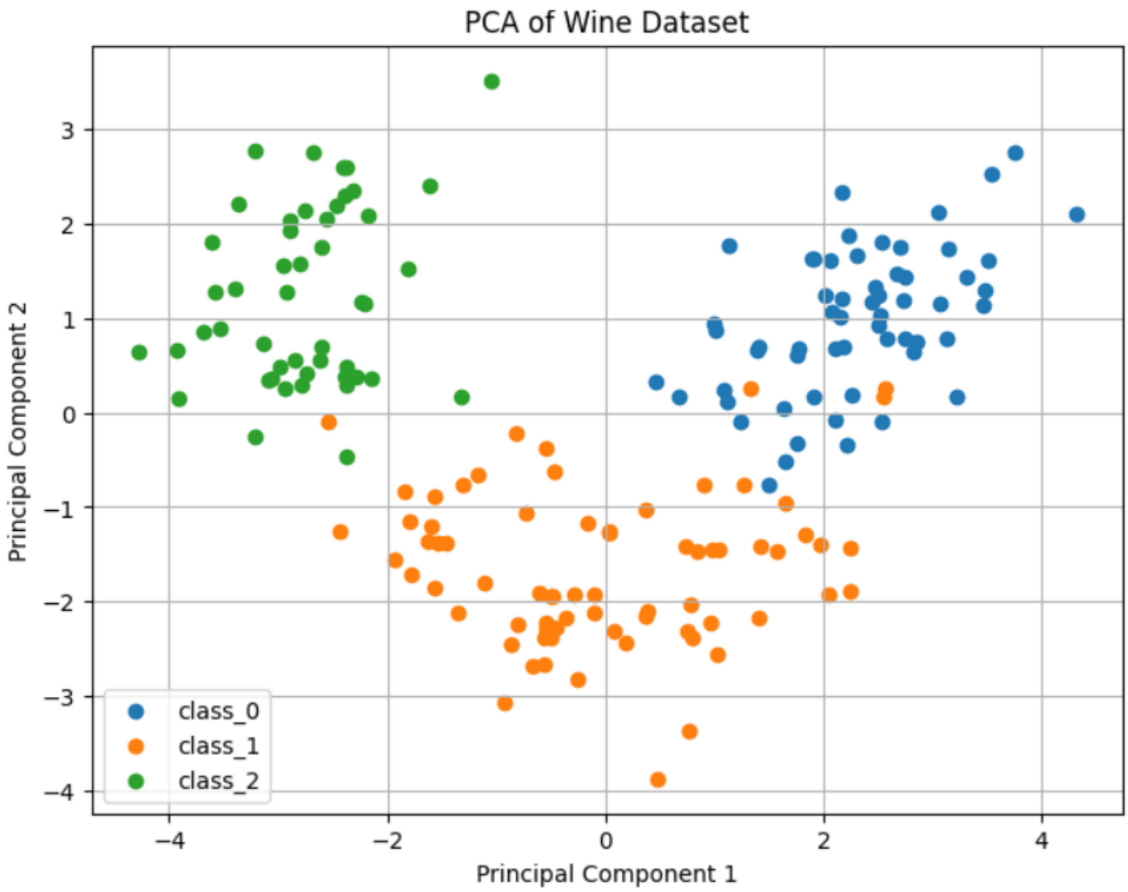
Practical No.: 01
Name: Anuj Shailendra Naikodi
Roll No: 41

```
# PCA on Wine dataset
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_wine
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

# Step 1: Load the Wine dataset
wine = load_wine()
X = pd.DataFrame(wine.data, columns=wine.feature_names)
y = wine.target  # 0, 1, 2 (different wine classes)

print("Dataset shape:", X.shape)
print(X.head(), "\n")

# Step 2: Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
print("Data standardized.\n")

# Step 3: Apply PCA
pca = PCA(n_components=2)  # Reduce to 2 components for visualization
X_pca = pca.fit_transform(X_scaled)

print("Explained variance ratio:", pca.explained_variance_ratio_)
print("Sum of explained variance:", np.sum(pca.explained_variance_ratio_), "\n")
```

```
# Step 4: Visualize the principal components
plt.figure(figsize=(8, 6))
for class_value in np.unique(y):
    plt.scatter(
        X_pca[y == class_value, 0],
        X_pca[y == class_value, 1],
        label=wine.target_names[class_value]
    )
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.title("PCA of Wine Dataset")
plt.legend()
plt.grid(True)
plt.show()
```

