciunas/files/2019/12/Taikomoji-statistika2-1.pdf.

153. THE EDITORS OF ENCYCLOPÆDIA BRITANNICA. Absolute zero. *Britannica* [interaktyvus]. 2025, [žiūrėta 2025-05-19]. Prieiga per: https://www.britannica.com/science/absolute-zero.

154. MEKŠRIŪNAITĖ, Sandra. Stačiakampė diagrama. *Duomenų vaizdavimas visuomenės sveikatos stebėsenos praktikoje. Rekomendacijos* [interaktyvus]. Vilnius: Higienos institutas, 2015, p. 16. [žiūrėta 2025-05-21]. Prieiga per: https://www.hi.lt/uploads/Products/product_200/rekom_duom_vaizd.pdf.

155. COVID-19 pandemija. *Visuotinė lietuvių enciklopedija* [interaktyvus]. [žiūrėta 2025-05-22]. Prieiga per: https://www.vle.lt/straipsnis/covid-19-pandemija/.

156. pandas.Series.interpolate. *pandas* [interaktyvus]. [žiūrėta 2025-05-22]. Prieiga per: https://pandas.pydata.org/docs/reference/api/pandas.Series.interpolate.html.

157. WILLK. Pandas interpolate „time“ vs „linear“ for equally-spaced times. *Stack Overflow* [interaktyvus]. 2025, [žiūrėta 2025-05-22]. Prieiga per: https://stackoverflow.com/q/54985896.

158. HYNDMANN, Rob J. ir George ATHANASOPOULOS. 6.1 Time series components. *Forecasting: Principles and Practice* [interaktyvus]. 2. Ed.Melbourne, Australia: OTexts, 2018, [žiūrėta 2025-05-22]. Prieiga per: https://otexts.com/fpp2/components.html.

159. RSTUDIO. Time series decomposition. *RPubs* [interaktyvus]. [žiūrėta 2025-05-22]. Prieiga per: https://rpubs.com/oddskid/ts_decomposition.

160. Sacharos dulkės Lietuvoje – geologinis ryšys tarp žemynų. *LIETUVOS GEOLOGIJOS TARNYBA PRIE APLINKOS MINISTERIJOS* [interaktyvus]. 2025, [žiūrėta 2025-05-25]. Prieiga per: https://lgt.lrv.lt/lt/naujienos/sacharos-dulkes-lietuvoje-geologinis-rysys-tarp-zemynu/.

161. Sinoptikai įspėja: į Lietuvą atkeliauja Sacharos dulkės. *Kas vyksta Kaune* [interaktyvus]. 2021, [žiūrėta 2025-05-25]. Prieiga per: https://kaunas.kasvyksta.lt/2021/02/22/gamta/sinoptikai-ispeja-i-lietuva-atkeliauja-sacharos-dulkes/.

162. ELTA. Lietuvoje gali pablogėti oro kokybė – vėjas atneš Sacharos dulkes. *lrt.lt* [interaktyvus]. 2025, [žiūrėta 2025-05-25]. Prieiga per: https://www.lrt.lt/naujienos/lietuvoje/2/2538966/lietuvoje-gali-pablogeti-oro-kokybe-vejas-atnes-sacharos-dulkes.

163. VILDŽIŪNAITĖ, Rūta. Į Lietuvą plūsta Sacharos dulkės: kvėpuojant gali patekti net į kraują ir baigtis labai blogai. *tv3.lt* [interaktyvus]. [žiūrėta 2025-05-25]. Prieiga per: https://www.tv3.lt/naujiena/lietuva/i-lietuva-plusta-sacharos-dulkes-kvepuojant-gali-patekti-net-i-krauja-ir-baigtis-labai-blogai-n1413529.

164. AMRHEIN, Valentin. Sander GREENLAND. ir Blake MCSHANE. Scientists rise up against statistical significance. *Nature* [interaktyvus]. 2019, Vol. 567, no. 7748, p. 305–307. [žiūrėta 2025-05-22]. Prieiga per: https://media.nature.com/original/magazine-assets/d41586-019-00857-9/d41586-019-00857-9.pdf.

165. WASSERSTEIN, Ronald L. Allen L. SCHIRM. ir Nicole A. LAZAR. Moving to a World Beyond „p<0.05“. *The American Statistician* [interaktyvus]. 2019, Vol. 73, no. sup1, p. 1–19. [žiūrėta 2025-05-22]. Prieiga per: https://www.tandfonline.com/doi/full/10.1080/00031305.2019.1583913.

166. IBE, Oliver C. *Fundamentals of Applied Probability and Random Processes* [interaktyvus]. 2. Ed. [s.l.]: Elsevier, 2014, ISBN 978-0-12-800852-2.

167. FORD, Clay. Understanding QQ Plots | UVA Library. *University of Virginia Library* [interaktyvus]. 2015, [žiūrėta 2025-05-23]. Prieiga per: https://library.virginia.edu/data/articles/understanding-q-q-plots.

168. Rodiklių duomenų bazė - Į atmosferą išmestų teršalų kiekis. *Oficialiosios statistikos portalas* [interaktyvus]. [žiūrėta 2025-05-23]. Prieiga per: https://osp.stat.gov.lt/statistiniu-rodikliu-analize?hash=94299c35-2430-4f14-9ab8-176c804cc665#/.

169. [Interaktyvus]. [s.l.]: McMaster University [žiūrėta 2025-05-25]. Prieiga per: https://www.ece.mcmaster.ca/~xwu/part4.pdf.

170. KUMAR, Utpal. Numerical optimization based on the L-BFGS method. *TDS Archive* [interaktyvus]. 2022, [žiūrėta 2025-05-25]. Prieiga per: https://medium.com/data-science/numerical-optimization-based-on-the-l-bfgs-method-f6582135b0ca.

171. Unconstrained Nonlinear Optimization Algorithms. *MATLAB Help Center* [interaktyvus]. [žiūrėta 2025-05-25]. Prieiga per: https://www.mathworks.com/help/optim/ug/unconstrained-nonlinear-optimization-algorithms.html.

# Priedai

## 1 priedas. Python kodas, skirtas „Open-Meteo“ duomenų atsisiuntimui.

```python
import os1.

import time
import calendar
import pandas as pd
import requests
from requests.exceptions import HTTPError
from prepare_dataframes import prepare_dataframe

OPEN_METEO_URL = "https://archive-api.open-meteo.com/v1/archive"


def get_bulk_weather(
    latitude: float,
    longitude: float,
    start_date: str,
    end_date: str,
    max_retries: int = 5,
    backoff_factor: float = 1.0,
) -> pd.DataFrame:
    params = {
        "latitude": latitude,
        "longitude": longitude,
        "start_date": start_date,
        "end_date": end_date,
        "hourly": ",".join(
            [
                "temperature_2m",
                "surface_pressure",
                "relativehumidity_2m",
                "windspeed_10m",
                "winddirection_10m",
            ]
        ),
        "timezone": "UTC",
    }

    for attempt in range(1, max_retries + 1):
        resp = requests.get(OPEN_METEO_URL, params=params)
        try:
            resp.raise_for_status()
        except HTTPError:
            if resp.status_code == 429 and attempt < max_retries:
                wait = backoff_factor * (2 ** (attempt - 1))
                print(f"[429] rate limit – sleeping {wait:.1f}s (attempt {attempt})")
                time.sleep(wait)
                continue
            else:
                raise

        data = resp.json()["hourly"]
        times = pd.to_datetime(data["time"], utc=True)
        df_hr = pd.DataFrame(
            {
                "temperature_2m": data["temperature_2m"],
                "surface_pressure": data["surface_pressure"],
                "relativehumidity_2m": data["relativehumidity_2m"],
                "windspeed_10m": data["windspeed_10m"],
                "winddirection_10m": data["winddirection_10m"],
            },
            index=times,
        )

        return df_hr

    resp.raise_for_status()


```

```python
def transform(
    df: pd.DataFrame,
    important_list: list,
    output_csv: str,
    batch_size: int = 1000,
) -> pd.DataFrame:
    df = df.copy()
    df["ldatetime"] = pd.to_datetime(df["ldatetime"], format="%Y-%m-%dT%H:%M:%S")
    df = df.dropna(subset=important_list, how="all")
    cols_done = [
        "TEMP (temperature), K",
        "PRES (atm. pressure), Pa",
        "HUMI (relative humidity), %",
        "WV (wind velocity), m/s",
        "WD (wind direction), deg",
    ]
    df = df[~df[cols_done].notna().all(axis=1)]
    neighbors = []
    for idx, row in df.iterrows():
        dt = row["ldatetime"]
        lat = row["latitude"]
        lon = row["longitude"]
        if dt.minute == 0:
            window = pd.Timedelta(hours=4)
        elif dt.minute == 30:
            window = pd.Timedelta(hours=4.5)
        else:
            continue

        lower = dt - window
        upper = dt + window
        start = lower.ceil("H")
        end = upper.floor("H")
        t = start
        while t <= end:
            neighbors.append({"ldatetime": t, "latitude": lat, "longitude": lon})
            t += pd.Timedelta(hours=1)

    original_hours = df[df["ldatetime"].dt.minute == 0][
        ["ldatetime", "latitude", "longitude"]
    ]

    combos = pd.concat(
        [pd.DataFrame(neighbors), original_hours],
        ignore_index=True,
    ).drop_duplicates(subset=["ldatetime", "latitude", "longitude"])
    combos = combos[combos["ldatetime"].dt.minute == 0]

    if os.path.exists(output_csv):
        existing = pd.read_csv(output_csv, parse_dates=["ldatetime"])
        done_set = set(
            zip(
                existing["ldatetime"].dt.tz_localize(None),
                existing["latitude"],
                existing["longitude"],
            )
        )
    else:
        done_set = set()

    to_fetch = combos[
        ~combos.apply(
            lambda r: (r["ldatetime"], r["latitude"], r["longitude"]) in done_set,
            axis=1,
        )
    ]

    if to_fetch.empty:
        print("No new (ldatetime, lat, lon) to fetch.")
        return pd.read_csv(output_csv, parse_dates=["ldatetime"])

    buffer = []
    for (lat, lon), sub in to_fetch.groupby(["latitude", "longitude"], sort=False):
```

```
        sub = sub.sort_values("ldatetime")
        sub["period"] = sub["ldatetime"].dt.to_period("M")
        for period, slice_ in sub.groupby("period", sort=False):
            year, month = period.year, period.month
            start = f"{year}-{month:02d}-01"
            end = f"{year}-{month:02d}-{calendar.monthrange(year, month)[1]:02d}"
            df_hr = get_bulk_weather(lat, lon, start, end)

            for dt in slice_["ldatetime"]:
                idx = pd.Timestamp(dt, tz="UTC")
                if idx in df_hr.index:
                    row = df_hr.loc[idx]
                    buffer.append(
                        {
                            "ldatetime": dt,
                            "latitude": lat,
                            "longitude": lon,
                            "pressure, Pa": row["surface_pressure"] * 100.0,
                            "temperature, K": row["temperature_2m"] + 273.15,
                            "relative_humidity, %": row["relativehumidity_2m"],
                            "wind_speed, m/s": row["windspeed_10m"],
                            "wind_direction, deg": row["winddirection_10m"],
                        }
                    )
                else:
                    print(f"Missing {dt} @ {lat},{lon}")

                if len(buffer) >= batch_size:
                    _flush(buffer, output_csv)
                    done_set.update(
                        (r["ldatetime"], r["latitude"], r["longitude"]) for r in buffer
                    )
                    buffer.clear()

    if buffer:
        _flush(buffer, output_csv)

    return pd.read_csv(output_csv, parse_dates=["ldatetime"])


def _flush(records: list[dict], output_csv: str):
    df_new = pd.DataFrame(records)
    header = not os.path.exists(output_csv)
    df_new.to_csv(output_csv, mode="a", index=False, header=header)


if __name__ == "__main__":
    averages_df = pd.read_csv(
        "./Duomenys/Monitoringas/Averages.csv"
    ).drop(columns=["_type", "_page.next"])
    quantity_df = pd.read_csv(
        "./Duomenys/Monitoringas/Quantity.csv"
    ).drop(columns=["_type", "_page.next"])
    quantity_units_df = pd.read_csv(
        "./Duomenys/Monitoringas/QuantityUnits.csv"
    ).drop(columns=["_type", "_page.next"])
    station_df = (
        pd.read_csv("./Duomenys/Monitoringas/Station.csv")
        .drop(columns=["_type", "_page.next"])
        .dropna(subset=["latitude"])
    )
    units_df = pd.read_csv(
        "./Duomenys/Monitoringas/Units.csv"
    ).drop(columns=["_type", "_page.next"])

    df = prepare_dataframe(
        quantity_units_df, averages_df, units_df, quantity_df, station_df, full=True
    )

    values_list = [
        101,
        201,
        301,
```

```
        401,
        503,
        1201,
        1202,
        1302,
        2001,
        2003,
        4403,
        8803,
        8903,
        9003,
    ]
    out = transform(
        df=df, important_list=values_list, output_csv="weather_data_full.csv"
    )
    print(out.head())
```

