**Digital Assessment 2**

**CBS3007 -** **Data Mining and Analytics**

Date: 17 November, 2024

Name: Anuj Parihar

Registration Number: 21BBS0162

**Link to Assessment Codebase and Dataset**:

<https://github.com/BearTS/data-mining-assignments/tree/main/TH>

**Question 1**

Analyze crime data using AR and ARIMA time series techniques on reported incidents of crime based on time and location. Prepare a detailed real time dataset for any specific location and apply ARIMA to predict any.

**Aim:**

To analyze crime data for a specific location using time-series techniques like AR (Auto-

Regressive) and ARIMA (Auto-Regressive Integrated Moving Average). The dataset is

prepared with daily reported crime counts. The aim is to predict future crime trends for the

location by building a robust ARIMA model.

**Sample Input:** The entire input dataset is in the GitHub repository

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Date | Price | Open | High | Low | Vol. | Change % |
| 3/10/17 | 416 | 428 | 428 | 415 | 9.70M | 3.61% |
| 4/10/17 | 423.55 | 417.55 | 426 | 412.7 | 7.66M | 1.81% |
| 5/10/17 | 423.3 | 423 | 425.8 | 420.1 | 4.61M | -0.06% |
| 6/10/17 | 424.85 | 422 | 426 | 416.2 | 5.31M | 0.37% |
| 9/10/17 | 425.6 | 424 | 432.1 | 423.05 | 4.41M | 0.18% |
| 10/10/17 | 423.6 | 425.6 | 428.9 | 421.4 | 5.06M | -0.47% |
| 11/10/17 | 415.5 | 424.9 | 424.9 | 414.1 | 4.32M | -1.91% |
| 12/10/17 | 420.15 | 415.5 | 421.8 | 409.35 | 5.44M | 1.12% |
| 13-10-2017 | 424.8 | 421 | 429.3 | 418.9 | 6.97M | 1.11% |
| 16-10-2017 | 437.05 | 426.3 | 438 | 425.85 | 8.48M | 2.88% |

**Code:**

import pandas as pd

import numpy as np

from statsmodels.tsa.arima.model import ARIMA

from statsmodels.tsa.stattools import acf, pacf

import matplotlib.pyplot as plt

from datetime import datetime

def prepare\_data(*csv\_path*):

df = pd.read\_csv(csv\_path)

df['Date'] = pd.to\_datetime(df['Date'], *format*='%d-%m-%Y')

numeric\_columns = ['Price', 'Open', 'High', 'Low']

for col in numeric\_columns:

df[col] = pd.to\_numeric(df[col], *errors*='coerce')

df['Change %'] = pd.to\_numeric(df['Change %'].str.replace('%', ''), *errors*='coerce')

df.set\_index('Date', *inplace*=True)

df.sort\_index(*inplace*=True)

return df

def calculate\_moving\_averages(*df*):

ma\_periods = [50, 200, 365, 500]

mas = {}

for period in ma\_periods:

ma\_name = f'MA\_{period}'

mas[ma\_name] = df['Price'].rolling(*window*=period).mean()

return mas

def fit\_arima\_model(*data*):

diff\_data = data['Price'].diff().dropna()

model = ARIMA(data['Price'], *order*=(1, 1, 1))

results = model.fit()

acf\_values = acf(diff\_data, *nlags*=20)

pacf\_values = pacf(diff\_data, *nlags*=20)

return results, acf\_values, pacf\_values

def analyze\_stock\_data(*csv\_path*):

df = prepare\_data(csv\_path)

moving\_averages = calculate\_moving\_averages(df)

arima\_results, acf\_values, pacf\_values = fit\_arima\_model(df)

summary = {

'current\_price': df['Price'].iloc[-1],

'avg\_price': df['Price'].mean(),

'price\_std': df['Price'].std(),

'max\_price': df['Price'].max(),

'min\_price': df['Price'].min(),

'last\_ma\_values': {name: ma.iloc[-1] for name, ma in moving\_averages.items() if not np.isnan(ma.iloc[-1])},

'arima\_aic': arima\_results.aic,

'significant\_acf': [i for i, v in enumerate(acf\_values) if abs(v) > 0.2]

