Name	**Bodhisatya Ghosh**
UID no.	**2021700026**

Experiment 10	
HONOUR PLEDGE	I hereby declare that the documentation, code and output attached with this lab experiment has been completed by me in accordance with the highest standards of honesty. I confirm that I have not plagiarized OR used unauthorized materials OR given or received illegitimate help for completing this experiment. I will uphold equity and honesty in the evaluation of my work and if found guilty of plagiarism or dishonesty, will bear the consequences as outlined in the 'integrity' section of the lab rubrics. I am doing so in order to maintain a community built around this code of honour. B Ghosh Bodhisatya Ghosh
PROBLEM STATEMENT :	Forecasting using ARIMA(p, d, q) Create an ARIMA forecast model for the stocks dataset used by you in Experiment 8.

	Following things need to be done: 1. Check stationarity of dataset using **Augmented Dickey-Fuller** test. If data is non-stationary, identify the value of 'd' which converts data to stationary data 2. Identify coefficients 'p' and 'q' using Auto-correlation Function (**ACF**) & Partial auto-correlation function (**PACF**) plots 3. Fit an ARIMA model on 80% of the historic data (train) using the p,q and d parameters and use the recent 20% data as 'test' 4. Evaluate the fitted model on various statistical metrics for error on 'train' and 'test' 5. Assess the model on metrics that calculate goodness of fit on 'train' and 'test' 6. Compare the performance of this model with your previously trained OLS model in Experiment 8 7. Compute Theil's coefficient of the 2 forecasts (OLS, ARIMA) for any one stock forecast Add ACF, PACF plots and plots of the Actuals, Predictions and Residuals for each of the stocks
THEORY:	**Augmented Dickey Fuller Test:** Statistical tests make strong assumptions about your data. They can only be used to inform the degree to which a null hypothesis can be rejected or fail to be rejected. The result must be interpreted for a given problem to be meaningful. However, they provide a quick check and confirmatory evidence that the time series is stationary or non-stationary. The *Augmented Dickey-Fuller* test is a type of statistical test called a *unit root test*. In probability theory and statistics, a unit root is a feature of some stochastic

processes (such as random walks) that can cause problems in statistical inference involving time series models. *In simple terms, the unit root is non-stationary* but does not always have a trend component.

ADF test is conducted with the following assumptions:

- *Null Hypothesis (HO)*: Series is non-stationary, or series has a unit root.

- *Alternate Hypothesis(HA)*: Series is stationary, or series has no unit root.

If the null hypothesis fails to be rejected, this test may provide evidence that the series is non-stationary.

Conditions to Reject Null Hypothesis(HO)

- If Test statistic < Critical Value and p-value < 0.05 – *Reject Null Hypothesis(HO), i.e.,* time series does not have a unit root, meaning it is stationary. It does not have a time-dependent structure.

Autocorrelation Function (ACF)

The ACF plot is a graphical representation of the correlation of a time series with itself at different lags. The correlation coefficient is a measure of how closely two variables are related. A correlation coefficient of 1 indicates a perfect positive relationship, while a correlation coefficient of -1 indicates a perfect negative relationship. A correlation coefficient of 0 indicates no relationship between the two variables.

The ACF plot can be used to identify the order of an AR model. The order of an AR model is the number of lags that are included in the model. The ACF plot will show spikes at the lags that are included in the model.

Partial Autocorrelation Function (PACF)

The PACF plot is a graphical representation of the correlation of a time series with itself at different lags, after removing the effects of the previous lags. The PACF plot can be used to identify the order of an MA model. The order of an MA model is the number of lags that are included in the model. The PACF plot will show spikes at the lags that are included in the model.

What is Autoregressive Integrated Moving Average (ARIMA)?

An autoregressive integrated moving average (ARIMA) model is a statistical tool utilized for analyzing time series data, aimed at gaining deeper insights into the dataset or forecasting forthcoming trends.

In this tutorial, We will talk about how to develop an ARIMA model for time series forecasting in Python.

An ARIMA model is a class of statistical models for analyzing and forecasting time series data. It is really simplified in terms of using it, Yet this model is really powerful.

ARIMA stands for Auto-Regressive Integrated Moving Average.

The parameters of the ARIMA model are defined as follows:

- p: The number of lag observations included in the model, also called the lag order.
- d: The number of times that the raw observations are differenced, also called the degree of difference.
- q: The size of the moving average window, also called the order of moving average.

