MATAS BRAZYS 1-os Užduoties ataskaita:

Turinys:

Užduotis:

1) Duomenų nuskaitymas vyksta per atskirą klasę. Į šį objektą nurdoma vietovės kodas, API URL. Ši klasė turi turėti du metodus:

a) Istorinių duomenų nuskaitymas už nurodytą laiko intervalą nuo - iki;

b) Prognozės duomenų nuskaitymas;

Abiem atvejais duomenys turėtų būti gražinami pandas. DataFrame formatu, kur indeksas yra laikas (pd.DatetimeIndex) su įvertinta laiko zona;

2) Nuskaičius istorinius duomenis už praeitus metus (laikotarpis nuo šiandien iki metai atgal) suskaičiuoti ir atvaizduoti šiuos rodiklius:

a) Vidutinė metų temperatūra, oro drėgmė;

b) Vidutinė metų dienos, ir nakties temperatūra priimant kad skaičiuojama LT laiko zonoje ir diena yra tarp 08:00 ir 20:00;

c) Kiek savaitgalių (šeštadienis/sekmadienis - 1 savaitgalis) per šį laikotarpį buvo prognozuojama kad lis;

3) Nuskaičius prognozės duomenis juos apjungti su istoriniais. Atvaizduoti grafiką, kuris rodo paskutinės savaitės išmatuotą temperatūrą ir ateinančio periodo prognozuojama temperatūrą.

4) Visi nuskaityti duomenys yra valandiniai. Parašyti funkciją, į kurią padavus temperatūros pandas. Series suskaičiuotų tarpines reikšmes ir pagražintų rezultatą pandas. Series kurio dažnis yra 5 minutės. Tarpines reikšmes interpoliuoti.

Sprendimo eiga:

Naudoti resursai:

- Duomenų šaltinis meteo api https://api.meteo.lt/v1
- Bibliotekos : requests, pandas, time, matplotlib.pyplot

1 dalis:

- Sukurta MeteoReader klasė.
  Jos tikslas – patogiai atsisiųsti bei apdoroti istorinius ir prognozės duomenis iš Lietuvos hidrometeorologijos tarnybos API (meteo.lt).

- Inicializavimas (__init__).
  Kadangi API užklausos istoriniams duomenims ir prognozei šiek tiek skiriasi, naudojami du atskiri parametrai:
  - station_code – naudojamas dirbant su meteorologinėmis stotimis ir istoriniais duomenimis. Pvz., Vilniaus stoties kodas yra vilniaus-ams.
  - place_code – naudojamas prognozės duomenims gauti pagal vietovę. Pvz., Vilniaus miesto kodas yra vilnius.

  (base_url – pagrindinis API adresas, numatytasis https://api.meteo.lt/v1.)

  Tokiu būdu užtikrinama, kad duomenų gavimas tiek iš istorinių, tiek iš prognozės API endpointų būtų lankstus ir aiškus.

Kodas:

```
def __init__(self, station_code:str, place_code: str, base_url: str= "https://api.meteo.lt/v1"):
    self.station_code=station_code
    self.place_code=place_code
    self.base_url = base_url
```

- Sukurtas metodas get_history, kuris priima du kintamuosius – date_from ir date_to. Iš pradžių sugeneruojamas dienų diapazonas ir sudedamas į kintamąjį dates. Tuomet atliekamas ciklas, kuris kreipiasi į API naudojant requests. Užklausos atliekamos su time.sleep, kad nebūtų pažeistos meteo.lt naudojimo taisyklės. Gauti rezultatai sudedami į DataFrame, sukuriamas time atributas, nustatomas kaip indeksas ir pridedama laiko juostos žyma (utc=True).