## 2 priedas. Python kodas, skirtas modelio treniravimui.

```python
import pandas as pd
from pyproj import Transformer
from prepare_dataframes import prepare_dataframe
from prepare_elements import prepare_statistical_dataframe
import numpy as np
from sklearn.neighbors import BallTree
from scipy.optimize import minimize

import matplotlib.pyplot as plt

# skipped loading quantity_units_df, averages_df, units_df, quantity_df, station_df

df_prepared = prepare_dataframe(
    quantity_units_df, averages_df, units_df, quantity_df, station_df, True
)
df_clean, pres_stats, temp_stats = prepare_statistical_dataframe(df_prepared, True)
df_clean_dropped = df_clean.dropna(
    subset=[
        "PM25, ug/m3",
        "PM10, ug/m3",
        "NO2, ug/m3",
        "SO2, ug/m3",
        "CO, mg/m3",
        "O3, ug/m3",
    ],
    how="all",
)
transformer = Transformer.from_crs("EPSG:3346", "EPSG:4326", always_xy=True)

def convert_lks94_to_wgs84(point_str):
    point_str = point_str.replace("POINT (", "").replace(")", "")
    x, y = map(float, point_str.split())
    lon, lat = transformer.transform(y, x)
    return lat, lon

tarsos_saltiniai_df = pd.read_csv(
    "./KTU/Duomenys/TarsosZidinys.csv"
).drop(columns=["_type", "_id", "_revision", "_page.next"])
tarsos_saltiniai_df[["latitude", "longitude"]] = tarsos_saltiniai_df["koord"].apply(
    lambda x: pd.Series(convert_lks94_to_wgs84(x))
)
tarsos_saltiniai_clean = tarsos_saltiniai_df.drop(
    columns=["koord", "objecto_nr", "ptz_adresas", "ptz_anketos_nr"]
)

if "tarsos_saltiniai_clean" not in globals() or "df_clean_dropped" not in globals():
    raise RuntimeError(
        "Please load 'tarsos_saltiniai_clean' and 'df_clean_dropped' into your session."
    )

sources = tarsos_saltiniai_clean.rename(
    columns={"latitude": "lat_src", "longitude": "lon_src"}
).copy()
stations = (
    df_clean_dropped.rename(
        columns={
            "latitude": "lat_stat",
            "longitude": "lon_stat",
            "ldatetime": "datetime",
        }
    )
    .reset_index(drop=True)
    .copy()
)
stations["wind_rad"] = np.deg2rad(stations["wind_direction, deg"])
stations["Vx"] = stations["wind_speed, m/s"] * np.sin(stations["wind_rad"])
stations["Vy"] = stations["wind_speed, m/s"] * np.cos(stations["wind_rad"])
src_rad = np.deg2rad(sources[["lat_src", "lon_src"]].values)
stat_rad = np.deg2rad(stations[["lat_stat", "lon_stat"]].values)
tree = BallTree(src_rad, metric="haversine")
radius = 10_000 / 6_371_000
```

```
idxs = tree.query_radius(stat_rad, r=radius)
stat_idx = np.concatenate([np.full(len(lst), i) for i, lst in enumerate(idxs)])
src_idx = np.concatenate(idxs)
sub_stat = stations.iloc[stat_idx].reset_index(drop=True)
sub_src = sources.iloc[src_idx].reset_index(drop=True)
df = pd.concat([sub_stat, sub_src], axis=1)
df["datetime"] = pd.to_datetime(df["datetime"])
df["ptz_anketos_data"] = pd.to_datetime(df["ptz_anketos_data"])
df["hour_of_day"] = df["datetime"].dt.hour
df["age_days"] = (df["datetime"] - df["ptz_anketos_data"]).dt.days
df = df[df["age_days"] >= 0].copy()
pollutant = "PM10, ug/m3"
df = df[df[pollutant].notna()].copy()
all_pols = [
    "PM25, ug/m3",
    "PM10, ug/m3",
    "NO2, ug/m3",
    "SO2, ug/m3",
    "CO, mg/m3",
    "O3, ug/m3",
]
for col in all_pols:
    if col != pollutant:
        df.drop(columns=[col], inplace=True)

df.rename(columns={pollutant: "Y"}, inplace=True)
df.drop(columns=["ptz_anketos_data"], inplace=True)
df["dx"] = (df["lat_stat"] - df["lat_src"]) * 111000
df["dy"] = (df["lon_stat"] - df["lon_src"]) * 111000 * np.cos(np.deg2rad(df["lat_src"]))
df["sx"], df["sy"] = df["dx"], df["dy"]
df["distance"] = np.hypot(df["sx"], df["sy"])
S = 1.0
df["s_norm_x"] = df["sx"] / df["distance"] * S
df["s_norm_y"] = df["sy"] / df["distance"] * S
df["wx"] = df["s_norm_x"] + df["Vx"]
df["wy"] = df["s_norm_y"] + df["Vy"]
df["w_norm"] = np.hypot(df["wx"], df["wy"])
df["V_norm"] = np.hypot(df["Vx"], df["Vy"])
df["dot_ws"] = df["wx"] * df["s_norm_x"] + df["wy"] * df["s_norm_y"]
df["cos_phi"] = df["dot_ws"] / (df["w_norm"] * S)
df["sgn_cos"] = np.sign(df["cos_phi"]).fillna(0)
df = df[df["sgn_cos"] != 0].copy()
df["proj"] = (df["Vx"] * df["s_norm_x"] + df["Vy"] * df["s_norm_y"]) / S
df["v_perp"] = np.sqrt(np.maximum(0, df["V_norm"] ** 2 - df["proj"] ** 2))
df["sigma"] = np.where(
    df["V_norm"] != 0,
    np.maximum(0, df["w_norm"] * df["sgn_cos"] / (S + df["V_norm"] + df["v_perp"])),
    1.0,
)
df["disp_factor"] = df["sigma"] * np.exp(df["w_norm"] / df["distance"])

for col in ["ptz_objekto_bukle", "ptz_objekto_tipas", "ptz_pavojingumas_aplinkai"]:
    df[col] = df[col].fillna("Nėra reikšmės")

y = df.set_index(["stat_num", "datetime"])["Y"]
meta_vars = [
    "relative_humidity, %",
    "TEMP (temperature), K",
    "PRES (atm. pressure), Pa",
    "hour_of_day",
    "age_days",
]
df["bukle_code"], bukle_uniques = pd.factorize(df["ptz_objekto_bukle"])
df["tipas_code"], tipas_uniques = pd.factorize(df["ptz_objekto_tipas"])
df["pavoj_code"], pavoj_uniques = pd.factorize(df["ptz_pavojingumas_aplinkai"])
n1, n2, n3, r = (
    len(bukle_uniques),
    len(tipas_uniques),
    len(pavoj_uniques),
    len(meta_vars),
)

def loss(params):
```

```
    k1 = params[:n1]
    k2 = params[n1 : n1 + n2]
    k3 = params[n1 + n2 : n1 + n2 + n3]
    P = params[n1 + n2 + n3 : n1 + n2 + n3 + r]
    k_prod = (
        k1[df["bukle_code"].values]
        * k2[df["tipas_code"].values]
        * k3[df["pavoj_code"].values]
    )
    p_sum = df[meta_vars].values.dot(P)
    contribs = df["disp_factor"].values * k_prod * p_sum
    agg = (
        pd.DataFrame(
            {
                "stat_num": df["stat_num"],
                "datetime": df["datetime"],
                "contrib": contribs,
            }
        )
        .groupby(["stat_num", "datetime"])["contrib"]
        .sum()
    )
    y_hat = agg.reindex(y.index).fillna(0).values
    return np.sum(np.abs(y.values - y_hat))

initial = np.ones(n1 + n2 + n3 + r)
res = minimize(loss, initial, method="L-BFGS-B", options={"maxiter": 500, "disp": True})
opt = res.x
k1_map = dict(zip(bukle_uniques, opt[:n1]))
k2_map = dict(zip(tipas_uniques, opt[n1 : n1 + n2]))
k3_map = dict(zip(pavoj_uniques, opt[n1 + n2 : n1 + n2 + n3]))
P_map = dict(zip(meta_vars, opt[n1 + n2 + n3 :]))
print("k1 (status):", k1_map)
print("k2 (type):", k2_map)
print("k3 (hazard):", k3_map)
print("P (meteo):", P_map)
k1_opt = opt[:n1]
k2_opt = opt[n1 : n1 + n2]
k3_opt = opt[n1 + n2 : n1 + n2 + n3]
P_opt = opt[n1 + n2 + n3 :]
k_prod = (
    k1_opt[df["bukle_code"].values]
    * k2_opt[df["tipas_code"].values]
    * k3_opt[df["pavoj_code"].values]
)
p_sum = df[meta_vars].values.dot(P_opt)
df["contrib"] = df["disp_factor"] * k_prod * p_sum
agg = df.groupby(["stat_num", "datetime"])["contrib"].sum()
y_hat = agg.reindex(y.index).fillna(0)
residuals = y - y_hat

def plot_hbar(coef_map, title):
    items = sorted(coef_map.items(), key=lambda x: x[1])
    labels, values = zip(*items)
    plt.figure(dpi=300)
    plt.barh(labels, values)
    plt.xlabel("Koeficiento vertė")
    plt.title(title)
    plt.tight_layout()
    plt.show()

k1_map = dict(zip(bukle_uniques, k1_opt))
k2_map = dict(zip(tipas_uniques, k2_opt))
k3_map = dict(zip(pavoj_uniques, k3_opt))
P_map = dict(zip(meta_vars, P_opt))
plot_hbar(k1_map, "Objekto būklės koeficientai")
plot_hbar(k2_map, "Objekto tipo koeficientai")
plot_hbar(k3_map, "Objekto pavojingumo aplinkai koeficientai")
plot_hbar(P_map, "Papildomų rodiklių koeficientai")
plt.figure()
plt.hist(residuals.values, bins=30, log=True, edgecolor="black")
plt.xlabel("Likutis (Reali vertė - Spėjimas)")
plt.ylabel("Kiekis (log skalė)")
```

```
218. plt.title("Likučių pasiskirtymas (log skalė)")
219. plt.tight_layout()
220. plt.show()
```