Output:

```
Dataset shape: (178, 13)
   alcohol  malic_acid   ash  alcalinity_of_ash  magnesium  total_phenols  \
0    14.23        1.71  2.43               15.6      127.0           2.80
1    13.20        1.78  2.14               11.2      100.0           2.65
2    13.16        2.36  2.67               18.6      101.0           2.80
3    14.37        1.95  2.50               16.8      113.0           3.85
4    13.24        2.59  2.87               21.0      118.0           2.80

   flavanoids  nonflavanoid_phenols  proanthocyanins  color_intensity   hue  \
0        3.06                  0.28             2.29             5.64  1.04
1        2.76                  0.26             1.28             4.38  1.05
2        3.24                  0.30             2.81             5.68  1.03
3        3.49                  0.24             2.18             7.80  0.86
4        2.69                  0.39             1.82             4.32  1.04

   od280/od315_of_diluted_wines  proline
0                          3.92   1065.0
1                          3.40   1050.0
2                          3.17   1185.0
3                          3.45   1480.0
4                          2.93    735.0

Data standardized.

Explained variance ratio: [0.36198848 0.1920749 ]
Sum of explained variance: 0.5540633835693526
```



PCA of Wine Dataset
Principal Component 1
Principal Component 2
class_0
class_1
class_2
−4
−2
0
2
4
3
2
1
0
−1
−2
−3
−4

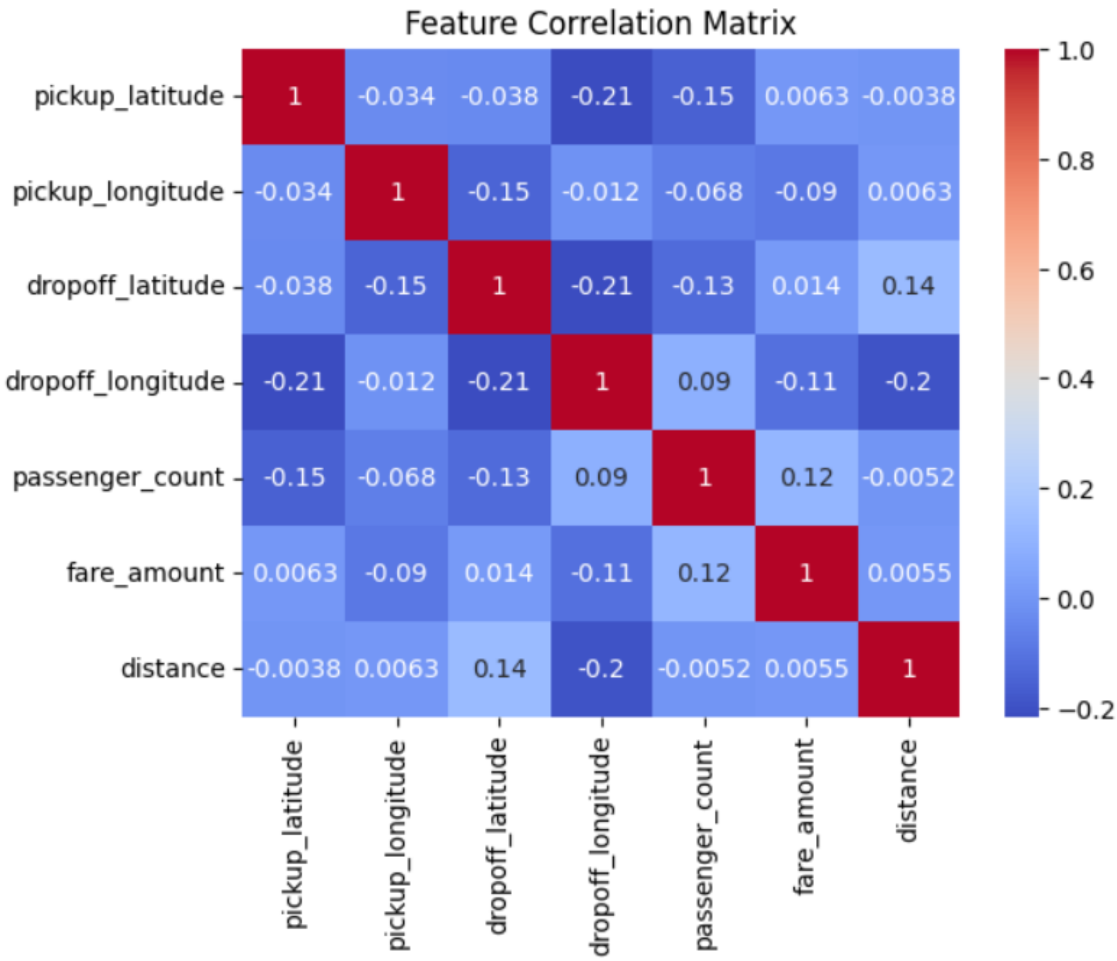
Practical No.: 02
Name: Anuj Shailendra Naikodi
Roll No:41

```
# Uber Fare Prediction using Linear, Ridge & Lasso Regression

# Import Libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression, Ridge, Lasso

from sklearn.metrics import r2_score, mean_squared_error

# Step 1: Create Sample Uber Dataset (Synthetic)

np.random.seed(41)

data = pd.DataFrame({

    'pickup_latitude': np.random.uniform(40.70, 40.85, 100),

    'pickup_longitude': np.random.uniform(-74.02, -73.93, 100),

    'dropoff_latitude': np.random.uniform(40.70, 40.85, 100),

    'dropoff_longitude': np.random.uniform(-74.02, -73.93, 100),

    'passenger_count': np.random.randint(1, 5, 100),

    'fare_amount': np.random.uniform(5, 50, 100)

})

print("Sample dataset:\n", data.head(), "\n")

# Step 2: Preprocess & Feature Engineering

# Calculate simple Euclidean distance feature

data['distance'] = np.sqrt(

    (data['dropoff_latitude'] - data['pickup_latitude'])**2 +

    (data['dropoff_longitude'] - data['pickup_longitude'])**2

)
```

```
# Drop original lat/long columns and keep main features
X = data[['distance', 'passenger_count']]
y = data['fare_amount']

# Step 3: Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=41)

# Step 4: Implement Regression Models
models = {
    'Linear': LinearRegression(),
    'Ridge': Ridge(alpha=1.0),
    'Lasso': Lasso(alpha=0.1)
}

for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    print(f"\n{name} Regression:")
    print("R2 Score:", round(r2_score(y_test, y_pred), 3))
    print("RMSE:", round(np.sqrt(mean_squared_error(y_test, y_pred)), 3))
# Step 5: Correlation & Visualization
corr = data.corr(numeric_only=True)
print("\nCorrelation Matrix:\n", corr, "\n")

sns.heatmap(corr, annot=True, cmap='coolwarm')
plt.title("Feature Correlation Matrix")
plt.show()
```

