VIST01511
AI & MACHINE LEARNING

Lab 7
Time Series

What you will learn / do in this lab

1. *Analyzing time series data using Pandas*
2. *Analyzing time series data using Statsmodel*
3. *Perform time series data forecasting with various models*

TABLE OF CONTENTS

1. OVERVIEW 1

1.1 Introduction Time Series 1

1.2 Applications of Time series forecast 1

2. ANALYSING TIME SERIES DATA IN PANDAS....................... 3

2.1 Creating time series Index in pandas 3

2.2 Creating time series white noise data 5

2.3 Simulating time series random walk 6

2.4 Summarizing time series data 7

3. ANALYSING TIME SERIES DATA WITH STATSMODEL.......... 9

3.1 Checking on time series stationarity 9

3.2 Make the time series stationary by differencing 13

4. PERFORM TIME SERIES FORECASTING 14

4.1 Build a simple moving average model 14

4.2 Build Holt-Winters exponential smoothing model 15

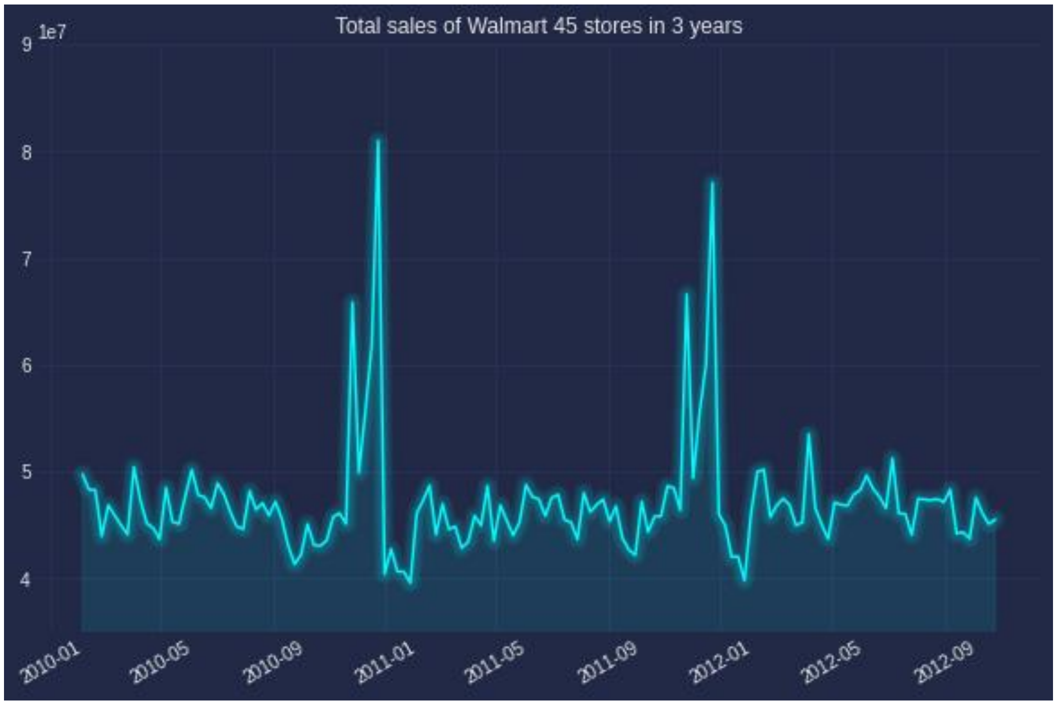
4.3 Build ARIMA forecast model 17

1.
OVERVIEW

In this practical, we will be exploring what is time series data and how to analyze time series data. We will also be building different time series model to perform time series forecasting.

1.1 INTRODUCTION TIME SERIES

Whether we wish to predict the trend in financial markets or electricity consumption, time is an important factor that must be considered in our models. For example, it would be interesting to forecast at what hour during the day is there going to be a peak consumption in electricity, such as to adjust the price or the production of electricity.



1.2 APPLICATIONS OF TIME SERIES FORECAST

The following are a list of applications for time series forecast:

- *Sports streaming platform to forecast online users.*
- *Forecasting traffic with sensor data of number of vehicles.*
- *Online retailer to forecast user-spending habits.*
- *Company to forecast staff turnover rate.*
- *Energy provider to predict user energy consumption.*
- *Many more…*

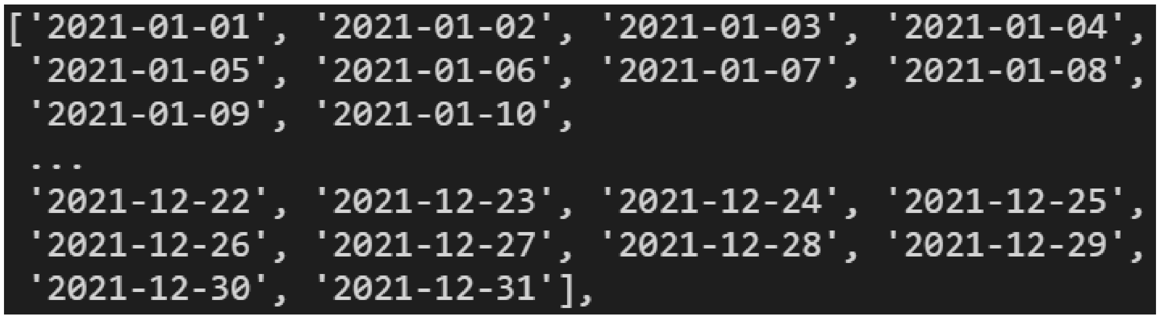
2.
ANALYSING TIME SERIES DATA IN PANDAS

In the section, we will learn how to create time series data using Pandas. Pandas was developed in the context of financial modelling, so as you might expect, it contains an extensive set of tools for working with dates, time, and time-indexed data.