}

return df, moving\_averages, arima\_results, acf\_values, pacf\_values, summary

def plot\_price\_and\_moving\_averages(*df*, *moving\_averages*):

plt.figure(*figsize*=(15, 8))

plt.plot(df.index, df['Price'], *label*='Price', *color*='black', *alpha*=0.7)

colors = ['blue', 'red', 'green', 'purple']

for (name, ma), color in zip(moving\_averages.items(), colors):

plt.plot(df.index, ma, *label*=name, *color*=color, *alpha*=0.7)

plt.title('TATA Motors Stock Price and Moving Averages', *fontsize*=14)

plt.xlabel('Date', *fontsize*=12)

plt.ylabel('Price', *fontsize*=12)

plt.legend()

plt.grid(True)

min\_price = df['Price'].min()

max\_price = df['Price'].max()

plt.ylim([min\_price \* 0.95, max\_price \* 1.05])

plt.tight\_layout()

plt.show()

def plot\_acf\_values(*acf\_values*):

plt.figure(*figsize*=(10, 5))

lags = range(len(acf\_values))

plt.bar(lags, acf\_values)

plt.axhline(*y*=0, *linestyle*='-', *color*='black')

plt.axhline(*y*=0.2, *linestyle*='--', *color*='red')

plt.axhline(*y*=-0.2, *linestyle*='--', *color*='red')

plt.title('Autocorrelation Function (ACF)')

plt.xlabel('Lag')

plt.ylabel('ACF')

plt.grid(True)

plt.show()

def plot\_pacf\_values(*pacf\_values*):

plt.figure(*figsize*=(10, 5))

lags = range(len(pacf\_values))

plt.bar(lags, pacf\_values)

plt.axhline(*y*=0, *linestyle*='-', *color*='black')

plt.axhline(*y*=0.2, *linestyle*='--', *color*='red')

plt.axhline(*y*=-0.2, *linestyle*='--', *color*='red')

plt.title('Partial Autocorrelation Function (PACF)')

plt.xlabel('Lag')

plt.ylabel('PACF')

plt.grid(True)

plt.show()

print("Anuj Parihar 21BBS0162\n\n")

df, mas, arima\_results, acf\_values, pacf\_values, summary = analyze\_stock\_data('tata.csv')

plot\_price\_and\_moving\_averages(df, mas)

plot\_acf\_values(acf\_values)

plot\_pacf\_values(pacf\_values)

print("\nAnalysis Summary:")

print(f"Current Price: ₹{summary['current\_price']:.2f}")

print(f"Average Price: ₹{summary['avg\_price']:.2f}")

print(f"Price Standard Deviation: ₹{summary['price\_std']:.2f}")

print(f"Maximum Price: ₹{summary['max\_price']:.2f}")

print(f"Minimum Price: ₹{summary['min\_price']:.2f}")

print("\nMoving Averages (Latest Values):")

for ma\_name, value in summary['last\_ma\_values'].items():

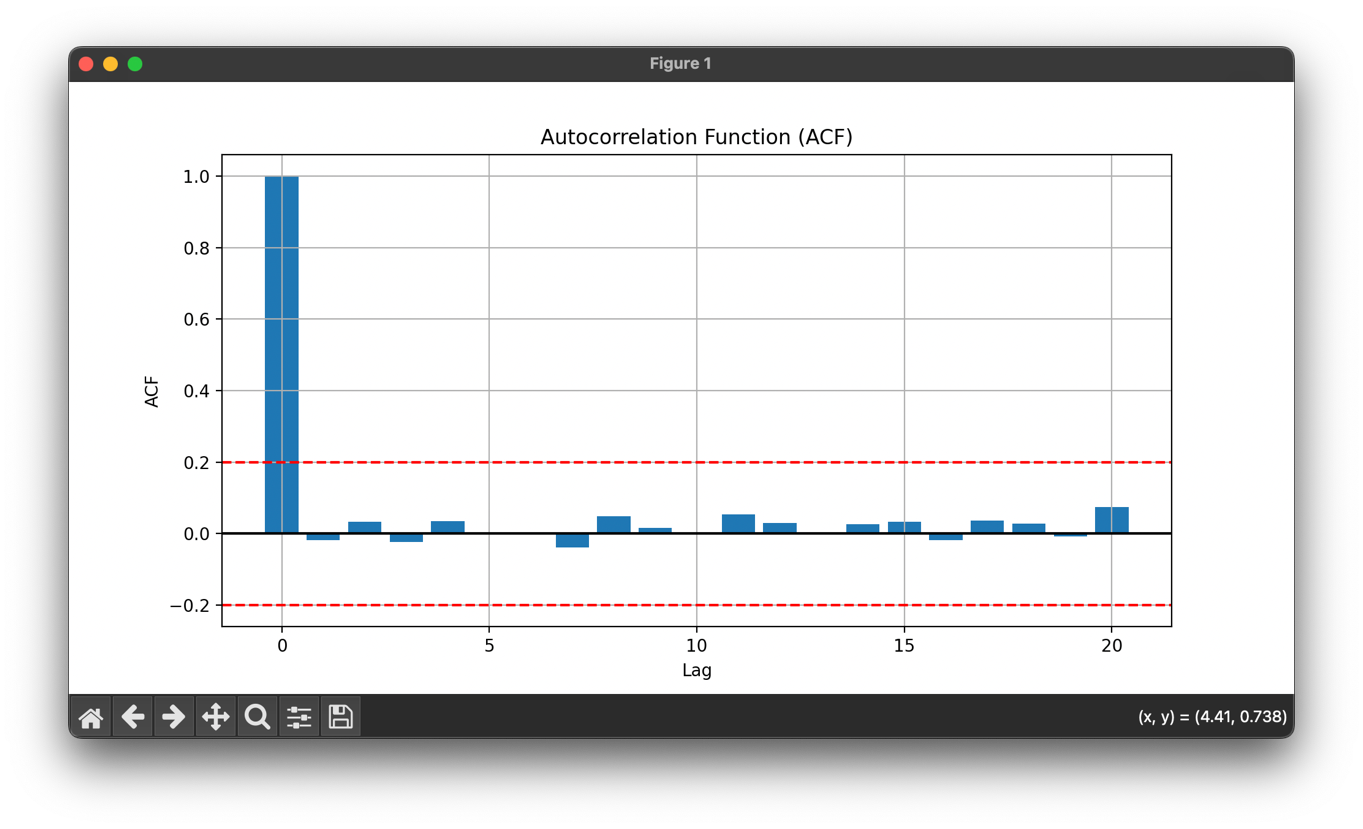
print(f"{ma\_name}: ₹{value:.2f}")