Output:

```
Sample dataset:
   pickup_latitude  pickup_longitude  dropoff_latitude  dropoff_longitude  \
0        40.756181        -74.017171         40.796305         -74.015349
1        40.842607        -73.962723         40.712621         -73.972178
2        40.809799        -73.991708         40.724244         -73.971343
3        40.789799        -73.974229         40.834783         -73.962631
4        40.723403        -73.938319         40.790964         -73.954652

   passenger_count  fare_amount
0                1    39.971611
1                2    30.128191
2                1    24.089990
3                1    45.785947
4                2    10.003887


Linear Regression:
R2 Score: -0.162
RMSE: 13.34

Ridge Regression:
R2 Score: -0.158
RMSE: 13.317

Lasso Regression:
R2 Score: -0.153
RMSE: 13.288


Correlation Matrix:
                   pickup_latitude  pickup_longitude  dropoff_latitude  \
pickup_latitude           1.000000         -0.034033         -0.037654
pickup_longitude         -0.034033          1.000000         -0.146354
dropoff_latitude         -0.037654         -0.146354          1.000000
dropoff_longitude        -0.211882         -0.011783         -0.214816
passenger_count          -0.154968         -0.068367         -0.128260
fare_amount               0.006286         -0.090083          0.014173
distance                 -0.003818          0.006334          0.135684

                  dropoff_longitude  passenger_count  fare_amount  distance
pickup_latitude           -0.211882        -0.154968     0.006286 -0.003818
pickup_longitude          -0.011783        -0.068367    -0.090083  0.006334
dropoff_latitude          -0.214816        -0.128260     0.014173  0.135684
dropoff_longitude          1.000000         0.090288    -0.110850 -0.195602
passenger_count            0.090288         1.000000     0.115456 -0.005189
fare_amount               -0.110850         0.115456     1.000000  0.005464
distance                  -0.195602        -0.005189     0.005464  1.000000
```