2.1 CREATING TIME SERIES INDEX IN PANDAS

(1) Define a time series index using pandas function, starting from 2021-01-01, ending at 2021-12-31, and with "day" as time interval. Your output should look like below:

```
ts1 = pd.date_range('2021-01-01', '2021-12-01', freq='D')
print(ts1)
```

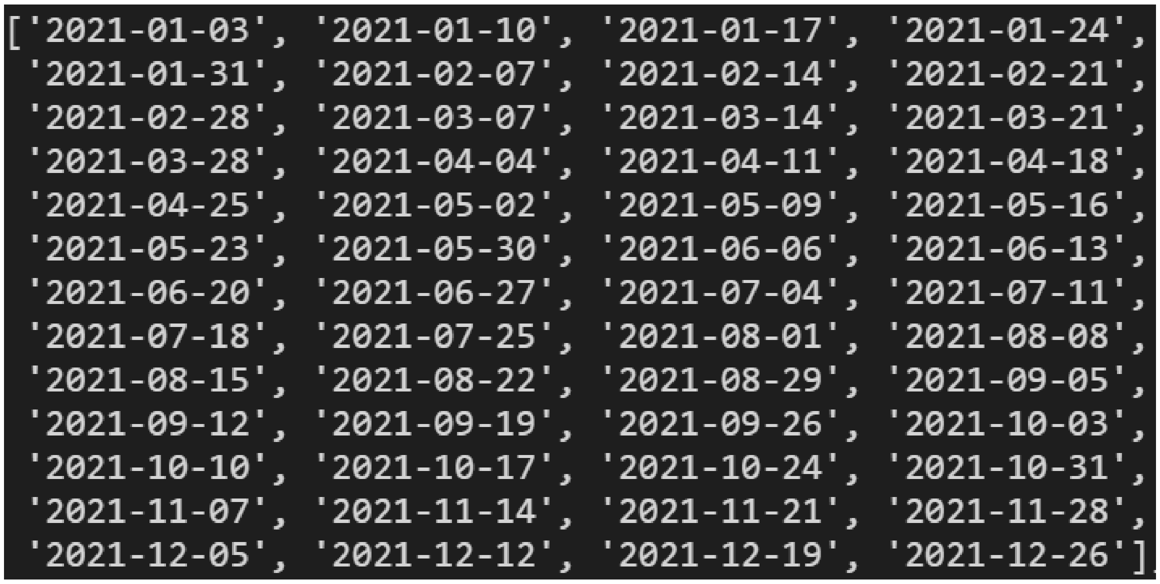


```
['2021-01-01', '2021-01-02', '2021-01-03', '2021-01-04',
 '2021-01-05', '2021-01-06', '2021-01-07', '2021-01-08',
 '2021-01-09', '2021-01-10',
 ...
 '2021-12-22', '2021-12-23', '2021-12-24', '2021-12-25',
 '2021-12-26', '2021-12-27', '2021-12-28', '2021-12-29',
 '2021-12-30', '2021-12-31'],
```

(2) Define a time series index using pandas function, starting from 2021-01-01, ending at 2021-12-31, and with "week" as time interval. Your output should look like below:

```
ts2 = pd.date_range('2021-01-01', '2021-12-01', freq='W')
```

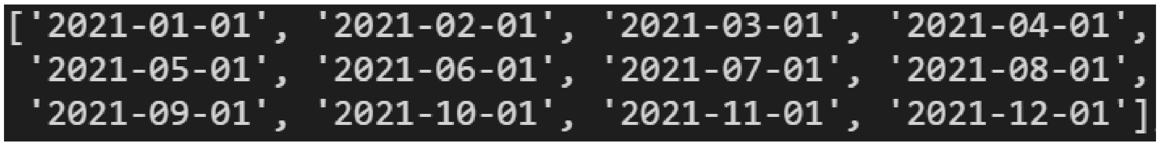
```
print(ts2)
```



```
['2021-01-03', '2021-01-10', '2021-01-17', '2021-01-24',
 '2021-01-31', '2021-02-07', '2021-02-14', '2021-02-21',
 '2021-02-28', '2021-03-07', '2021-03-14', '2021-03-21',
 '2021-03-28', '2021-04-04', '2021-04-11', '2021-04-18',
 '2021-04-25', '2021-05-02', '2021-05-09', '2021-05-16',
 '2021-05-23', '2021-05-30', '2021-06-06', '2021-06-13',
 '2021-06-20', '2021-06-27', '2021-07-04', '2021-07-11',
 '2021-07-18', '2021-07-25', '2021-08-01', '2021-08-08',
 '2021-08-15', '2021-08-22', '2021-08-29', '2021-09-05',
 '2021-09-12', '2021-09-19', '2021-09-26', '2021-10-03',
 '2021-10-10', '2021-10-17', '2021-10-24', '2021-10-31',
 '2021-11-07', '2021-11-14', '2021-11-21', '2021-11-28',
 '2021-12-05', '2021-12-12', '2021-12-19', '2021-12-26']
```

(3) Define a time series index using pandas function, starting from 2021-01-01, ending at 2021-12-01, and with “month” as time interval. Your output should look like below:

```
ts3 = pd.date_range('2021-01-01', '2021-12-01', freq='MS')
print(ts3)
```