## 3 priedas. Python kodas, skirtas skaičiavimams ir grafikams.

```python
import pandas as pd
import matplotlib.pyplot as plt
from scipy.stats import skew, kurtosis

from statsmodels.tsa.seasonal import seasonal_decompose
import calendar

import numpy as np
from calendar import isleap
import matplotlib.dates as mdates
from matplotlib.dates import DateFormatter
from scipy.stats import shapiro, normaltest
import math
from scipy import stats
from scipy.stats import pearsonr, spearmanr, kendalltau
from statsmodels.tsa.stattools import grangercausalitytests
import seaborn as sns
from sklearn.feature_selection import mutual_info_regression
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score


def prepare_temperature_column(df: pd.DataFrame) -> pd.DataFrame:
    df["620_kelvin"] = df[620] + 273.15
    df["TEMP (temperature), K"] = df[615].combine_first(df["620_kelvin"])
    return df.drop(columns=[615, 620, "620_kelvin"])


def prepare_pressure_column(df: pd.DataFrame) -> pd.DataFrame:
    df["PRES (atm. pressure), Pa"] = df[906] * 100
    return df.drop(columns=[906])


def prepare_humidity_wind(df: pd.DataFrame) -> pd.DataFrame:
    df = df.rename(columns={708: "HUMI (relative humidity), %"})
    df = df.rename(columns={1004: "WV (wind velocity), m/s"})
    df = df.rename(columns={1105: "WD (wind direction), deg"})
    return df


def prepare_dataframe(
    quantity_units_df, averages_df, units_df, quantity_df, station_df, full=False
):
    all_posssible_codes = quantity_units_df["code_combi"].unique()
    all_used_codes = []
    all_new_codes = []
    odd_quantity_codes = []
    all_legit_codes = []

    for i in averages_df["stat_num._id"].unique():
        df = averages_df[averages_df["stat_num._id"] == i]
        df_list = df["code_combi"].unique()
        for e in df_list:
            if e not in all_used_codes:
                all_used_codes.append(e)

    for i in all_used_codes:
        if i not in all_posssible_codes:
            all_new_codes.append(i)

    for i in all_new_codes:
        quantity_code = str(i)[0:2]
        if quantity_code not in odd_quantity_codes:
            odd_quantity_codes.append(quantity_code)

    for i in all_used_codes:
        if i not in all_new_codes:
            all_legit_codes.append(i)

    units_df_1 = units_df.drop(columns=["_revision", "id", "code_unit"])
    quantity_df_1 = quantity_df.drop(
```

```
        columns=["_revision", "code_quantity", "id", "code_unit._id"]
    )

    quantity_units_df_1 = quantity_units_df[
        quantity_units_df["code_combi"].isin(all_legit_codes)
    ].drop(columns=["_id", "_revision"])
    quantity_units_df_2 = quantity_units_df_1.merge(
        units_df_1, left_on="code_unit._id", right_on="_id", how="left"
    ).drop(columns=["_id", "code_unit._id"])
    quantity_units_df_3 = quantity_units_df_2.merge(
        quantity_df_1, left_on="code_quantity._id", right_on="_id", how="left"
    ).drop(columns=["_id", "code_quantity._id"])

    averages_df_1 = averages_df.drop(columns=["_id", "_revision", "id", "atribut"])
    averages_df_2 = averages_df_1.merge(
        station_df, left_on="stat_num._id", right_on="_id", how="left"
    ).drop(columns=["_id", "_revision", "stat_num._id"])
    averages_df_3 = averages_df_2.dropna(subset=["stat_num"])

    averages_df_4 = averages_df_3[
        averages_df_3["code_combi"].isin(quantity_units_df_3["code_combi"])
    ].dropna()
    convert_dict = {"code_combi": int, "stat_num": int}
    averages_df_5 = averages_df_4.astype(convert_dict)

    pivoted_df = averages_df_5.pivot_table(
        index=["ldatetime", "stat_num", "latitude", "longitude"],
        columns="code_combi",
        values="lvalue",
    ).reset_index()

    pivoted_df["stat_num"] = pivoted_df["stat_num"].astype(str)
    pivoted_df = pivoted_df[pivoted_df["stat_num"].str.match(r"^-?\d+(\.\d+)?$")]

    df_temp = prepare_temperature_column(pivoted_df)
    df_pres = prepare_pressure_column(df_temp)

    if full:
        df_pres = prepare_humidity_wind(df_pres)

    return df_pres


def prepare_metiniai_duomenys():
    df = pd.read_csv(
        "./KTU/Duomenys/Oro_tersalu_apskaita/IAtmosferaIsmetamiTersalai.csv"
    )
    df = df.rename(
        columns={"metai": "year", "kiekis": "exp_value", "tersalai": "exp_label"}
    ).drop(columns=["_type", "_id", "_revision", "_page.next", "eil_nr"])
    df["year"] = pd.to_datetime(df["year"], format="%Y")
    needed_pollutants = [
        "Pb",
        "Sox ",
        "PM10",
        "Nox ",
        "PM2_5",
        "CO",
        "Benzo_a",
        "benzo_k",
        "Benzo_b",
        "HCB",
    ]
    df_small = df.where(df["exp_label"].isin(needed_pollutants)).dropna()
    df_small["exp_label"] = df_small["exp_label"].replace("Sox ", "Sox")
    df_small["exp_label"] = df_small["exp_label"].replace("Nox ", "Nox")
    df_co_ts = (
        df_small[df_small["exp_label"] == "CO"]
        .drop(columns=["exp_label"])
        .rename(columns={"nfr_kodas": "exp_label"})
        .set_index(["exp_label", "year"])
        .sort_index()
        .drop_duplicates()
```

```
    )
    df_pm25_ts = (
        df_small[df_small["exp_label"] == "PM2_5"]
        .drop(columns=["exp_label"])
        .rename(columns={"nfr_kodas": "exp_label"})
        .set_index(["exp_label", "year"])
        .sort_index()
        .drop_duplicates()
    )
    df_pm10_ts = (
        df_small[df_small["exp_label"] == "PM10"]
        .drop(columns=["exp_label"])
        .rename(columns={"nfr_kodas": "exp_label"})
        .set_index(["exp_label", "year"])
        .sort_index()
        .drop_duplicates()
    )
    df_nox_ts = (
        df_small[df_small["exp_label"] == "Nox"]
        .drop(columns=["exp_label"])
        .rename(columns={"nfr_kodas": "exp_label"})
        .set_index(["exp_label", "year"])
        .sort_index()
        .drop_duplicates()
    )
    df_pb_ts = (
        df_small[df_small["exp_label"] == "Pb"]
        .drop(columns=["exp_label"])
        .rename(columns={"nfr_kodas": "exp_label"})
        .set_index(["exp_label", "year"])
        .sort_index()
        .drop_duplicates()
    )
    df_sox_ts = (
        df_small[df_small["exp_label"] == "Sox"]
        .drop(columns=["exp_label"])
        .rename(columns={"nfr_kodas": "exp_label"})
        .set_index(["exp_label", "year"])
        .sort_index()
        .drop_duplicates()
    )
    df_hcb_ts = (
        df_small[df_small["exp_label"] == "HCB"]
        .drop(columns=["exp_label"])
        .rename(columns={"nfr_kodas": "exp_label"})
        .set_index(["exp_label", "year"])
        .sort_index()
        .drop_duplicates()
    )
    df_viso = df_small[df_small["nfr_kodas"] == "Iš viso"]
    df_viso_ts = (
        df_viso.set_index(["exp_label", "year"])
        .sort_index()
        .drop_duplicates()
        .drop(columns=["nfr_kodas"])
    )

    df_sox_pivoted = plot_experiments_cumulative(
        df_sox_ts, r"$SO_{X}$ taršos kategorijų kumuliacinis grafikas"
    )
    df_nox_pivoted = plot_experiments_cumulative(
        df_nox_ts, r"$NO_{X}$ taršos kategorijų kumuliacinis grafikas"
    )
    df_co_pivoted = plot_experiments_cumulative(
        df_co_ts, r"$CO$ taršos kategorijų kumuliacinis grafikas"
    )
    df_pm25_pivoted = plot_experiments_cumulative(
        df_pm25_ts, r"$PM_{2.5}$ taršos kategorijų kumuliacinis grafikas"
    )
    df_pm10_pivoted = plot_experiments_cumulative(
        df_pm10_ts, r"$PM_{10}$ taršos kategorijų kumuliacinis grafikas"
    )
    df_pb_pivoted = plot_experiments_cumulative(
```

```
        df_pb_ts, r"$Pb$ taršos kategorijų kumuliacinis grafikas", n_columns=1
    )
    df_hcb_pivoted = plot_experiments_cumulative(
        df_hcb_ts,
        r"$C_{6}Cl_{6}$ (heksachlorobenzeno) taršos kategorijų kumuliacinis grafikas",
        n_columns=1,
    )

    return (
        df_co_ts,
        df_pm25_ts,
        df_pm10_ts,
        df_nox_ts,
        df_pb_ts,
        df_sox_ts,
        df_hcb_ts,
        df_viso_ts,
    )


def plot_experiments_cumulative(
    df: pd.DataFrame,
    title=r"taršos kategorijų kumuliacinis grafikas",
    n_columns: int = 2,
):
    fig, ax = plt.subplots(figsize=(12, 8), dpi=300)
    all_exp_labels = df.index.get_level_values("exp_label").unique()
    regular_exps = [e for e in all_exp_labels if e != "Iš viso"]
    n_cats = len(regular_exps)
    years = sorted(df.index.get_level_values("year").unique())

    tmp = df.reset_index()[["year", "exp_label", "exp_value"]].query(
        "exp_label != 'Iš viso'"
    )

    pivot = tmp.pivot(index="year", columns="exp_label", values="exp_value")
    pivot = pivot.reindex(index=years, columns=regular_exps)
    pivot0 = pivot.fillna(0)
    first_row = pivot0.iloc[0]
    sorted_exps = first_row.sort_values(ascending=False).index.tolist()
    pivot0 = pivot0[sorted_exps]
    y_data = pivot0.values
    cmap = plt.get_cmap("gnuplot")
    colors = [cmap(i / len(sorted_exps)) for i in range(len(sorted_exps))]

    ax.stackplot(
        years,
        y_data.T,
        labels=sorted_exps,
        colors=colors,
        alpha=0.8,
        edgecolor="none",
    )

    full_idx = pd.MultiIndex.from_product(
        [sorted_exps, years], names=["exp_label", "year"]
    )

    if "Iš viso" in all_exp_labels:
        total_series = df.loc["Iš viso", "exp_value"].reindex(years)
        total_filtered = total_series.dropna()
        ax.plot(
            total_filtered.index,
            total_filtered.values,
            "k-",
            linewidth=3,
            label="Iš viso",
            marker="o",
        )

    ax.set_title(title, fontsize=16, pad=20)
    ax.text(
        0.5,
```

```
        1.01,
        f"Kategorijų kiekis: {n_cats}",
        transform=ax.transAxes,
        ha="center",
        va="bottom",
        fontsize=12,
    )

    if isinstance(years[0], pd.Timestamp):
        ax.xaxis.set_major_locator(mdates.YearLocator())
        ax.xaxis.set_major_formatter(DateFormatter("%Y"))
    else:
        ax.set_xticks(years)
        ax.set_xticklabels([str(y) for y in years])

    ax.set_xlabel("Metai", fontsize=14)
    unit = df["matavimo_vnt"].unique()[0]
    ax.set_ylabel(f"Kiekis, {unit}", fontsize=14)
    ax.grid(True, alpha=0.3)

    ax.legend(
        title="Kategorija",
        fontsize=8,
        title_fontsize=10,
        loc="upper left",
        bbox_to_anchor=(1, 1),
        ncol=n_columns,
    )

    orig = df.copy()
    grid = pd.DataFrame(index=full_idx, columns=["exp_value"], dtype=float)
    for exp, yr in grid.index:
        try:
            grid.loc[(exp, yr), "exp_value"] = orig.loc[(exp, yr), "exp_value"]
        except KeyError:
            grid.loc[(exp, yr), "exp_value"] = np.nan

    missing_counts = grid["exp_value"].isna().groupby(level="year").sum()

    ax2 = ax.secondary_xaxis("bottom")
    ax2.set_xlim(ax.get_xlim())
    ax2.set_xticks(years)
    ax2.set_xticklabels([int(missing_counts.loc[yr]) for yr in years])
    ax2.set_xlabel("Kategorijų, su trūkstamomis vertėmis, skaičius", fontsize=12)
    ax2.spines["bottom"].set_position(("outward", 50))

    plt.tight_layout(rect=[0, 0.05, 1, 0.95])
    plt.show()

    full_index = pd.MultiIndex.from_product(
        [regular_exps, years], names=["exp_label", "year"]
    )
    complete_data = pd.DataFrame(index=full_index, columns=["exp_value"])

    for exp in regular_exps:
        for year in years:
            try:
                val = orig.loc[(exp, year), "exp_value"]
                complete_data.loc[(exp, year), "exp_value"] = val
            except (KeyError, TypeError):
                pass

    pivot_ready = complete_data.reset_index()
    pivoted_df = pivot_ready.pivot(
        index="year", columns="exp_label", values="exp_value"
    )

    if "Iš viso" in all_exp_labels:
        is_viso_data = df.loc["Iš viso"].copy()
        for year in pivoted_df.index:
            try:
                pivoted_df.loc[year, "Iš viso"] = is_viso_data.loc[year, "exp_value"]
            except (KeyError, TypeError):
```

```
                pass

    column_order = sorted_exps
    if "Iš viso" in pivoted_df.columns:
        column_order = column_order + ["Iš viso"]

    pivoted_df = pivoted_df[column_order]

    for col in pivoted_df.columns:
        pivoted_df[col] = pd.to_numeric(pivoted_df[col], errors="coerce")

    return pivoted_df


def _is_normal(series, alpha=0.05, nmin_shapiro=3, nmax_shapiro=5000, nmin_dag=8):
    x = series.dropna()
    n = len(x)
    if n < nmin_shapiro:
        return False
    if n <= nmax_shapiro:
        stat, p = shapiro(x)
    elif n >= nmin_dag:
        stat, p = normaltest(x)
    else:
        return False
    return p > alpha


def fill_missing_nrwc_with_unit(
    df,
    label_col="exp_label",
    year_col="year",
    value_col="exp_value",
    unit_col="matavimo_vnt",
    alpha=0.05,
):
    units = df[unit_col].unique()
    if len(units) != 1:
        raise ValueError(f"Expected exactly one unique unit in {unit_col}, got {units}")
    unit = units[0]
    df0 = df.reset_index()
    wide = df0.pivot(index=year_col, columns=label_col, values=value_col)
    normal_flags = {col: _is_normal(wide[col], alpha=alpha) for col in wide.columns}
    method = "pearson" if all(normal_flags.values()) else "spearman"
    print(f"{method}")
    corr = wide.corr(method=method)
    if method == "pearson":
        korel = "Pearson'o"
    else:
        korel = "Spearman'o"

    mask = wide.notna().astype(int)
    overlap = mask.T.dot(mask)
    labels = wide.columns
    W = pd.DataFrame(np.nan, index=labels, columns=labels)
    for t in labels:
        for j in labels:
            if j == t:
                continue
            n = overlap.at[t, j]
            r = corr.at[t, j]
            if n > 2 and abs(r) < 1:
                W.at[t, j] = (n - 2) * r**2 / (1 - r**2)

    filled = wide.copy()
    for t in labels:
        for yr in filled.index[filled[t].isna()]:
            refs = [
                j
                for j in labels
                if not np.isnan(W.at[t, j]) and not np.isnan(filled.at[yr, j])
            ]
            if not refs:
```

```
                continue
            wj = np.array([W.at[t, j] for j in refs])
            yj = np.array([filled.at[yr, j] for j in refs])
            filled.at[yr, t] = (wj * yj).sum() / wj.sum()

    out = filled.stack().to_frame(value_col).reorder_levels([1, 0]).sort_index()
    out[unit_col] = unit
    return out, korel


def fill_missing_nrwc_with_unit_(
    df,
    label_col="exp_label",
    year_col="year",
    value_col="exp_value",
    unit_col="matavimo_vnt",
):
    units = df[unit_col].unique()
    if len(units) != 1:
        raise ValueError(f"Expected exactly one unique unit in {unit_col}, got {units}")
    unit = units[0]
    df0 = df.reset_index()
    print(df0.columns)
    wide = df0.pivot(index=year_col, columns=label_col, values=value_col)
    corr = wide.corr()
    mask = wide.notna().astype(int)
    overlap = mask.T.dot(mask)
    labels = wide.columns
    W = pd.DataFrame(np.nan, index=labels, columns=labels)
    for t in labels:
        for j in labels:
            if j == t:
                continue
            n = overlap.at[t, j]
            r = corr.at[t, j]
            if n > 2 and abs(r) < 1:
                W.at[t, j] = (n - 2) * r**2 / (1 - r**2)

    filled = wide.copy()
    for t in labels:
        for yr in filled.index[filled[t].isna()]:
            refs = [
                j
                for j in labels
                if not np.isnan(W.at[t, j]) and not np.isnan(wide.at[yr, j])
            ]
            if not refs:
                continue
            wj = np.array([W.at[t, j] for j in refs])
            yj = np.array([wide.at[yr, j] for j in refs])
            filled.at[yr, t] = (wj * yj).sum() / wj.sum()

    out = filled.stack().to_frame(value_col).reorder_levels([1, 0]).sort_index()
    out[unit_col] = unit
    return out


def fill_missing_by_interpolation(
    df,
    label_col="exp_label",
    year_col="year",
    value_col="exp_value",
    unit_col="matavimo_vnt",
    interp_method="linear",
    limit_direction="both",
):
    if label_col not in df.columns or year_col not in df.columns:
        df0 = df.reset_index()
    else:
        df0 = df.copy()

    units = df0[unit_col].unique()
    if len(units) != 1:
```

```
            raise ValueError(
                f"Expected exactly one unique unit in {unit_col}, got {units!r}"
            )
        unit = units[0]
        wide = df0.pivot(index=year_col, columns=label_col, values=value_col)
        filled_wide = wide.interpolate(
            method=interp_method, axis=0, limit_direction=limit_direction
        )
        out = filled_wide.stack().to_frame(value_col).reorder_levels([1, 0]).sort_index()
        out[unit_col] = unit
        return out


def plot_corrected_cumulative(
        df_wide: pd.DataFrame,
        matavimo_vnt: str,
        title: str = "Taršos kategorijų kumuliacinis grafikas",
        n_columns: int = 2,
):
        df_long = (
            df_wide.reset_index()
            .melt(id_vars="year", var_name="exp_label", value_name="exp_value")
            .set_index(["exp_label", "year"])
        )
        fig, ax = plt.subplots(figsize=(12, 8), dpi=300)
        all_labels = df_long.index.get_level_values("exp_label").unique()
        cats = [e for e in all_labels if e != "Iš viso"]
        years = sorted(df_long.index.get_level_values("year").unique())
        tmp = df_long.reset_index().query("exp_label!='Iš viso'")
        pivot = tmp.pivot(index="year", columns="exp_label", values="exp_value")
        pivot = pivot.reindex(index=years, columns=cats).fillna(0)
        first = pivot.iloc[0].sort_values(ascending=False).index.tolist()
        pivot = pivot[first]
        ydat = pivot.values.T
        cmap = plt.get_cmap("gnuplot")
        colors = [cmap(i / len(first)) for i in range(len(first))]
        ax.stackplot(
            years,
            ydat,
            labels=first,
            colors=colors,
            alpha=0.8,
            edgecolor="none",
        )
        if "Iš viso" in all_labels:
            total = df_long.loc["Iš viso", "exp_value"].reindex(years)
            ax.plot(years, total, "k-", lw=3, label="Iš viso", marker="o")

        ax.set_title(title, fontsize=16, pad=20)
        ax.text(
            0.5,
            1.01,
            f"Kategorijų kiekis: {len(cats)}",
            transform=ax.transAxes,
            ha="center",
            va="bottom",
            fontsize=12,
        )
        if isinstance(years[0], pd.Timestamp):
            ax.xaxis.set_major_locator(mdates.YearLocator())
            ax.xaxis.set_major_formatter(DateFormatter("%Y"))
        else:
            ax.set_xticks(years)
            ax.set_xticklabels([str(y) for y in years])
        ax.set_xlabel("Metai", fontsize=14)
        ax.set_ylabel(f"Kiekis, {matavimo_vnt}", fontsize=14)
        ax.grid(True, alpha=0.3)

        ax.legend(
            title="Kategorija",
            fontsize=8,
            title_fontsize=10,
            loc="upper left",
```

```
        bbox_to_anchor=(1, 1),
        ncol=n_columns,
    )
    full = pd.DataFrame(
        index=pd.MultiIndex.from_product([cats, years], names=["exp_label", "year"]),
        columns=["exp_value"],
    )
    full["exp_value"] = df_long["exp_value"]
    miss = full["exp_value"].isna().groupby(level="year").sum()

    ax2 = ax.secondary_xaxis("bottom")
    ax2.set_xlim(ax.get_xlim())
    ax2.set_xticks(years)
    ax2.set_xticklabels([int(miss.loc[y]) for y in years])
    ax2.set_xlabel("Kategorijų, su trūkstamomis vertėmis, skaičius", fontsize=12)
    ax2.spines["bottom"].set_position(("outward", 50))

    plt.tight_layout(rect=[0, 0.05, 1, 0.95])
    plt.show()
    pivot_ready = full.reset_index().pivot(
        index="year", columns="exp_label", values="exp_value"
    )
    if "Iš viso" in all_labels:
        pivot_ready["Iš viso"] = df_long.loc["Iš viso", "exp_value"].reindex(years)

    pivot_ready = pivot_ready.apply(pd.to_numeric, errors="coerce")
    order = first + (["Iš viso"] if "Iš viso" in pivot_ready.columns else [])
    return pivot_ready[order]


def hampel_outliers(x, k=2, t0=3):
    x = pd.Series(x).astype(float).reset_index(drop=True)
    n = len(x)
    L = 1.4826
    out = []
    for i in range(n):
        start = max(0, i - k)
        end = min(n, i + k + 1)
        window = x.iloc[start:end]
        med = window.median()
        mad = L * np.median(np.abs(window - med))
        if mad > 0 and abs(x.iat[i] - med) > t0 * mad:
            out.append(i)
    return out


def sum_interpol_method(
    input_df: pd.DataFrame, eps: float = 0.05, k: int = 2, t0: float = 1
) -> pd.DataFrame:
    df_before = input_df.copy()
    df = input_df.copy()
    if "Iš viso" not in df.columns:
        raise KeyError("DataFrame must contain an 'Iš viso' column")

    cats = [c for c in df.columns if c != "Iš viso"]
    idx_list = list(df.index)

    def is_hampel_outlier(arr, i, k, t0):
        x = pd.Series(arr).astype(float).reset_index(drop=True)
        L = 1.4826
        start, end = max(0, i - k), min(len(x), i + k + 1)
        w = x.iloc[start:end]
        med = w.median()
        mad = L * np.median(np.abs(w - med))
        return mad > 0 and abs(x.iat[i] - med) > t0 * mad

    ratios = df[cats].sum(axis=1) / df["Iš viso"]
    bad_rows = ratios[(ratios < 1 - eps) | (ratios > 1 + eps)].index.tolist()
    print(f"Rows outside [1±{eps}]: {len(bad_rows)} → {bad_rows}")
    replacements = []
    T = len(df)
    for idx in bad_rows:
        i = idx_list.index(idx)
```

```
        for col in cats:
            ser = df[col].astype(float).copy()
            old = ser.iat[i]
            ser.iat[i] = np.nan
            if 0 < i < T - 1:
                cand = ser.interpolate().iat[i]
            else:
                nbrs = ser.dropna()
                cand = nbrs.iat[0] if i == T - 1 else nbrs.iat[-1]

            if is_hampel_outlier(ser.fillna(old).values, i, k, t0):
                new_sum = df.loc[idx, cats].sum() - old + cand
                new_ratio = new_sum / df.loc[idx, "Iš viso"]
                df.at[idx, col] = cand
                replacements.append((idx, col, old, cand))
                if 1 - eps <= new_ratio <= 1 + eps:
                    break

    print(f"Replacements made: {len(replacements)}")
    for idx, col, old, new in replacements:
        print(f" • Row {idx!r}, '{col}': {old:.5g} → {new:.5g}")

    fig, ax = plt.subplots(figsize=(12, 8), dpi=300)
    years = df.index
    ax.plot(years, ratios.values, "o-", label="Santykis")
    ax.axhline(1 + eps, color="red", linestyle="--", label=r"1+$\epsilon$")
    ax.axhline(1 - eps, color="red", linestyle="--", label=r"1-$\epsilon$")
    if pd.api.types.is_datetime64_any_dtype(years):
        ax.xaxis.set_major_locator(mdates.YearLocator())
        ax.xaxis.set_major_formatter(DateFormatter("%Y"))
        fig.autofmt_xdate()
    ax.set_ylabel("Santykis")
    ax.set_xlabel("Metai")
    ax.set_title(
        "Kiekvienų metų kategorijų sumos ir bendros sumos tinkamo santykio ribos"
    )
    ax.legend()
    plt.tight_layout()
    plt.show()
    for idx, col, old, new in replacements:
        ser0 = df_before[col].astype(float)
        med = ser0.rolling(2 * k + 1, center=True, min_periods=1).median()
        mad = ser0.rolling(2 * k + 1, center=True, min_periods=1).apply(
            lambda z: 1.4826 * np.median(np.abs(z - np.median(z))), raw=True
        )
        hi, lo = med + t0 * mad, med - t0 * mad
        fig, ax = plt.subplots(figsize=(12, 8))
        ax.fill_between(
            ser0.index, lo, hi, color="lightgray", alpha=0.5, label=f"±{t0}×MAD (k={k})"
        )
        ax.plot(ser0.index, ser0.values, "-o", color="black", alpha=0.7, label=col)
        # old vs new
        ax.scatter([idx], [old], color="red", s=100, label="Senoji vertė", zorder=5)
        ax.scatter([idx], [new], color="black", s=100, label="Naujoji vertė", zorder=5)

        if pd.api.types.is_datetime64_any_dtype(ser0.index):
            ax.xaxis.set_major_locator(mdates.YearLocator())
            ax.xaxis.set_major_formatter(DateFormatter("%Y"))
            fig.autofmt_xdate()

        disp = idx.strftime("%Y") if isinstance(idx, pd.Timestamp) else str(idx)
        ax.set_title(f"'{col}' kategorijos atnaujinimas {disp} metais")
        ax.set_ylabel(col)
        ax.set_xlabel("Metai")
        ax.legend(loc="best")
        plt.tight_layout()
        plt.show()

    return df


def prepare_final_dfs():
    df_co_ts, df_pm25_ts, df_pm10_ts, df_nox_ts, df_pb_ts, df_sox_ts, df_hcb_ts, _ = (
```

```
729.         prepare_metiniai_duomenys()
730.     )
731.
732.     df_sox_interpolated = fill_missing_by_interpolation(df_sox_ts)
733.     df_co_interpolated = fill_missing_by_interpolation(df_co_ts)
734.     df_nox_interpolated = fill_missing_by_interpolation(df_nox_ts)
735.     df_pm25_interpolated = fill_missing_by_interpolation(df_pm25_ts)
736.     df_pm10_interpolated = fill_missing_by_interpolation(df_pm10_ts)
737.     df_pb_interpolated = fill_missing_by_interpolation(df_pb_ts)
738.     df_hcb_interpolated = fill_missing_by_interpolation(df_hcb_ts)
739.
740.     df_sox_interpolated_pivoted = plot_experiments_cumulative(
741.         df_sox_interpolated,
742.         r"$SO_{X}$ taršos kategorijų kumuliacinis grafikas, užpildytas naudojant interpoliavimą",
743.     )
744.     df_co_interpolated_pivoted = plot_experiments_cumulative(
745.         df_co_interpolated,
746.         r"$CO$ taršos kategorijų kumuliacinis grafikas, užpildytas naudojant interpoliavimą",
747.     )
748.     df_nox_interpolated_pivoted = plot_experiments_cumulative(
749.         df_nox_interpolated,
750.         r"$NO_{X}$ taršos kategorijų kumuliacinis grafikas, užpildytas naudojant interpoliavimą",
751.     )
752.     df_pm25_interpolated_pivoted = plot_experiments_cumulative(
753.         df_pm25_interpolated,
754.         r"$PM_{2.5}$ taršos kategorijų kumuliacinis grafikas, užpildytas naudojant
interpoliavimą",
755.     )
756.     df_pm10_interpolated_pivoted = plot_experiments_cumulative(
757.         df_pm10_interpolated,
758.         r"$PM_{10}$ taršos kategorijų kumuliacinis grafikas, užpildytas naudojant
interpoliavimą",
759.     )
760.     df_pb_interpolated_pivoted = plot_experiments_cumulative(
761.         df_pb_interpolated,
762.         r"$Pb$ taršos kategorijų kumuliacinis grafikas, užpildytas naudojant interpoliavimą",
763.     )
764.     df_hcb_interpolated_pivoted = plot_experiments_cumulative(
765.         df_hcb_interpolated,
766.         r"$HCB$ taršos kategorijų kumuliacinis grafikas, užpildytas naudojant interpoliavimą",
767.     )
768.
769.     str_co = r"$CO$ taršos kategorijų sutvarkytas grafikas, užpildytas naudojant sumų-
interpoliavimo metodą."
770.     str_sox = r"$SO_{X}$ taršos kategorijų sutvarkytas grafikas, užpildytas naudojant sumų-
interpoliavimo metodą."
771.     str_nox = r"$NO_{X}$ taršos kategorijų sutvarkytas grafikas, užpildytas naudojant sumų-
interpoliavimo metodą."