Feature Correlation Matrix
pickup_latitude
pickup_longitude
dropoff_latitude
dropoff_longitude
passenger_count
fare_amount
distance
1 -0.034 -0.038 -0.21 -0.15 0.0063 -0.0038
-0.034 1 -0.15 -0.012 -0.068 -0.09 0.0063
-0.038 -0.15 1 -0.21 -0.13 0.014 0.14
-0.21 -0.012 -0.21 1 0.09 -0.11 -0.2
-0.15 -0.068 -0.13 0.09 1 0.12 -0.0052
0.0063 -0.09 0.014 -0.11 0.12 1 0.0055
-0.0038 0.0063 0.14 -0.2 -0.0052 0.0055 1
1.0
0.8
0.6
0.4
0.2
0.0
−0.2

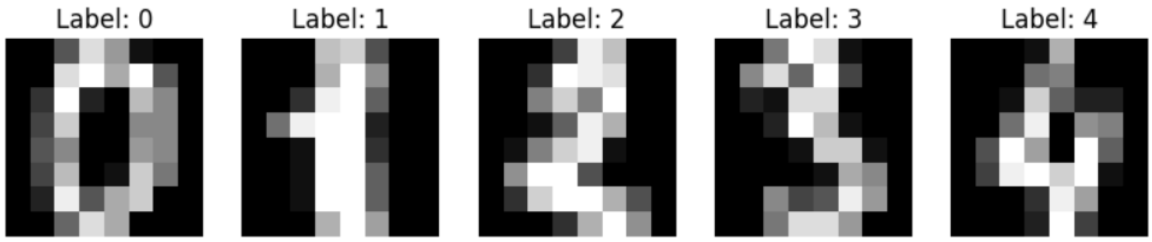
Practical No.: 03
Name: Anuj Shailendra Naikodi
Roll No:41

```
# SVM for Handwritten Digit Classification (0–9)

# Import libraries

import matplotlib.pyplot as plt

from sklearn.datasets import load_digits

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.svm import SVC

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report


# Step 1: Load the digits dataset

digits = load_digits()

X = digits.data        # Flattened 8x8 images → 64 features

y = digits.target      # Labels 0–9


print("Dataset shape:", X.shape)

print("Sample labels:", y[:10], "\n")


# Visualize first few digits

plt.figure(figsize=(10, 2))

for i in range(5):

    plt.subplot(1, 5, i + 1)

    plt.imshow(digits.images[i], cmap='gray')

    plt.title(f"Label: {y[i]}")

    plt.axis('off')

plt.show()

# Step 2: Preprocess data

scaler = StandardScaler()
```

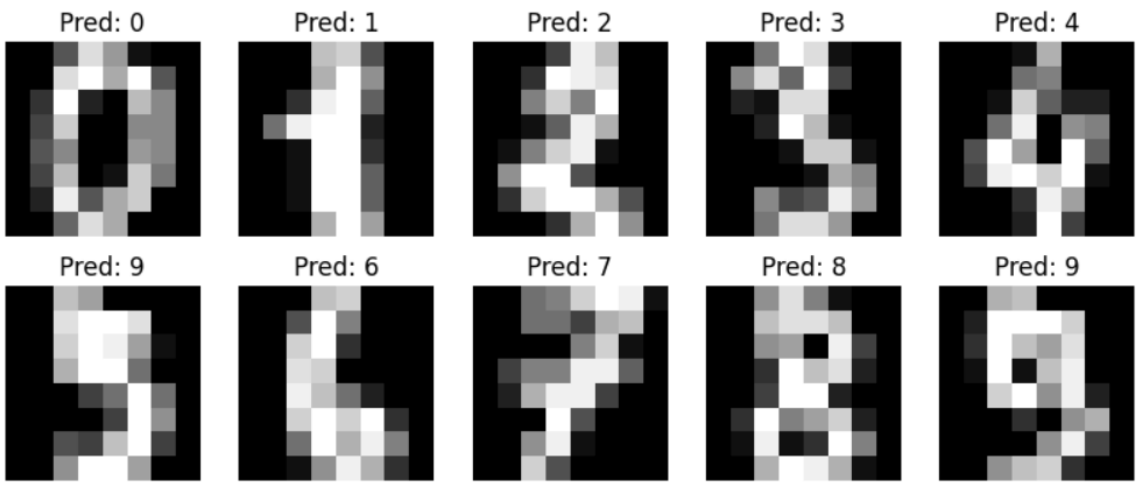
```
X_scaled = scaler.fit_transform(X)  # Scale features for SVM
print("Data standardized.\n")
# Step 3: Split dataset
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.3, random_state=41
)
# Step 4: Train SVM Classifier
svm_clf = SVC(kernel='rbf', gamma='scale')  # RBF kernel
svm_clf.fit(X_train, y_train)
print("SVM model trained.\n")

# Step 5: Evaluate the Model
y_pred = svm_clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("SVM Accuracy on test set:", round(accuracy, 3))
# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", cm)
# Classification Report
print("\nClassification Report:\n", classification_report(y_test, y_pred))
# Step 6: Visualize some predictions
plt.figure(figsize=(10, 4))
for i in range(10):
    plt.subplot(2, 5, i + 1)
    plt.imshow(digits.images[i], cmap='gray')
    pred_label = svm_clf.predict([X_scaled[i]])[0]
    plt.title(f"Pred: {pred_label}")
    plt.axis('off')
plt.show()
```