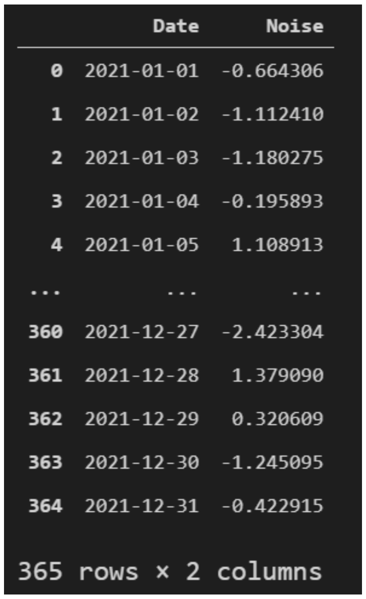
```
['2021-01-01', '2021-02-01', '2021-03-01', '2021-04-01',
 '2021-05-01', '2021-06-01', '2021-07-01', '2021-08-01',
 '2021-09-01', '2021-10-01', '2021-11-01', '2021-12-01']
```

2.2 CREATING TIME SERIES WHITE NOISE DATA

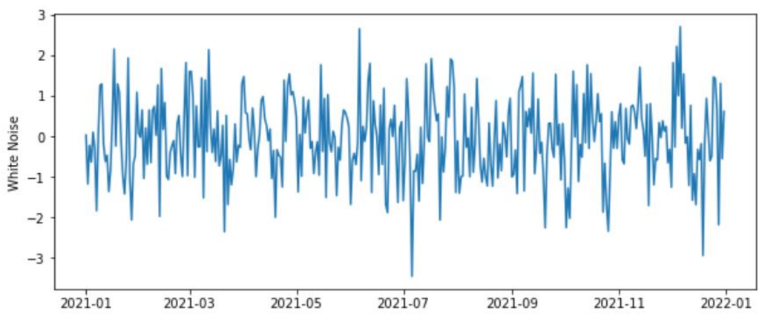
Consider a time series $\{w_t, t = 1, 2, \ldots, n\}$. If the element of the series w_i are independent and identically distributed, with a mean of zero, and no correlation to each other, then we say the time series is white noise.

In the task, you will create a dataframe called df, with two columns, “Date” and “Noise”. “Date” column is timestamp from 2021-01-01 to 2021-12-31, with time interval as day. “Noise” column is white noise data, with mean of 0 and standard deviation of 1 (hint: for white noise data, you can use numpy or random function). After creating dataframe, plot the white noise versus date in a graph to have a look. Your output should look like below

```
df = pd.DataFrame({'Date': pd.date_range('2021-01-01', '2021-12-31', freq='D')
,
                   'Noise': [random.gauss(0, 1) for i in range(365)]})
plt.figure(figsize=(10, 4))
plt.plot(df['Date'], df['Noise'])
plt.ylabel('White Noise')
plt.show()
```



```
           Date      Noise
  0  2021-01-01  -0.664306
  1  2021-01-02  -1.112410
  2  2021-01-03  -1.180275
  3  2021-01-04  -0.195893
  4  2021-01-05   1.108913
...         ...        ...
360  2021-12-27  -2.423304
361  2021-12-28   1.379090
362  2021-12-29   0.320609
363  2021-12-30  -1.245095
364  2021-12-31  -0.422915

365 rows × 2 columns
```



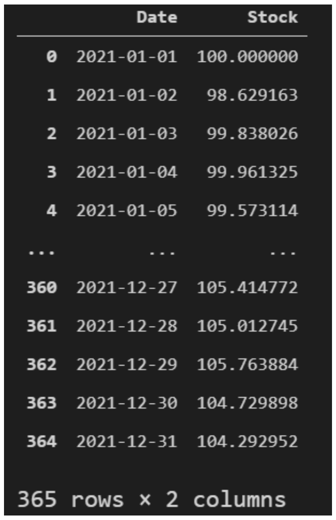
2.3 SIMULATING TIME SERIES RANDOM WALK

A random walk is another time series model where the current observation is equal to the previous observation with a random step up or down.

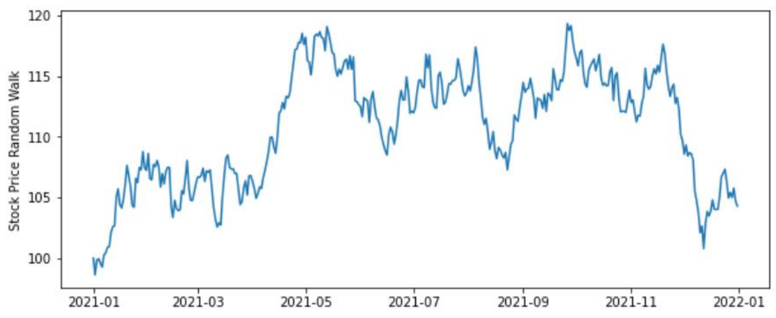
$$x_t = x_{t-1} + w_t, where\ w_t\ is\ white\ noise$$

