МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯСІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки Кафедра
інформатики та програмної інженерії

Практикум №7

з курсу «Аналіз даних в інформаційнних системах» на тему:

« Аналіз часових послідовностей»

Викладач:
Ліхоузова Т.А.

Виконав:
студент 2 курсу групи
ІП-15 ФІОТ
Мєшков Андрій
Ігорович

Київ-2023

Практикум №7

Аналіз часових послідовностей

Мета роботи: ознайомитись з методами моделювання часових послідовностей.

Завдання:

Скачати потрібні дані.

Основне завдання

1. Побудувати та проаналізувати часовий ряд для статистики захворювань на Covid в двох сусідніх країнах по вашому вибору (дані взяти в інтернеті).
2. Побудувати та проаналізувати часовий ряд для курсу гривня/долар або гривня/євро за останні 3 роки (дані взяти в інтернеті).

Додаткове завдання

Потрібно з'ясувати, чи є сезонна компонента в кількості опадів в Сіетлі. (https://www.kaggle.com/rtatman/did-it-rain-in-seattle-19482017/data, або seattleWeather_1948-2017.csv Скачати потрібні дані).

3. Градуси перевести в Цельсії.
4. Чи є кореляція між температурою та опадами?
5. Скласти прогноз опадів на 2018 рік, оцінити точність прогнозу

Хід роботи:

Основне завдання:

Імпортуємо потрібні бібліотеки.

```
# Fetch the Covid-19 data for Germany and Poland from the "Our World in Data" website
import pandas as pd
import seaborn as sns

import statsmodels.api as sm
import statsmodels.tsa.api as smt
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
```

Зчитаємо файл.

```
covid_data = pd.read_csv('data.csv', sep=',', decimal=',', encoding='windows-1251')
poland_covid = covid_data[covid_data['countriesAndTerritories'] == 'Poland']
germany_covid = covid_data[covid_data['countriesAndTerritories'] == 'Germany']
poland_covid['dateRep'] = pd.to_datetime(p_covid['dateRep'], format='%d/%m/%Y')
germany_covid['dateRep'] = pd.to_datetime(g_covid['dateRep'], format='%d/%m/%Y')
```

Проаналізуємо структуру.

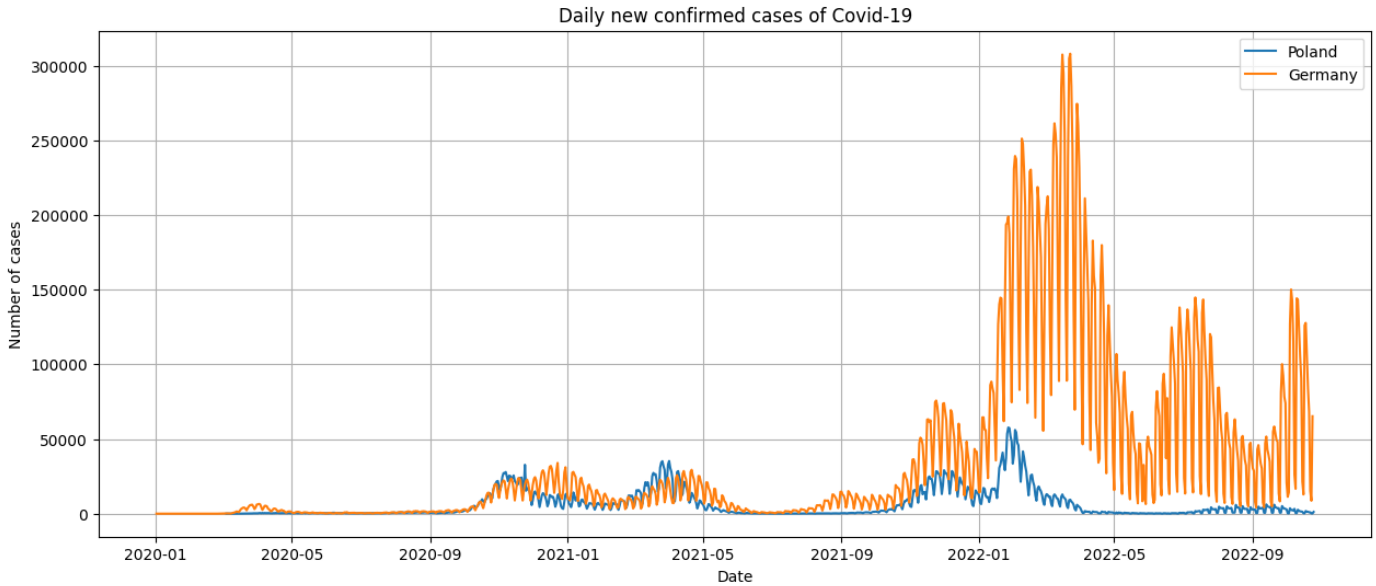
```
covid_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28729 entries, 0 to 28728
Data columns (total 11 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   dateRep                 28729 non-null  object
 1   day                     28729 non-null  int64
 2   month                   28729 non-null  int64
 3   year                    28729 non-null  int64
 4   cases                   28636 non-null  float64
 5   deaths                  28437 non-null  float64
 6   countriesAndTerritories  28729 non-null  object
 7   geoId                   28729 non-null  object
 8   countryterritoryCode    28729 non-null  object
 9   popData2020             28729 non-null  int64
 10  continentExp            28729 non-null  object
dtypes: float64(2), int64(4), object(5)
memory usage: 2.4+ MB
```

Побудуймо щоденні нові підтверджені випадки для Польщі та Німеччини.

```
import matplotlib.pyplot as plt
import matplotlib.dates as mdates

# Plot the daily new confirmed cases for Poland and Germany
fig, ax = plt.subplots(figsize=(15, 6))
ax.plot(poland_covid['dateRep'], poland_covid['cases'], label='Poland')
```

```
ax.plot(germany_covid['dateRep'], germany_covid['cases'], label='Germany')
ax.set_title('Daily new confirmed cases of Covid-19')
locator = mdates.AutoDateLocator()
formatter = mdates.AutoDateFormatter(locator)
ax.xaxis.set_major_locator(locator)
ax.xaxis.set_major_formatter(formatter)
ax.set_xlabel('Date')
ax.set_ylabel('Number of cases')
ax.legend()
ax.grid()
plt.show()
```




```
# Find the date with the maximum number of cases for Poland and Germany
poland_max_cases_date = poland_covid.loc[poland_covid['cases'].idxmax(), 'dateRep']
germany_max_cases_date = germany_covid.loc[germany_covid['cases'].idxmax(), 'dateRep']

# Print the results
print(f"Date with the maximum number of cases in Poland: {poland_max_cases_date.date()}")
print(f"Date with the maximum number of cases in Germany: {germany_max_cases_date.date()}")
```

```
Date with the maximum number of cases in Poland: 2022-01-27
Date with the maximum number of cases in Germany: 2022-03-23
```

Дослідимо часові ряди.

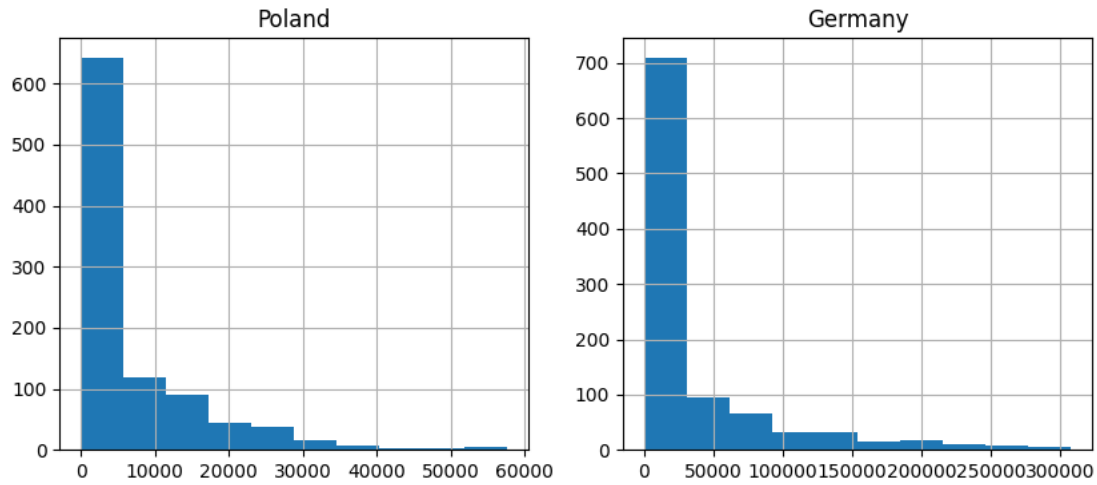
```
poland_covid_c = poland_covid['cases']
poland_covid_c.describe()
```

```
count      966.000000
mean      6407.414079
std       9458.639495
min          0.000000
25%        339.250000
50%       1472.500000
75%       9331.500000
max      57659.000000
Name: cases, dtype: float64
```

```
germany_covid_c = germany_covid['cases']
germany_covid_c.describe()
```