PROGRAM:

exp 10.ipynb

BAP > exp 10 > exp 10.ipynb > Check stationarity of dataset using Augmented Dickey-Fuller test. If data is non-stationary, identify the value of 'd' which converts data to stationary data

```python
import pandas as pd
import numpy as np
import math
import yfinance as yf
import matplotlib.pyplot as plt
```

[8] 0.0s

```python
stock_symbol = 'AAPL'
stock_data = (yf.download(stock_symbol, start="2010-01-01", end=pd.Timestamp.now())).resample('3M').mean()['Open']
```

[9] 0.9s

```
[*********************100%%**********************]  1 of 1 completed
```

Check stationarity of dataset using Augmented Dickey-Fuller test. If data is non-stationary, identify the value of 'd' which converts data to stationary data

```python
from statsmodels.tsa.stattools import adfuller

# Function to perform Augmented Dickey-Fuller test
def adf_test(timeseries):
    result = adfuller(timeseries, autolag='AIC')
    print('ADF Statistic: {:.3f}'.format(result[0]))
    print('p-value: {:.3f}'.format(result[1]))
    print('Critical Values:')
    for key, value in result[4].items():
        print('\t{}: {:.3f}'.format(key, value))

# Perform Augmented Dickey-Fuller test
adf_test(stock_data)
```

[10] 0.6s

```
ADF Statistic: 0.680
p-value: 0.989
Critical Values:
        1%: -3.581
        5%: -2.927
        10%: -2.602
```

```python
# Take first difference to make data stationary
stock_prices_diff = stock_data.diff().dropna()

# Perform Augmented Dickey-Fuller test on differenced data
adf_test(stock_prices_diff)
```

[11] 0.0s

```
ADF Statistic: -0.832
p-value: 0.809
Critical Values:
        1%: -3.581
        5%: -2.927
        10%: -2.602
```

```python
# Take second difference to make data stationary
stock_prices_diff2 = stock_prices_diff.diff().dropna()

# Perform Augmented Dickey-Fuller test on differenced data
adf_test(stock_prices_diff2)
```

[12] 0.0s

```
ADF Statistic: -4.999
p-value: 0.000
Critical Values:
        1%: -3.581
        5%: -2.927
        10%: -2.602
```

d = 2

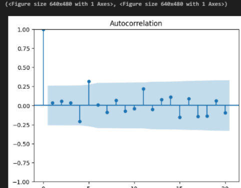
Identify coefficients 'p' and 'q' using Auto-correlation Function (ACF) & Partial auto-correlation function (PACF) plots

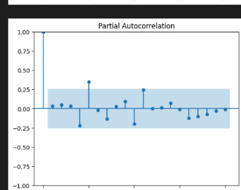
```python
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

# Plot ACF
plot_acf(stock_prices_diff, lags=20),plot_pacf(stock_prices_diff, lags=20)
```

[25] 0.3s

```
(<Figure size 640x480 with 1 Axes>, <Figure size 640x480 with 1 Axes>)
```





p = 5, q = 5

Fit an ARIMA model on 80% of the historic data (train) using the p,q and d parameters and use the recent 20% data as 'test'

```python
from statsmodels.tsa.arima.model import ARIMA

# Split data into train and test sets
train_size = int(len(stock_data) * 0.8)
train, test = stock_data[:train_size], stock_data[train_size:]

# Fit ARIMA model
model = ARIMA(train, order=(2,1,5)) # Example order, replace with identified values
model_fit = model.fit()
print(model_fit.summary())
```

[26] ✓ 0.3s Python

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                   Open   No. Observations:                   46
Model:                 ARIMA(2, 1, 5)   Log Likelihood                -139.567
Date:                Wed, 01 May 2024   AIC                            295.134
Time:                        17:09:21   BIC                            309.588
Sample:                    01-31-2010   HQIC                           300.522
                         - 04-30-2021
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.3481      1.857     -0.187      0.851      -3.988       3.292
ar.L2          0.5136      0.880      0.583      0.560      -1.212       2.239
ma.L1          0.6787      1.988      0.341      0.733      -3.217       4.574
ma.L2         -0.3736      0.468     -0.798      0.425      -1.292       0.544
ma.L3          0.4664      0.275      1.695      0.090      -0.073       1.006
ma.L4          0.6041      1.257      0.481      0.631      -1.860       3.068
ma.L5          0.0259      0.492      0.053      0.958      -0.938       0.989
sigma2        28.0870      7.599      3.696      0.000      13.192      42.982
===================================================================================
Ljung-Box (L1) (Q):                   0.07   Jarque-Bera (JB):                 3.00
Prob(Q):                              0.79   Prob(JB):                         0.22
Heteroskedasticity (H):               5.82   Skew:                             0.33
Prob(H) (two-sided):                  0.00   Kurtosis:                         4.08
===================================================================================
```

Evaluate the fitted model on various statistical metrics for error on 'train' and 'test'