In this task, you are going to simulate stock price. The date range is from 2021-01-01 to 2021-12-31, with day as frequency. The stock price starts from 100 on day 1, and you will simulate the stock price using random walk. Your output should look like below

```
df = pd.DataFrame({'Date': pd.date_range('2021-01-01', '2021-12-31', freq='D')
,
                   'Noise': [100]+[random.gauss(0, 1) for i in range(364)]})
df['Stock'] = df['Noise'].cumsum()
plt.figure(figsize=(10, 4))
plt.plot(df['Date'], df['Stock'])
plt.ylabel('Stock Price Random Walk')
plt.show()
```



```
           Date       Stock
  0  2021-01-01  100.000000
  1  2021-01-02   98.629163
  2  2021-01-03   99.838026
  3  2021-01-04   99.961325
  4  2021-01-05   99.573114
...         ...         ...
360  2021-12-27  105.414772
361  2021-12-28  105.012745
362  2021-12-29  105.763884
363  2021-12-30  104.729898
364  2021-12-31  104.292952

365 rows × 2 columns
```



2.4 SUMMARIZING TIME SERIES DATA

In this task, you will use the dataset BikeSharing.csv. You are provided hourly bike rental data spanning two years. The description of data fields are as below:

dteday - hourly date + timestamp
season - 1 = spring, 2 = summer, 3 = fall, 4 = winter
holiday - whether the day is considered a holiday
workingday - whether the day is neither a weekend nor holiday
weather - 1: Clear, 2: Mist + Cloudy, 3: Light Snow + Light Rain, 4: Heavy Rain
temp – normalized temperature in Celsius
atemp - normalized "feels like" temperature in Celsius
humidity – normalized relative humidity
windspeed – normalized wind speed
casual - number of non-registered user rentals initiated
registered - number of registered user rentals initiated
count - number of total rentals

you are going to manipulate this time series data with Pandas function, and answer the following questions in order to extract useful insights.

1. *Compute the monthly average temperature.*
2. *Compute the daily sum of casual, register and total rentals.*
3. *Compute the monthly sum of total rentals in different weathers.*
4. *What is the peak hours of bike rental for casual users and registered users, respectively?*

```
# load the dataset
df = pd.read_csv('data/bikesharing.csv')
df['dteday'] = pd.to_datetime(df['dteday'])
df.set_index('dteday', inplace=True)

# insight 1: compute the montly average temperature
monthly_temp = df.resample('M')[['temp']].mean()

# insight 2: compute the daily sum of casual, register and total rentals
daily_rental = df.resample('D')[['casual', 'registered', 'cnt']].sum()

# insight 3: compute the monthly sum of total rentals in different weathers
monthly_rental_weather = df.groupby([pd.Grouper(freq='M'), 'weather'])[['cnt']
].sum()

# insight 4: what is the peak hours of bike rental for casual users and regist
ered users, respectively?
df['hour'] = df.index.hour
peak_reg = df.groupby('hour')[['registered']].mean().sort_values('registered',
 ascending=False)
peak_cas = df.groupby('hour')[['casual']].mean().sort_values('casual', ascendi
ng=False)
```

3.
ANALYSING TIME SERIES DATA WITH STATSMODEL

In the section, we will learn how to analyze time series data with Python Statsmodel. Statsmodel contains models and functions that are useful for time series analysis.

3.1 CHECKING ON TIME SERIES STATIONARITY

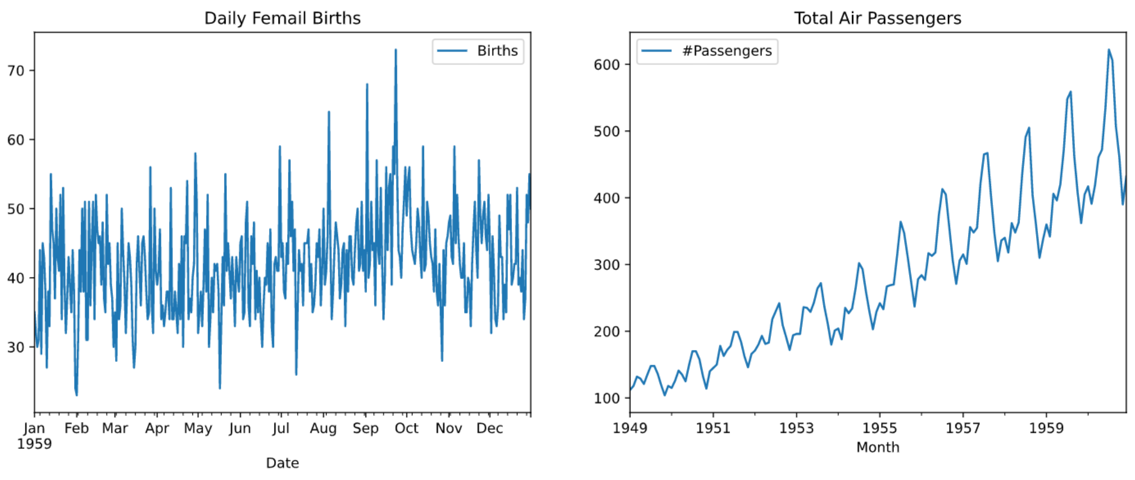
Time series are stationary if they do not have trend or seasonal effects. Summary statistics calculated on the time series are consistent over time, like the mean or the variance of the observations. Load the two datasets **daily-total-female-births.csv** and **AirPassengers.csv**, and use the multiple methods to check whether the datasets are stationary.

```
# load the two datasets
df1 = pd.read_csv('data/daily-total-female-births.csv')
df2 = pd.read_csv('data/airpassengers.csv')
df1['Date'] = pd.to_datetime(df1['Date'])
df2['Month'] = pd.to_datetime(df2['Month'])
df1.set_index('Date', inplace=True)
df2.set_index('Month', inplace=True)
```