Kodas:

```python
def get_history(self, date_from:str, date_to:str)->pd.DataFrame:
    dates=pd.date_range(date_from, date_to, freq="D")
    frames=[]

    for d in dates:
        url = f"{self.base_url}/stations/{self.station_code}/observations/{d.date()}"
        #print(d.date())
        r = requests.get(url)
        if r.status_code != 200:
            print(f"Klaida gaunant duomenis {d.date()}: {r.status_code}")
            continue
        #nes site sake kad 180 per minute
        time.sleep(0.5)

        data = r.json().get("observations", [])
        if not data:
            continue

        df = pd.DataFrame(data)
        df["time"] = pd.to_datetime(df["observationTimeUtc"], utc=True)
        df = df.set_index("time").drop(columns=["observationTimeUtc"])
        frames.append(df)

    return pd.concat(frames).sort_index() if frames else pd.DataFrame()
```

- Sukurtas metodas get_forecast, kuris priima vieną kintamąjį – forecast_type (numatytasis „long-term“). Metodas sudaro API užklausos URL pagal vietovės kodą (place_code) ir pageidaujamą prognozės tipą. Vykdoma užklausa su requests.get, patikrinama, ar užklausa pavyko (raise_for_status). Gautas JSON rezultatas paverčiamas į DataFrame, sukuriamas time stulpelis iš forecastTimeUtc, pridedama laiko juosta (utc=True) ir nustatoma kaip indeksas. Metodas grąžina prognozės duomenų lentelę su temperatūra, krituliais ir kitais meteorologiniais parametrais.

Kodas:

```python
def get_forecast(self, forecast_type: str = "long-term") -> pd.DataFrame:
    url = f"{self.base_url}/places/{self.place_code}/forecasts/{forecast_type}"
    r = requests.get(url)
    r.raise_for_status()
    data = r.json()
    df = pd.DataFrame(data.get("forecastTimestamps", []))
    df["time"] = pd.to_datetime(df["forecastTimeUtc"], utc=True)
    df = df.set_index("time").drop(columns=["forecastTimeUtc"])
    return df
```

• Sukurtas MeteoReader klasės objektas reader, inicijuojant jį su stoties kodu vilniaus-ams istoriniams duomenims ir vietovės kodu vilnius prognozės duomenims. Pirmiausia iškviečiamas metodas get_history su nurodytu laikotarpiu nuo 2024-08-28 iki 2025-08-28, ir gauti istorinių duomenų pirmieji įrašai išvedami su head(). Tuomet iškviečiamas metodas get_forecast su numatytuoju prognozės tipu long-term, o gauti prognozės duomenys taip pat pateikiami ekrane su head().

Kodas:

```
]: reader = MeteoReader(station_code="vilniaus-ams", place_code="vilnius")

   print("Istoriniai duomenys:")
   history_df = reader.get_history("2024-08-28", "2025-08-28")
   print(history_df.head())
   print("Forecastas:")
   forecast_df = reader.get_forecast("long-term")
   print(forecast_df.head())
```

```
Istoriniai duomenys:
                           airTemperature  feelsLikeTemperature  windSpeed  \
time
2024-08-28 00:00:00+00:00            14.4                  14.4        1.2
2024-08-28 01:00:00+00:00            14.2                  14.2        1.7
2024-08-28 02:00:00+00:00            13.6                  13.6        1.6
2024-08-28 03:00:00+00:00            13.2                  13.2        1.9
2024-08-28 04:00:00+00:00            13.5                  13.5        1.8

                           windGust  windDirection  cloudCover  \
time
2024-08-28 00:00:00+00:00       2.8            349         0.0
2024-08-28 01:00:00+00:00       3.4             17         0.0
2024-08-28 02:00:00+00:00       3.4            357         0.0
2024-08-28 03:00:00+00:00       3.7             19         0.0
2024-08-28 04:00:00+00:00       4.3             17         0.0

                           seaLevelPressure  relativeHumidity  precipitation  \
time
2024-08-28 00:00:00+00:00            1026.6                75            0.0
2024-08-28 01:00:00+00:00            1026.7                76            0.0
2024-08-28 02:00:00+00:00            1026.8                79            0.0
2024-08-28 03:00:00+00:00            1027.1                81            0.0
2024-08-28 04:00:00+00:00            1027.3                79            0.0

                          conditionCode
time
2024-08-28 00:00:00+00:00         clear
2024-08-28 01:00:00+00:00         clear
```

2dalis:

• Pirmiausia istorinių ir prognozės duomenų indeksai konvertuojami į Lietuvos laiko zoną (Europe/Vilnius) naudojant tz_convert, kad vėlesni skaičiavimai būtų teisingi pagal vietinį laiką.

