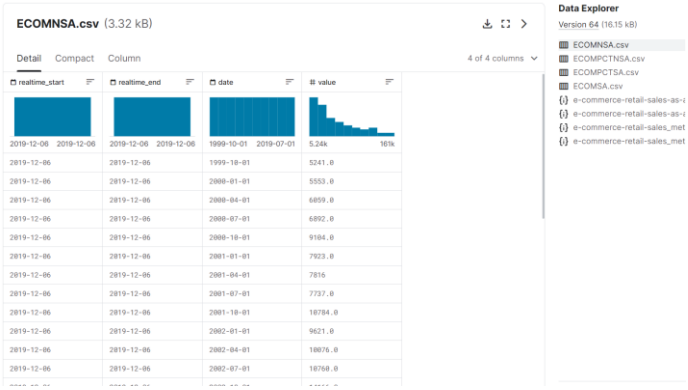
Report для тестового завдання компанії Uvik. Виконав Колесник Андрій.

Для початку я розглянув набір даних, наданий для аналізу та моделювання. Набір даних містив в собі дані для аналізу і прогнозування часових рядів для роздрібних продаів електронної комерції. Він містив 4 набори, з різним характером даних, 2 з сезонністю(в одному відсоткові, а в іншому абсолютні значення) та 2 без сезонності(аналогічно відсотковий і абсолютний).

Але для цікавості я обрав набір ECOMNSA, бо він має сезонність та абсолютні значення.



Загрузка даних та аналіз.

Отримавши набір, я подивився на їх структуру, метадані та стовпці. Далі я перевів мітку часу в формат datetime, та виніс час початку і час кінця вимірювання даних на випадок, якщо вони знадобляться. На виході, я виділив дату і значення в цю дату.

Дані представляють собою прибуток для е комерції поквартально для кожного кварталу, тобто маємо тільки 01.01.рік, 01.04.рік, 01.07.рік, 01.10.рік значення для дати, я зробив це індексом та задав частоту.

```
import numpy as np
import pandas as pd
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

import warnings
warnings.simplefilter('ignore')
```

```
data=pd.read_csv('ECOMNSA.csv')
meta=pd.read_json('e-commerce-retail-sales_metadata.json')['seriess'][0]
meta
```

```
{'id': 'ECOMNSA',
 'realtime_start': '2019-12-06',
 'realtime_end': '2019-12-06',
 'title': 'E-Commerce Retail Sales',
 'observation_start': '1999-10-01',
 'observation_end': '2019-07-01',
 'frequency': 'Quarterly',
 'frequency_short': 'Q',
 'units': 'Millions of Dollars',
 'units_short': 'Mil. of $',
 'seasonal_adjustment': 'Not Seasonally Adjusted',
 'seasonal_adjustment_short': 'NSA',
 'last_updated': '2019-11-19 09:16:02-06',
 'popularity': 31,
 'notes': 'E-commerce sales are sales of goods and services where  the buyer places an order, or the price and terms  of the sa
le  are negotiated over an Internet, mobile device  (M-commerce),  extranet,  Electronic  Data  Interchange  (EDI)  network,  e
lectronic  mail,  or  other  comparable  online  system.  Payment  may  or  may  not  be  made online.'}
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 80 entries, 0 to 79
Data columns (total 4 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   realtime_start  80 non-null     object
 1   realtime_end    80 non-null     object
 2   date            80 non-null     object
 3   value           80 non-null     float64
dtypes: float64(1), object(3)
memory usage: 2.6+ KB
```

```
data.head(7)
```

	realtime_start	realtime_end	date	value
0	2019-12-06	2019-12-06	1999-10-01	5241.0
1	2019-12-06	2019-12-06	2000-01-01	5553.0
2	2019-12-06	2019-12-06	2000-04-01	6059.0
3	2019-12-06	2019-12-06	2000-07-01	6892.0
4	2019-12-06	2019-12-06	2000-10-01	9104.0
5	2019-12-06	2019-12-06	2001-01-01	7923.0
6	2019-12-06	2019-12-06	2001-04-01	7816.0

```
data['date']=pd.to_datetime(data['date'])
realtime_start=data.realtime_start[0]
realtime_end=data.realtime_end[0]
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 80 entries, 1999-10-01 to 2019-07-01
Freq: QS-OCT
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   value   80 non-null     float64
dtypes: float64(1)
memory usage: 1.2 KB
```

```
df=data[['date','value']]
df.set_index("date",inplace=True)
df.index.freq = 'QS-OCT'
df.info()
```