1. Visual Check on plots

You are required to use Matplotlib to plot line plots for the two time series datasets. You can review the line plots of your data and visually check if there are any obvious trends or seasonality. Your output should look like below:

```
# Visual Check
fig, ax = plt.subplots(1, 2, figsize=(14, 5))
df1.plot(ax=ax[0])
df2.plot(ax=ax[1])
ax[0].set_title('Daily Femail Births')
ax[1].set_title('Total Air Passengers')
plt.show()
```

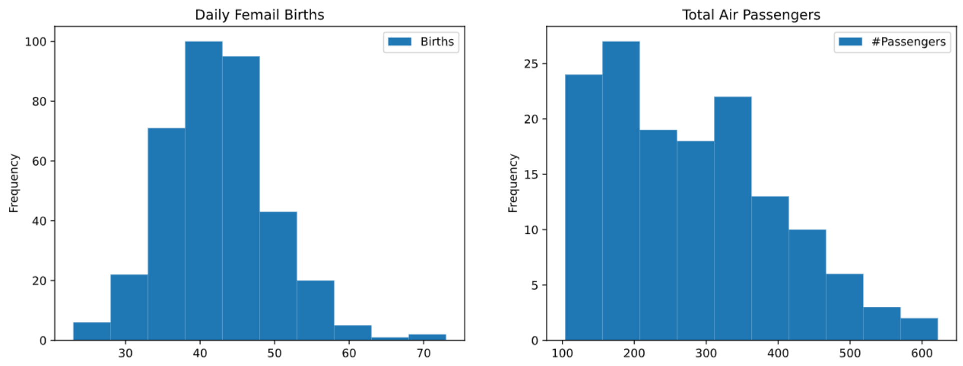


2. *Summary Statistics*

A quick check to see if time series is non-stationary is to review summary statistics.

As a first step, you are required to use Matplotlib to plot histograms for both datasets. If the time series is stationary, the data should conform to Gaussian distribution (bell curve) otherwise it is not.

```
# Summary Statistics
fig, ax = plt.subplots(1, 2, figsize=(14, 5))
df1.plot(ax=ax[0], kind='hist')
df2.plot(ax=ax[1], kind='hist')
ax[0].set_title('Daily Femail Births')
ax[1].set_title('Total Air Passengers')
plt.show()
```



Next, for each of the dataset, you are required to randomly split the dataset into two groups. You are going to calculate the mean and variance of each group and compare the values. Remember for a stationary time series, they should always have the same mean and variance values. Your output should look like below:

```
# Summary Statistics
X1 = df1.iloc[:len(df1)//2, 0]
X2 = df1.iloc[len(df1)//2:, 0]
print(f'For Female Birth Dataset, group 1 mean = {X1.mean():.2f}, group 2 mean = {X1.me
an():.2f}, \ngroup 1 variance = {X1.var():.2f}, group 2 variance = {X2.var():.2f}')
print()
X3 = df2.iloc[:len(df2)//2, 0]
X4 = df2.iloc[len(df2)//2:, 0]
print(f'For Air Passenger Dataset, group 1 mean = {X3.mean():.2f}, group 2 mean = {X4.m
ean():.2f}, \ngroup 1 variance = {X3.var():.2f}, group 2 variance = {X4.var():.2f}')
```

```
For Female Birth Dataset, group 1 mean = 39.76, group 2 mean = 39.76,
group 1 variance = 49.49, group 2 variance = 48.98

For Air Passenger Dataset, group 1 mean = 182.90, group 2 mean = 377.69,
group 1 variance = 2275.69, group 2 variance = 7471.74
```

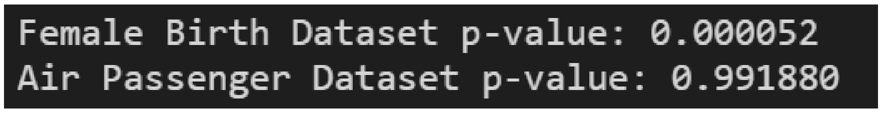
3. *Statistical Test*

The Augmented Dickey-Fuller test is a type of statistical test called a unit root test.

The intuition behind a unit root test is that it determines how strongly a time series is defined by a trend. The null hypothesis of the test is that the time series can be represented by a unit root, that it is not stationary (has some time-dependent structure). The alternate hypothesis (rejecting the null hypothesis) is that the time series is stationary.

You are required to use the function from Statsmodel and to perform Augmented Dickey-Fuller test on both of the time series datasets.

```
# Augmented Dickey-Fuller Test
from statsmodels.tsa.stattools import adfuller

result1 = adfuller(df1['Births'])
result2 = adfuller(df2['#Passengers'])
print('Female Birth Dataset p-value: %f' % result1[1])
print('Air Passenger Dataset p-value: %f' % result2[1])
```



```
Female Birth Dataset p-value: 0.000052
Air Passenger Dataset p-value: 0.991880
```

- *P-value > 0.05: Fail to reject the null hypothesis (H0), the data has a unit root and is **non-stationary**.*
- *P-value <= 0.05: Reject the null hypothesis (H0), the data does not have a unit root and is **stationary**.*