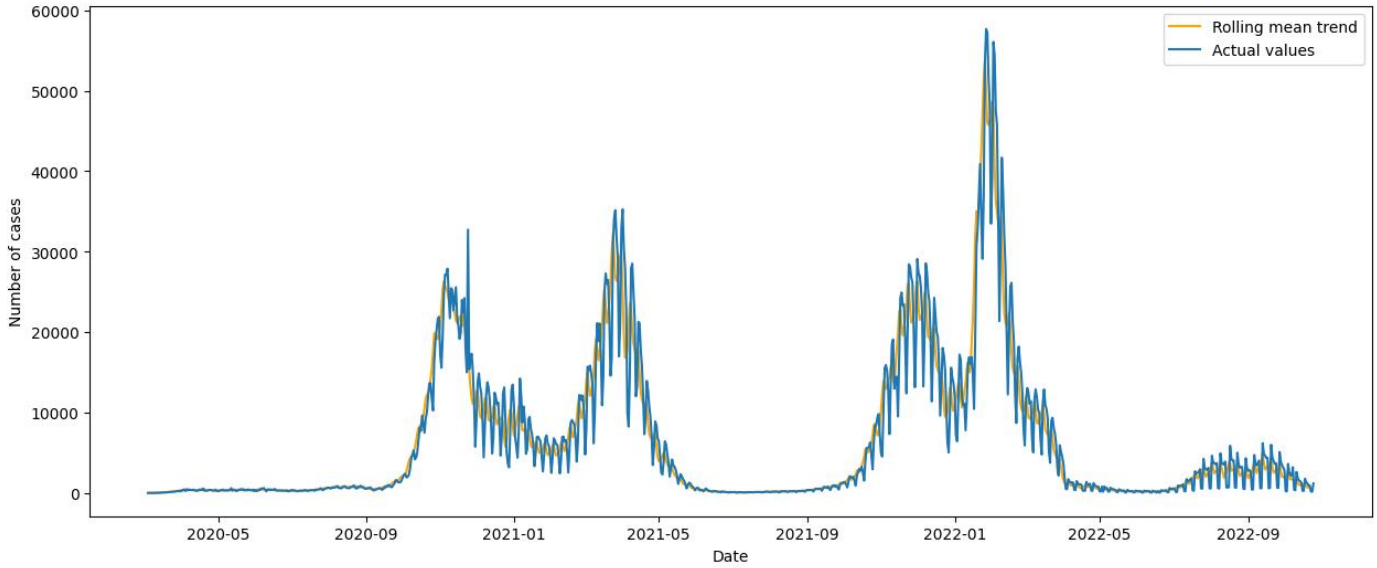
```
count       992.000000
mean      35572.268145
std       56682.109086
min           1.000000
25%        2088.000000
50%       11359.000000
75%       41831.750000
max      307914.000000
Name: cases, dtype: float64
```

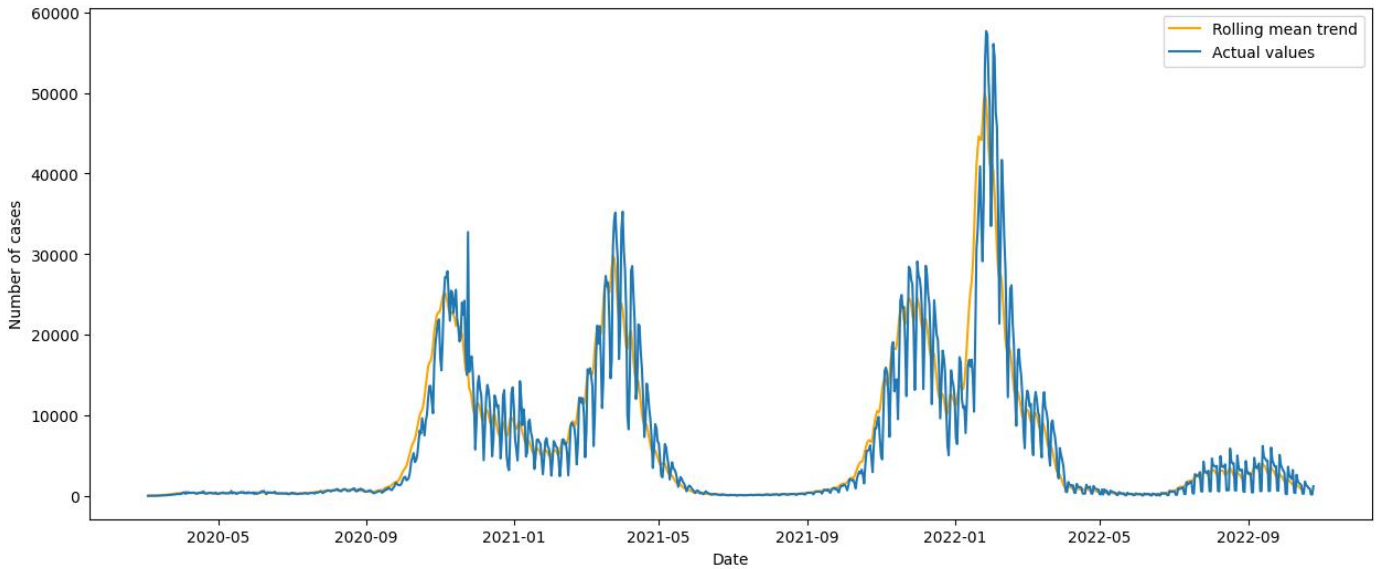
```
fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(10, 4))
poland_covid_c.hist(ax=ax1)
ax1.set_title('Poland')
germany_covid_c.hist(ax=ax2)
ax2.set_title('Germany')
plt.show()
```

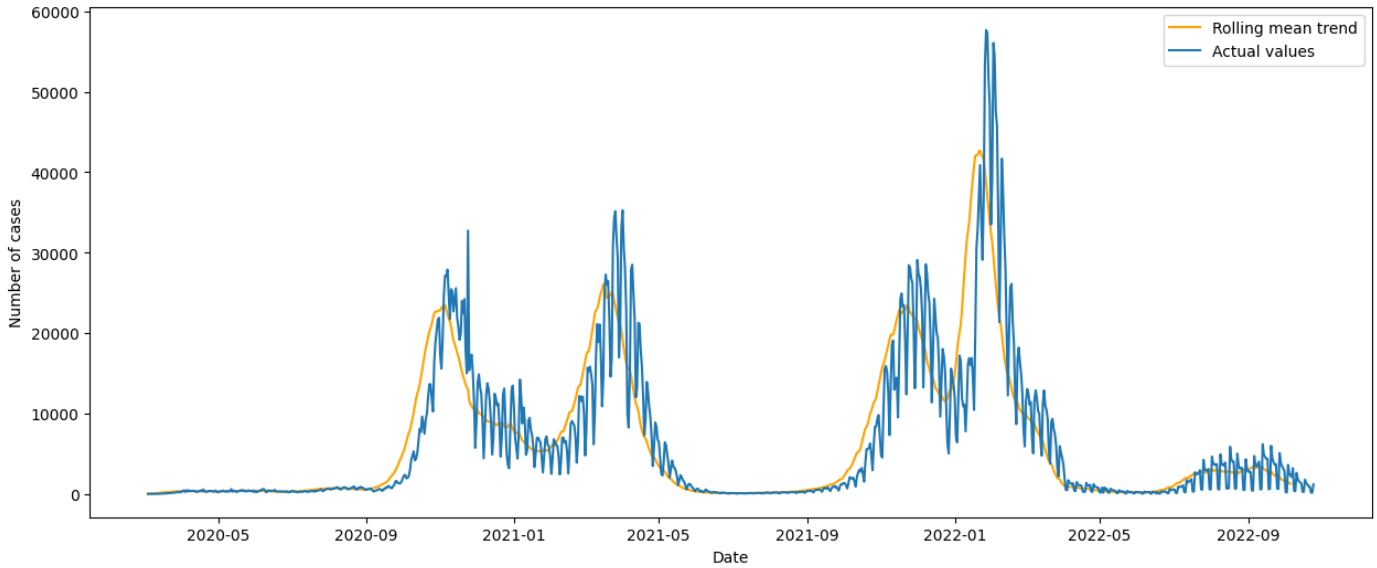


Для кращої візуалізації властивостей ряду (трендів, сезонності тощо) застосуємо згладжування за допомогою ковзаючого середнього.

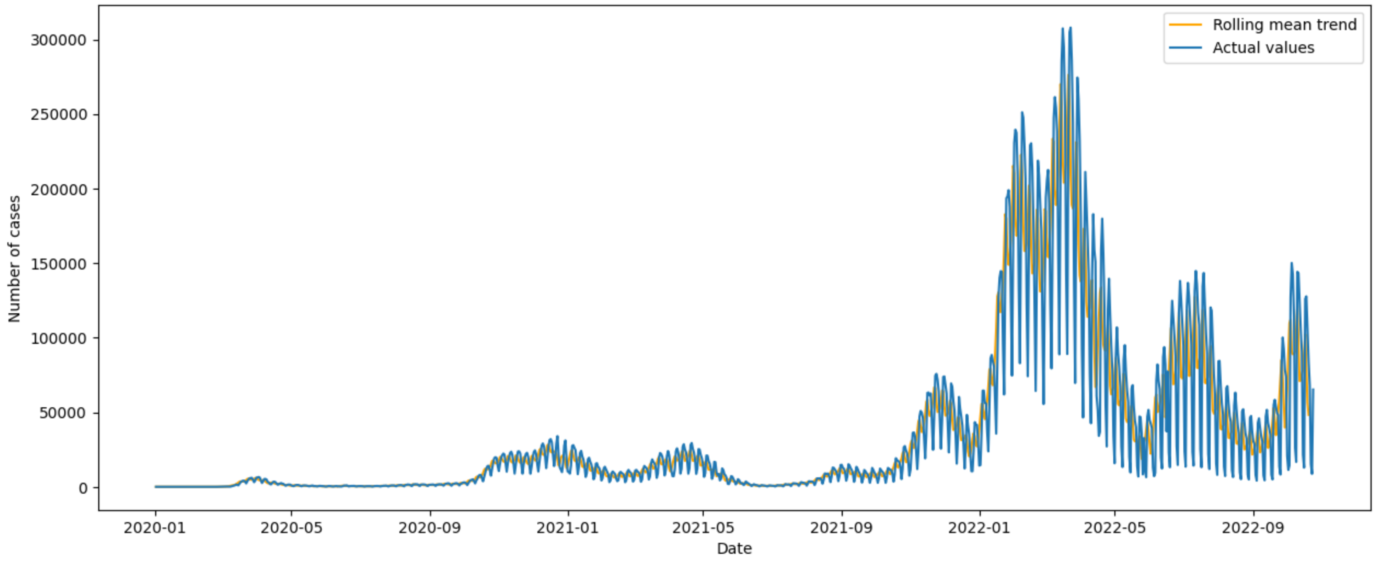
```
def plot_moving_average(series, n):
    rolling_mean = series['cases'].rolling(window=n).mean()
    fig, ax = plt.subplots(figsize=(15, 6))
    ax.set_xlabel('Date')
    ax.set_ylabel('Number of cases')
    ax.plot(series['dateRep'], rolling_mean, c='orange', label='Rolling mean trend')
    ax.plot(series['dateRep'], series['cases'], label='Actual values')
    ax.legend(loc='upper left')
    ax.grid(True)
    ax.legend()
    ax.grid()
    plt.show()
print('Poland: ')
plot_moving_average(poland_covid, 5)
plot_moving_average(poland_covid, 10)
plot_moving_average(poland_covid, 20)
```