Output:

```
Dataset shape: (1797, 64)
Sample labels: [0 1 2 3 4 5 6 7 8 9]
```




```
Data standardized.

SVM model trained.

SVM Accuracy on test set: 0.981
Confusion Matrix:
 [[53  0  0  0  0  0  0  0  0  0]
 [ 0 50  0  0  0  0  0  0  0  0]
 [ 0  0 47  0  0  0  0  0  0  0]
 [ 0  0  2 51  0  1  0  0  0  0]
 [ 0  0  0  0 60  0  0  0  0  0]
 [ 0  0  0  0  0 66  0  0  0  0]
 [ 0  0  0  0  0  0 53  0  0  0]
 [ 0  0  0  0  0  0  0 54  0  1]
 [ 0  0  1  1  0  0  0  0 41  0]
 [ 0  0  0  0  0  1  1  0  2 55]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        53
           1       1.00      1.00      1.00        50
           2       0.94      1.00      0.97        47
           3       0.98      0.94      0.96        54
           4       1.00      1.00      1.00        60
           5       0.97      1.00      0.99        66
           6       0.98      1.00      0.99        53
           7       1.00      0.98      0.99        55
           8       0.95      0.95      0.95        43
           9       0.98      0.93      0.96        59

    accuracy                           0.98       540
   macro avg       0.98      0.98      0.98       540
weighted avg       0.98      0.98      0.98       540
```



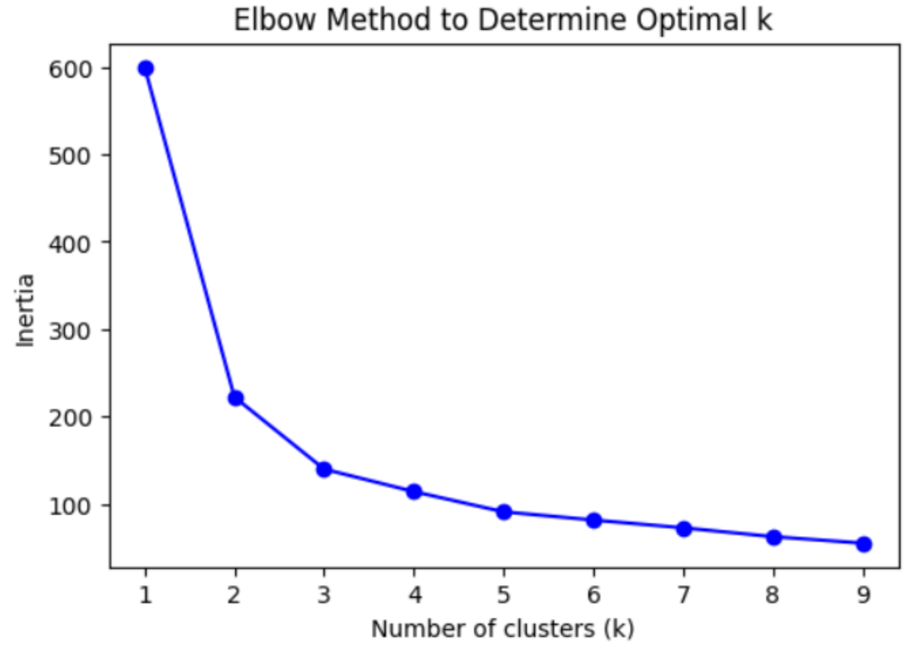
Practical No.: 04
Name: Anuj Shailendra Naikodi
Roll No:41

```
# K-Means Clustering on Iris Dataset

# Import Libraries

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

import seaborn as sns

# Step 1: Load Iris Dataset

iris = sns.load_dataset('iris')  # built-in Iris dataset

X = iris.iloc[:, :-1]            # Drop species column

print("Dataset shape:", X.shape)

print(X.head(), "\n")

# Step 2: Preprocess Data

scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)  # Scale features for clustering

# Step 3: Determine Optimal Clusters (Elbow Method)

inertia = []

K_range = range(1, 10)

for k in K_range:

    kmeans = KMeans(n_clusters=k, random_state=41, n_init=10)

    kmeans.fit(X_scaled)

    inertia.append(kmeans.inertia_)


# Plot Elbow Curve

plt.figure(figsize=(6, 4))

plt.plot(K_range, inertia, 'bo-')

plt.xlabel('Number of clusters (k)')
```