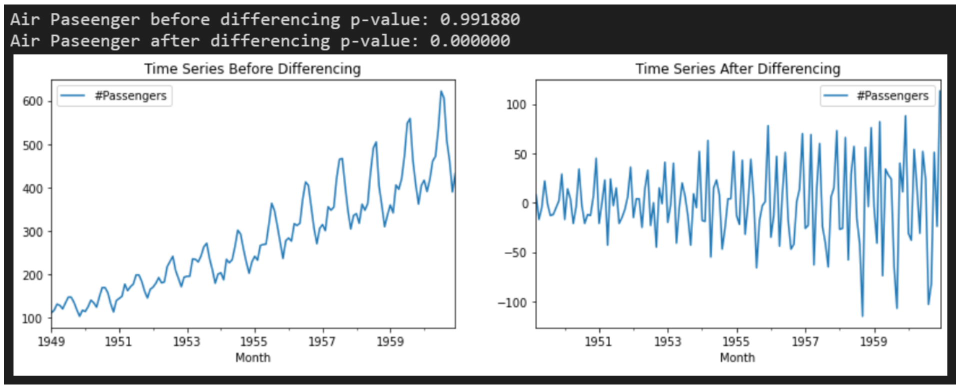
3.2 MAKE THE TIME SERIES STATIONARY BY DIFFERENCING

As we can see from the above stationarity test, the time series data **AirPassengers.csv** is non-stationary. One way to make the time series stationary is by differencing the time series.

Therefore, you are required to use Statsmodel function to difference the time series data.
(Hint: from statsmodels.tsa.statespace.tools import diff)

After the differencing operation, you are required to run Augmented Dickey-Fuller test to see whether the new time series is stationary. Also, you should plot the time series before and after differencing operation. Your output should look similar to below

```
from statsmodels.tsa.statespace.tools import diff
from statsmodels.tsa.stattools import adfuller

df = pd.read_csv('data/airpassengers.csv')
df['Month'] = pd.to_datetime(df['Month'])
df.set_index('Month', inplace=True)

# perform differencing operation
df_diff = diff(df, k_diff=2)
result1 = adfuller(df)
result2 = adfuller(df_diff)

print('Air Paseenger before differencing p-value: %f' % result1[1])
print('Air Paseenger after differencing p-value: %f' % result2[1])

fig, ax = plt.subplots(1, 2, figsize=(14, 4))
df.plot(ax=ax[0])
df_diff.plot(ax=ax[1])
ax[0].set_title('Time Series Before Differencing')
ax[1].set_title('Time Series After Differencing')
plt.show()
```



```
Air Paseenger before differencing p-value: 0.991880
Air Paseenger after differencing p-value: 0.000000
```

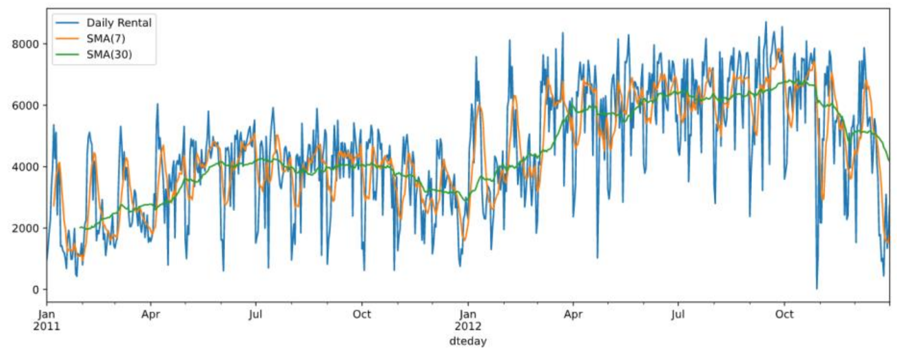
4. PERFORM TIME SERIES FORECASTING

4.1 BUILD A SIMPLE MOVING AVERAGE MODEL

In this task, you are required to build a simple moving average model on the bikesharing.csv dataset. The purpose of the simple moving average is to smooth the fluctuating time series data, in order to visualize trends and generate insights.

Step 1: Calculate the daily total bike rental (using groupby function)
Step 2: Plot the daily total bike rental data, and see if you can identify any trend or insights.
Step 3: Produce a 7-day moving average and a 30-day moving average data and plot on top of the daily total bike rental data. Can you see any trend and insight this time?

```
df = pd.read_csv('data/bikesharing.csv')
df['dteday'] = pd.to_datetime(df['dteday'])
df.set_index('dteday', inplace=True)

daily_rental = df.resample('D')[['cnt']].sum()
ax = daily_rental.plot(figsize=(14, 5))
daily_rental.rolling(window=7).mean().plot(ax=ax)
daily_rental.rolling(window=30).mean().plot(ax=ax)
ax.legend(labels=['Daily Rental', 'SMA(7)', 'SMA(30)'])
plt.show()
```



4.2 BUILD HOLT-WINTERS EXPONENTIAL SMOOTHING MODEL

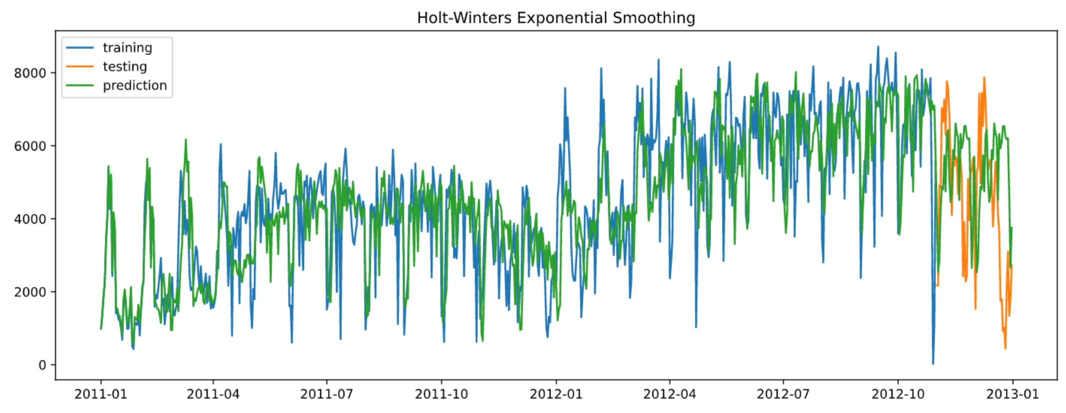
In this task, you are required to build a Holt-Winters exponential smoothing model on the bikesharing.csv dataset. The model should be able to capture historical data and predict future daily bike rental.