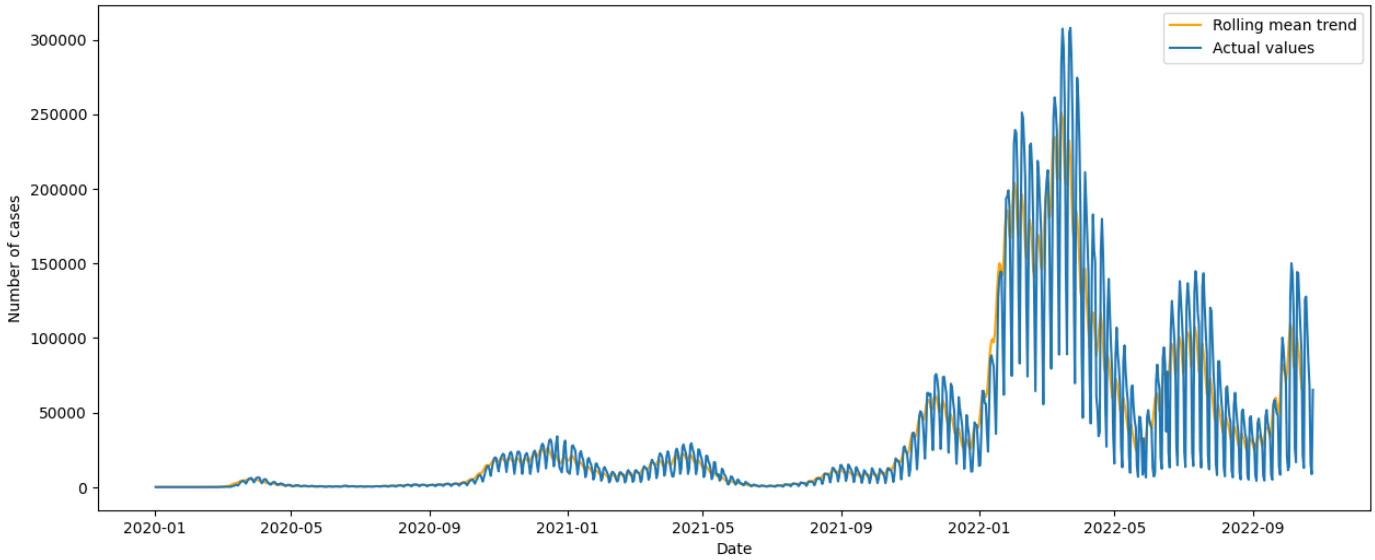


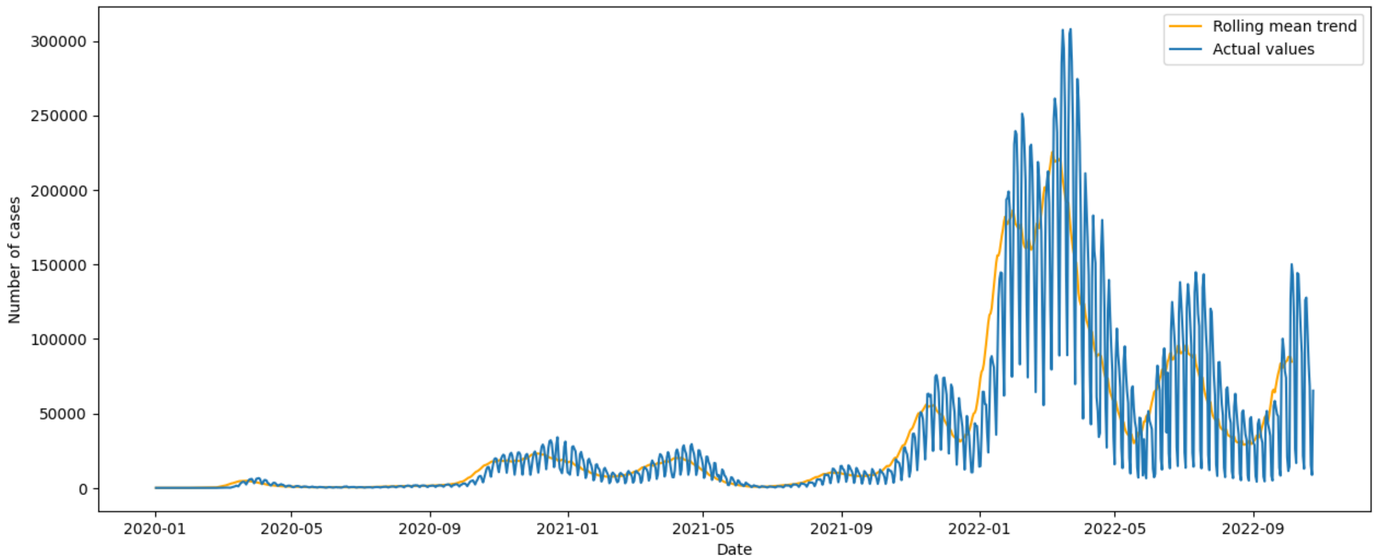





```
print('Germany: ')
plot_moving_average(germany_covid, 5)
plot_moving_average(germany_covid, 10)
plot_moving_average(germany_covid, 20)
```



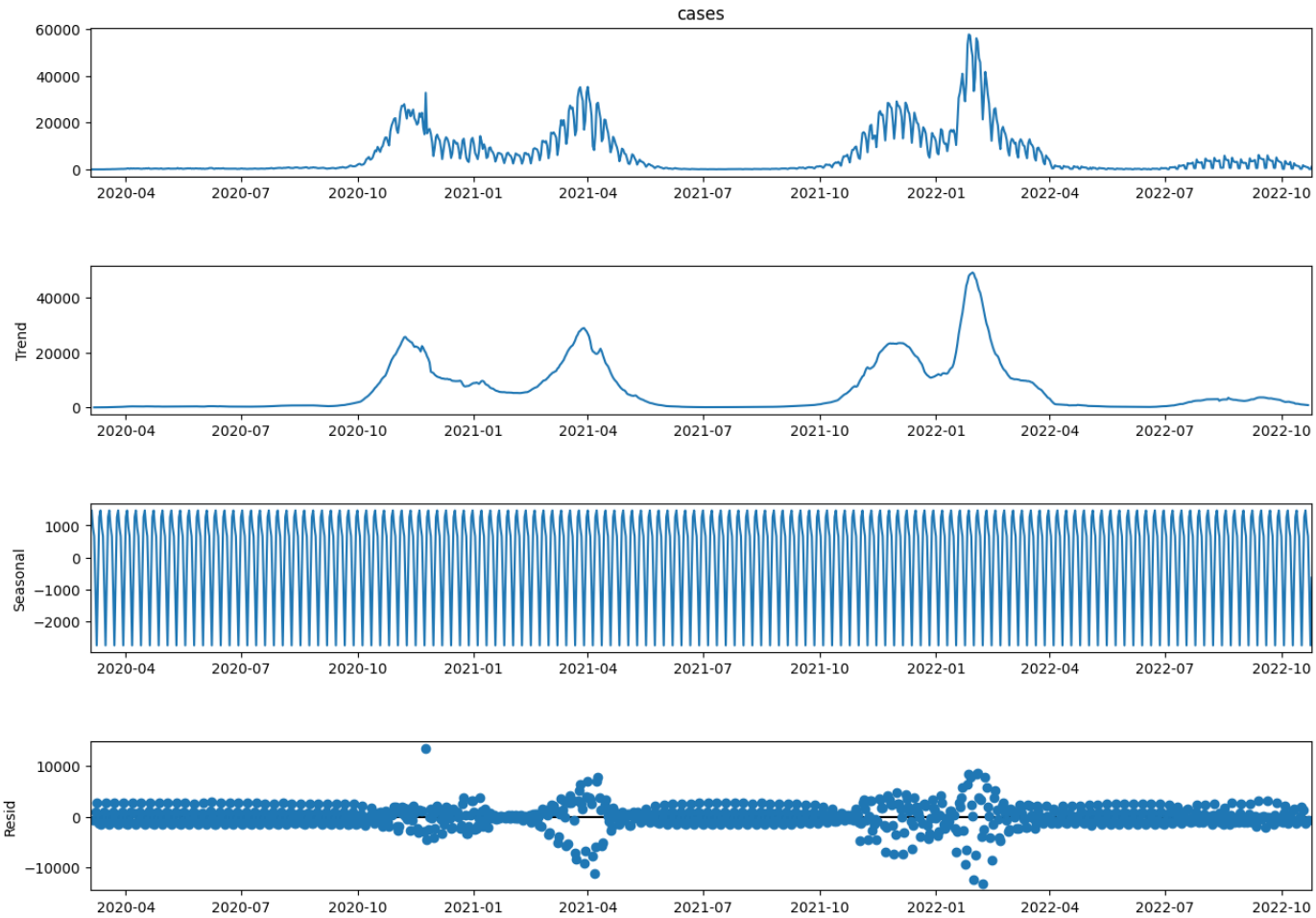




Візуалізуємо декомпозицію ряду на тренд, сезонність та залишки.