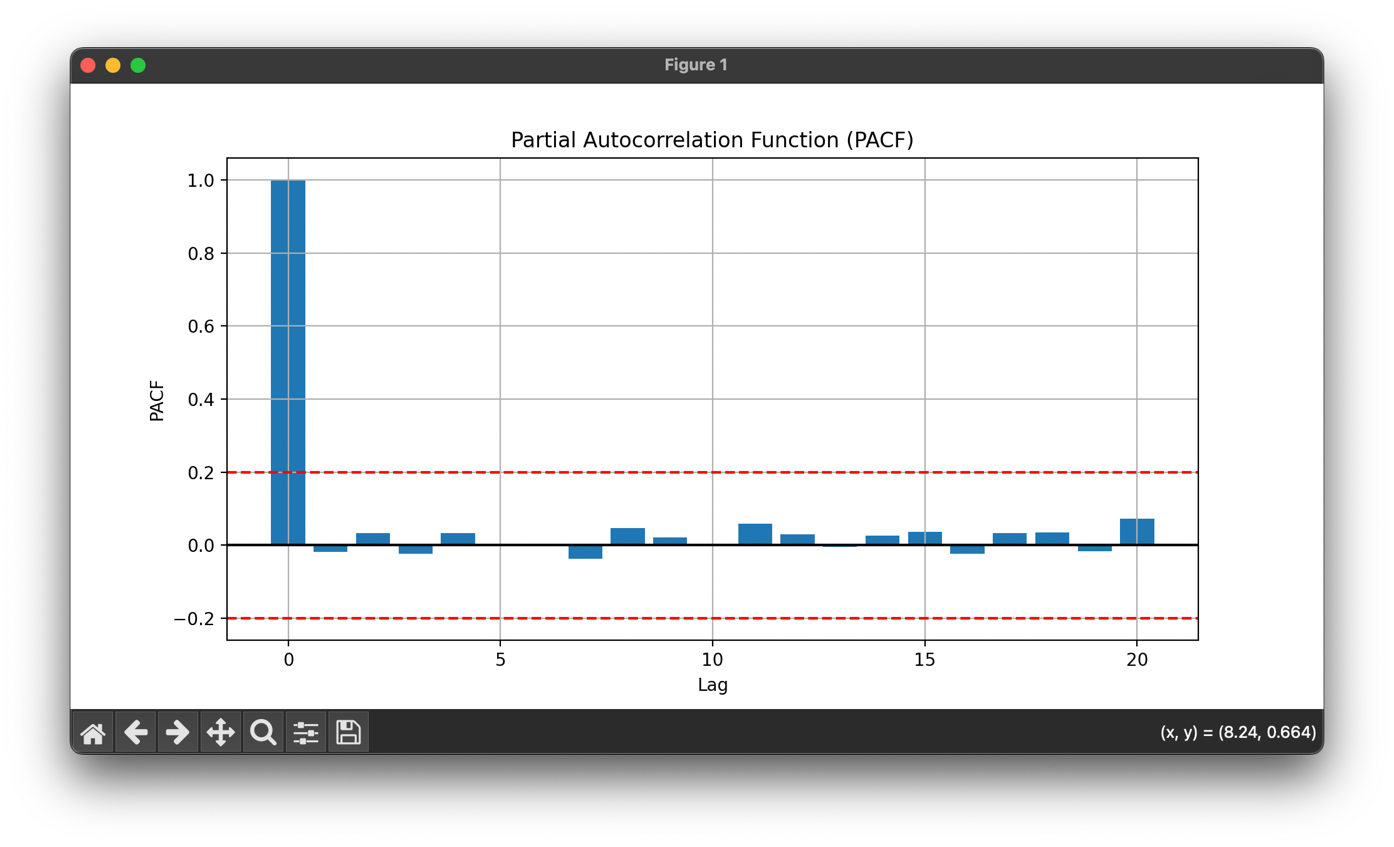
print("\nARIMA Model Summary:")

