DAV PRACTICALS

1. Calculate the average, variance and standard deviation in Python using NumPy
   Program:

```
import numpy as np

# Define an array of numbers
numbers = np.array([2, 4, 6, 8, 10])

# Calculate the average
average = np.mean(numbers)

# Calculate the variance
variance = np.var(numbers)

# Calculate the standard deviation
std_deviation = np.std(numbers)

# Print the results
print("Average:", average)
print("Variance:", variance)
print("Standard deviation:", std_deviation)
```

---

2. How to calculate probability in a normal distribution given mean and standard deviation in Python?
   Program:

```
from scipy.stats import norm

# Define the mean and standard deviation
mean = 10
std_dev = 2

# Calculate the probability of a value less than or equal to 12
prob = norm.cdf(12, mean, std_dev)

# Print the result
print("Probability:", prob)
```

3. How to Make a Time Series Plot with Rolling Average in Python?
   Program:

```
import pandas as pd
import matplotlib.pyplot as plt

# Load data into a Pandas DataFrame
data = pd.read_csv('data.csv', index_col=0, parse_dates=True)

# Calculate the rolling average with a window of 7 days
rolling_avg = data.rolling(window=7).mean()

# Plot the original data and the rolling average
plt.plot(data, label='Original Data')
plt.plot(rolling_avg, label='Rolling Average (7 days)')
plt.legend()
plt.show()
```

---

4. Any 4 plotting methods using pandas and matplotlib.
   Program:

```
import pandas as pd
import matplotlib.pyplot as plt

# Load the data into a Pandas DataFrame
data = pd.read_csv('example_data.csv')

# Plot a line chart
plt.plot(data['x'], data['y1'])
plt.xlabel('x')
plt.ylabel('y1')
plt.title('Line Chart')

# Plot a scatter plot
plt.figure()
plt.scatter(data['x'], data['y2'])
plt.xlabel('x')
plt.ylabel('y2')
plt.title('Scatter Plot')

# Plot a bar chart
plt.figure()
plt.bar(data['category'], data['value'])
plt.xlabel('Category')
plt.ylabel('Value')
plt.title('Bar Chart')

# Plot a histogram
plt.figure()
plt.hist(data['x'], bins=10)
plt.xlabel('x')
plt.ylabel('Frequency')
plt.title('Histogram')

# Show all the plots
plt.show()
```

---

5. Perform linear regression on the data set mydata.csv.
   Program:

```
import pandas as pd
from sklearn.linear_model import LinearRegression

# Load the data into a Pandas DataFrame
data = pd.read_csv('mydata.csv')

# Split the data into the input (X) and output (y) variables
X = data['input_variable'].values.reshape(-1, 1)
y = data['output_variable'].values.reshape(-1, 1)

# Create a LinearRegression model and fit it to the data
model = LinearRegression()
model.fit(X, y)

# Print the model coefficients and intercept
print("Coefficients:", model.coef_)
print("Intercept:", model.intercept_)
```

6. Perform multiple linear regression on any dataset and obtain the values of coefficients and intercept
   Program:

```
import pandas as pd
from sklearn.linear_model import LinearRegression

# Load the data into a Pandas DataFrame
data = pd.read_csv('mydata.csv')

# Split the data into the input (X) and output (y) variables
X = data[['input_variable1', 'input_variable2']].values
y = data['output_variable'].values.reshape(-1, 1)

# Create a LinearRegression model and fit it to the data
model = LinearRegression()
model.fit(X, y)

# Print the model coefficients and intercept
print("Coefficients:", model.coef_)
print("Intercept:", model.intercept_)
```

---

7. Perform ARIMA model and predict the future values on temp.csv.
   Program:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.arima_model import ARIMA

# Load the data into a Pandas DataFrame
data = pd.read_csv('temp.csv', index_col='Date', parse_dates=True)

# Visualize the time series
plt.plot(data)
plt.xlabel('Date')
plt.ylabel('Temperature')
plt.show()

# Split the data into training and testing sets
train = data.iloc[:200]
test = data.iloc[200:]
```

```
# Fit the ARIMA model to the training data
model = ARIMA(train, order=(1, 1, 1))
results = model.fit()

# Print the model summary
print(results.summary())

# Make predictions on the testing data
predictions = results.predict(start=test.index[0], end=test.index[-1], dynamic=False)

# Visualize the predictions and actual values
plt.plot(predictions, label='Predictions')
plt.plot(test, label='Actual')
plt.legend()
plt.show()
```

---

8. Different plotting using ggplot in r programming.

Program:

```
# Load the ggplot2 package
library(ggplot2)

# Example data for all plots
data <- data.frame(
  x = 1:10,
  y1 = rnorm(10),
  y2 = rnorm(10),
  category = LETTERS[1:10],
  value = rpois(10, lambda = 5)
)

# Line plot
ggplot(data, aes(x = x, y = y1)) +
  geom_line() +
  ggtitle("Line Plot")

# Scatter plot
ggplot(data, aes(x = x, y = y2)) +
  geom_point() +
  ggtitle("Scatter Plot")

# Bar plot
ggplot(data, aes(x = category, y = value)) +
  geom_bar(stat = "identity") +
  ggtitle("Bar Plot")

# Histogram
ggplot(data, aes(x = y1)) +
  geom_histogram(bins = 10) +
  ggtitle("Histogram")

# Box plot
ggplot(data, aes(x = category, y = y1)) +
  geom_boxplot() +
  ggtitle("Box Plot")
```