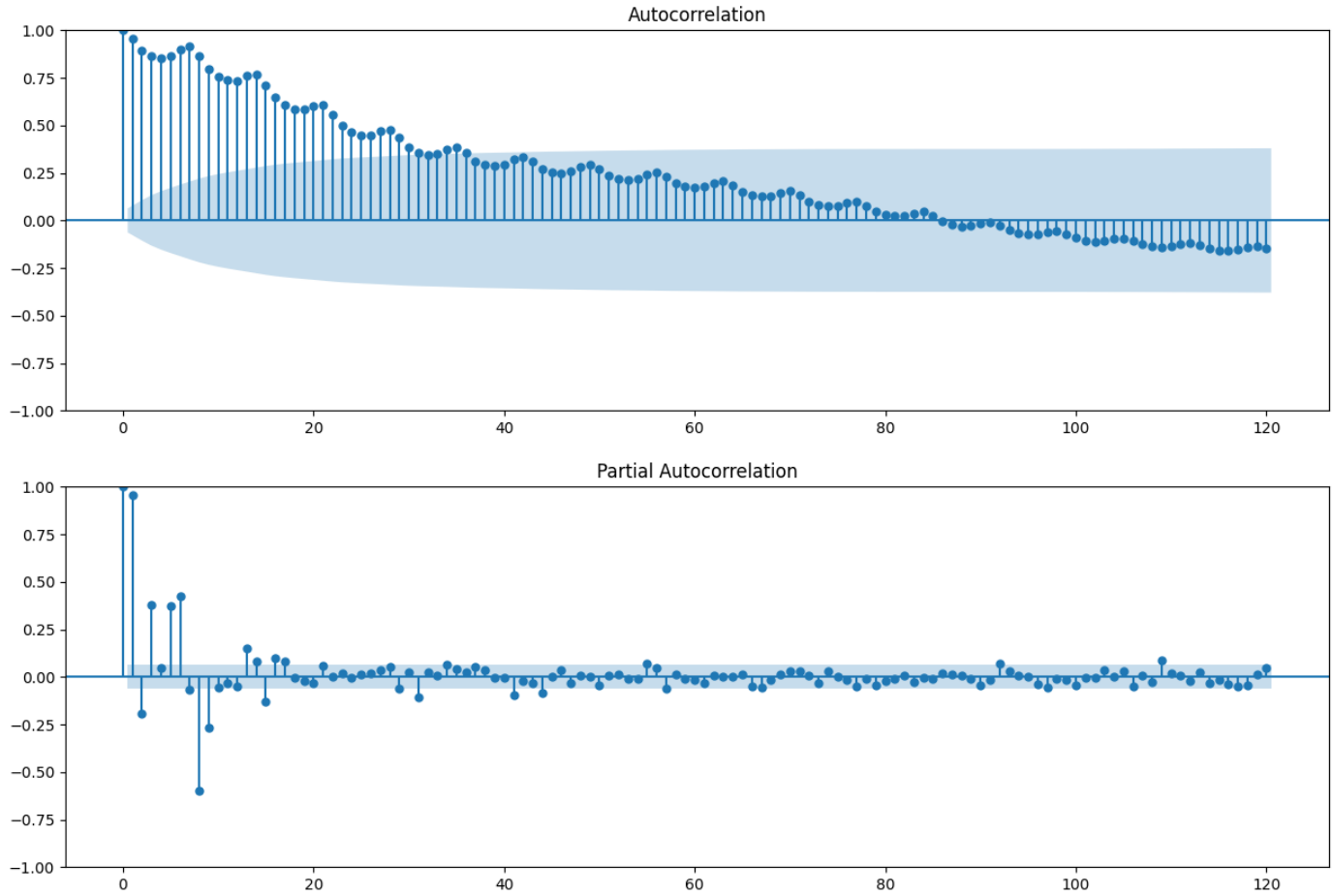
```
poland_covid_n = poland_covid[['dateRep','cases']]
poland_covid_n = poland_covid_n.dropna()
poland_covid_n = poland_covid_n.sort_values(by='dateRep')
decomposition = smt.seasonal_decompose(poland_covid_n.set_index('dateRep')['cases'])

# Plot the results
```

```
print('Poland: ')
fig = decomposition.plot()
fig.set_size_inches(15, 10)
plt.show()
```



Будуємо графіки автокореляції та часткової автокореляції.

```
poland_covid_n = poland_covid[['dateRep', 'cases']]
fig, ax = plt.subplots(2, figsize=(15, 10))
ax[0] = plot_acf(poland_covid_n['cases'], ax=ax[0], lags=120)
ax[1] = plot_pacf(poland_covid_n['cases'], ax=ax[1], lags=120)
plt.show()
```



Перевіримо ряд на стаціонарність за допомогою доповненого тесту Дікі-Фуллера.

```
def dickey_fuller_test(series):
    test = smt.adfuller(series, autolag='AIC')
    print('adf: ', test[0])
    print('p-value: ', test[1])
    print('Critical values: ', test[4])
    if test[0] > test[4]['5%']:
        print(' There are unit roots, the series is not stationary.')
    else:
        print(' There are no unit roots, the series is stationary.')

dickey_fuller_test(poland_covid_n['cases'])
print('\n')
dickey_fuller_test(germany_covid['cases'])
```

```
adf:  -3.389276145156373
p-value:  0.011329924879352258
Critical values:  {'1%': -3.43725945868569, '5%': -2.8645903751292536, '10%': -2.5683941938438886}
There are no unit roots, the series is stationary


adf:  -1.9916387092353385
p-value:  0.29025122490318106
Critical values:  {'1%': -3.437116468121892, '5%': -2.864527318984441, '10%': -2.5683606077036214}
There are unit roots, the series is not stationary.
```

+ Code + Markdown

Зчитаємо файл

```
df = pd.read_csv('uah-usd.csv', sep=',', decimal=',', encoding='windows-1251')
```

```
# Convert data types to the appropriate types
df['Date'] = pd.to_datetime(df['Date'], format='%Y-%m-%d')
df[['Open', 'High', 'Low', 'Close', 'Adj Close']] = df[['Open', 'High', 'Low', 'Close', 'Adj Close']].astype(float)

# Sort the dataframe by date
df.sort_values(by='Date', inplace=True)
df.set_index('Date', inplace=True)
df.index.freq = pd.date_range(start=df.index[0], end=df.index[-1], periods=len(df)).inferred_freq
```

Будуємо динаміку ціни, найвищої та найнижчої за добу

```
fig, ax = plt.subplots(figsize=(15, 12))

df[['Open', 'High', 'Low']].plot(ax=ax, subplots=True)
ax.grid()
plt.show()
```

Використовуємо згладжування для дослідження характеристик рядів

```
def moving_average(series, n):
    rolling_mean = series.rolling(window=n).mean()
    plt.figure(figsize=(15, 5))
    plt.plot(rolling_mean, c='orange', label='Rolling mean trend')
    plt.plot(series[n:], label='Actual values')
    plt.legend(loc='upper left')
```

```
    plt.grid(True)
  moving_average(df['Open'], 30)
moving_average(df['High'], 30)
moving_average(df['Low'], 30)
```