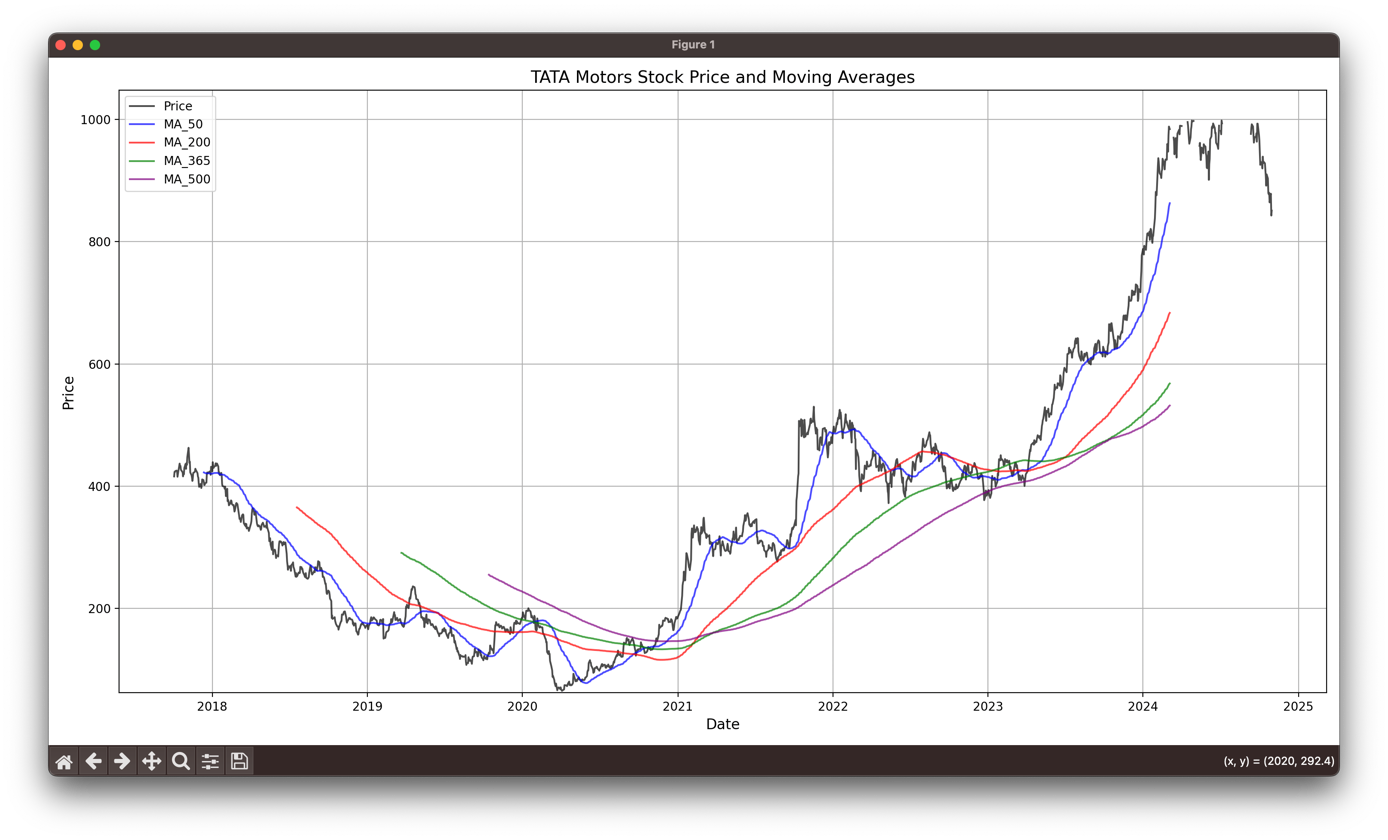
print(arima\_results.summary())