In order to verify the model accuracy, you should use data from 2011-01-01 to 2012-10-31 as your training data, and data from 2012-11-01 to 2012-12-31 as testing data.

Using Holt-Winters exponential smoothing model, your output should look similar to below:
(Hint: from statsmodels.tsa.holtwinters import ExponentialSmoothing)

```
# load the dataset
df = pd.read_csv('data/bikesharing.csv')
df['dteday'] = pd.to_datetime(df['dteday'])
df.set_index('dteday', inplace=True)

# group and calculate daily rental
daily_rental = df.resample('D')[['cnt']].sum()

# define training and testing dataset
train_data = daily_rental[daily_rental.index<'2012-11']
test_data = daily_rental[daily_rental.index>='2012-11']
```

```
# Exponential Smoothing Model
from statsmodels.tsa.holtwinters import ExponentialSmoothing as HWES
from sklearn.metrics import mean_absolute_percentage_error

es_model = HWES(train_data, seasonal_periods=31, trend='add', seasonal='add',
                damped_trend=True, freq='D').fit()
pred1 = es_model.predict(start='2011-01-01', end='2012-12-31')

fig, ax = plt.subplots(figsize=(14, 5))
ax.plot(train_data, label='training')
ax.plot(test_data, label='testing')
ax.plot(pred1, label='prediction')
plt.legend()
plt.title('Holt-Winters Exponential Smoothing')
plt.show()
```



```
# Evaluation
from sklearn.metrics import mean_absolute_percentage_error

mape_train = mean_absolute_percentage_error(train_data, pred1[pred1.index<'2012-11'])
mape_test = mean_absolute_percentage_error(test_data, pred1[pred1.index>='2012-11'])
print(f'Model Mean Absolute Percentage Error on training data is {mape_train*100:.2f}%'
)
print(f'Model Mean Absolute Percentage Error on testing data is {mape_test*100:.2f}%')
```

```
Model Mean Absolute Percentage Error on training data is 76.50%
Model Mean Absolute Percentage Error on testing data is 90.48%
```

4.3 BUILD ARIMA FORECAST MODEL

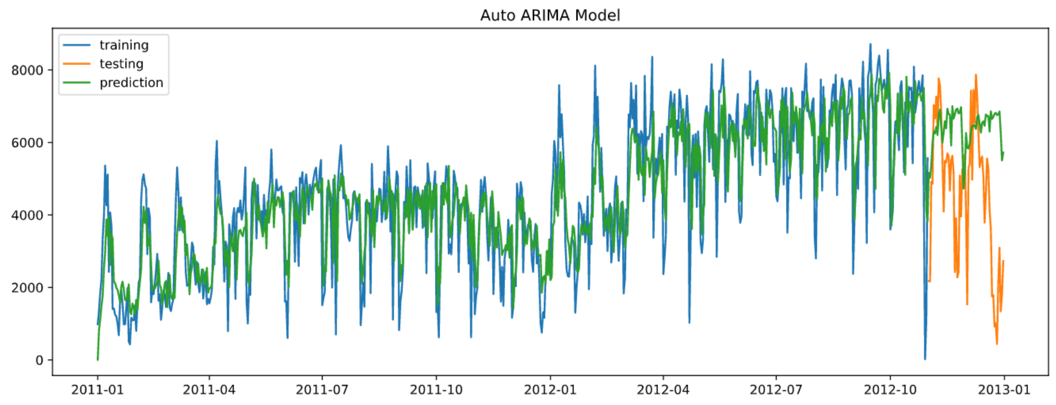
In this task, you are required to build a Seasonal ARIMA model on the bikesharing.csv dataset. The model should be able to capture historical data and predict future daily bike rental.

In order to verify the model accuracy, you should use data from 2011-01-01 to 2012-10-31 as your training data, and data from 2012-11-01 to 2012-12-31 as testing data.

Using Auto ARIMA model, you output should look similar as below.

```
# load the dataset
df = pd.read_csv('data/bikesharing.csv')
df['dteday'] = pd.to_datetime(df['dteday'])
df.set_index('dteday', inplace=True)

# group and calculate daily rental
daily_rental = df.resample('D')[['cnt']].sum()

# define training dataset
train_data = daily_rental[daily_rental.index<'2012-11']
test_data = daily_rental[daily_rental.index>='2012-11']
```

```
# auto-arima model
from pmdarima.arima import auto_arima
arima_model = auto_arima(train_data, seasonal=True, m=30, n_jobs=-1)

# use the model to make predictions
pred_in_sample = arima_model.predict_in_sample()
pred_out_sample = arima_model.predict(n_periods=61)
pred_full = np.concatenate((pred_in_sample, pred_out_sample))

# Plot the results
fig, ax = plt.subplots(figsize=(14, 5))
ax.plot(train_data, label='training')
ax.plot(test_data, label='testing')
ax.plot(daily_rental.index, pred_full, label='prediction')
plt.legend()
plt.title('Auto ARIMA Model')
plt.show()
```