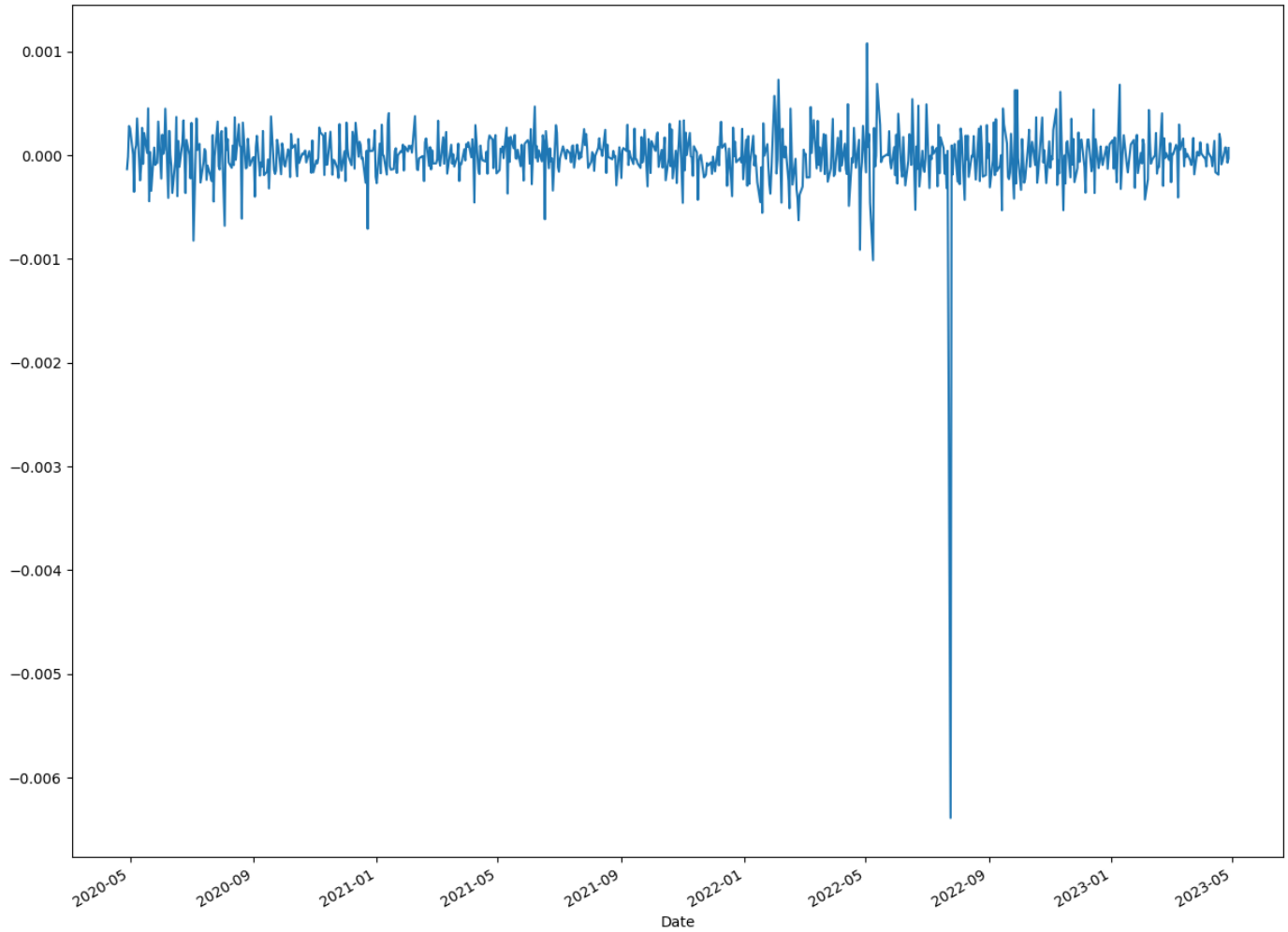
Перевіримо ряд на стаціонарність за допомогою доповненого тесту Дікі-Фуллера.

```
dickey_fuller_test(df['Open'])
```

```
adf:  -0.4096657335338527
p-value:  0.9084723942617245
Critical values:  {'1%': -3.4387291412780177, '5%': -2.8652383048736056, '10%': -2.568739332674375}
There are unit roots, the series is not stationary.
```

Зробимо стаціонарним

```
currencies_price_df_diff = df['Open'].diff(periods=1).dropna()
fig, ax = plt.subplots(figsize=(15, 12))

currencies_price_df_diff.plot(ax=ax)

plt.show()
```

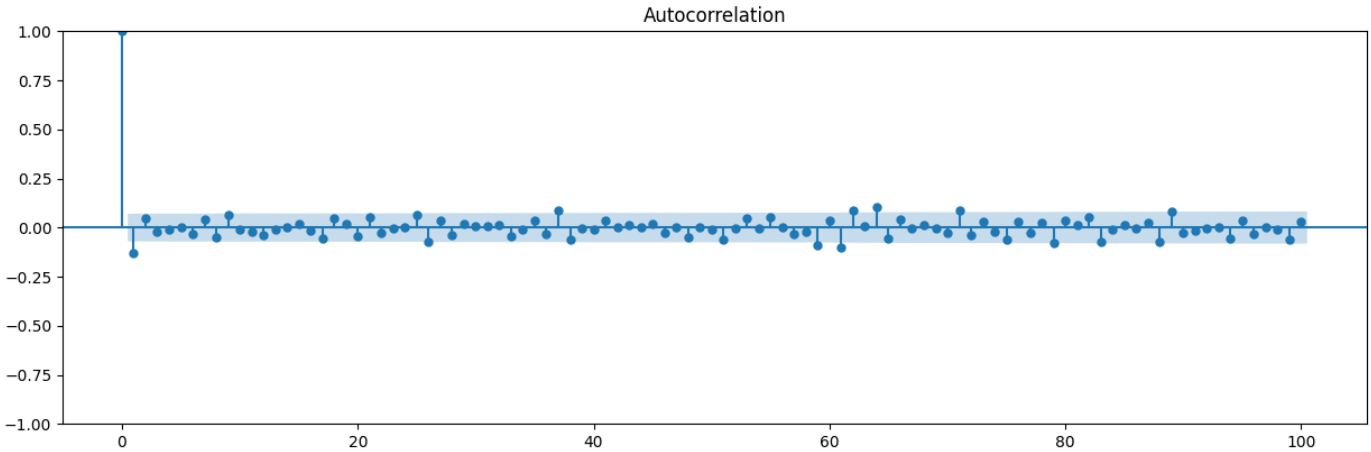


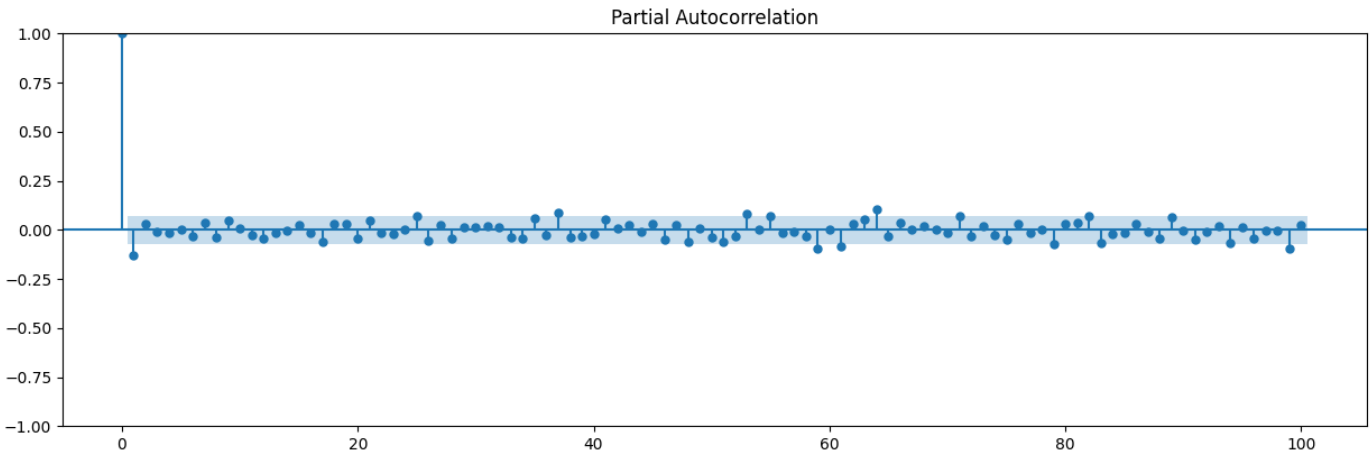

```
dickey_fuller_test(currencies_price_df_diff)
```

```
adf:  -31.798977358781638
p-value:  0.0
Critical values:  {'1%': -3.4387291412780177, '5%': -2.8652383048736056, '10%': -2.568739332674375}
There are no unit roots, the series is stationary
```

Будуємо графіки автокореляції та часткової автокореляції

```
fig, ax = plt.subplots(2, figsize=(15, 10))
ax[0] = plot_acf(currencies_price_df_diff, ax=ax[0], lags=100)
ax[1] = plot_pacf(currencies_price_df_diff, ax=ax[1], lags=100)
```





Будуємо модель ARIMA для прогнозу значення ціни на тиждень вперед

```
train_data = df['Open'][:-7]
model = smt.ARIMA(train_data, order=(1, 1, 1)).fit()
model.summary()
```

SARIMAX Results

Dep. Variable:	Open	No. Observations:	778
Model:	ARIMA(1, 1, 1)	Log Likelihood	5169.218
Date:	Fri, 28 Apr 2023	AIC	-10332.435
Time:	00:28:09	BIC	-10318.469
Sample:	04-27-2020	HQIC	-10327.063
	- 04-19-2023		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	-0.3888	0.010	-37.585	0.000	-0.409	-0.369
ma.L1	0.2669	0.010	26.154	0.000	0.247	0.287
sigma2	9.71e-08	4.74e-10	204.684	0.000	9.62e-08	9.8e-08

Ljung-Box (L1) (Q):	0.00	Jarque-Bera (JB):	1589552.96
Prob(Q):	0.96	Prob(JB):	0.00
Heteroskedasticity (H):	5.66	Skew:	-10.92
Prob(H) (two-sided):	0.00	Kurtosis:	223.50

```
pred = model.predict(df['Open'].index[-7], df['Open'].index[-1])
test_data = currencies_price_df_diff[-7:]
forecasts = model.forecast(7)
forecasts
```

```
2023-04-20    0.027413
2023-04-21    0.027418
2023-04-24    0.027416
2023-04-25    0.027417
2023-04-26    0.027416
2023-04-27    0.027416
2023-04-28    0.027416
Freq: B, Name: predicted_mean, dtype: float64
```

Додаткове завдання:

Імпортуємо потрібні бібліотеки.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import statsmodels.api as sm
import statsmodels.tsa.api as smt
```

```
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.graphics.tsaplots import plot_predict
```

Завантажемо файл.