772.     str_pm25 = r"$PM_{2.5}$ taršos kategorijų sutvarkytas grafikas, užpildytas naudojant sumų-
interpoliavimo metodą."
773.     str_pm10 = r"$PM_{10}$ taršos kategorijų sutvarkytas grafikas, užpildytas naudojant sumų-
interpoliavimo metodą."
774.     str_pb = r"$Pb$ taršos kategorijų sutvarkytas grafikas, užpildytas naudojant sumų-
interpoliavimo metodą."
775.     str_hcb = r"$C_{6}Cl_{6}$ taršos kategorijų sutvarkytas grafikas, užpildytas naudojant sumų-
interpoliavimo metodą."
776.
777.     df_co_summed = sum_interpol_method(
778.         input_df=df_co_interpolated_pivoted, eps=0.05, k=2, t0=1
779.     )
780.     co_final = plot_corrected_cumulative(
781.         df_co_summed, matavimo_vnt="kt", title=str_co, n_columns=2
782.     )
783.
784.     df_sox_summed = sum_interpol_method(
785.         input_df=df_sox_interpolated_pivoted, eps=0.05, k=2, t0=3
786.     )
787.     sox_final = plot_corrected_cumulative(
788.         df_sox_summed, matavimo_vnt="kt", title=str_sox, n_columns=2
789.     )
790.
791.     df_nox_summed = sum_interpol_method(
792.         input_df=df_nox_interpolated_pivoted, eps=0.05, k=2, t0=3
```

```
    )
    nox_final = plot_corrected_cumulative(
        df_nox_summed, matavimo_vnt="kt", title=str_nox, n_columns=2
    )

    df_pm25_summed = sum_interpol_method(
        input_df=df_pm25_interpolated_pivoted, eps=0.05, k=2, t0=3
    )
    pm25_final = plot_corrected_cumulative(
        df_pm25_summed, matavimo_vnt="kt", title=str_pm25, n_columns=2
    )

    df_pm10_summed = sum_interpol_method(
        input_df=df_pm10_interpolated_pivoted, eps=0.05, k=2, t0=3
    )
    pm10_final = plot_corrected_cumulative(
        df_pm10_summed, matavimo_vnt="kt", title=str_pm10, n_columns=2
    )

    df_pb_summed = sum_interpol_method(
        input_df=df_pb_interpolated_pivoted, eps=0.05, k=2, t0=3
    )
    pb_final = plot_corrected_cumulative(
        df_pb_summed, matavimo_vnt="t", title=str_pb, n_columns=2
    )

    df_hcb_summed = sum_interpol_method(
        input_df=df_hcb_interpolated_pivoted, eps=0.05, k=2, t0=3
    )
    hcb_final = plot_corrected_cumulative(
        df_hcb_summed, matavimo_vnt="kg", title=str_hcb, n_columns=2
    )

    dataframes = [
        co_final,
        sox_final,
        nox_final,
        pm25_final,
        pm10_final,
        pb_final,
        hcb_final,
    ]
    names = [
        "CO metinis, kt",
        "SOX metinis, kt",
        "NOX metinis, kt",
        "PM25 metinis, kt",
        "PM10 metinis, kt",
        "Pb metinis, t",
        "HCB metinis, kg",
    ]

    renamed_dfs = []
    for df, name in zip(dataframes, names):
        temp_df = df[["Iš viso"]].copy()
        temp_df = temp_df.rename(columns={"Iš viso": f"{name}"})
        renamed_dfs.append(temp_df)

    result = pd.concat(renamed_dfs, axis=1)
    result.index.name = None
    return result


def daily_means(df, time_col="ldatetime"):
    df = df.copy()
    df[time_col] = pd.to_datetime(df[time_col])
    df = df.set_index(time_col)
    df_num = df.select_dtypes(include=[np.number])
    daily = df_num.resample("D").mean()
    return daily


def analyze_timeseries(
```

```
    df: pd.DataFrame,
    aqg_in: float,
    it_1: float,
    it_2: float,
    it_3: float,
    it_4: float,
    column_name: str,
    time_col: str = "ldatetime",
) -> pd.DataFrame:
    df = df.copy()
    df[time_col] = pd.to_datetime(df[time_col])
    df = df.set_index(time_col).sort_index()

    neg = df[df[column_name] < 0][column_name]
    if not neg.empty:
        print(f"Found {len(neg)} negative readings; sample values:")
        print(neg.value_counts().to_frame(name="count").head())
        df[column_name] = df[column_name].clip(lower=0)
    else:
        print("No negative readings found.")

    coverage = {}
    for year, group in df[column_name].groupby(df.index.year):
        present = group.notna().sum()
        days = 366 if isleap(year) else 365
        possible = days * 48
        pct = present / possible * 100
        coverage[year] = (present, possible, pct)
        print(f"{year}: {present}/{possible} samples " f"({pct:.1f} %) present")

    daily = daily_means(df.reset_index(), time_col=time_col)
    if not isinstance(daily.index, pd.DatetimeIndex):
        daily.index = pd.to_datetime(daily.index)

    thresholds = {
        "AQG": aqg_in,
        "IT4": it_4,
        "IT3": it_3,
        "IT2": it_2,
        "IT1": it_1,
    }
    for year, grp in daily.groupby(daily.index.year):
        total_days = grp.shape[0]
        present_days = grp[column_name].count()
        missing_days = total_days - present_days
        print(f"\n=== {year} daily-summary ===")
        print(
            f"Days with data:   {present_days}/{total_days} ({present_days/total_days*100:.1f}
%)"
        )
        print(
            f"Days missing:     {missing_days}/{total_days} ({missing_days/total_days*100:.1f}
%)"
        )

        for name, thresh in thresholds.items():
            above = (grp[column_name] > thresh).sum()
            below = (grp[column_name] < thresh).sum()
            print(
                f"Days > {name} ({thresh}): {above}/{present_days} "
                f"({above/present_days*100:.1f} %),  "
                f"< {name}: {below}/{present_days} "
                f"({below/present_days*100:.1f} %)"
            )

    top10 = df[column_name].nlargest(10)
    print("\nTop 10 individual readings:")
    print(top10.to_frame(name=column_name))

    plt.figure(figsize=(10, 6))
    ax = plt.gca()
    for year, grp in daily.groupby(daily.index.year):
        x = grp.index.dayofyear
```

```
        y = grp[column_name]
        ax.plot(x, y, marker="", label=str(year))

    for name, thresh in thresholds.items():
        ax.axhline(thresh, linestyle="--", label=name)

    ax.set_xlabel("Day of Year")
    ax.set_ylabel(column_name)
    ax.set_title(f"Daily means of {column_name} by Year")
    ax.legend(loc="upper right", fontsize="small")
    plt.tight_layout()
    plt.show()
    return daily


def summarize_timeseries(
    df: pd.DataFrame, column_name: str, thresholds: dict, time_col: str = "ldatetime"
) -> pd.Series:
    df = df.copy()
    df[time_col] = pd.to_datetime(
        df[time_col], format="%Y-%m-%dT%H:%M:%S", errors="coerce"
    )
    df = df.dropna(subset=[time_col]).set_index(time_col).sort_index()
    neg = df[column_name][df[column_name] < 0]
    neg_count = int(len(neg))
    neg_values = neg.unique().tolist()
    df[column_name] = df[column_name].clip(lower=0)
    years = df.index.year.unique()
    possible = sum((366 if isleap(yr) else 365) * 48 for yr in years)
    present = int(df[column_name].notna().sum())
    present_pct = present / possible * 100 if possible else np.nan
    daily = daily_means(df.reset_index(), time_col=time_col)
    total_days = int(len(daily))
    days_present = int(daily[column_name].count())
    days_missing = total_days - days_present
    days_present_pct = days_present / total_days * 100 if total_days else np.nan
    days_missing_pct = days_missing / total_days * 100 if total_days else np.nan
    stats = {}
    for name, thr in thresholds.items():
        above = int((daily[column_name] > thr).sum())
        below = int((daily[column_name] < thr).sum())
        stats[f"days_above_{name}"] = above
        stats[f"days_above_{name}_pct"] = (
            above / total_days * 100 if total_days else np.nan
        )
        stats[f"days_below_{name}"] = below
        stats[f"days_below_{name}_pct"] = (
            below / total_days * 100 if total_days else np.nan
        )

    top10 = df[column_name].nlargest(10)
    top10_vals = top10.tolist()
    top10_timestamps = [ts.isoformat() for ts in top10.index]
    data = {
        "neg_count": neg_count,
        "neg_values": neg_values,
        "halfhour_present": present,
        "halfhour_possible": possible,
        "halfhour_present_pct": present_pct,
        "daily_days_total": total_days,
        "daily_days_present": days_present,
        "daily_days_present_pct": days_present_pct,
        "daily_days_missing": days_missing,
        "daily_days_missing_pct": days_missing_pct,
        **stats,
        "top10_values": top10_vals,
        "top10_timestamps": top10_timestamps,
    }
    return pd.Series(data)


def make_all_station_summaries(
    df_clean: pd.DataFrame,
```

```
    threshold_dict: dict,
    station_col: str = "stat_num",
    time_col: str = "ldatetime",
) -> pd.DataFrame:
    stations = sorted(df_clean[station_col].dropna().unique())
    records, index = [], []

    for sid in stations:
        df_stat = df_clean[df_clean[station_col] == sid]
        for col, thr in threshold_dict.items():
            if col not in df_stat.columns:
                continue
            summary = summarize_timeseries(df_stat, col, thr, time_col=time_col)
            if summary["halfhour_present"] != 0:
                records.append(summary)
                index.append((sid, col))

    mi = pd.MultiIndex.from_tuples(index, names=[station_col, "column"])
    return pd.DataFrame(records, index=mi)


def make_station_df(df: pd.DataFrame, station_id: str) -> pd.DataFrame:
    return df[df["stat_num"] == station_id].drop(
        columns=["stat_num", "latitude", "longitude"]
    )


def merge_fill_compare(
    original_df: pd.DataFrame,
    helper_df: pd.DataFrame,
    join_cols=["ldatetime", "latitude", "longitude"],
    orig_pres_col: str = "PRES (atm. pressure), Pa",
    orig_temp_col: str = "TEMP (temperature), K",
    helper_pres_col: str = "pressure, Pa",
    helper_temp_col: str = "temperature, K",
    full: bool = False,
) -> tuple[pd.DataFrame, pd.DataFrame, pd.DataFrame]:
    if full:
        df = original_df.merge(
            helper_df[
                join_cols
                + [
                    helper_pres_col,
                    helper_temp_col,
                    "relative_humidity, %",
                    "wind_speed, m/s",
                    "wind_direction, deg",
                ]
            ],
            on=join_cols,
            how="left",
        )
    else:
        df = original_df.merge(
            helper_df[join_cols + [helper_pres_col, helper_temp_col]],
            on=join_cols,
            how="left",
        )

    orig_pres = df[orig_pres_col]
    helper_pres = df[helper_pres_col]
    orig_temp = df[orig_temp_col]
    helper_temp = df[helper_temp_col]

    percentiles = [0.01, 0.05, 0.1, 0.25, 0.5, 0.75, 0.9, 0.95, 0.99]
    pres_desc_o = orig_pres.describe(percentiles=percentiles)
    pres_desc_h = helper_pres.describe(percentiles=percentiles)
    pres_stats = pd.DataFrame({"original": pres_desc_o, "helper": pres_desc_h})
    pres_stats.loc["skew"] = [orig_pres.skew(), helper_pres.skew()]
    pres_stats.loc["kurtosis"] = [orig_pres.kurtosis(), helper_pres.kurtosis()]

    pres_diff = orig_pres.subtract(helper_pres)
    temp_diff = orig_temp.subtract(helper_temp)
```

```
1083.
1084.     print("\n--- Pressure differences (orig – helper) ---")
1085.     print(pres_diff.describe(percentiles=percentiles).to_string())
1086.     print("\n--- Temperature differences (orig – helper) ---")
1087.     print(temp_diff.describe(percentiles=percentiles).to_string())
1088.
1089.     temp_desc_o = orig_temp.describe(percentiles=percentiles)
1090.     temp_desc_h = helper_temp.describe(percentiles=percentiles)
1091.     temp_stats = pd.DataFrame({"original": temp_desc_o, "helper": temp_desc_h})
1092.     temp_stats.loc["skew"] = [orig_temp.skew(), helper_temp.skew()]
1093.     temp_stats.loc["kurtosis"] = [orig_temp.kurtosis(), helper_temp.kurtosis()]
1094.
1095.     print("\n=== Full descriptive statistics for Pressure (Pa) ===")
1096.     print(pres_stats.to_string())
1097.     print("\n=== Full descriptive statistics for Temperature (K) ===")
1098.     print(temp_stats.to_string())
1099.
1100.     pres_mask_fill = df[helper_pres_col].notna()
1101.     temp_mask_fill = df[helper_temp_col].notna()
1102.
1103.     n_pres_inserted = pres_mask_fill.sum()
1104.     n_temp_inserted = temp_mask_fill.sum()
1105.
1106.     df.loc[pres_mask_fill, orig_pres_col] = df.loc[pres_mask_fill, helper_pres_col]
1107.     df.loc[temp_mask_fill, orig_temp_col] = df.loc[temp_mask_fill, helper_temp_col]
1108.
1109.     print(f"\nInserted {n_pres_inserted} pressure values")
1110.     print(f"Inserted {n_temp_inserted} temperature values")
1111.
1112.     plt.figure(figsize=(6, 4), dpi=300)
1113.     plt.hist(pres_diff.dropna(), bins=50, log=True)
1114.     plt.title("Pradinio ir pagalbinio slėgių skirtumo logaritminė histograma")
1115.     plt.xlabel("Δ slėgio [Pa]")
1116.     plt.ylabel("Kiekis")
1117.     plt.tight_layout()
1118.     plt.show()
1119.
1120.     plt.figure(figsize=(6, 6), dpi=300)
1121.     mask = orig_pres.notna() & helper_pres.notna()
1122.     plt.scatter(helper_pres[mask], orig_pres[mask], s=10, alpha=0.5)
1123.     lims = [
1124.         min(helper_pres.min(), orig_pres.min()),
1125.         max(helper_pres.max(), orig_pres.max()),
1126.     ]
1127.     plt.plot(lims, lims, "r--", label="y = x")
1128.     plt.title("Pradinio ir pagalbinio slėgių palyginimai")
1129.     plt.xlabel("Pagalbinis slėgis [Pa]")
1130.     plt.ylabel("Pradinis slėgis [Pa]")
1131.     plt.legend()
1132.     plt.tight_layout()
1133.     plt.show()
1134.
1135.     plt.figure(figsize=(6, 4), dpi=300)
1136.     plt.hist(temp_diff.dropna(), bins=50, log=True)
1137.     plt.title("Pradinės ir pagalbinės temperatūrų skirtumo logaritminė histograma")
1138.     plt.xlabel("Δ temperatūros [K]")
1139.     plt.ylabel("Kiekis")
1140.     plt.tight_layout()
1141.     plt.show()
1142.
1143.     plt.figure(figsize=(6, 6), dpi=300)
1144.     mask_t = orig_temp.notna() & helper_temp.notna()
1145.     plt.scatter(helper_temp[mask_t], orig_temp[mask_t], s=10, alpha=0.5)
1146.     lims_t = [
1147.         min(helper_temp.min(), orig_temp.min()),
1148.         max(helper_temp.max(), orig_temp.max()),
1149.     ]
1150.     plt.plot(lims_t, lims_t, "r--", label="y = x")
1151.     plt.title("Pradinės ir pagalbinės temperatūrų palyginimai")
1152.     plt.xlabel("Pagalbinė temperatūra [K]")
1153.     plt.ylabel("Pradinė temperatūra [K]")
1154.     plt.legend()
1155.     plt.tight_layout()
```

```
    plt.show()

    df_filled = df.drop(columns=[helper_pres_col, helper_temp_col])
    return df_filled, pres_stats, temp_stats


def coalesce_temperature_and_pressure(
    original_df: pd.DataFrame, helper_df: pd.DataFrame
) -> pd.DataFrame:
    merged_df = original_df.join(
        helper_df[
            ["ldatetime", "latitude", "longitude", "pressure, Pa", "temperature, K"]
        ],
        on=["ldatetime", "latitude", "longitude"],
        how="left",
    )

    merged_df["PRES (atm. pressure), Pa"] = merged_df[
        "PRES (atm. pressure), Pa"
    ].fillna(merged_df["pressure, Pa"])
    merged_df["TEMP (temperature), K"] = merged_df["TEMP (temperature), K"].fillna(
        merged_df["temperature, K"]
    )
    merged_df = merged_df.drop(columns=["pressure, Pa", "temperature, K"])
    return merged_df


def ppb_to_ugm3(
    df: pd.DataFrame,
    ppb_column,
    molar_mass: float,
    output_column: str,
    p_column: str = "PRES (atm. pressure), Pa",
    temp_k_column: str = "TEMP (temperature), K",
) -> pd.DataFrame:
    multiply_factor = 273.15 / (22.711 * 1e3 * 1e2)
    df[output_column] = (
        df[ppb_column] * molar_mass * df[p_column] / df[temp_k_column] * multiply_factor
    )
    return df


def prepare_only_renamed_columns(df: pd.DataFrame) -> pd.DataFrame:
    df = df.rename(columns={503: "PM10, ug/m3"})
    df = df.rename(columns={8803: "PM1, ug/m3"})
    df = df.rename(columns={8903: "PM4, ug/m3"})
    df = df.rename(columns={9003: "PM_total, ug/m3"})
    df = df.rename(columns={4403: "PM25, ug/m3"})
    return df


def prepare_transformed_columns(df: pd.DataFrame) -> pd.DataFrame:
    # O3
    df["1202_ppb"] = df[1202] * 1000
    df["O3 (ozon), ppb"] = df[1201].combine_first(df["1202_ppb"])
    df = ppb_to_ugm3(
        df=df,
        ppb_column="O3 (ozon), ppb",
        molar_mass=48.0,
        p_column="PRES (atm. pressure), Pa",
        temp_k_column="TEMP (temperature), K",
        output_column="O3, ug/m3",
    )

    # SO2
    df = ppb_to_ugm3(
        df=df,
        ppb_column=101,
        molar_mass=64.066,
        output_column="SO2, ug/m3",
    )

    # NO2
```

```
    df = ppb_to_ugm3(
        df=df,
        ppb_column=301,
        molar_mass=46.0055,
        output_column="NO2, ug/m3",
    )

    # CO
    df = ppb_to_ugm3(
        df=df,
        ppb_column=1302,
        molar_mass=28.01,
        output_column="CO, mg/m3",
    )

    return df


def prepare_clean_dataframe(df: pd.DataFrame) -> pd.DataFrame:
    columns_to_keep = [
        "ldatetime",
        "stat_num",
        "latitude",
        "longitude",
        "O3, ug/m3",
        "SO2, ug/m3",
        "NO2, ug/m3",
        "CO, mg/m3",
        "PM1, ug/m3",
        "PM4, ug/m3",
        "PM10, ug/m3",
        "PM_total, ug/m3",
        "PM25, ug/m3",
    ]
    helper_df = pd.read_csv("./KTU/weather_data_full_with_halves.csv")

    df_1 = coalesce_temperature_and_pressure(original_df=df, helper_df=helper_df)
    df_2 = prepare_only_renamed_columns(df=df_1)
    df_3 = prepare_transformed_columns(df=df_2)
    return df_3[columns_to_keep]


def prepare_statistical_dataframe(df: pd.DataFrame, full: bool = False) -> pd.DataFrame:
    columns_to_keep = [
        "ldatetime",
        "stat_num",
        "latitude",
        "longitude",
        "O3, ug/m3",
        "SO2, ug/m3",
        "NO2, ug/m3",
        "CO, mg/m3",
        "PM10, ug/m3",
        "PM25, ug/m3",
    ]
    columns_to_keep_full = [
        "ldatetime",
        "stat_num",
        "latitude",
        "longitude",
        "O3, ug/m3",
        "SO2, ug/m3",
        "NO2, ug/m3",
        "CO, mg/m3",
        "PM10, ug/m3",
        "PM25, ug/m3",
        "relative_humidity, %",
        "wind_speed, m/s",
        "wind_direction, deg",
        "TEMP (temperature), K",
        "PRES (atm. pressure), Pa",
    ]
    helper_df = pd.read_csv("./KTU/weather_data_full_with_halves.csv")
```

```
    helper_df["ldatetime"] = pd.to_datetime(helper_df["ldatetime"], errors="coerce")
    helper_df["ldatetime"] = helper_df["ldatetime"].dt.strftime("%Y-%m-%dT%H:%M:%S")

    df_1, pres_stats, temp_stats = merge_fill_compare(
        original_df=df, helper_df=helper_df, full=True
    )
    df_2 = prepare_only_renamed_columns(df=df_1)
    df_3 = prepare_transformed_columns(df=df_2)
    if full:
        return df_3[columns_to_keep_full], pres_stats, temp_stats
    else:
        return df_3[columns_to_keep], pres_stats, temp_stats


def prepare_medicinos_duomenys():
    df_statistika = pd.read_excel(
        "./KTU/Duomenys/Sveikatos/report-2.xls",
        header=[1, 2],
        index_col=0,
    ).T
    df_total = (
        df_statistika[
            df_statistika.index.get_level_values(1)
            == "Mirčių skaičius pagal ilgąjį sąrašą"
        ]
        .droplevel(1)
        .fillna(0)
    )
    df_per_100 = (
        df_statistika[
            df_statistika.index.get_level_values(1)
            == "Mirčių skaičius pagal ilgąjį sąrašą 100 000 gyventojų"
        ]
        .droplevel(1)
        .fillna(0)
    )
    relevant_columns = [
        "  1 A00-Y89, U07 Iš viso",
        " 16 C00-C96 Piktybiniai navikai, iš jų:",
        " 88 I00-I99 Kraujotakos sistemos ligos, iš jų:",
        "124 J00-J99 Kvėpavimo sistemos ligos, iš jų:",
    ]
    df_per_100.index = pd.to_datetime(df_per_100.index.astype(str), format="%Y")
    df_per_100_short = df_per_100[relevant_columns].rename(
        columns={
            "  1 A00-Y89, U07 Iš viso": "Visos mirtys",
            " 16 C00-C96 Piktybiniai navikai, iš jų:": "Navikai",
            " 88 I00-I99 Kraujotakos sistemos ligos, iš jų:": "Kraujotakos sistemos ligos",
            "124 J00-J99 Kvėpavimo sistemos ligos, iš jų:": "Kvėpavimo sistemos ligos",
        }
    )
    return df_total, df_per_100_short


def make_all_aggregates(df: pd.DataFrame, cols: list[str]):
    df = df.copy()
    df["ldatetime"] = pd.to_datetime(df["ldatetime"])
    df = df.sort_values(["stat_num", "ldatetime"])
    df["date"] = df["ldatetime"].dt.normalize()
    df = df.set_index("ldatetime")
    df["O3_8h_av"] = (
        df.groupby("stat_num")["O3, ug/m3"]
        .rolling("8H")
        .mean()
        .reset_index(level=0, drop=True)
    )
    df = df.reset_index()

    daily_max_8h = (
        df.groupby(["date", "stat_num"])["O3_8h_av"]
        .max()
        .reset_index()
        .rename(columns={"date": "ldatetime", "O3_8h_av": "O3 (8 val.), ug/m3"})
```

```
    )

    daily_df = (
        df.groupby(["date", "stat_num"])[cols]
        .mean()
        .reset_index()
        .rename(columns={"date": "ldatetime"})
    )
    daily_df = daily_df.merge(daily_max_8h, on=["ldatetime", "stat_num"], how="left")
    monthly_df = (
        daily_df.assign(
            month=lambda x: x["ldatetime"].dt.to_period("M").dt.to_timestamp()
        )
        .groupby(["month", "stat_num"])[cols]
        .mean()
        .reset_index()
        .rename(columns={"month": "ldatetime"})
    )
    yearly_df = (
        monthly_df.assign(
            year=lambda x: x["ldatetime"].dt.to_period("Y").dt.to_timestamp()
        )
        .groupby(["year", "stat_num"])[cols]
        .mean()
        .reset_index()
        .rename(columns={"year": "ldatetime"})
    )
    weekday_df = (
        daily_df.assign(weekday=lambda x: x["ldatetime"].dt.weekday + 1)
        .groupby(["weekday", "stat_num"])[cols]
        .mean()
        .reset_index()
    )
    half_hour_df = (
        df.assign(slot=lambda x: x["ldatetime"].dt.floor("30T").dt.time)
        .groupby(["slot", "stat_num"])[cols]
        .mean()
        .reset_index()
    )

    years_df = (
        yearly_df.assign(
            year=lambda x: x["ldatetime"].dt.to_period("Y").dt.to_timestamp()
        )
        .groupby(["year"])[cols]
        .mean()
        .reset_index()
        .rename(columns={"year": "ldatetime"})
        .set_index("ldatetime")
    )
    years_df.index.name = None

    return {
        "daily_df": daily_df,
        "monthly_df": monthly_df,
        "yearly_df": yearly_df,
        "weekday_df": weekday_df,
        "half_hour_df": half_hour_df,
        "year_df": years_df,
    }


def get_gnuplot_colors(n=17):
    cmap = plt.get_cmap("tab20c", n)
    return [cmap(i) for i in range(n)]


def describe_and_boxplot(
    df: pd.DataFrame,
    cols: list[str],
    group_col: str,
    color_map: dict[str, tuple],
    min_count: int = 30,
```

```
    subtitle: str | None = None,
    max_value: float | None = None,
    pollutant: str = "x-axis",
    concen: str = "x-axis",
):
    groups = list(color_map.keys())

    for col in cols:
        records = []
        for g in groups:
            s = df.loc[df[group_col] == g, col].dropna()
            if len(s) < min_count:
                continue
            desc = s.describe(percentiles=[0.25, 0.5, 0.75])
            records.append(
                {
                    group_col: g,
                    "count": len(s),
                    "mean": desc["mean"],
                    "std": desc["std"],
                    "min": desc["min"],
                    "25%": desc["25%"],
                    "50%": desc["50%"],
                    "75%": desc["75%"],
                    "max": desc["max"],
                    "skew": skew(s),
                    "kurtosis": kurtosis(s, fisher=True),
                }
            )

        stats_df = pd.DataFrame.from_records(
            records,
            columns=[
                group_col,
                "count",
                "mean",
                "std",
                "min",
                "25%",
                "50%",
                "75%",
                "max",
                "skew",
                "kurtosis",
            ],
        )

        def fmt(x):
            return f"{x:.2f}".replace(".", ",")

        for fld in [
            "mean",
            "std",
            "min",
            "25%",
            "50%",
            "75%",
            "max",
            "skew",
            "kurtosis",
        ]:
            stats_df[fld] = stats_df[fld].apply(fmt)

        print(f"\n— Descriptive stats for '{col}' (≥{min_count} pts) —")
        print(stats_df.to_string(index=False))

        valid = [
            g
            for g in groups
            if df.loc[df[group_col] == g, col].dropna().size >= min_count
        ]
        data = []
        for g in valid:
```

```
            series = df.loc[df[group_col] == g, col].dropna()
            if max_value is not None:
                series = series.clip(upper=max_value)
            data.append(series.values)

        fig, ax = plt.subplots(figsize=(8, 5), dpi=300)
        bp = ax.boxplot(
            data, tick_labels=valid, vert=False, patch_artist=True, showfliers=True
        )
        for patch, g in zip(bp["boxes"], valid):
            patch.set_facecolor(color_map[g])

        if max_value is not None:
            ax.set_xlim(0, max_value)

        if subtitle is not None:
            _subtitle = "su visomis pilnomis diagramomis ir išskirtimis."
        else:
            _subtitle = f"su nevisomis pilnomis diagramomis, apribota ties {max_value}."
        ax.set_title(f"Stačiakampė {pollutant} teršalo diagrama,\n{_subtitle}")
        ax.set_xlabel(f"Koncentracija, {concen}")
        print(f"Stačiakampė {col} teršalo diagrama, {_subtitle}")
        plt.tight_layout()
        plt.show()


def plot_event_timeline(
    df: pd.DataFrame,
    date_col: str = "ldatetime",
    group_col: str = "station_name",
    color_map: dict = None,
    vlines: list = None,
    vline_kwargs: dict = None,
    figsize: tuple = (12, 8),
):
    df = df.copy()
    df[date_col] = pd.to_datetime(df[date_col])

    vline_kwargs = vline_kwargs or {"color": "gray", "linestyle": "--", "linewidth": 1}

    groups = [
        g
        for g in (color_map.keys() if color_map else df[group_col].unique())
        if g in df[group_col].unique()
    ]
    n = len(groups)
    y_positions = {g: n - i for i, g in enumerate(groups)}

    fig, ax = plt.subplots(figsize=figsize, dpi=300)
    if vlines:
        for vd in vlines:
            ax.axvline(pd.to_datetime(vd), **vline_kwargs)

    for group in groups:
        dates = df.loc[df[group_col] == group, date_col].sort_values()
        if dates.empty:
            continue
        y = y_positions[group]
        start, end = dates.min(), dates.max()
        color = color_map.get(group, "C0") if color_map else "C0"

        ax.hlines(y, start, end, color=color, linewidth=2)
        ax.scatter(dates, [y] * len(dates), color=color, edgecolor="black", zorder=3)

    ax.set_yticks(list(y_positions.values()))
    ax.set_yticklabels(groups)
    ax.invert_yaxis()
    ax.set_xlabel("Metai")
    ax.set_ylabel("Matavimo stotis")
    ax.set_title("Matavimų laikotarpis")
    plt.tight_layout()
    plt.show()