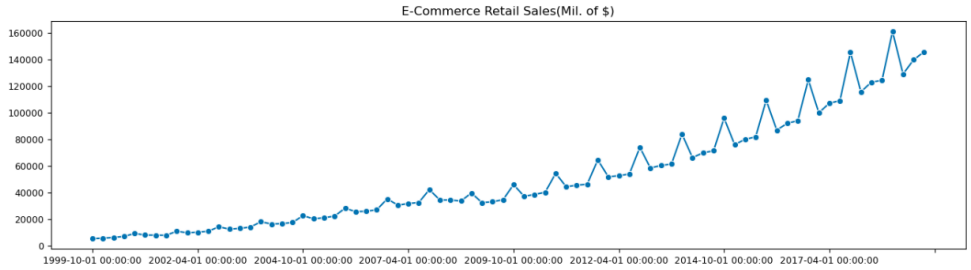
```
df.head(7)
```

	value
date	
1999-10-01	5241.0
2000-01-01	5553.0
2000-04-01	6059.0
2000-07-01	6892.0
2000-10-01	9104.0
2001-01-01	7923.0
2001-04-01	7816.0

Далі я вирішив візуалізувати дані та проаналізувати характер даних.

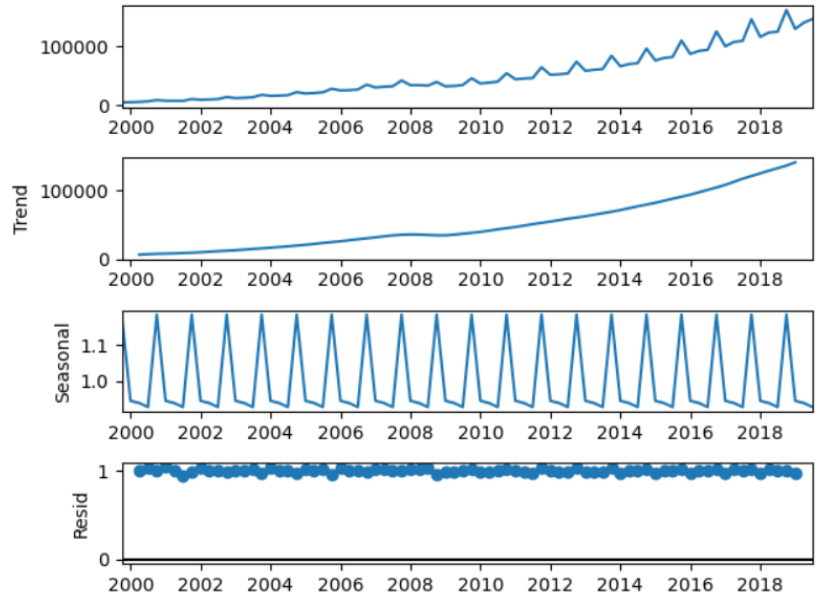
```
from sktime.utils.plotting import plot_series
import matplotlib.pyplot as plt

plt.figure(figsize=(18,5))
plot_series(df)
plt.title('E-Commerce Retail Sales(Mil. of $)')
plt.tick_params(labelsize=9)
```

```
<Figure size 1800x500 with 0 Axes>
```



Ми бачимо, що є явна сезонність та тренд. Сезон 4 квартали, 1 рік, де кожен 3 квартал має найбільше значення. Для підтвердження цього я зробив декомпозицію (мультиплікативна, бо збільшується амплітуда в сезоні). Тренд це зростання, але ми бачимо перегиб, скоріш за все це зв'язано з світовою кризою в 2008.