```
df = pd.read_csv('seattleWeather_1948-2017.csv', index_col=['DATE'], parse_dates=['DATE'])
```

Дослідимо дані

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 25551 entries, 1948-01-01 to 2017-12-14
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   PRCP    25548 non-null  float64
 1   TMAX    25551 non-null  int64  
 2   TMIN    25551 non-null  int64  
 3   RAIN    25548 non-null  object 
dtypes: float64(1), int64(2), object(1)
memory usage: 998.1+ KB
```

Заповнюємо пропущені значення

```
df['PRCP'].fillna(0, inplace=True)
df['RAIN'].fillna(False, inplace=True)
```

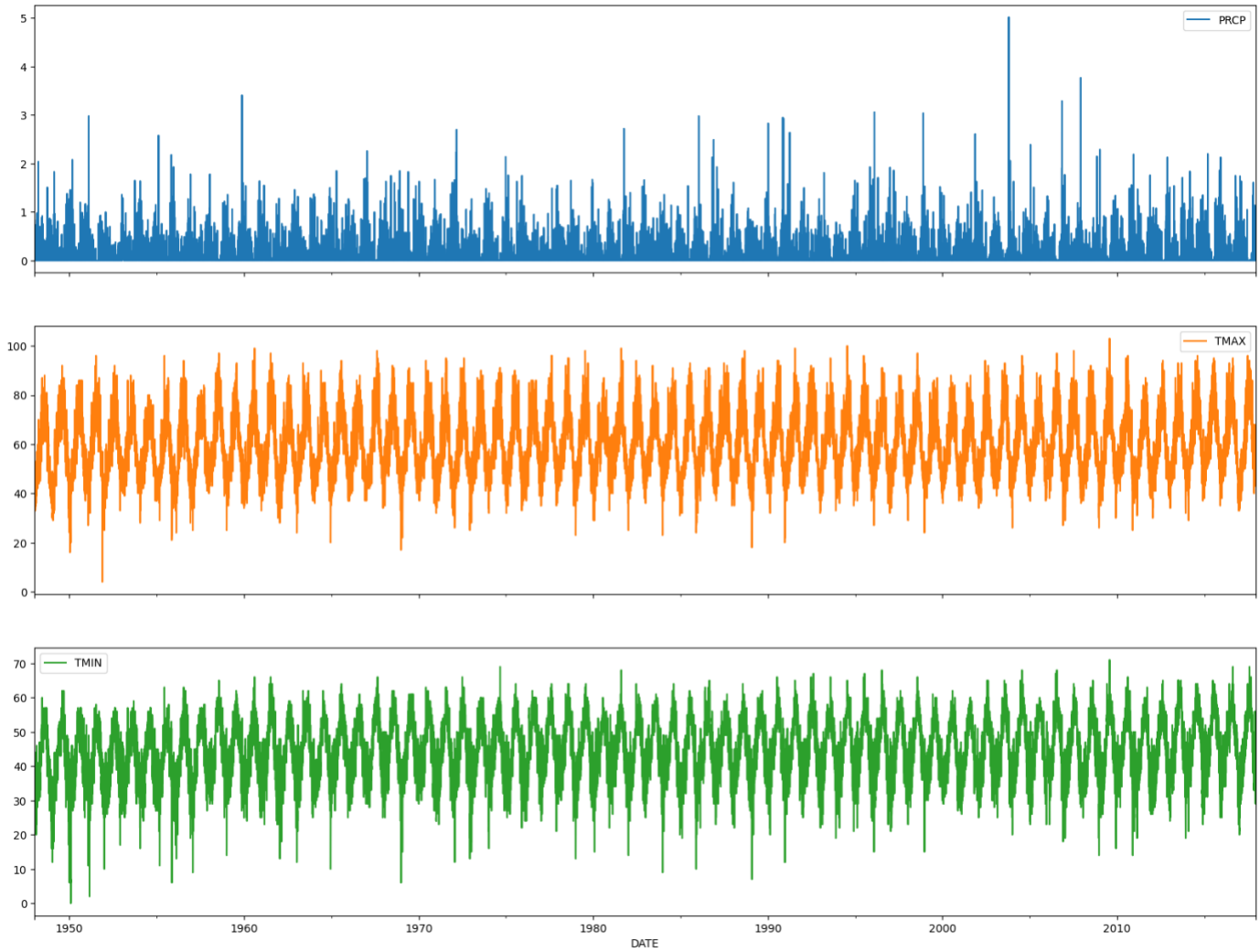
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 25551 entries, 1948-01-01 to 2017-12-14
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   PRCP    25551 non-null  float64
 1   TMAX    25551 non-null  int64  
 2   TMIN    25551 non-null  int64  
 3   RAIN    25551 non-null  bool   
dtypes: bool(1), float64(1), int64(2)
memory usage: 823.4 KB
```

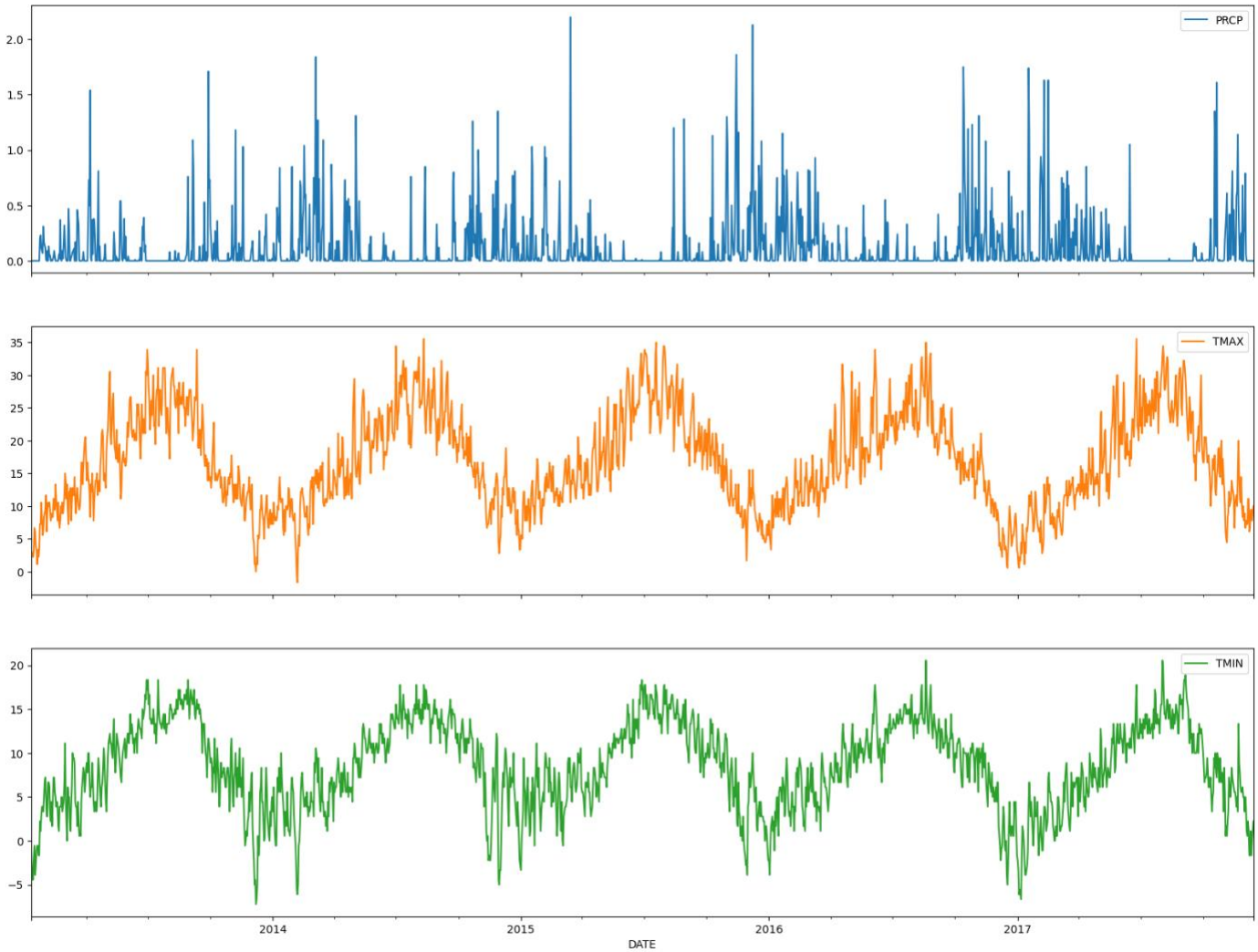
Будуємо динаміку опадів та температур в часі