```

```python

def plot_decomp_per_station_segments(
    daily_df: pd.DataFrame,
    value_col: str,
    nice_name: str,
    nice_value: str,
    name_maps,
    date_col: str = "ldatetime",
    group_col: str = "station_name",
    color_map: dict[str, tuple] = None,
    period: int = 365,
    max_interp_frac: float = 0.8,
    break_days: int = 30,
    plottings: list[bool] = [True, True, True, True],
    minimum_cycles: int = 0,
):
    # broken because of the maps
    stations = list(color_map.keys())
    min_len = 2 * period

    for station in stations:
        s = (
            daily_df.loc[daily_df[group_col] == station, [date_col, value_col]]
            .dropna(subset=[value_col])
            .set_index(date_col)
            .sort_index()[value_col]
        )
        if s.empty:
            print(f"  {station}: no data → skip")
            continue

        dates = s.index
        gaps = dates.to_series().diff().dt.days.fillna(0)
        seg_ids = (gaps > break_days).cumsum()

        for seg_id, seg_dates in s.groupby(seg_ids):
            start, end = seg_dates.index.min(), seg_dates.index.max()
            total_days = (end - start).days + 1
            if total_days < min_len:
                print(
                    f"  {station}·seg{seg_id}: only {total_days}d (<{min_len}) → skip"
                )
                continue

            idx = pd.date_range(start, end, freq="D")
            ts = seg_dates.reindex(idx).asfreq("D").interpolate(method="time")
            orig_days = len(seg_dates)
            interp_frac = 1 - orig_days / len(idx)
            if interp_frac > max_interp_frac:
                pct = interp_frac * 100
                print(
                    f"  {station}·seg{seg_id}: {pct:.1f}% interpolated (>
{max_interp_frac*100:.0f}%) → skip"
                )
                continue

            ts = ts.dropna()
            if len(ts) < min_len:
                print(
                    f"  {station}·seg{seg_id}: {len(ts)}d after dropna (<{min_len}) → skip"
                )
                continue

            pct = interp_frac * 100
            print(
                f"Used! {station}·seg{seg_id}: {pct:.1f}% interpolated, {len(ts)}d after dropna"
            )

            res = seasonal_decompose(
                ts, model="additive", period=period, extrapolate_trend="freq"
            )
            week_numbers = pd.to_datetime(ts.index).isocalendar().week
            n_cycles = week_numbers.nunique()
```

```
1666.
1667.             comps = ["observed", "trend", "seasonal", "resid"]
1668.             _observed = "Stebėta reišmė, " + nice_value
1669.             _trend = "Tendencija, " + nice_value
1670.             _seasonal = "Sezoniškumas, " + nice_value
1671.             _residual = "Liekana, " + nice_value
1672.             labels = [_observed, _trend, _seasonal, _residual]
1673.             sup_t = f"{nice_name} taršos koncentracijos dekompozicija oro matavimo stotyje:
{station}.\nAtkarpa nuo {start.date()} iki {end.date()}."
1674.             _seasonal_title = f"Pagal sezoniškumą pakoreguotos {nice_name} taršos koncentracija
oro matavimo stotyje: {station}.\nAtkarpa nuo {start.date()} iki {end.date()}."
1675.             _concentration_value = "Koncentracija, " + nice_value
1676.             _trend_tilte = f"Tendencinė {nice_name} taršos koncentracija oro matavimo stotyje:
{station}.\nAtkarpa nuo {start.date()} iki {end.date()}."
1677.             _nice_title = f"Mėnesiais išskaidyta {nice_name} taršos koncentracija su panaikintu
sezoniškumu oro matavimo stotyje: {station}.\nAtkarpa nuo {start.date()} iki {end.date()}."
1678.             _nice_weekly_title = f"Savaitės dienomis išskaidyta {nice_name} taršos koncentracija
su panaikintu sezoniškumu oro matavimo stotyje: {station}.\nAtkarpa nuo {start.date()} iki
{end.date()}."
1679.             _laikotarpio_label = "Metai"
1680.
1681.             c = name_maps[station]
1682.
1683.             if plottings[0]:
1684.                 plot_seasonal_adjusted(
1685.                     ts,
1686.                     res,
1687.                     c,
1688.                     _seasonal_title,
1689.                     date_col=_laikotarpio_label,
1690.                     value_col=_concentration_value,
1691.                 )
1692.
1693.             if plottings[1]:
1694.                 plot_trend_cycle(
1695.                     ts,
1696.                     res,
1697.                     c,
1698.                     _trend_tilte,
1699.                     date_col=_laikotarpio_label,
1700.                     value_col=_concentration_value,
1701.                 )
1702.
1703.             if plottings[2]:
1704.                 gg_subseries_py(
1705.                     res,
1706.                     color=c,
1707.                     nice_value=nice_value,
1708.                     nice_title=_nice_title,
1709.                     date_label="Metai",
1710.                     figsize=(20, 4),
1711.                 )
1712.
1713.             if plottings[3] and n_cycles >= minimum_cycles:
1714.                 gg_subseries_weekday_daily(
1715.                     res,
1716.                     color=c,
1717.                     nice_konc=_concentration_value,
1718.                     nice_title=_nice_weekly_title,
1719.                     date_label="Savaitės numeris",
1720.                     figsize=(16, 4),
1721.                     minimum_cycles=minimum_cycles,
1722.                 )
1723.             elif plottings[3]:
1724.                 print(
1725.                     f"  {station}·seg{seg_id}: only {n_cycles} weeks (<{minimum_cycles}) →
skipping weekday plot"
1726.                 )
1727.
1728.             if plottings[4] and n_cycles >= minimum_cycles:
1729.                 fig, axes = plt.subplots(4, 1, figsize=(10, 8), sharex=True, dpi=300)
1730.
1731.                 for ax, comp, lbl in zip(axes, comps, labels):
```

```
                    series = getattr(res, comp)
                    ax.plot(
                        series.index, series.values, color=color_map[station], lw=1.3
                    )
                    ax.set_ylabel(lbl)

                axes[-1].set_xlabel(_laikotarpio_label)

                fig.suptitle(
                    sup_t,
                    fontsize=14,
                )
                plt.tight_layout(rect=[0, 0, 1, 0.93])
                plt.show()
            elif plottings[4]:
                print(
                    f"  {station}·seg{seg_id}: only {n_cycles} weeks (<{minimum_cycles}) → skipping final decomposition plot"
                )


def plot_seasonal_adjusted(
    ts, res, color, nice_name="", date_col="Date", value_col="Value", figsize=(10, 4)
):
    sa = ts - res.seasonal

    plt.figure(figsize=figsize, dpi=300)
    plt.plot(ts.index, ts.values, color="grey", linewidth=1, label="Stebėti duomenys")
    plt.plot(
        sa.index,
        sa.values,
        color=color,
        linewidth=2,
        label="Pagal sezoniškumą pakoreguoti duomenys",
    )
    plt.title(nice_name, fontsize=12)
    plt.xlabel(date_col)
    plt.ylabel(value_col)
    plt.legend()
    plt.tight_layout()
    plt.show()


def plot_trend_cycle(
    ts,
    res,
    color,
    nice_name="",
    date_col="Date",
    value_col="Value",
    figsize=(10, 4),
):
    tc = res.trend  # + res.seasonal
    # tc = ts - res.seasonal

    plt.figure(figsize=figsize, dpi=300)
    plt.plot(ts.index, ts.values, color="grey", linewidth=1, label="Stebėti duomenys")
    plt.plot(tc.index, tc.values, color=color, linewidth=2, label="Tendencija")
    plt.title(nice_name, fontsize=12)
    plt.xlabel(date_col)
    plt.ylabel(value_col)
    plt.legend()
    plt.tight_layout()
    plt.show()


def gg_subseries_py(
    res,
    color,
    nice_value,
    nice_title,
    date_label: str = "Year",
    figsize=(20, 4),
```

```
):
    ts = res.trend + res.resid

    df = ts.dropna().reset_index(name="seasonal").rename(columns={"index": date_label})
    df["year"] = df[date_label].dt.year
    df["month"] = df[date_label].dt.month
    df["m_abb"] = df["month"].map(lambda m: calendar.month_abbr[m])
    months = list(calendar.month_abbr)[1:]
    fig, axes = plt.subplots(1, 12, figsize=figsize, sharey=True, dpi=300)
    for ax, m in zip(axes, months):
        sub = df[df["m_abb"] == m]
        if sub.empty:
            ax.set_visible(False)
            continue

        annual = sub.groupby("year")["seasonal"].mean().reset_index()
        years = annual["year"].values
        vals = annual["seasonal"].values

        ax.scatter(years, vals, color=color, edgecolor="black", zorder=3)
        ax.plot(years, vals, color=color, linewidth=1.5)
        mean_val = vals.mean()
        ax.hlines(
            mean_val, xmin=years.min(), xmax=years.max(), color="blue", linewidth=2
        )

        ax.set_title(m)
        ax.set_xticks(years)
        ax.set_xticklabels(years, rotation=90, fontsize=8)

        if ax is not axes[0]:
            ax.tick_params(labelleft=False)

    nice_konc = "Koncentracija, " + nice_value
    fig.suptitle(
        nice_title,
        fontsize=16,
        y=1.02,
    )
    fig.supxlabel(date_label, fontsize=12)
    fig.supylabel(nice_konc, fontsize=12, x=0.02)
    fig.subplots_adjust(left=0.04)
    plt.tight_layout(rect=[0, 0, 1, 0.93])
    plt.show()


def gg_subseries_weekday_daily(
    res,
    color,
    nice_konc: str,
    nice_title: str,
    date_label: str = "Savaitė",
    figsize=(14, 4),
    minimum_cycles: int = 0,
    tick_step: int = 5,
):
    ts = res.seasonal  # res.trend + res.resid
    df = ts.dropna().reset_index(name="seasonal").rename(columns={"index": "date"})
    df["weekday"] = df["date"].dt.weekday
    lt_days = [
        "Pirmadienis",
        "Antradienis",
        "Trečiadienis",
        "Ketvirtadienis",
        "Penktadienis",
        "Šeštadienis",
        "Sekmadienis",
    ]
    df["wd_lt"] = df["weekday"].map(lambda d: lt_days[d])
    df["week"] = df["date"].dt.isocalendar().week

    fig, axes = plt.subplots(1, 7, figsize=figsize, sharey=True, dpi=300)