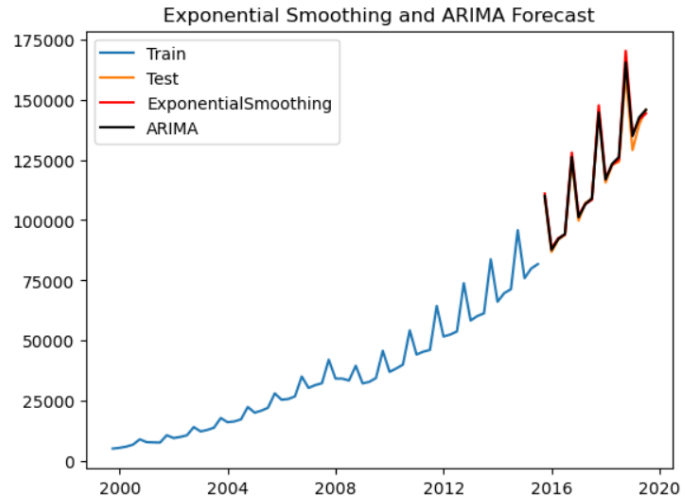
```
import statsmodels.tsa.seasonal as decomp

decomp_series=decomp.seasonal_decompose(df,model='multiplicative')
decomp_series.plot()
plt.show()
```



Далі, перед мною стало питання вибору алгоритму для прогнозування, це могли бути наївний баєс або більш складні типу Рекурентних нейромереж. Але з різних причин я обрав ARIMA та ExponentialSmoothing.

Я обрав тренувальний набір як 0.8 частину, і 0.2 як тестову відповідно. Для метрики оцінки я обрав метрику MAPE(mean absolute percentage error).

```
import pandas as pd
from statsmodels.tsa.holtwinters import ExponentialSmoothing
import matplotlib.pyplot as plt
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_absolute_percentage_error
from sklearn.metrics import mean_squared_error

train_size = int(len(df) * 0.8)
train, test = df[:train_size], df[train_size:]

ES = ExponentialSmoothing(train, trend="mul", seasonal="mul", seasonal_periods=4).fit()
ES_predictions = ES.forecast(steps=len(test))
ES_error=mean_absolute_percentage_error(test,ES_predictions)

Arima_model = ARIMA(train, order=(1, 1, 0), seasonal_order=(2, 1, 2, 4)).fit()
ARM_predictions = Arima_model.forecast(steps=len(test))
ARM_error=mean_absolute_percentage_error(test,ARM_predictions)

# start_date = y_test.index.min()
# end_date = y_test.index.max()
# predictions = model_fit.predict(start=start_date, end=end_date)


plt.plot(train.index, train.values, label='Train')
plt.plot(test.index, test.values, label='Test')
plt.plot(ES_predictions.index, ES_predictions, label='ExponentialSmoothing', color='red')
plt.plot(ARM_predictions.index, ARM_predictions, label='ARIMA', color='black')
plt.legend()
plt.title('Exponential Smoothing Forecast')
plt.show()

print("ES MAPE ",ES_error)
print("ARM MAPE",ARM_error)
```

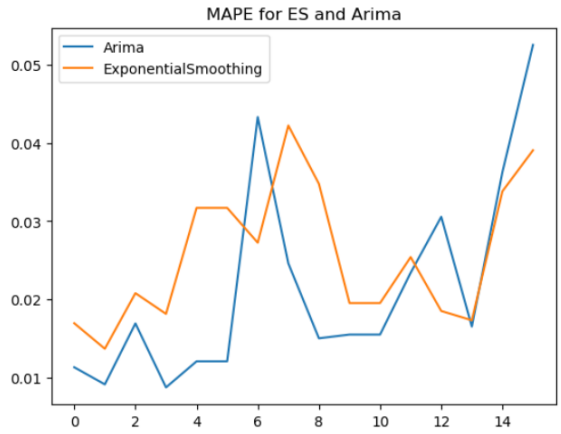


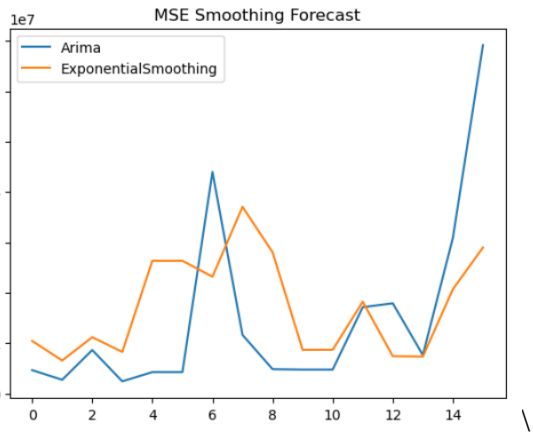

```
ES MAPE  0.016948638345269388
ARM MAPE 0.011315777346619068
```

Видно, що Аріма має дещо меншу помилку, але що буде якщо я збільшу кількість тестових зразків, скажімо, якщо задача це прогнозувати більш довгі послідовності.