A. Vidutinė oro temperatūra ir drėgmė
• Iš istorinių duomenų apskaičiuojamas stulpelių airTemperature ir relativeHumidity vidurkis naudojant mean().
• Rezultatas atspindi visų metų vidutines oro sąlygas.

B. Vidutinė dienos ir nakties temperatūra
• Iš laiko indekso išskiriama valanda (hour) ir pagal ją priskiriama „day“ (08:00–20:00) arba „night“ kategorija (day_or_night).
• Apskaičiuojami vidurkiai atskirai dienos ir nakties laikotarpiams.
• Rezultatai pateikiami kaip vidutinė dienos ir nakties temperatūra.

C. Savaitgalių su lietumi skaičius
• Sukuriamas stulpelis is_rain, kuris žymi, ar kritulių kiekis didesnis už 0.
• Iš laiko indekso sukuriamas date stulpelis (tik data) ir weekday stulpelis (0=pn, 5=šeštadienis, 6=sekmadienis).
• Atrenkamos tik savaitgalio dienos, kur lijo (is_rain=True).
• Pasinaudojus pd.Series ir to_period('W'), suskaičiuojamas unikalių savaitgalių skaičius, kai buvo kritulių.
• Rezultatas pateikiamas ekrane.

Kodas:

```
#2 uzd----------------------------------------------------------------
history_lt = history_df.tz_convert("Europe/Vilnius")
forecast_lt=forecast_df.tz_convert("Europe/Vilnius")

#A:
print("Oro ir dregmes vidurkiai:")
means=history_lt[["airTemperature", "relativeHumidity"]].mean()
print(means)

#B:
history_lt["hour"] = history_lt.index.hour
history_lt["day_or_night"] = history_lt["hour"].apply(lambda h: "day" if 8 <= h < 20 else "night")
day_mean = history_lt[history_lt["day_or_night"] == "day"]["airTemperature"].mean()
night_mean = history_lt[history_lt["day_or_night"] == "night"]["airTemperature"].mean()

print(f"Vidutinė dienos temperatūra: {day_mean:.2f} °C")
print(f"Vidutinė nakties temperatūra: {night_mean:.2f} °C")

#C:
history_lt["is_rain"] = history_lt["precipitation"] > 0

history_lt["date"] = history_lt.index.normalize()
history_lt["weekday"] = history_lt.index.weekday
rain_weekend_days = history_lt[(history_lt["is_rain"]) & (history_lt["weekday"] >= 5)]["date"].unique()
rain_weekend_series = pd.Series(rain_weekend_days)
num_rainy_weekends = len(rain_weekend_series.dt.to_period('W').unique())
print(f"Savaitgalių, kai lijo: {num_rainy_weekends}")
```

Rezultatas:

```
Oro ir dregmes vidurkiai:
airTemperature       8.620328
relativeHumidity    78.507696
dtype: float64
Vidutinė dienos temperatūra: 10.09 °C
Vidutinė nakties temperatūra: 7.15 °C
Savaitgalių, kai lijo: 34
```

3 dalis

• Nustatomas paskutinės savaitės laikotarpis: nuo didžiausios istorinių duomenų datos atimant 7 dienas (last_week_start).
• Atrenkami tik paskutinės savaitės istorinių duomenų įrašai (last_week).
• Iš istorinių duomenų išrenkamas stulpelis airTemperature ir pavadinamas Measured.
• Iš prognozės duomenų išrenkamas stulpelis airTemperature ir pavadinamas Forecast.
• Istoriniai ir prognozės duomenys sujungiami į vieną DataFrame (combined).

• Sukuriamas grafikas (plt.figure) su dydžiu 12x5 colių:

Paskutinės savaitės išmatuota temperatūra nubrėžiama su ženklais o.

