```
# 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



# Load the Covid-19 data for Poland and Germany from a CSV file
covid_data = pd.read_csv('data.csv', sep=',', decimal=',', encoding='windows-1251')

# p_covid = covid_data[covid_data['countriesAndTerritories'] == 'Poland']
# g_covid = covid_data[covid_data['countriesAndTerritories'] == 'Germany']
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')


covid_data.info()

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()}")
poland_covid_c = poland_covid['cases']
poland_covid_c.describe()


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



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()

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'])
# Fetch the exchange rate data for hryvnia/dollar and hryvnia/euro from the National Bank of Ukraine website
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



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

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

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

```
test_data = currencies_price_df_diff[-7:]



forecasts = model.forecast(7)
forecasts
```

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

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

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

plt.show()
df.loc[df.index[-1800:]].plot(figsize=(20, 15), subplots=True)
plt.show()
df[['PRCP', 'TMIN', 'TMAX']].loc[df.index[-1800:]].resample('W').mean().plot(figsize=(20, 15), subplots=True)
plt.show()
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()
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)
df[['TMIN', 'TMAX']] = (df[['TMIN', 'TMAX']] - 32) * 5 / 9
df[['TMIN', 'TMAX']].describe()
corrmat = df[['PRCP', 'TMIN', 'TMAX']].corr()

corrmat
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(df['PRCP'].loc['2010-01-01':])
train_data = df['PRCP'].loc['2010-01-01':]
train_data.describe()
model = smt.ARIMA(train_data, order=(3, 0, 1)).fit()
model.summary()
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()
```