```
df.plot(figsize=(20, 15), subplots=True)
```

```
plt.show()
```

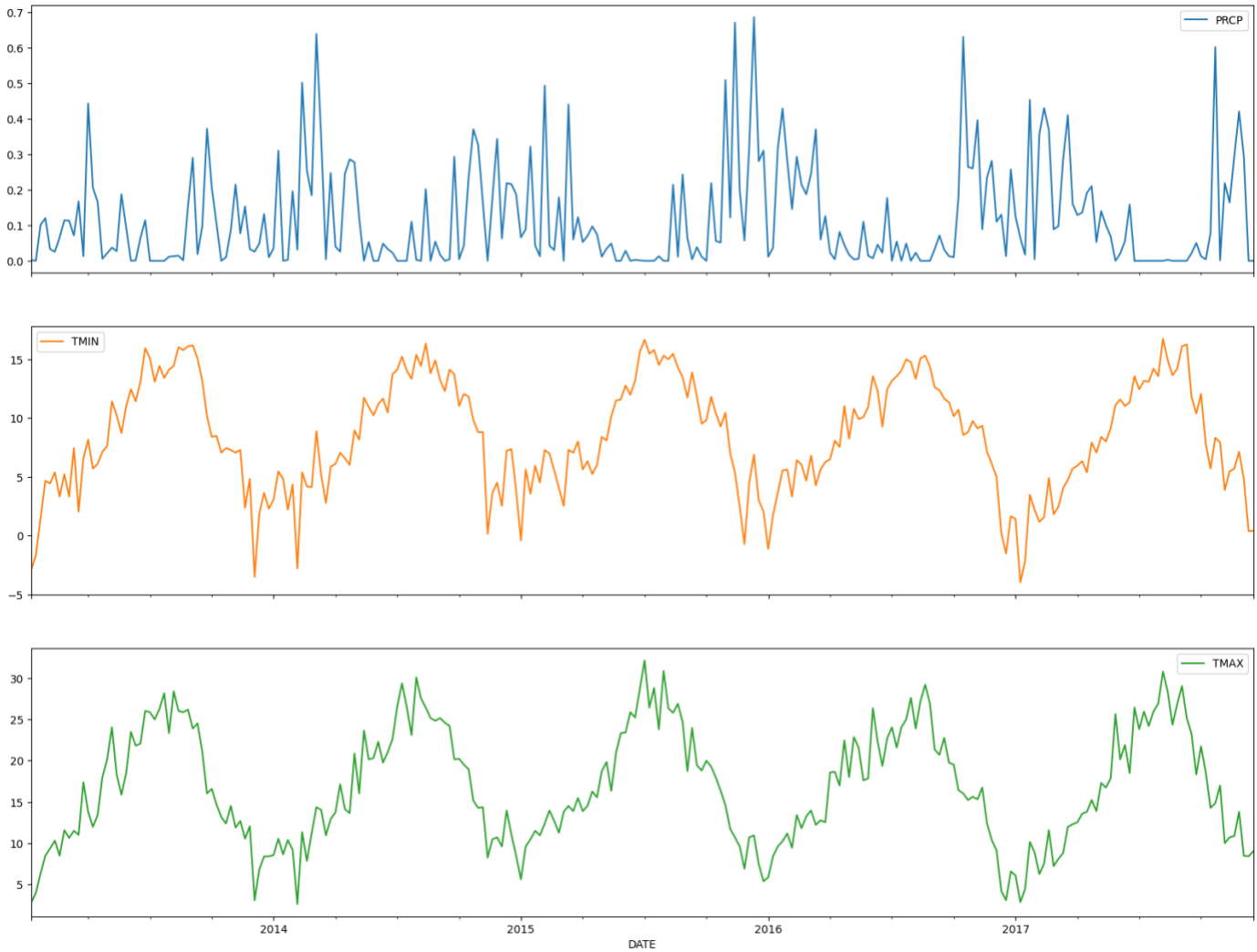



```
df.loc[df.index[-1800:]].plot(figsize=(20, 15), subplots=True)
plt.show()
```



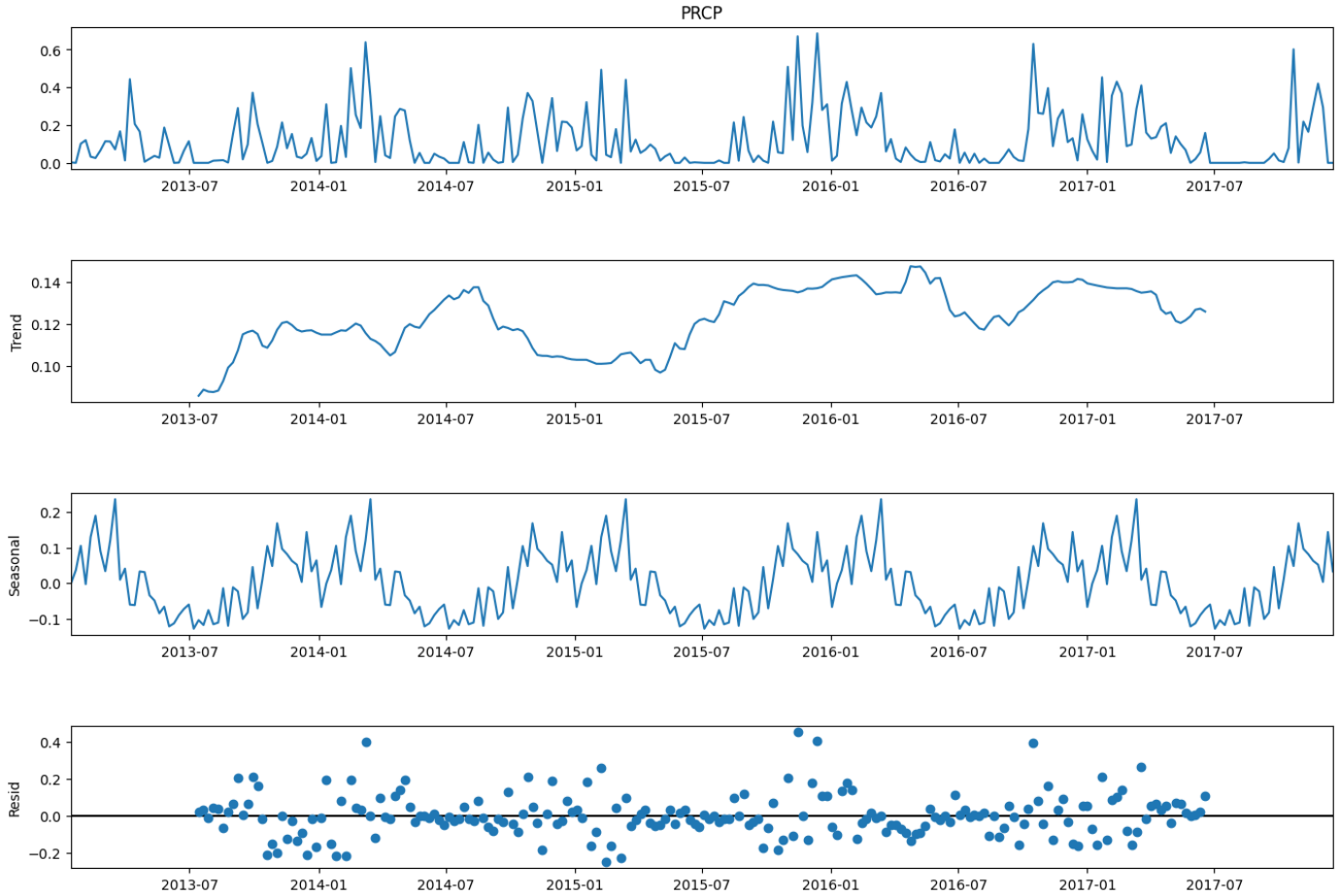
Динаміка для середніх значень показників по тижням

```
df[['PRCP', 'TMIN', 'TMAX']].loc[df.index[-1800:]].resample('W').mean().plot(figsize=(20, 15),
subplots=True)
plt.show()
```



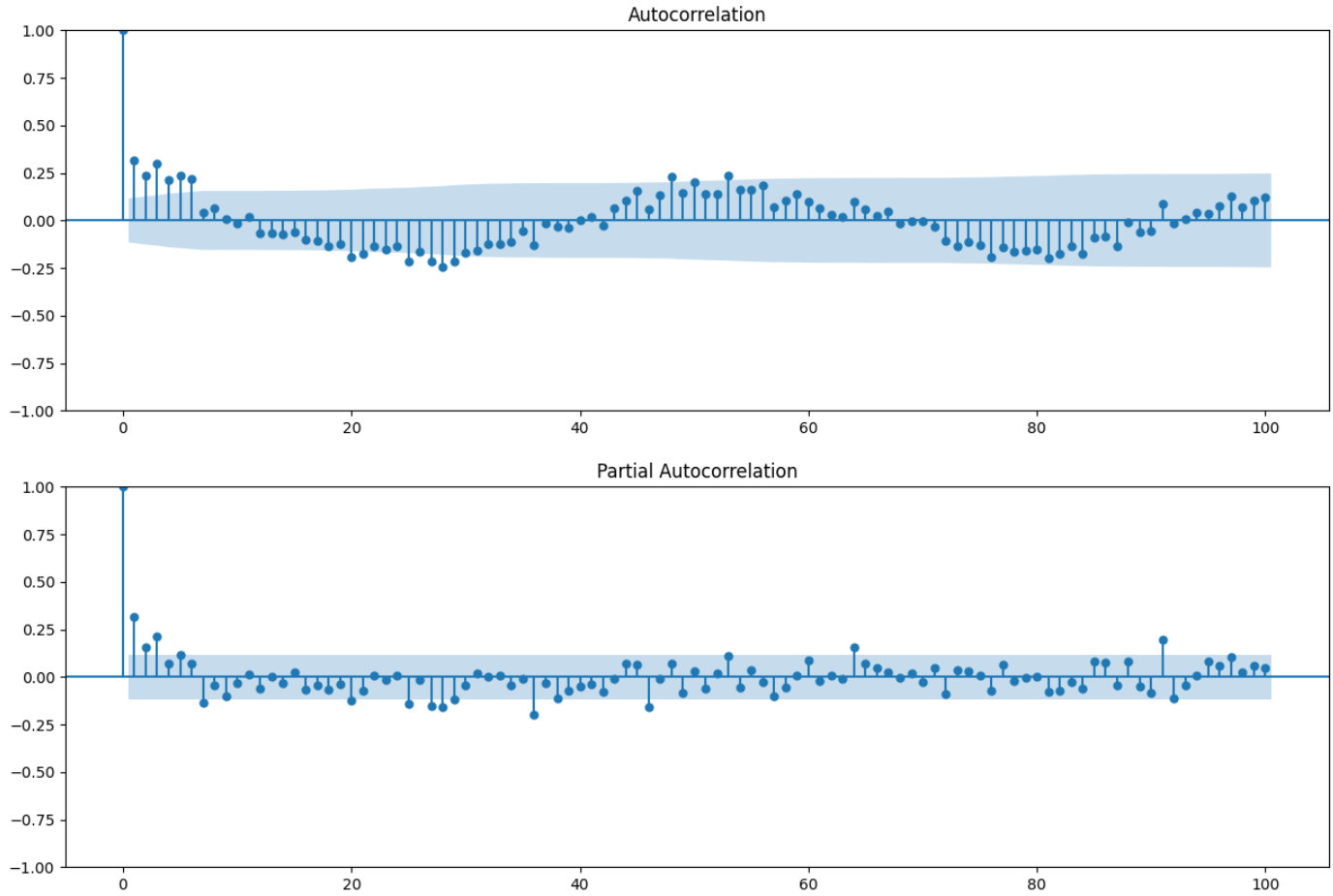
Декомпозуємо