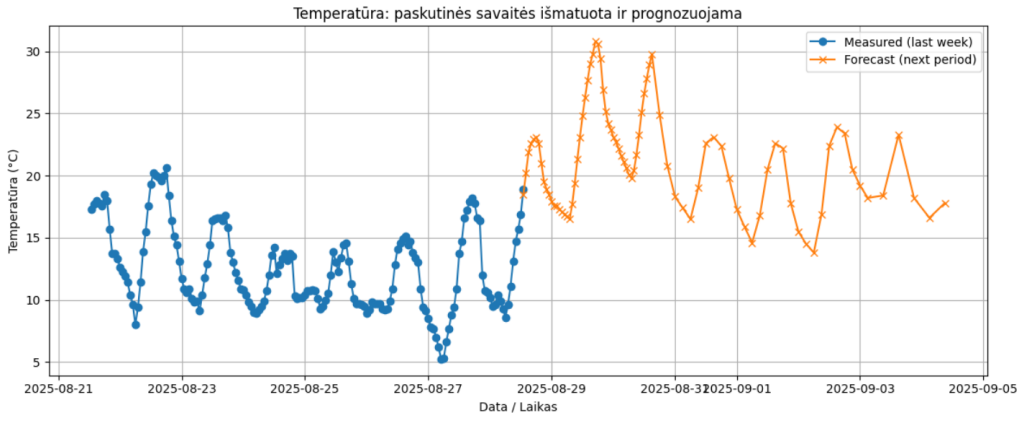
Ateinančios prognozės temperatūra nubrėžiama su ženklais x.

Priedu pridedamas pavadinimas, ašių aprašymai, legenda ir tinklelis.
• Grafikas vizualiai pateikia istorinius ir prognozės duomenis, leidžiant palyginti faktinę paskutinės savaitės temperatūrą su artimiausios prognozės reikšmėmis.

Kodas:

```
#3uzd:
last_week_start = history_lt.index.max() - pd.Timedelta(days=7)
last_week = history_lt[history_lt.index >= last_week_start]
history_temp = last_week["airTemperature"].rename("Measured")
forecast_temp = forecast_lt["airTemperature"].rename("Forecast")
combined = pd.concat([history_temp, forecast_temp])
plt.figure(figsize=(12,5))
plt.plot(history_temp.index, history_temp, label="Measured (last week)", marker='o')
plt.plot(forecast_temp.index, forecast_temp, label="Forecast (next period)", marker='x')
plt.title("Temperatūra: paskutinės savaitės išmatuota ir prognozuojama")
plt.xlabel("Data / Laikas")
plt.ylabel("Temperatūra (°C)")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

Rezultatas:



4dalis

• Sukurta funkcija interpolate_to_5min, kuri priima pandas.Series objektą (temperatūros laikotarpį).
• Funkcijos veikimas:

- Sukuriamas naujas indeksas su 5 minučių dažniu (freq='5T') nuo originalaus laikotarpio pradžios iki pabaigos, išlaikant originalią laiko juostą (tz=series.index.tz).
- Originali serija priskiriama naujam indeksui naudojant reindex.
- Tarpinės reikšmės interpoliuojamos pagal laiką (interpolate(method='time')).
  • Praktinis pavyzdys: paskutinės savaitės temperatūros serija (last_week_temp) buvo interpoliuota iki 5 minučių dažnio, ir rezultatas pateiktas ekrane (print(temp_5min.head(50))).

• Tokiu būdu gauta temperatūros serija turi tankesnį laiko dažnį, kas leidžia tiksliau vizualizuoti ir analizuoti temperatūros pokyčius per savaitę.

Kodas:

```python
#4uzd:
def interpolate_to_5min(series: pd.Series) -> pd.Series:
    new_index = pd.date_range(start=series.index.min(),
                              end=series.index.max(),
                              freq='5T',
                              tz=series.index.tz)

    series_5min = series.reindex(new_index)
    series_5min = series_5min.interpolate(method='time')
    return series_5min

# Paskutine savaite
last_week_temp = history_lt["airTemperature"].last("7D")
temp_5min = interpolate_to_5min(last_week_temp)

print(temp_5min.head(50))
```

Rezultatai:

```
2025-08-21 14:00:00+03:00    17.700000
2025-08-21 14:05:00+03:00    17.725000
2025-08-21 14:10:00+03:00    17.750000
2025-08-21 14:15:00+03:00    17.775000
2025-08-21 14:20:00+03:00    17.800000
2025-08-21 14:25:00+03:00    17.825000
2025-08-21 14:30:00+03:00    17.850000
2025-08-21 14:35:00+03:00    17.875000
2025-08-21 14:40:00+03:00    17.900000
2025-08-21 14:45:00+03:00    17.925000
2025-08-21 14:50:00+03:00    17.950000
```