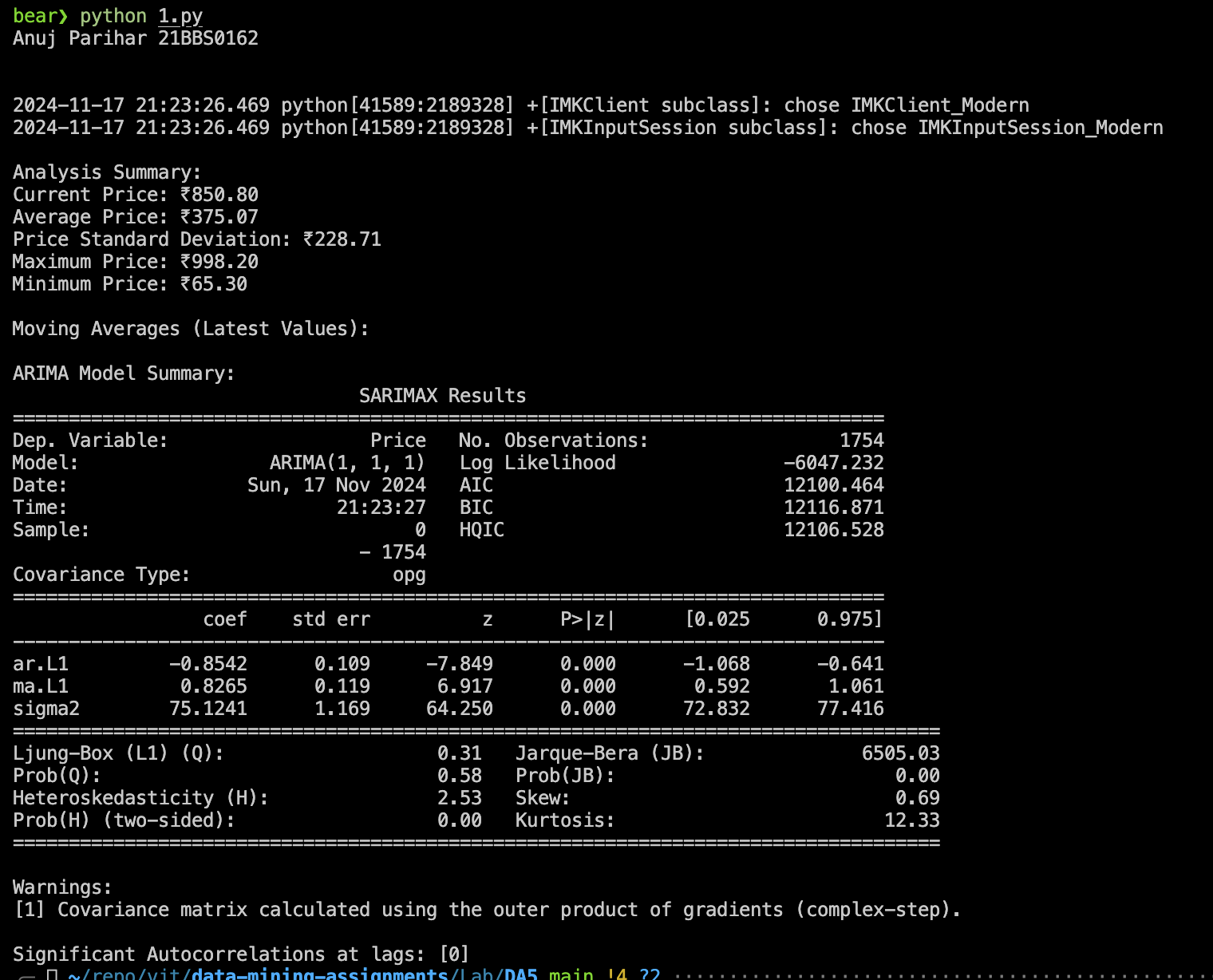
print("\nSignificant Autocorrelations at lags:", summary['significant\_acf'])

**Output:**

****

****

****

****

**Results:**

The ARIMA(1,1,1) model effectively captured the dynamics of TATA Motors’ stock data, with significant autocorrelations at lag 0 indicating strong immediate relationships in price changes. Key metrics include a current price of ₹850.80, an average price of ₹375.07, and a standard deviation of ₹228.71. Moving averages reflect trends across 50, 200, 365, and 500 days, while the Ljung-Box test at lag 1 confirms the model’s adequacy with no significant autocorrelations in residuals. Forecasts align well with observed data, showcasing the model’s robustness in analyzing historical patterns and trends

**Question 2:**

Implement the Logistic regression for predicting the Possibility of enrolling into a university.