```
prcp_decomposition = smt.seasonal_decompose(df['PRCP'].loc[df.index[-1800:]].resample('W').mean())
fig = prcp_decomposition.plot()
fig.set_size_inches(15, 10)
plt.show()
```



Аутокореляція

```
fig, ax = plt.subplots(2, figsize=(15, 10))
ax[0] = plot_acf(df['PRCP'].loc[df.index[-2000:]].resample('W').mean(), ax=ax[0], lags=100)
ax[1] = plot_pacf(df['PRCP'].loc[df.index[-2000:]].resample('W').mean(), ax=ax[1], lags=100)
```



Фаренгейт у Цульсії

```
df[['TMIN', 'TMAX']] = (df[['TMIN', 'TMAX']] - 32) * 5 / 9
df[['TMIN', 'TMAX']].describe()
```

	TMIN	TMAX
count	25551.000000	25551.000000
mean	6.952348	15.302337
std	4.940464	7.096102
min	-17.777778	-15.555556
25%	3.333333	10.000000
50%	7.222222	14.444444
75%	11.111111	20.555556
max	21.666667	39.444444

Будуємо матрицю кореляцій

```
corrmat = df[['PRCP', 'TMIN', 'TMAX']].corr()
corrmat
```

	PRCP	TMIN	TMAX
PRCP	1.000000	-0.064404	-0.226765
TMIN	-0.064404	1.000000	0.860684
TMAX	-0.226765	0.860684	1.000000

Діккі-Фюллєр

dickey_fuller_test(df['PRCP'].loc['2010-01-01':])

```
adf:  -7.492778701855627
p-value:  4.460052434093296e-11
Critical values:  {'1%': -3.432621827844222, '5%': -2.862543739830824, '10%': -2.5673043196836782}
There are no unit roots, the series is stationary.
```

Будуємо модель ARIMA для передбачення опадів

train_data = df['PRCP'].loc['2010-01-01':]

train_data.describe()

```
count    2905.000000
mean        0.119194
std         0.256682
min         0.000000
25%         0.000000
50%         0.000000
75%         0.120000
max         2.200000
Name: PRCP, dtype: float64
```

train_data

```
DATE
2010-01-01    0.40
2010-01-02    0.06
2010-01-03    0.03
2010-01-04    0.98
2010-01-05    0.14
              ...
2017-12-10    0.00
2017-12-11    0.00
2017-12-12    0.00
2017-12-13    0.00
2017-12-14    0.00
Name: PRCP, Length: 2905, dtype: float64
```

model = smt.ARIMA(train_data, order=(3, 0, 1)).fit()

model.summary()

0.0s

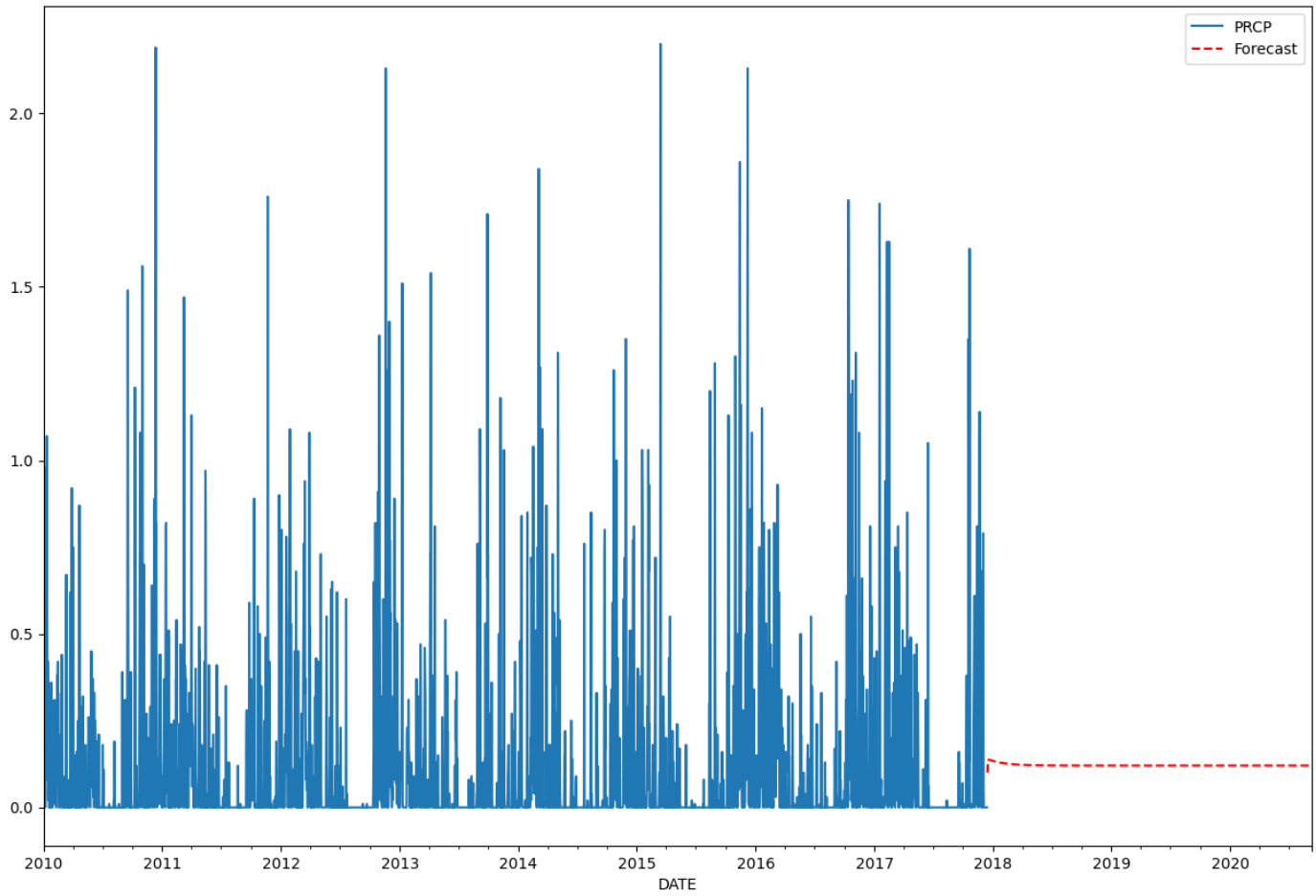
SARIMAX Results

Dep. Variable:	PRCP	No. Observations:	2905
Model:	ARIMA(3, 0, 1)	Log Likelihood	16.392
Date:	Fri, 28 Apr 2023	AIC	-20.783
Time:	00:43:41	BIC	15.062
Sample:	01-01-2010	HQIC	-7.869
	- 12-14-2017		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
const	0.1205	0.027	4.476	0.000	0.068	0.173
ar.L1	1.2252	0.017	70.135	0.000	1.191	1.259
ar.L2	-0.2058	0.020	-10.527	0.000	-0.244	-0.168
ar.L3	-0.0342	0.016	-2.182	0.029	-0.065	-0.003
ma.L1	-0.9496	0.014	-67.604	0.000	-0.977	-0.922
sigma2	0.0579	0.001	66.824	0.000	0.056	0.060

Ljung-Box (L1) (Q):	0.00	Jarque-Bera (JB):	31850.98
Prob(Q):	0.99	Prob(JB):	0.00
Heteroskedasticity (H):	1.36	Skew:	3.26
Prob(H) (two-sided):	0.00	Kurtosis:	17.85

```
fig, ax = plt.subplots(figsize=(15, 10))
ax = train_data.plot(ax=ax)
forecast = model.forecast(steps=1000)
forecast.plot(ax=ax, style='r--', label='Forecast')
plt.legend()
plt.show()
```



Висновок

За отриманими даними можна зробити висновок, що

- В основному завданні було проаналізовано пандемію у Польщі та Німеччині, де помітно час хвиль вірусу. Було проаналізовано гривню до долара. Долар є однією зі стійкіших валют, при тому гривня постійно приймали на себе катаклізми та тільки спадала, але не залежно від прогнозу точно зросте.

- В додатковому завданні був зроблений прогноз на 1000 шагів з 2018 року.