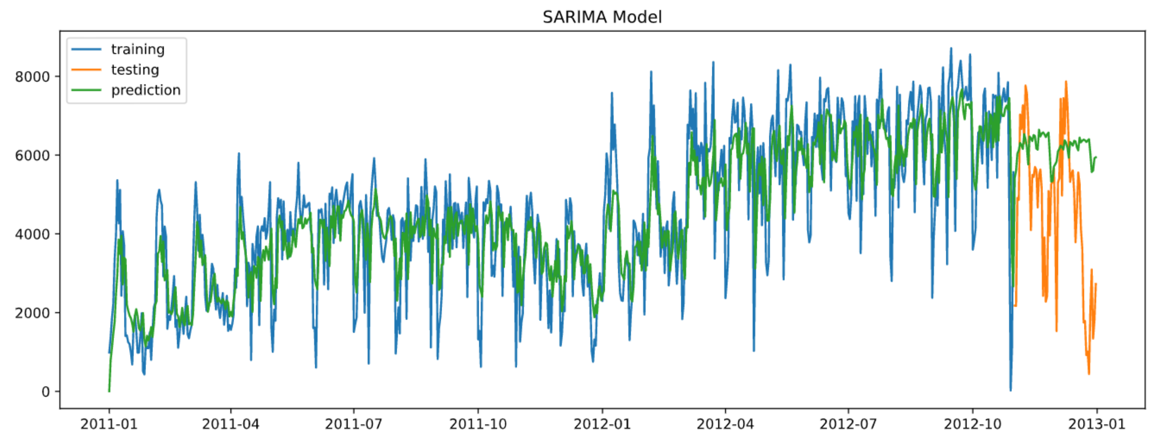
```
# To evaluate the auto arima model
from sklearn.metrics import mean_absolute_percentage_error

mape_train = mean_absolute_percentage_error(train_data, pred_in_sample)
mape_test = mean_absolute_percentage_error(test_data, pred_out_sample)
print(f'Model Mean Absolute Percentage Error on training data is {mape_train*1
00:.2f}%')
print(f'Model Mean Absolute Percentage Error on testing data is {mape_test*100
:.2f}%')
```

```
Model Mean Absolute Percentage Error on training data is 68.13%
Model Mean Absolute Percentage Error on testing data is 105.84%
```

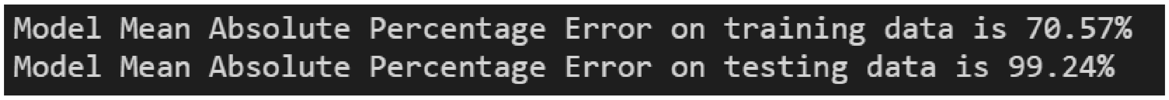
Alternatively, we can use SARIMA model (ARIMA with seasonality), your model output should look similar to below. You can see that the prediction accuracy improves slightly.

```
# load the dataset
df = pd.read_csv('data/bikesharing.csv')
df['dteday'] = pd.to_datetime(df['dteday'])
df.set_index('dteday', inplace=True)

# group and calculate daily rental
daily_rental = df.resample('D')[['cnt']].sum()

# define training dataset
train_data = daily_rental[daily_rental.index<'2012-11']
test_data = daily_rental[daily_rental.index>='2012-11']
```

```
# SARIMAX modeel
from statsmodels.tsa.statespace.sarimax import SARIMAX

sarima_model = SARIMAX(train_data, order=(2,1,1),seasonal_order=(2,0,1,30)).fi
t()
pred = sarima_model.get_prediction(start='2011-01-01', end='2012-12-31').predi
cted_mean

fig, ax = plt.subplots(figsize=(14, 5))
ax.plot(train_data, label='training')
ax.plot(test_data, label='testing')
ax.plot(pred, label='prediction')
plt.legend()
plt.title('SARIMA Model')
plt.show()
```



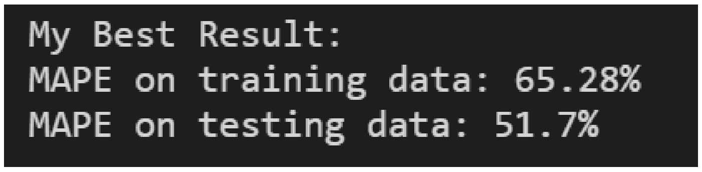
```
# To evaluate the auto arima model
from sklearn.metrics import mean_absolute_percentage_error

mape_train = mean_absolute_percentage_error(train_data, pred[pred.index<'2012-
11'])
mape_test = mean_absolute_percentage_error(test_data, pred[pred.index>='2012-1
1'])
print(f'Model Mean Absolute Percentage Error on training data is {mape_train*1
00:.2f}%')
print(f'Model Mean Absolute Percentage Error on testing data is {mape_test*100
:.2f}%')
```



```
Model Mean Absolute Percentage Error on training data is 70.57%
Model Mean Absolute Percentage Error on testing data is 99.24%
```

4.4 OPTIMIZED YOUR ARIMA FORECAST MODEL (CHALLENGE EXERCISE)

This is an optional challenging task for you to try. How do you optimize your ARIMA forecast model in order to maximize the accuracy on the testing data? Try to break my record!



```
My Best Result:
MAPE on training data: 65.28%
MAPE on testing data: 51.7%
```