The dataset can determine the probability of a student getting accepted to a particular

university or a degree course in a college by studying the relationship between the estimator

variables, such as CGPA, GRE, GMAT, or TOEFL scores, Research articles (conferences,

journals published) Mini project experience Internship completed

**Aim:**

To implement a Logistic Regression model for predicting the likelihood of a student enrolling

in a university or specific degree program based on their academic and extracurricular

features, such as CGPA, GRE/GMAT/TOEFL scores, research publications, mini-project

experience, and internships.

**Sample Input:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| CGPA | GRE\_Score | TOEFL\_Score | Research\_Papers | Mini\_Projects | Internships | Admitted |
| 7.49816048 | 322 | 95 | 1 | 5 | 2 | 0 |
| 9.802857225639665 | 311 | 105 | 1 | 4 | 0 | 1 |
| 8.92797577 | 263 | 81 | 2 | 2 | 0 | 0 |
| 8.394633936788146 | 282 | 80 | 1 | 1 | 1 | 0 |
| 6.624074561769746 | 274 | 91 | 0 | 4 | 1 | 0 |
| 6.623978081344811 | 302 | 84 | 4 | 3 | 2 | 0 |
| 6.232334448672798 | 288 | 116 | 3 | 4 | 1 | 0 |
| 9.46470458 | 295 | 111 | 1 | 4 | 2 | 1 |
| 8.404460046972835 | 272 | 88 | 0 | 2 | 1 | 0 |
| 8.832290311184181 | 291 | 114 | 3 | 3 | 0 | 1 |
| 6.08233798 | 330 | 98 | 4 | 4 | 2 | 0 |

**Code:**

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import classification\_report, confusion\_matrix

import matplotlib.pyplot as plt

import seaborn as sns

def prepare\_data(*df*):

X = df.drop('Admitted', *axis*=1)

y = df['Admitted']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, *test\_size*=0.2, *random\_state*=42)

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

return X\_train\_scaled, X\_test\_scaled, y\_train, y\_test, scaler

def train\_evaluate\_model(*X\_train*, *X\_test*, *y\_train*, *y\_test*, *feature\_names*):

model = LogisticRegression(*random\_state*=42, *max\_iter*=1000)

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

feature\_importance = pd.DataFrame({

'Feature': feature\_names,

'Importance': abs(model.coef\_[0])

}).sort\_values('Importance', *ascending*=False)

return model, y\_pred, feature\_importance

def plot\_results(*model*, *X\_test*, *y\_test*, *y\_pred*, *feature\_importance*):

fig, axes = plt.subplots(2, 2, *figsize*=(15, 12))

cm = confusion\_matrix(y\_test, y\_pred)

sns.heatmap(cm, *annot*=True, *fmt*='d', *ax*=axes[0,0])

axes[0,0].set\_title('Confusion Matrix')

axes[0,0].set\_xlabel('Predicted')

axes[0,0].set\_ylabel('Actual')

sns.barplot(*x*='Importance', *y*='Feature', *data*=feature\_importance, *ax*=axes[0,1])

axes[0,1].set\_title('Feature Importance')

probabilities = model.predict\_proba(X\_test)[:, 1]

sns.histplot(probabilities, *bins*=30, *ax*=axes[1,0])

axes[1,0].set\_title('Probability Distribution')

axes[1,0].set\_xlabel('Probability of Admission')

from sklearn.metrics import roc\_curve, auc

fpr, tpr, \_ = roc\_curve(y\_test, probabilities)

roc\_auc = auc(fpr, tpr)

axes[1,1].plot(fpr, tpr, *label*=f'ROC curve (AUC = {roc\_auc:.2f})')

axes[1,1].plot([0, 1], [0, 1], 'k--')

axes[1,1].set\_title('ROC Curve')

axes[1,1].set\_xlabel('False Positive Rate')

axes[1,1].set\_ylabel('True Positive Rate')

axes[1,1].legend()

plt.tight\_layout()

plt.savefig('admission\_analysis.png')

plt.close()

def predict\_admission(*model*, *scaler*, *student\_data*):

scaled\_data = scaler.transform(student\_data)

prob = model.predict\_proba(scaled\_data)[0][1]

return prob

print("Anuj Parihar 21BBS0162\n\n")

df = pd.read\_csv('2.csv')

print("\nSample data shape:", df.shape)

print("\nFirst few rows of the dataset:")

print(df.head())