```

```
    for ax, lt_day in zip(axes, lt_days):
        sub = df[df["wd_lt"] == lt_day]
        n_cycles = sub["week"].nunique()
        if n_cycles < minimum_cycles or sub.empty:
            ax.set_visible(False)
            continue

        ax.scatter(
            sub["week"], sub["seasonal"], color=color, alpha=0.9, s=20, edgecolor="none"
        )

        med = sub.groupby("week")["seasonal"].median().reset_index()
        ax.plot(med["week"], med["seasonal"], color=color, linewidth=2)
        mean_val = sub["seasonal"].mean()
        ax.hlines(
            mean_val,
            xmin=med["week"].min(),
            xmax=med["week"].max(),
            color="blue",
            linestyle="--",
            linewidth=1,
        )

        ax.set_title(lt_day)
        weeks = sorted(med["week"].unique())
        tick_locs = weeks[::tick_step]
        ax.set_xticks(tick_locs)
        ax.set_xticklabels(tick_locs, rotation=90, fontsize=8)

        if ax is not axes[0]:
            ax.tick_params(labelleft=False)

    fig.suptitle(nice_title, fontsize=16, y=1.02)
    fig.supxlabel(date_label, fontsize=12)
    fig.supylabel(nice_konc, fontsize=12, x=0.02)
    fig.subplots_adjust(left=0.04)
    plt.tight_layout(rect=[0, 0, 1, 0.93])
    plt.show()


def run_decomp(
    df,
    pollutant,
    nice_name,
    nice_value,
    period,
    max_interp,
    breakdays,
    plottings,
    minimum_cycles,
    name_maps,
    color_map,
):
    df_in = df.dropna(subset=[pollutant])
    plot_decomp_per_station_segments(
        daily_df=df_in,
        value_col=pollutant,
        nice_name=nice_name,
        nice_value=nice_value,
        date_col="ldatetime",
        group_col="stat_num",
        name_maps=name_maps,
        color_map=color_map,
        period=period,
        max_interp_frac=max_interp,
        break_days=breakdays,
        plottings=plottings,
        minimum_cycles=minimum_cycles,
    )


# skipped loading quantity_units_df, averages_df, units_df, quantity_df, station_df