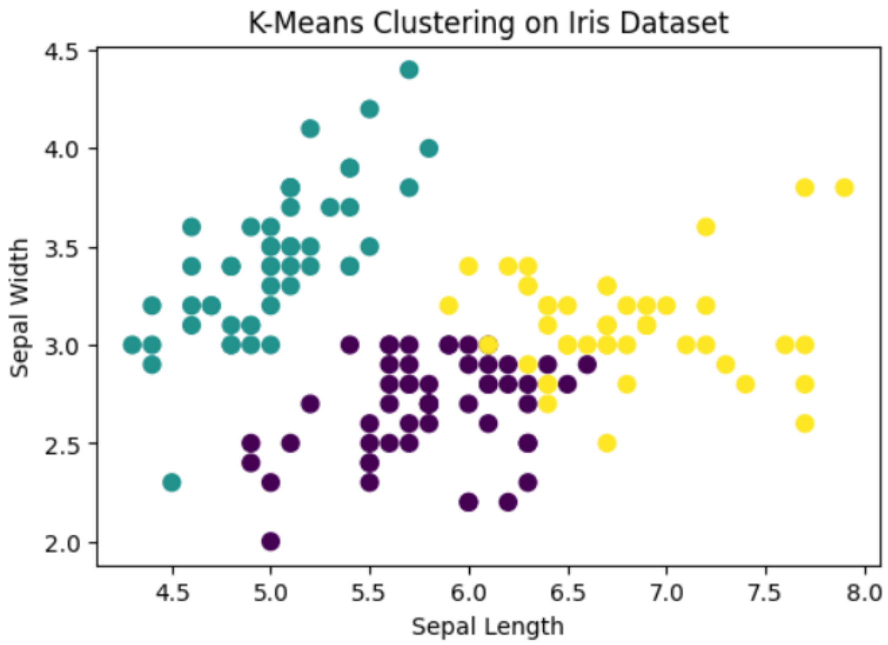
```
plt.ylabel('Inertia')
plt.title('Elbow Method to Determine Optimal k')
plt.show()
# Step 4: Apply K-Means Clustering
optimal_k = 3  # From elbow observation
kmeans = KMeans(n_clusters=optimal_k, random_state=41, n_init=10)
clusters = kmeans.fit_predict(X_scaled)
# Step 5: Add Cluster Labels to Dataset
iris['Cluster'] = clusters
print("Dataset with cluster labels:\n", iris.head())
# Step 6: Visualize Clusters
plt.figure(figsize=(6, 4))
plt.scatter(
    iris['sepal_length'], iris['sepal_width'],
    c=iris['Cluster'], cmap='viridis', s=50
)
plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
plt.title('K-Means Clustering on Iris Dataset')
plt.show()
```

Output:

```
Dataset shape: (150, 4)
   sepal_length  sepal_width  petal_length  petal_width
0           5.1          3.5           1.4          0.2
1           4.9          3.0           1.4          0.2
2           4.7          3.2           1.3          0.2
3           4.6          3.1           1.5          0.2
4           5.0          3.6           1.4          0.2
```




```
Dataset with cluster labels:
   sepal_length  sepal_width  petal_length  petal_width species  Cluster
0           5.1          3.5           1.4          0.2  setosa        1
1           4.9          3.0           1.4          0.2  setosa        1
2           4.7          3.2           1.3          0.2  setosa        1
3           4.6          3.1           1.5          0.2  setosa        1
4           5.0          3.6           1.4          0.2  setosa        1
```