Я написав код для обчислення помилки, де тренувальний набір від 0.8 змешується до 0.65 з кроків 0.01

```
ES_error_arr=[]
ES_mean_squared_error=[]

ARM_error_arr=[]
ARM_mean_squared_error=[]

for i in np.arange(.8, .65, -0.01, dtype=float):

    train_size = int(len(df) * i)
    train, test = df[:train_size], df[train_size:]

    ES = ExponentialSmoothing(train, trend="mul", seasonal="mul", seasonal_periods=4).fit()
    ES_predictions = ES.forecast(steps=len(test))

    ES_error_arr.append(mean_absolute_percentage_error(test,ES_predictions))
    ES_mean_squared_error.append(mean_squared_error(test,ES_predictions))


    Arima_model = ARIMA(train, order=(1, 1, 0), seasonal_order=(2, 1, 2, 4)).fit()
    ARM_predictions = Arima_model.forecast(steps=len(test))


    ARM_error_arr.append(mean_absolute_percentage_error(test,ARM_predictions))
    ARM_mean_squared_error.append(mean_squared_error(test,ARM_predictions))
```





\

Arima в більшості випадків має меншу помилку, тому для прогнозування в інтерфейсі я зберіг цю модель.

```
train_size = int(len(df) * 0.8)
train, test = df[:train_size], df[train_size:]

Arima_model = ARIMA(train, order=(1, 1, 0), seasonal_order=(2, 1, 2, 4)).fit()
Arima_model.save('model.pkl')
```

Для інтерфейсу я написав код з використанням Flask, код наведений в файлах main.py, index.html.

В інтерфейсі задається проміжок(рік+дата/квартал), після натискання на кнопку, з’являється графік та значення прогнозів під ним.

Sales forecast by ARIMA

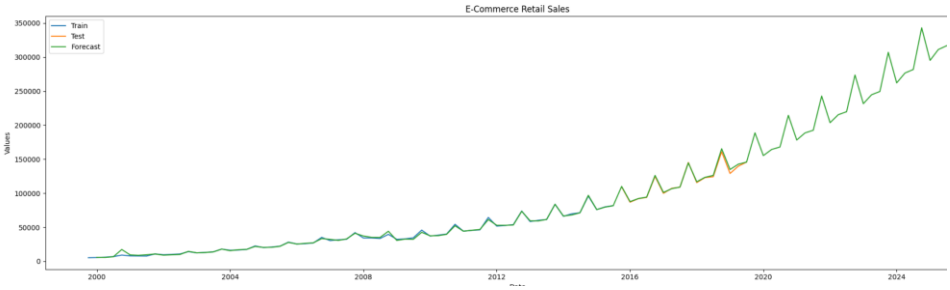
Set a range for forecasting:

From (year 2000-2025): 2000 From (Date for each quartal): 01-01

To (year 2000-2025): 2025 To (Date for each quartal): 07-01

Forecast

Prediction for E-Commerce Retail Sales:



Date	Predicted values
2000-01-01	5783.0
2000-04-01	5639.0
2000-07-01	6638.0
2000-10-01	17301.0
2001-01-01	9385.0
2001-04-01	8800.0
2001-07-01	9583.0
2001-10-01	10712.0
2002-01-01	8966.0
2002-04-01	9506.0
2002-07-01	10003.0
2002-10-01	14562.0
2003-01-01	12425.0
2003-04-01	12839.0
2003-07-01	13738.0
2003-10-01	17881.0
2004-01-01	15570.0
2004-04-01	16956.0
2004-07-01	17505.0
2004-10-01	21806.0
2005-01-01	20344.0
2005-04-01	20530.0
2005-07-01	22001.0
2005-10-01	27739.0
2006-01-01	25354.0
2006-04-01	26366.0
2006-07-01	27107.0
2006-10-01	33314.0
2007-01-01	32293.0