```python
# Evaluate on statistical metrics
train_rmse = np.sqrt(np.mean(train_errors**2))
test_rmse = np.sqrt(np.mean(test_errors**2))
print('Train RMSE:', train_rmse)
print('Test RMSE:', test_rmse)
```

[28] ✓ 0.0s Python

```
Train RMSE: 5.280593529188685
Test RMSE: 20.319001759708826
```

Assess the model on metrics that calculate goodness of fit on 'train' and 'test'

```python
# Assess goodness of fit
train_r_squared = 1 - (np.var(train_errors) / np.var(train))
test_r_squared = 1 - (np.var(test_errors) / np.var(test))
print('Train R-squared:', train_r_squared)
print('Test R-squared:', test_r_squared)
```

[29] ✓ 0.0s Python

```
Train R-squared: 0.9680901085383098
Test R-squared: 0.11335504442920685
```

Compare the performance of this model with your previously trained OLS model in Experiment 8

```python
from sklearn.linear_model import LinearRegression

# Generate features for OLS (e.g., lagged values)
lag = 7  # Example lag value
stock_prices_df = pd.DataFrame(stock_data.copy(), columns=['Open'])
for i in range(1, lag+1):
    stock_prices_df[f'Lag_{i}'] = stock_prices_df['Open'].shift(i)

# Split data into train and test sets
train_size = int(len(stock_prices_df) * 0.8)
train_df, test_df = stock_prices_df.iloc[:train_size], stock_prices_df.iloc[train_size:]

# Drop missing values
train_df.dropna(inplace=True)
test_df.dropna(inplace=True)

# Define features and target for OLS
X_train = train_df.drop(columns=['Open'])
y_train = train_df['Open']
X_test = test_df.drop(columns=['Open'])

# Fit OLS model
ols_model = LinearRegression()
ols_model.fit(X_train, y_train)

# Predictions
ols_train_predictions = ols_model.predict(X_train)
ols_test_predictions = ols_model.predict(X_test)
```

[30] ✓ 0.0s Python

Compute Theil's coefficient of the 2 forecasts (OLS, ARIMA) for any one stock forecast

```python
# Assuming you have another forecast (OLS) available
# Calculate Theil's coefficient
def theil_coefficient(ols_forecast, arima_forecast, actual):
    numerator = np.sqrt(np.mean((arima_forecast - actual)**2))
    denominator = np.sqrt(np.mean((ols_forecast - actual)**2))
    return numerator / denominator

# Compute Theil's coefficient
theil_coeff = theil_coefficient(ols_test_predictions, test_predictions, test)
print('Theil\'s coefficient:', theil_coeff)
```

[31] ✓ 0.0s Python

```
Theil's coefficient: 0.839278680832617
```

<table>
<tr><td>RESULT:</td><td><code>Theil's coefficient: 0.839278680832617</code></td></tr>
<tr><td>References:</td><td>Statistical Tests to Check Stationarity in Time Series (analyticsvidhya.com)

Scraping Reddit Data Using Python and PRAW : A Beginner’s Guide | by Archana Kokate | Mar, 2024 | Medium</td></tr>
<tr><td colspan="2">CONCLUSION:
A Theil's coefficient value of 0.83 indicates that the ARIMA forecast is about 83% as accurate as the OLS (Ordinary Least Squares) forecast when compared against the actual observed values.

Here's what it means in more detail:

- If Theil's coefficient is closer to 1, it suggests that the two forecasts are equally accurate.
- A value less than 1 indicates that the ARIMA forecast is more accurate compared to the OLS forecast.
- Conversely, a value greater than 1 suggests that the ARIMA forecast is less accurate compared to the OLS forecast.

In this case, with a Theil's coefficient of 0.83, it means that the ARIMA forecast is about 83% as accurate as the OLS forecast in predicting the test data. So, while the ARIMA model is still providing reasonably accurate forecasts, the OLS model might be slightly more accurate for this particular dataset and forecasting horizon.</td></tr>
</table>