```

```
df_prepared = prepare_dataframe(
    quantity_units_df, averages_df, units_df, quantity_df, station_df
)

df_clean, pres_stats, temp_stats = prepare_statistical_dataframe(df_prepared)

pollutants = [
    "PM25, ug/m3",
    "PM10, ug/m3",
    "NO2, ug/m3",
    "SO2, ug/m3",
    "CO, mg/m3",
    "O3, ug/m3",
]

name_maps = {
    "4": "Vilnius, Savanorių pr.",
    "2": "Vilnius, Lazdynai",
    "41": "Kaunas, Petrašiūnai",
    "31": "Klaipėda, Centras",
    "1": "Vilnius, Senamiestis",
    "3": "Vilnius, Žirmūnai",
    "42": "Jonava",
    "43": "Kėdainiai",
    "21": "Naujoji Akmenė",
    "22": "Šiauliai",
    "23": "Mažeikiai",
    "33": "Klaipėda, Šilutės plentas",
    "12": "Panevėžys",
    "53": "Žemaitija",
    "52": "Dzūkija",
    "51": "Aukštaitija",
    "44": "Kaunas, Noreikiškės",
}

palette17 = get_gnuplot_colors(17)

color_maps = {
    "Vilnius, Savanorių pr.": palette17[0],
    "Vilnius, Lazdynai": palette17[1],
    "Kaunas, Petrašiūnai": palette17[2],
    "Klaipėda, Centras": palette17[3],
    "Vilnius, Senamiestis": palette17[4],
    "Vilnius, Žirmūnai": palette17[5],
    "Jonava": palette17[6],
    "Kėdainiai": palette17[7],
    "Naujoji Akmenė": palette17[8],
    "Šiauliai": palette17[9],
    "Mažeikiai": palette17[10],
    "Klaipėda, Šilutės plentas": palette17[11],
    "Panevėžys": palette17[12],
    "Žemaitija": palette17[13],
    "Dzūkija": palette17[14],
    "Aukštaitija": palette17[15],
    "Kaunas, Noreikiškės": palette17[16],
}

df_clean_dropped = df_clean.dropna(
    subset=[
        "PM25, ug/m3",
        "PM10, ug/m3",
        "NO2, ug/m3",
        "SO2, ug/m3",
        "CO, mg/m3",
        "O3, ug/m3",
    ],
    how="all",
)

df_clean_dropped = df_clean_dropped.replace({"stat_num": name_maps})

"""
thresholds = {
```

```
    "PM25, ug/m3": {"AQG": 15.0, "IT1": 75.0, "IT2": 50.0, "IT3": 37.5, "IT4": 25.0},
    "PM10, ug/m3": {"AQG": 45.0, "IT1": 150.0, "IT2": 100.0, "IT3": 75.0, "IT4": 50.0},
    "NO2, ug/m3": {"AQG": 25.0, "IT1": 120.0, "IT2": 50.0, "IT3": 50.0, "IT4": 50.0},
    "SO2, ug/m3": {"AQG": 40.0, "IT1": 125.0, "IT2": 50.0, "IT3": 50.0, "IT4": 50.0},
    "CO, mg/m3": {"AQG": 4.0, "IT1": 7.0, "IT2": 7.0, "IT3": 7.0, "IT4": 7.0},
    "O3, ug/m3": {"AQG": 4.0, "IT1": 7.0, "IT2": 7.0, "IT3": 7.0, "IT4": 7.0},
}

df_summary = make_all_station_summaries(df_clean_dropped, thresholds)
"""


df_clean_dropped[pollutants] = df_clean_dropped[pollutants].clip(lower=0)

df_clean_dropped.drop(columns=["latitude", "longitude"], inplace=True)

aggs = make_all_aggregates(df_clean_dropped, cols=pollutants)
daily = aggs["daily_df"]
monthly = aggs["monthly_df"]
yearly = aggs["yearly_df"]
weekday = aggs["weekday_df"]
half_hour = aggs["half_hour_df"]
year = aggs["year_df"]

_subset = [
    "PM25, ug/m3",
    "PM10, ug/m3",
    "NO2, ug/m3",
    "SO2, ug/m3",
    "CO, mg/m3",
    "O3, ug/m3",
]

df_for_corr = monthly
print("Monthly")
for i in range(5):
    for j in range(i, 5):
        ii = _subset[i]
        jj = _subset[j + 1]
        temp_subset = [ii, jj]
        _p = (
            df_for_corr[temp_subset].dropna(how="any").corr(method="pearson").iloc[0][1]
        )
        _s = (
            df_for_corr[temp_subset]
            .dropna(how="any")
            .corr(method="spearman")
            .iloc[0][1]
        )
        _k = (
            df_for_corr[temp_subset].dropna(how="any").corr(method="kendall").iloc[0][1]
        )
        print(f"Koreliacija tarp {ii} ir {jj}: {round(_s,4)}")

df["ldatetime"] = pd.to_datetime(df["ldatetime"])
df["month"] = df["ldatetime"].dt.month


def get_monthly_plot(pollutant):
    df = monthly
    df["ldatetime"] = pd.to_datetime(df["ldatetime"])
    df["month"] = df["ldatetime"].dt.month
    valid_df = df[df[pollutant].notna()]
    month_counts = valid_df.groupby("stat_num")["month"].nunique()
    valid_stat_nums = month_counts[month_counts == 12].index
    filtered_df = df[df["stat_num"].isin(valid_stat_nums)].copy()
    monthly_medians = filtered_df.groupby("month")[pollutant].median()
    max_median = monthly_medians.max()
    ylim_top = math.floor(max_median * 2 / 10) * 10
    fig, axes = plt.subplots(1, 12, figsize=(36, 4), sharey=True, dpi=300)

    for month in range(1, 13):
        ax = axes[month - 1]
```

```
        monthly_data = filtered_df[filtered_df["month"] == month]

        for stat, group in monthly_data.groupby("stat_num"):
            y = group[pollutant]
            x = group["ldatetime"]
            mask = y.notna()
            if mask.sum() > 0:
                ax.scatter(x[mask], y[mask], color="black", s=20)

        valid_vals = monthly_data[pollutant].dropna()
        if not valid_vals.empty:
            median_val = valid_vals.median()
            ax.axhline(median_val, color="blue", linestyle="--", linewidth=2)

        ax.set_ylim(top=ylim_top)
        ax.set_title(pd.to_datetime(f"2024-{month:02d}-01").strftime("%b"))
        ax.tick_params(axis="x", rotation=45)

    fig.suptitle(
        f"Visos teritorijos kasmėnesiniai teršalo {pollutant} matavimai ir jų medianos.",
        fontsize=16,
    )

    plt.tight_layout()
    plt.subplots_adjust(top=0.85, bottom=0.25)
    plt.show()


for i in _subset:
    get_monthly_plot(i)

data = {
    ("PM2.5", "Ilgalaikis"): {"IT1": 35, "IT2": 25, "IT3": 15, "IT4": 10, "AQG": 5},
    ("PM2.5", "Trumpalaikis"): {
        "IT1": 75,
        "IT2": 50,
        "IT3": 37.5,
        "IT4": 25,
        "AQG": 15,
    },
    ("PM10", "Ilgalaikis"): {"IT1": 70, "IT2": 50, "IT3": 30, "IT4": 20, "AQG": 15},
    ("PM10", "Trumpalaikis"): {"IT1": 150, "IT2": 100, "IT3": 75, "IT4": 50, "AQG": 45},
    ("O3", "Ilgalaikis"): {"IT1": 100, "IT2": 70, "AQG": 60},
    ("O3", "Trumpalaikis"): {"IT1": 160, "IT2": 120, "AQG": 100},
    ("NO2", "Ilgalaikis"): {"IT1": 40, "IT2": 30, "IT3": 20, "AQG": 10},
    ("NO2", "Trumpalaikis"): {"IT1": 120, "IT2": 50, "AQG": 25},
    ("SO2", "Trumpalaikis"): {"IT1": 125, "IT2": 50, "AQG": 40},
    ("CO", "Trumpalaikis"): {"IT1": 7, "AQG": 4},
}

thresholds_df = pd.DataFrame(data).T
thresholds_df.index.names = ["Pollutant", "Period"]

col_map = {
    "PM25, ug/m3": ("PM2.5", "Trumpalaikis", "Ilgalaikis"),
    "PM10, ug/m3": ("PM10", "Trumpalaikis", "Ilgalaikis"),
    "NO2, ug/m3": ("NO2", "Trumpalaikis", "Ilgalaikis"),
    "SO2, ug/m3": ("SO2", "Trumpalaikis", None),
    "CO, mg/m3": ("CO", "Trumpalaikis", None),
    "O3, ug/m3": ("O3", "Trumpalaikis", "Ilgalaikis"),
}

colors = ["green", "yellow", "orange", "red", "darkred", "black"]


def classify_value(value, thresholds):
    if pd.isna(value):
        return None
    aqg = thresholds.get("AQG", float("-inf"))
    it1 = thresholds.get("IT1", float("inf"))
    it2 = thresholds.get("IT2", float("inf"))
    it3 = thresholds.get("IT3", float("inf"))
    it4 = thresholds.get("IT4", float("inf"))
```

```
2169.
2170.     if value <= aqg:
2171.         return 0
2172.     if "IT4" in thresholds:
2173.         if value <= it4:
2174.             return 1
2175.         elif value <= it3:
2176.             return 2
2177.         elif value <= it2:
2178.             return 3
2179.         elif value <= it1:
2180.             return 4
2181.         else:
2182.             return 5
2183.     elif "IT3" in thresholds:
2184.         if value <= it3:
2185.             return 1
2186.         elif value <= it2:
2187.             return 2
2188.         elif value <= it1:
2189.             return 3
2190.         else:
2191.             return 4
2192.     elif "IT2" in thresholds:
2193.         if value <= it2:
2194.             return 1
2195.         elif value <= it1:
2196.             return 2
2197.         else:
2198.             return 3
2199.     else:
2200.         if value <= it1:
2201.             return 1
2202.         else:
2203.             return 2
2204.
2205.
2206. def safe_int(value):
2207.     try:
2208.         return int(value)
2209.     except (ValueError, TypeError):
2210.         return None
2211.
2212.
2213. def generate_chart_final_polish_fixed(
2214.     df,
2215.     column,
2216.     thresholds_short,
2217.     thresholds_long=None,
2218.     special_case=None,
2219.     group_dict=None,
2220.     name_map=None,
2221. ):
2222.     df["ldatetime"] = pd.to_datetime(df["ldatetime"])
2223.     short_term = df[["ldatetime", "stat_num", column]].dropna()
2224.     short_term["classification"] = short_term[column].apply(
2225.         lambda x: classify_value(x, thresholds_short)
2226.     )
2227.     short_counts = (
2228.         short_term.groupby(["stat_num", "classification"]).size().unstack(fill_value=0)
2229.     )
2230.     short_totals = short_counts.sum(axis=1)
2231.     short_percentages = short_counts.div(short_totals, axis=0) * 100
2232.     short_percentages = short_percentages.dropna(how="all")
2233.     df["year"] = df["ldatetime"].dt.year
2234.     long_term = df[["ldatetime", "year", "stat_num", column]].dropna()
2235.
2236.     if thresholds_long:
2237.         if special_case == "O3":
2238.             long_term["month"] = (
2239.                 long_term["ldatetime"].dt.to_period("M").dt.to_timestamp()
2240.             )
2241.             monthly_avg = (
```

```
                long_term.groupby(["stat_num", "year", "month"])[column]
                .mean()
                .reset_index()
            )
            long_term_max = []
            for (stat, year), group in monthly_avg.groupby(["stat_num", "year"]):
                group = group.set_index("month").resample("D").interpolate()
                max_6m = group[column].rolling("180D").mean().max()
                if pd.notna(max_6m):
                    long_term_max.append((stat, year, max_6m))
            long_df = pd.DataFrame(long_term_max, columns=["stat_num", "year", "value"])
        else:
            long_avg = (
                long_term.groupby(["stat_num", "year"])[column].mean().reset_index()
            )
            long_avg = long_avg.rename(columns={column: "value"})
            long_df = long_avg

        long_df["classification"] = long_df["value"].apply(
            lambda x: classify_value(x, thresholds_long)
        )
        per_station = pd.get_dummies(long_df["classification"])
        per_station["stat_num"] = long_df["stat_num"].values
        station_grouped = per_station.groupby("stat_num").sum()
        station_grouped_percent = (
            station_grouped.div(station_grouped.sum(axis=1), axis=0) * 100
        )
        station_grouped_percent = station_grouped_percent.dropna(how="all")

    level_names = {
        0: "Iki AQG",
        1: "Viršija AQG",
        2: "Viršija IT4",
        3: "Viršija IT3",
        4: "Viršija IT2",
        5: "Viršija IT1",
    }
    short_percentages.columns = [
        level_names.get(c, c) for c in short_percentages.columns
    ]
    if thresholds_long:
        station_grouped_percent.columns = [
            level_names.get(c, c) for c in station_grouped_percent.columns
        ]

    nice_name = r"$O_{3}$"
    flag = True
    group_order, group_labels, group_display = [], [], []
    for group_name, stat_list in group_dict.items():
        for stat in stat_list:
            stat_str = str(stat)
            mapped = name_map.get(stat_str, stat_str) if name_map else stat_str
            if stat_str in short_percentages.index or (
                thresholds_long and stat_str in station_grouped_percent.index
            ):
                group_order.append(stat_str)
                group_labels.append(group_name)
                group_display.append(mapped)

    short_percentages.index = short_percentages.index.astype(str)
    short_percentages = short_percentages.loc[
        short_percentages.index.intersection(group_order)
    ]
    short_percentages = short_percentages.reindex(group_order)
    short_percentages.index = group_display

    short_totals = short_counts.sum(axis=1).reindex(group_order)
    short_totals.index = group_display

    if thresholds_long:
        station_grouped_percent.index = station_grouped_percent.index.astype(str)
        station_grouped_percent = station_grouped_percent.loc[
            station_grouped_percent.index.intersection(group_order)
```

```
        ]
        station_grouped_percent = station_grouped_percent.reindex(group_order)
        station_grouped_percent.index = group_display
        year_counts = long_df.groupby("stat_num").size().reindex(group_order)
        year_counts.index = group_display

    label_positions = {}
    for i, label in enumerate(group_labels):
        if label not in label_positions:
            label_positions[label] = i
    group_boundaries = [
        i for i in range(1, len(group_labels)) if group_labels[i] != group_labels[i - 1]
    ]

    fig, ax = plt.subplots(figsize=(14, 8))
    short_percentages.plot(
        kind="barh", stacked=True, ax=ax, color=colors[: short_percentages.shape[1]]
    )
    print_group_level_proportions(
        short_percentages, group_dict, name_map, "Trumpalaikės klasifikacijos grupėse"
    )
    if flag:
        title = f"Trumpalaikė tarša pagal PSO normas, rodiklis: " + nice_name
    else:
        title = f"Trumpalaikė tarša pagal PSO normas, rodiklis: {column}"

    print(f"Trumpalaikė tarša pagal PSO normas, rodiklis: {column}")
    ax.set_title(title)
    ax.set_xlabel("Procentai matuotų dienų")
    ax.set_ylabel("")
    for i, label in enumerate(short_percentages.index):
        count = short_totals.get(label)
        if pd.notna(count):
            ax.text(101, i, f"{int(count)}", va="center", fontsize=9)
    ax.set_xlim(0, 110)
    ax.legend(title="Lygis", bbox_to_anchor=(1.05, 1), loc="upper left")
    for pos in group_boundaries:
        ax.axhline(pos - 0.5, color="black", linewidth=2)
    for group, start_index in label_positions.items():
        ax.text(
            -20,
            start_index,
            group,
            va="bottom",
            ha="right",
            fontsize=11,
            fontweight="bold",
        )
    plt.tight_layout()
    plt.show()

    if thresholds_long:
        fig2, ax2 = plt.subplots(figsize=(14, 8))
        station_grouped_percent.plot(
            kind="barh",
            stacked=True,
            ax=ax2,
            color=colors[: station_grouped_percent.shape[1]],
        )
        print_group_level_proportions(
            station_grouped_percent,
            group_dict,
            name_map,
            "Ilgalaikės klasifikacijos grupėse",
        )
        if flag:
            title = f"Ilgalaikė tarša pagal PSO normas, rodiklis: " + nice_name
        else:
            title = f"Ilgalaikė tarša pagal PSO normas, rodiklis: {column}"

        print(f"Ilgalaikė tarša pagal PSO normas, rodiklis: {column}")
        ax2.set_title(title)
        ax2.set_xlabel("Procentai matuotų metų")
```

```
        ax2.set_ylabel("")
        for i, label in enumerate(station_grouped_percent.index):
            count = year_counts.get(name_map.get(label, label), year_counts.get(label))
            if pd.notna(count):
                ax2.text(101, i, f"{int(count)}", va="center", fontsize=9)
        ax2.set_xlim(0, 110)
        ax2.legend(title="Lygis", bbox_to_anchor=(1.05, 1), loc="upper left")
        for pos in group_boundaries:
            ax2.axhline(pos - 0.5, color="black", linewidth=2)
        for group, start_index in label_positions.items():
            ax2.text(
                -20,
                start_index,
                group,
                va="bottom",
                ha="right",
                fontsize=11,
                fontweight="bold",
            )
        plt.tight_layout()
        plt.show()


def print_group_level_proportions(dataframe, group_dict, name_map, title):
    print(f"\n{title}")
    reverse_name_map = {v: k for k, v in name_map.items()} if name_map else {}

    for group, stat_list in group_dict.items():
        mapped_stats = []
        for stat in stat_list:
            stat_str = str(stat)
            stat_name = name_map.get(stat_str, stat_str) if name_map else stat_str
            if stat_name in dataframe.index:
                mapped_stats.append(stat_name)

        if not mapped_stats:
            continue

        group_df = dataframe.loc[mapped_stats]
        proportions = group_df.sum().div(group_df.sum().sum()).round(2).to_dict()
        formatted = ", ".join(
            [f"{level}: {value}" for level, value in proportions.items()]
        )
        print(f"{group}: {formatted}")


place_maps = {
    "Didmiestis": ["4", "2", "41", "31", "1", "3", "33", "44"],
    "Miestas": ["42", "43", "21", "22", "23", "12"],
    "Kaimas": ["53", "52", "51"],
}

name_maps = {
    "4": "Vilnius, Savanorių pr.",
    "2": "Vilnius, Lazdynai",
    "41": "Kaunas, Petrašiūnai",
    "31": "Klaipėda, Centras",
    "1": "Vilnius, Senamiestis",
    "3": "Vilnius, Žirmūnai",
    "42": "Jonava",
    "43": "Kėdainiai",
    "21": "Naujoji Akmenė",
    "22": "Šiauliai",
    "23": "Mažeikiai",
    "33": "Klaipėda, Šilutės plentas",
    "12": "Panevėžys",
    "53": "Žemaitija",
    "52": "Dzūkija",
    "51": "Aukštaitija",
    "44": "Kaunas, Noreikiškės",
}

for col, (pollutant, short, long) in col_map.items():
```

```
    thresholds_short = thresholds_df.loc[(pollutant, short)].dropna().to_dict()
    thresholds_long = (
        thresholds_df.loc[(pollutant, long)].dropna().to_dict() if long else None
    )
    generate_chart_final_polish_fixed(
        daily,
        col,
        thresholds_short,
        thresholds_long,
        special_case=None,
        group_dict=place_maps,
        name_map=name_maps,
    )

generate_chart_final_polish_fixed(
    daily,
    "O3 (8 val.), ug/m3",
    thresholds_short,
    thresholds_long,
    special_case="O3",
    group_dict=place_maps,
    name_map=name_maps,
)

run_decomp(
    df=daily,
    pollutant="NO2, ug/m3",
    nice_name=r"$NO_{2}$",
    nice_value=r"$\mu g$/$m^3$",
    period=365,
    max_interp=0.1,
    breakdays=1,
    plottings=[False, False, False, True, True],
    minimum_cycles=30,
    name_maps=name_maps,
)

_, short = prepare_medicinos_duomenys()
bendri = prepare_final_dfs()
metiniai = pd.concat([short, year, bendri], axis=1, join="outer")


def correlation_with_pvalues(df, past_cols, present_cols, method="pearson"):
    relevant_cols = past_cols + present_cols
    df_subset = df[relevant_cols].dropna()

    if method == "pearson":
        corr_func = pearsonr
    elif method == "spearman":
        corr_func = spearmanr
    elif method == "kendall":
        corr_func = kendalltau

    n_past = len(past_cols)
    n_present = len(present_cols)

    corr_matrix = np.zeros((n_past, n_present))
    pval_matrix = np.zeros((n_past, n_present))

    for i, past_col in enumerate(past_cols):
        for j, present_col in enumerate(present_cols):
            if len(df_subset[[past_col, present_col]].dropna()) > 2:
                corr, pval = corr_func(
                    df_subset[past_col].dropna(), df_subset[present_col].dropna()
                )
                corr_matrix[i, j] = corr
                pval_matrix[i, j] = pval
            else:
                corr_matrix[i, j] = np.nan
                pval_matrix[i, j] = np.nan

    corr_df = pd.DataFrame(corr_matrix, index=past_cols, columns=present_cols)
    pval_df = pd.DataFrame(pval_matrix, index=past_cols, columns=present_cols)
```

```
2534.
2535.     return corr_df, pval_df
2536.
2537.
2538. def comprehensive_relationship_analysis(df, past_cols, present_cols):
2539.     results = {}
2540.     print("=== PEARSON CORRELATION ===")
2541.     pearson_corr, pearson_pval = correlation_with_pvalues(
2542.         df, past_cols, present_cols, "pearson"
2543.     )
2544.     results["pearson_corr"] = pearson_corr
2545.     results["pearson_pval"] = pearson_pval
2546.
2547.     print("Correlation Matrix:")
2548.     print(pearson_corr.round(4))
2549.     print("\nP-values Matrix:")
2550.     print(pearson_pval.round(4))
2551.
2552.     print("\n=== SPEARMAN CORRELATION ===")
2553.     spearman_corr, spearman_pval = correlation_with_pvalues(
2554.         df, past_cols, present_cols, "spearman"
2555.     )
2556.     results["spearman_corr"] = spearman_corr
2557.     results["spearman_pval"] = spearman_pval