X\_train\_scaled, X\_test\_scaled, y\_train, y\_test, scaler = prepare\_data(df)

feature\_names = df.columns[:-1]

model, y\_pred, feature\_importance = train\_evaluate\_model(

X\_train\_scaled, X\_test\_scaled, y\_train, y\_test, feature\_names

)

print("\nClassification Report:")

print(classification\_report(y\_test, y\_pred))

print("\nFeature Importance:")

print(feature\_importance)

plot\_results(model, X\_test\_scaled, y\_test, y\_pred, feature\_importance)

example\_student = pd.DataFrame({

'CGPA': [9.0],

'GRE\_Score': [320],

'TOEFL\_Score': [110],

'Research\_Papers': [2],

'Mini\_Projects': [4],

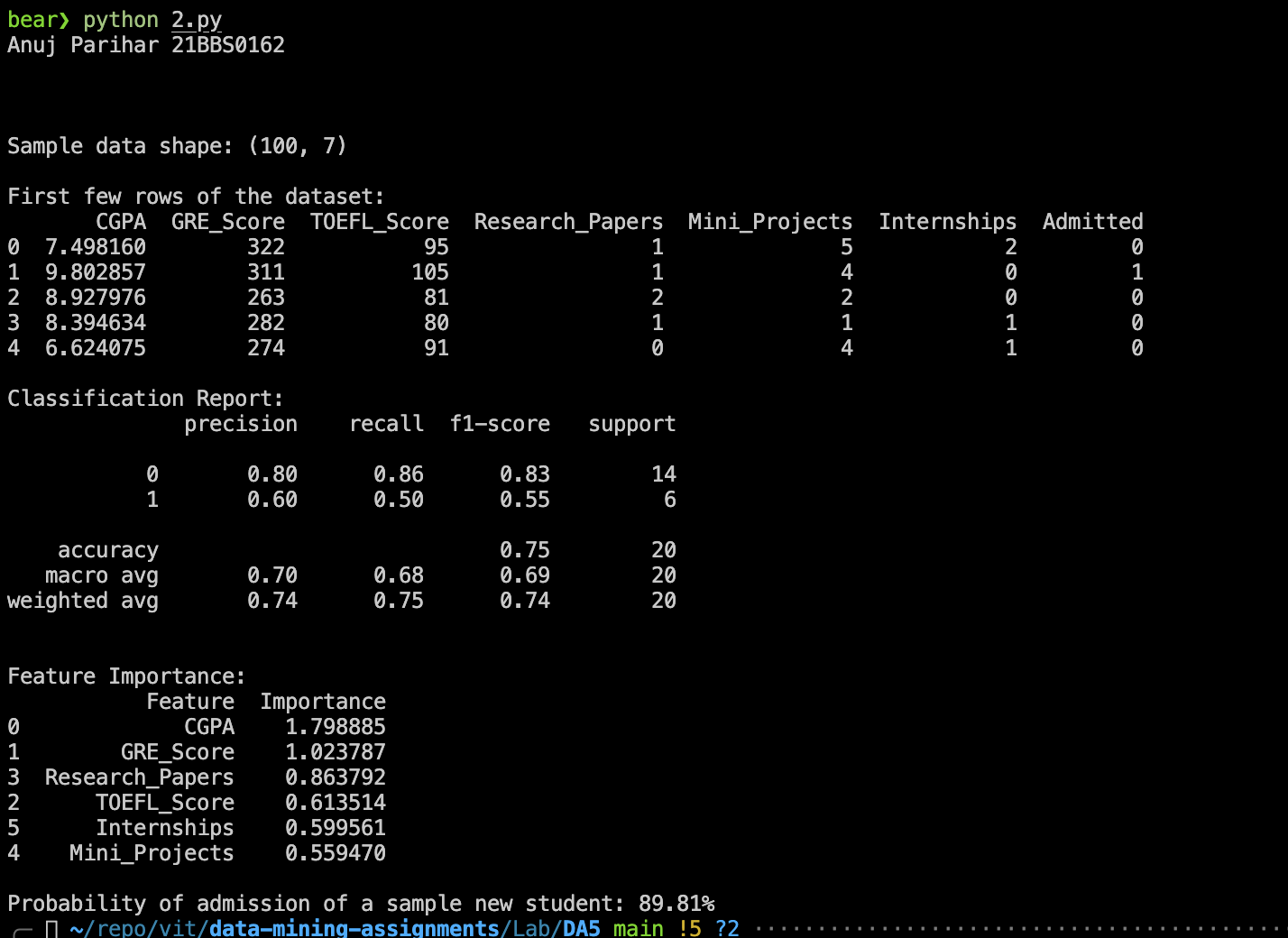
'Internships': [2]