Practical No.: 05
Name: Anuj Shailendra Naikodi
Roll No:41

```
# Boosting Algorithms on Iris Dataset

# Install XGBoost if not installed

# !pip install xgboost --quiet

# Import Libraries

import pandas as pd

import seaborn as sns

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import LabelEncoder

from sklearn.ensemble import AdaBoostClassifier, GradientBoostingClassifier

from xgboost import XGBClassifier

from sklearn.metrics import accuracy_score, classification_report

# Step 1: Load Iris Dataset

iris = sns.load_dataset('iris')

X = iris.iloc[:, :-1]  # Features

y = iris['species']    # Target


# Encode target labels

le = LabelEncoder()

y_encoded = le.fit_transform(y)

print("Target classes:", le.classes_, "\n")

# Step 2: Split Dataset

X_train, X_test, y_train, y_test = train_test_split(

    X, y_encoded, test_size=0.3, random_state=41

)

# Step 3: Define Classifiers

adaboost_clf = AdaBoostClassifier(n_estimators=50, random_state=41)

gbm_clf = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1,
random_state=41)
```

```
xgb_clf = XGBClassifier(use_label_encoder=False, eval_metric='mlogloss',
random_state=41)

# Step 4: Train and Predict

models = {

    'AdaBoost': adaboost_clf,

    'Gradient Boosting': gbm_clf,

    'XGBoost': xgb_clf

}

results = {}

for name, model in models.items():

    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)

    acc = accuracy_score(y_test, y_pred)

    results[name] = acc

    print(f"\n{name} Model Accuracy: {acc:.3f}")

    print(classification_report(y_test, y_pred, target_names=le.classes_))

# Step 5: Compare Model Performance

print("\nSummary of Model Accuracy:")

for model_name, acc in results.items():

    print(f"{model_name}: {acc:.3f}")
```

Output:

```
Target classes: ['setosa' 'versicolor' 'virginica']


AdaBoost Model Accuracy: 1.000
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        19
  versicolor       1.00      1.00      1.00        13
   virginica       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45


Gradient Boosting Model Accuracy: 1.000
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        19
  versicolor       1.00      1.00      1.00        13
   virginica       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45

/usr/local/lib/python3.12/dist-packages/xgboost/training.py:199: UserWarning: [10:53:44] WARNING: /workspace/src/learner.cc:790:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)


XGBoost Model Accuracy: 1.000
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        19
  versicolor       1.00      1.00      1.00        13
   virginica       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45


Summary of Model Accuracy:
AdaBoost: 1.000
Gradient Boosting: 1.000
XGBoost: 1.000
```

Practical No.: 06
Name: Anuj Shailendra Naikodi
Roll No:41

```python
# Tic-Tac-Toe with Q-Learning

import numpy as np

import random

# Step a: Setting up the environment

class TicTacToe:

    def __init__(self):

        self.board = [' ' for _ in range(9)]  # 3x3 board

        self.current_winner = None


    def available_moves(self):

        return [i for i, spot in enumerate(self.board) if spot == ' ']


    def make_move(self, square, letter):

        if self.board[square] == ' ':

            self.board[square] = letter

            if self.winner(square, letter):

                self.current_winner = letter

            return True

        return False


    def winner(self, square, letter):

        row_ind = square // 3

        row = self.board[row_ind * 3:(row_ind + 1) * 3]

        if all(s == letter for s in row): return True

        col_ind = square % 3

        col = [self.board[col_ind + i * 3] for i in range(3)]

        if all(s == letter for s in col): return True
```

```
        if square % 2 == 0:
            diag1 = [self.board[i] for i in [0, 4, 8]]
            diag2 = [self.board[i] for i in [2, 4, 6]]
            if all(s == letter for s in diag1) or all(s == letter for s in diag2):
                return True
        return False

    def reset(self):
        self.board = [' ' for _ in range(9)]
        self.current_winner = None

# Step b: Q-Learning Agent
class QLearningAgent:
    def __init__(self, alpha=0.3, gamma=0.9, epsilon=0.2):
        self.q_table = {}  # state-action pairs
        self.alpha = alpha
        self.gamma = gamma
        self.epsilon = epsilon

    def get_state(self, board):
        return ''.join(board)

    def choose_action(self, env):
        state = self.get_state(env.board)
        if random.uniform(0, 1) < self.epsilon:
            return random.choice(env.available_moves())
        q_values = [self.q_table.get((state, a), 0) for a in env.available_moves()]
        max_q = max(q_values)
        max_actions = [a for a, q in zip(env.available_moves(), q_values) if q == max_q]
        return random.choice(max_actions)
```

```
    def update_q(self, state, action, reward, next_state):
        old_q = self.q_table.get((state, action), 0)
        if next_state is not None:
            max_future_q = max([self.q_table.get((next_state, a), 0) for a in range(9)])
        else:
            max_future_q = 0
        self.q_table[(state, action)] = old_q + self.alpha * (reward + self.gamma * max_future_q
- old_q)
# Step c & d: Training the model
env = TicTacToe()
agent = QLearningAgent()
episodes = 5000

for _ in range(episodes):
    env.reset()
    state = agent.get_state(env.board)
    while True:
        action = agent.choose_action(env)
        env.make_move(action, 'X')
        next_state = agent.get_state(env.board)
        if env.current_winner == 'X':
            agent.update_q(state, action, 1, None)
            break
        elif not env.available_moves():
            agent.update_q(state, action, 0.5, None)
            break
        else:
            opp_action = random.choice(env.available_moves())
            env.make_move(opp_action, 'O')
```

```
            if env.current_winner == 'O':
                agent.update_q(state, action, -1, None)
                break
            else:
                agent.update_q(state, action, 0, next_state)
        state = next_state

print("Training completed!\n")
# Step e: Testing the model
def play_game(agent):
    env.reset()
    while True:
        print(f"Board: {env.board}")
        action = agent.choose_action(env)
        env.make_move(action, 'X')
        if env.current_winner == 'X':
            print(f"AI wins! Final board: {env.board}\n")
            break
        elif not env.available_moves():
            print(f"Draw! Final board: {env.board}\n")
            break
        opp_action = random.choice(env.available_moves())
        env.make_move(opp_action, 'O')
        if env.current_winner == 'O':
            print(f"Opponent wins! Final board: {env.board}\n")
            break

# Play 3 test games
for i in range(3):
    print(f"--- Test Game {i+1} ---")
```