2558.
2559.     print("Correlation Matrix:")
2560.     print(spearman_corr.round(4))
2561.     print("\nP-values Matrix:")
2562.     print(spearman_pval.round(4))
2563.
2564.     print("\n=== KENDALL'S TAU ===")
2565.     kendall_corr, kendall_pval = correlation_with_pvalues(
2566.         df, past_cols, present_cols, "kendall"
2567.     )
2568.     results["kendall_corr"] = kendall_corr
2569.     results["kendall_pval"] = kendall_pval
2570.
2571.     print("Correlation Matrix:")
2572.     print(kendall_corr.round(4))
2573.     print("\nP-values Matrix:")
2574.     print(kendall_pval.round(4))
2575.
2576.     return results
2577.
2578.
2579. def mutual_information_analysis(df, past_cols, present_cols):
2580.     print("\n=== MUTUAL INFORMATION ANALYSIS ===")
2581.     df_clean = df[past_cols + present_cols].dropna()
2582.
2583.     mi_matrix = np.zeros((len(past_cols), len(present_cols)))
2584.
2585.     for i, past_col in enumerate(past_cols):
2586.         X = df_clean[past_col].values.reshape(-1, 1)
2587.         for j, present_col in enumerate(present_cols):
2588.             y = df_clean[present_col].values
2589.             mi_score = mutual_info_regression(X, y, random_state=42)[0]
2590.             mi_matrix[i, j] = mi_score
2591.
2592.     mi_df = pd.DataFrame(mi_matrix, index=past_cols, columns=present_cols)
2593.     print("Mutual Information Matrix:")
2594.     print(mi_df.round(4))
2595.
2596.     return mi_df
2597.
2598.
2599. def regression_analysis(df, past_cols, present_cols):
2600.     print("\n=== REGRESSION ANALYSIS ===")
2601.
2602.     regression_results = []
2603.
2604.     for past_col in past_cols:
2605.         for present_col in present_cols:
2606.             data = df[[past_col, present_col]].dropna()
```

```
            if len(data) < 5:
                continue

            X = data[past_col].values.reshape(-1, 1)
            y = data[present_col].values
            reg = LinearRegression()
            reg.fit(X, y)
            y_pred = reg.predict(X)
            r2 = r2_score(y, y_pred)
            n = len(y)
            f_stat = (r2 / (1 - r2)) * (n - 2)
            f_pval = 1 - stats.f.cdf(f_stat, 1, n - 2)

            if f_pval < 0.005:
                regression_results.append(
                    {
                        "past_variable": past_col,
                        "present_variable": present_col,
                        "r_squared": r2,
                        "coefficient": reg.coef_[0],
                        "intercept": reg.intercept_,
                        "f_statistic": f_stat,
                        "f_pvalue": f_pval,
                        "n_observations": n,
                    }
                )

    reg_df = pd.DataFrame(regression_results)
    print("Regression Results:")
    print(
        reg_df[
            [
                "past_variable",
                "present_variable",
                "r_squared",
                "coefficient",
                "f_pvalue",
            ]
        ].round(4)
    )

    return reg_df


def granger_causality_test(df, past_cols, present_cols, maxlag=4):
    """
    Granger Causality Test (requires time series data)
    """
    print("\n=== GRANGER CAUSALITY TEST ===")
    print("Note: This test requires time series data with proper temporal ordering")

    granger_results = []

    for past_col in past_cols:
        for present_col in present_cols:
            try:
                data = df[[present_col, past_col]].dropna()
                if len(data) < maxlag * 3:
                    continue

                gc_result = grangercausalitytests(data, maxlag=maxlag, verbose=False)
                for lag in range(1, maxlag + 1):
                    if lag in gc_result:
                        f_pval = gc_result[lag][0]["ssr_ftest"][1]
                        if f_pval < 0.05:
                            granger_results.append(
                                {
                                    "past_variable": past_col,
                                    "present_variable": present_col,
                                    "lag": lag,
                                    "f_pvalue": f_pval,
                                    "significant": f_pval < 0.005,
                                }
```

```
                            )

            except Exception as e:
                print(
                    f"Could not perform Granger test for {past_col} -> {present_col}: {str(e)}"
                )

    if granger_results:
        granger_df = pd.DataFrame(granger_results)
        print("Granger Causality Results:")
        print(
            granger_df[
                ["past_variable", "present_variable", "lag", "f_pvalue", "significant"]
            ]
        )
        return granger_df
    else:
        print("No Granger causality tests could be performed")
        return None


def create_significance_summary(corr_results, alpha=0.005):
    """
    Create a summary of statistically significant relationships
    """
    print(f"\n=== SIGNIFICANCE SUMMARY (α = {alpha}) ===")

    methods = ["pearson", "spearman", "kendall"]

    for method in methods:
        if f"{method}_pval" in corr_results:
            pval_df = corr_results[f"{method}_pval"]
            corr_df = corr_results[f"{method}_corr"]

            # Find significant relationships
            significant_mask = pval_df < alpha

            print(f"\n{method.upper()} - Significant relationships:")
            for past_col in pval_df.index:
                for present_col in pval_df.columns:
                    if significant_mask.loc[past_col, present_col]:
                        corr_val = corr_df.loc[past_col, present_col]
                        pval = pval_df.loc[past_col, present_col]
                        print(
                            f"  {past_col} -> {present_col}: r={corr_val:.4f}, p={pval:.4f}"
                        )


def plot_correlation_heatmaps(corr_results, figsize=(15, 5)):
    methods = ["pearson", "spearman", "kendall"]
    fig, axes = plt.subplots(1, 3, figsize=figsize)

    for i, method in enumerate(methods):
        if f"{method}_corr" in corr_results:
            corr_df = corr_results[f"{method}_corr"]
            sns.heatmap(
                corr_df,
                annot=True,
                cmap="RdBu_r",
                center=0,
                ax=axes[i],
                fmt=".3f",
                cbar_kws={"shrink": 0.8},
            )
            axes[i].set_title(f"{method.upper()} Correlation")
            axes[i].set_xlabel("Present Variables")
            axes[i].set_ylabel("Past Variables")

    plt.tight_layout()
    plt.show()


# Example usage function
```

```
2753. def analyze_past_present_relationships(df, past_cols, present_cols):
2754.     print("COMPREHENSIVE ANALYSIS: PAST vs PRESENT VARIABLES")
2755.     print("=" * 60)
2756.     corr_results = comprehensive_relationship_analysis(df, past_cols, present_cols)
2757.     mi_results = mutual_information_analysis(df, past_cols, present_cols)
2758.     reg_results = regression_analysis(df, past_cols, present_cols)
2759.     # granger_results = granger_causality_test(df, past_cols, present_cols)
2760.     create_significance_summary(corr_results)
2761.     # plot_correlation_heatmaps(corr_results)
2762.     return {
2763.         "correlations": corr_results,
2764.         "mutual_information": mi_results,
2765.         "regression": reg_results,
2766.         # "granger": granger_results,
2767.     }
2768.
2769.
2770. def plot_regression_analysis(df, past_col, present_col, figsize=(12, 8), time_col=None):
2771.     data = df[[past_col, present_col]].dropna()
2772.     print(data)
2773.
2774.     if len(data) < 5:
2775.         print(f"Nepakanka duomenų taškų ({len(data)}) regresinei analizei")
2776.         return None
2777.
2778.     X = data[past_col].values
2779.     y = data[present_col].values
2780.     X_reshaped = X.reshape(-1, 1)
2781.     reg = LinearRegression()
2782.     reg.fit(X_reshaped, y)
2783.     X_range = np.linspace(X.min(), X.max(), 100)
2784.     y_pred_range = reg.predict(X_range.reshape(-1, 1))
2785.     y_pred = reg.predict(X_reshaped)
2786.     r2 = r2_score(y, y_pred)
2787.     pearson_corr, pearson_pval = spearmanr(X, y)
2788.     residuals = y - y_pred
2789.     n = len(y)
2790.     f_stat = (r2 / (1 - r2)) * (n - 2) if r2 != 1 else np.inf
2791.     f_pval = 1 - stats.f.cdf(f_stat, 1, n - 2) if f_stat != np.inf else 0
2792.     mse = np.mean(residuals**2)
2793.     se_coef = np.sqrt(mse / np.sum((X - np.mean(X)) ** 2))
2794.     t_stat = reg.coef_[0] / se_coef
2795.     t_pval = 2 * (1 - stats.t.cdf(abs(t_stat), n - 2))
2796.     fig = plt.figure(figsize=figsize, dpi=300)
2797.     gs = fig.add_gridspec(3, 3, hspace=0.4, wspace=0.3, height_ratios=[1, 1, 1])
2798.     ax_time = fig.add_subplot(gs[0, :])
2799.
2800.     if time_col and time_col in df.columns:
2801.         time_data = df[time_col]
2802.         time_label = time_col
2803.     else:
2804.         time_data = df.index
2805.         time_label = "Metai"
2806.
2807.     past_series = df[past_col]
2808.     present_series = df[present_col]
2809.     color1 = "tab:blue"
2810.     color2 = "tab:red"
2811.
2812.     ax_time.set_xlabel(f"{time_label}", fontsize=10)
2813.     ax_time.set_ylabel(f"{past_col}", color=color1, fontsize=10)
2814.     line1 = ax_time.plot(
2815.         time_data,
2816.         past_series,
2817.         color=color1,
2818.         marker="o",
2819.         markersize=4,
2820.         linewidth=2,
2821.         alpha=0.7,
2822.         label=past_col,
2823.     )
2824.     ax_time.tick_params(axis="y", labelcolor=color1)
2825.     ax_time.grid(True, alpha=0.3)
```

```
    ax_time2 = ax_time.twinx()
    ax_time2.set_ylabel(f"{present_col}", color=color2, fontsize=12)
    line2 = ax_time2.plot(
        time_data,
        present_series,
        color=color2,
        marker="s",
        markersize=4,
        linewidth=2,
        alpha=0.7,
        label=present_col,
    )
    ax_time2.tick_params(axis="y", labelcolor=color2)
    valid_mask = ~(past_series.isna() | present_series.isna())
    if valid_mask.sum() > 0:
        ax_time.scatter(
            time_data[valid_mask],
            past_series[valid_mask],
            color=color1,
            s=50,
            edgecolors="black",
            linewidth=2,
            zorder=5,
        )
        ax_time2.scatter(
            time_data[valid_mask],
            present_series[valid_mask],
            color=color2,
            s=50,
            edgecolors="black",
            linewidth=2,
            zorder=5,
        )

    ax_time.set_title(
        f'Laiko eilučių palyginimas: "{past_col}" ir "{present_col}"\n'
        f"Paryškinti koreliacijai naudoti taškai (n={valid_mask.sum()})",
        fontsize=12,
    )
    lines = line1 + line2
    labels = [l.get_label() for l in lines]
    ax_time.legend(lines, labels, loc="upper left")
    ax1 = fig.add_subplot(gs[1, :2])
    ax1.scatter(
        X, y, alpha=0.6, color="steelblue", s=50, edgecolors="white", linewidth=0.5
    )
    ax1.plot(X_range, y_pred_range, color="red", linewidth=2, label="Regresijos tiesė")
    residual_std = np.std(residuals)
    confidence_interval = 2.81 * residual_std
    ax1.fill_between(
        X_range,
        y_pred_range - confidence_interval,
        y_pred_range + confidence_interval,
        alpha=0.2,
        color="red",
    )

    ax1.set_xlabel(f"{past_col}", fontsize=10)
    ax1.set_ylabel(f"{present_col}", fontsize=10)
    ax1.set_title(
        f'Tiesinė regresija tarp "{past_col}" ir "{present_col}"',
        fontsize=12,
    )
    ax1.grid(True, alpha=0.3)
    ax1.legend()
    ax2 = fig.add_subplot(gs[1, 2])
    ax2.scatter(y_pred, residuals, alpha=0.6, color="green", s=40)
    ax2.axhline(y=0, color="red", linestyle="--", alpha=0.8)
    ax2.set_xlabel("Prognozuotos vertės", fontsize=10)
    ax2.set_ylabel("Liekanos", fontsize=10)
    ax2.set_title("Likučių priklausomybė nuo prognozuotų verčių", fontsize=12)
    ax2.grid(True, alpha=0.3)
    ax3 = fig.add_subplot(gs[2, 0])
```

```
    stats.probplot(residuals, dist="norm", plot=ax3)
    ax3.set_title("Kvartilių grafikas (Q-Q)", fontsize=12)
    ax3.set_xlabel("Teoriniai kvantiliai", fontsize=10)
    ax3.set_ylabel("Imties kvantiliai", fontsize=10)
    ax3.grid(True, alpha=0.3)
    ax4 = fig.add_subplot(gs[2, 1])
    ax4.hist(
        residuals,
        bins=min(20, len(residuals) // 3),
        alpha=0.7,
        color="orange",
        edgecolor="black",
    )
    ax4.axvline(x=0, color="red", linestyle="--", alpha=0.8)
    ax4.set_xlabel("Likučiai", fontsize=10)
    ax4.set_ylabel("Dažnis", fontsize=10)
    ax4.set_title("Likučių pasiskirstymo histograma", fontsize=12)
    ax4.grid(True, alpha=0.3)
    ax5 = fig.add_subplot(gs[2, 2])
    ax5.axis("off")
    total_rows = len(df)
    past_available = (~df[past_col].isna()).sum()
    present_available = (~df[present_col].isna()).sum()
    both_available = valid_mask.sum()
    stats_text = f"""APRAŠOMOJI REGRESIJOS STATISTIKA

R-kvadratas: {r2:.4f}

TIESINĖS REGRESIJOS LYGTIS
{present_col} = {reg.coef_[0]:.4f} × {past_col} + {reg.intercept_:.4f}

DUOMENŲ APRAŠYMAS
Bendrai eilučių: {total_rows}
{past_col}: {past_available} ({past_available/total_rows*100:.1f}%)
{present_col}: {present_available} ({present_available/total_rows*100:.1f}%)
Abu kintamieji: {both_available} ({both_available/total_rows*100:.1f}%)
"""

    ax5.text(
        0.05,
        0.95,
        stats_text,
        transform=ax5.transAxes,
        fontsize=9,
        verticalalignment="top",
        bbox=dict(boxstyle="round", facecolor="lightgray", alpha=0.8),
    )

    plt.suptitle(
        f'Visapusiškas ryšių tarp "{past_col}" ir "{present_col}" įvertinimas',
        fontsize=16,
        fontweight="bold",
        y=0.98,
    )

    plt.tight_layout()
    plt.show()
    return {
        "r_squared": r2,
        "correlation": pearson_corr,
        "correlation_pvalue": pearson_pval,
        "coefficient": reg.coef_[0],
        "intercept": reg.intercept_,
        "coefficient_stderr": se_coef,
        "coefficient_tstat": t_stat,
        "coefficient_pvalue": t_pval,
        "f_statistic": f_stat,
        "f_pvalue": f_pval,
        "n_observations": n,
        "residuals": residuals,
        "fitted_values": y_pred,
        "total_rows": total_rows,
        "past_available": past_available,
```

```
        "present_available": present_available,
        "both_available": both_available,
        "data_completeness": both_available / total_rows,
    }


def plot_multiple_regressions(df, past_cols, present_cols, cols_per_row=2):
    n_combinations = len(past_cols) * len(present_cols)
    n_rows = (n_combinations + cols_per_row - 1) // cols_per_row

    fig, axes = plt.subplots(
        n_rows, cols_per_row, figsize=(6 * cols_per_row, 4 * n_rows)
    )
    if n_combinations == 1:
        axes = [axes]
    elif n_rows == 1:
        axes = axes.reshape(1, -1)

    plot_idx = 0
    results_summary = []

    for past_col in past_cols:
        for present_col in present_cols:
            if plot_idx >= n_combinations:
                break

            row = plot_idx // cols_per_row
            col = plot_idx % cols_per_row
            ax = axes[row, col] if n_rows > 1 else axes[col]
            data = df[[past_col, present_col]].dropna()

            if len(data) < 5:
                ax.text(
                    0.5,
                    0.5,
                    f"Insufficient data\n({len(data)} points)",
                    ha="center",
                    va="center",
                    transform=ax.transAxes,
                )
                ax.set_title(f"{past_col} vs {present_col}")
                plot_idx += 1
                continue

            X = data[past_col].values
            y = data[present_col].values
            X_reshaped = X.reshape(-1, 1)
            reg = LinearRegression()
            reg.fit(X_reshaped, y)
            ax.scatter(X, y, alpha=0.6, s=30)
            X_range = np.linspace(X.min(), X.max(), 100)
            y_pred_range = reg.predict(X_range.reshape(-1, 1))
            ax.plot(X_range, y_pred_range, color="red", linewidth=2)
            y_pred = reg.predict(X_reshaped)
            r2 = r2_score(y, y_pred)
            pearson_corr, pearson_pval = spearmanr(X, y)
            ax.set_xlabel(past_col, fontsize=10)
            ax.set_ylabel(present_col, fontsize=10)
            ax.set_title(
                f"{past_col} vs {present_col}\nR² = {r2:.3f}, r = {pearson_corr:.3f}",
                fontsize=11,
            )
            ax.grid(True, alpha=0.3)
            results_summary.append(
                {
                    "past_variable": past_col,
                    "present_variable": present_col,
                    "r_squared": r2,
                    "correlation": pearson_corr,
                    "correlation_pvalue": pearson_pval,
                    "significant": pearson_pval < 0.005,
                }
            )
```

```
3045.
3046.             plot_idx += 1
3047.
3048.     for idx in range(plot_idx, n_rows * cols_per_row):
3049.         row = idx // cols_per_row
3050.         col = idx % cols_per_row
3051.         if n_rows > 1:
3052.             axes[row, col].set_visible(False)
3053.         else:
3054.             if idx < len(axes):
3055.                 axes[idx].set_visible(False)
3056.
3057.     plt.tight_layout()
3058.     plt.show()
3059.     summary_df = pd.DataFrame(results_summary)
3060.     print("\nREGRESSION SUMMARY:")
3061.     print("=" * 80)
3062.     print(
3063.         summary_df[
3064.             [
3065.                 "past_variable",
3066.                 "present_variable",
3067.                 "r_squared",
3068.                 "correlation",
3069.                 "correlation_pvalue",
3070.                 "significant",
3071.             ]
3072.         ]
3073.     )
3074.
3075.     return summary_df
3076.
3077.
3078. past_cols = [
3079.     "PM25, ug/m3",
3080.     "PM10, ug/m3",
3081.     "NO2, ug/m3",
3082.     "SO2, ug/m3",
3083.     "CO, mg/m3",
3084.     "O3, ug/m3",
3085.     "CO metinis, kt",
3086.     "SOX metinis, kt",
3087.     "NOX metinis, kt",
3088.     "PM25 metinis, kt",
3089.     "PM10 metinis, kt",
3090.     "Pb metinis, t",
3091.     "HCB metinis, kg",
3092. ]
3093. present_cols = [
3094.     "Visos mirtys",
3095.     "Navikai",
3096.     "Kraujotakos sistemos ligos",
3097.     "Kvėpavimo sistemos ligos",
3098. ]
3099.
3100. results = analyze_past_present_relationships(metiniai, past_cols, present_cols)
3101.
3102. plot_regression_analysis(metiniai, "CO metinis, kt", "Navikai")
3103.
3104. metiniai_added = metiniai.copy()
3105. metiniai_added.at["2021-01-01", "CO metinis, kt"] = 108.54688
3106. metiniai_added.at["2022-01-01", "CO metinis, kt"] = 101.51633
3107. metiniai_added.at["2021-01-01", "SOX metinis, kt"] = 52.62540
3108. metiniai_added.at["2022-01-01", "SOX metinis, kt"] = 49.45868
3109.
3110. results_added = analyze_past_present_relationships(
3111.     metiniai_added, past_cols, present_cols
3112. )
3113.
3114. plot_regression_analysis(metiniai_added, "CO metinis, kt", "Navikai")
3115.
3116. (
3117.     df_co_ts,
```

```
    df_pm25_ts,
    df_pm10_ts,
    df_nox_ts,
    df_pb_ts,
    df_sox_ts,
    df_hcb_ts,
    df_viso_ts,
) = prepare_metiniai_duomenys()

print("\nOutlier Detection (using IQR per experiment):")
outlier_threshold_upper = {}
outlier_threshold_lower = {}
for exp in df_viso_ts.index.get_level_values("exp_label").unique():
    q1 = df_viso_ts.loc[exp]["exp_value"].quantile(0.25)
    q3 = df_viso_ts.loc[exp]["exp_value"].quantile(0.75)
    iqr = q3 - q1
    upper_bound = q3 + 1.5 * iqr
    lower_bound = q1 - 1.5 * iqr
    outlier_threshold_upper[exp] = upper_bound
    outlier_threshold_lower[exp] = lower_bound
    outliers = df_viso_ts.loc[exp][
        (df_viso_ts.loc[exp]["exp_value"] < lower_bound)
        | (df_viso_ts.loc[exp]["exp_value"] > upper_bound)
    ]
    print(f"\nExperiment: {exp}")
    print("  Number of outliers:", len(outliers))

```