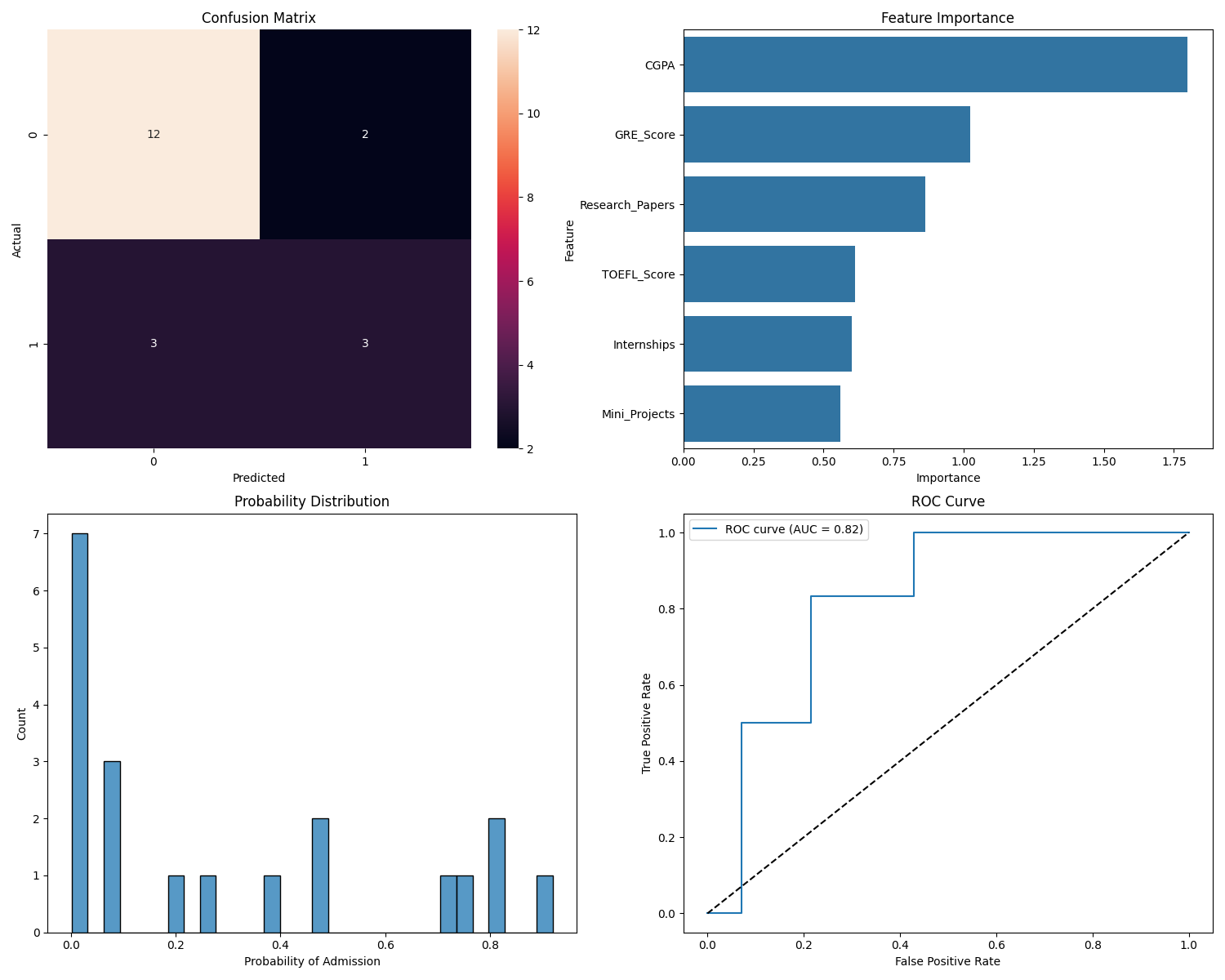
})

prob = predict\_admission(model, scaler, example\_student)

print(f"\nProbability of admission of a sample new student: {prob:.2%}")

**Output:**

****

****

**Results:**

The Logistic Regression model was successfully implemented to predict the likelihood of university admission. The model achieved an overall accuracy of 75% on the test set, with precision, recall, and F1-scores indicating balanced performance, especially for non-admitted cases.

Key feature importances show CGPA (1.80) and GRE Score (1.02) as the most influential predictors, followed by Research Papers (0.86) and TOEFL Score (0.61). The model was used to predict the probability of admission for a new sample student, yielding a high likelihood of 89.81%.