play_game(agent)

Output:

```
Training completed!

--- Test Game 1 ---
Board: [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ']
Board: ['O', 'X', ' ', ' ', ' ', ' ', ' ', ' ', ' ']
Board: ['O', 'X', 'X', ' ', ' ', ' ', ' ', ' ', 'O']
Board: ['O', 'X', 'X', ' ', ' ', 'O', 'X', ' ', 'O']
Board: ['O', 'X', 'X', 'X', ' ', 'O', 'X', 'O', 'O']
AI wins! Final board: ['O', 'X', 'X', 'X', 'X', 'O', 'X', 'O', 'O']

--- Test Game 2 ---
Board: [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ']
Board: ['O', 'X', ' ', ' ', ' ', ' ', ' ', ' ', ' ']
Board: ['O', 'X', ' ', ' ', ' ', ' ', 'X', 'O', ' ']
Board: ['O', 'X', ' ', 'O', ' ', ' ', 'X', 'O', 'X']
Board: ['O', 'X', 'O', 'O', 'X', ' ', 'X', 'O', 'X']
Draw! Final board: ['O', 'X', 'O', 'O', 'X', 'X', 'X', 'O', 'X']

--- Test Game 3 ---
Board: [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ']
Board: [' ', 'X', ' ', 'O', ' ', ' ', ' ', ' ', ' ']
Board: [' ', 'X', ' ', 'O', 'X', ' ', 'O', ' ', ' ']
Board: ['X', 'X', ' ', 'O', 'X', ' ', 'O', 'O', ' ']
AI wins! Final board: ['X', 'X', 'X', 'O', 'X', ' ', 'O', 'O', ' ']
```

Practical No.: 07
Name: Anuj Shailendra Naikodi
Roll No:41

import pandas as pd

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

df_csv = pd.read_csv("sales_data.csv")
df_excel = pd.read_excel("sales_data.xlsx")
df_json = pd.read_json("sales_data.json")

sales_data = pd.concat([df_csv, df_excel, df_json], ignore_index=True)
sales_data.fillna({'Region': 'Unknown'}, inplace=True)
sales_data.drop_duplicates(inplace=True)
sales_data.columns = sales_data.columns.str.title()
sales_data['Product'] = sales_data['Product'].astype('category')
sales_data['Region'] = sales_data['Region'].astype('category')
sales_data['Total_Sales'] = sales_data['Quantity'] * sales_data['Price']

grouped_data = sales_data.groupby(['Product', 'Region'])['Total_Sales'].sum().reset_index()
print(grouped_data)
print(sales_data.describe(include='all'))
print(sales_data.groupby('Product')['Total_Sales'].sum())
print(sales_data.groupby('Region')['Total_Sales'].sum())

sns.barplot(x='Product', y='Total_Sales', data=sales_data, estimator=sum)
plt.title("Total Sales by Product")
plt.show()

sns.barplot(x='Region', y='Total_Sales', data=sales_data, estimator=sum)
plt.title("Total Sales by Region")
plt.show()
```

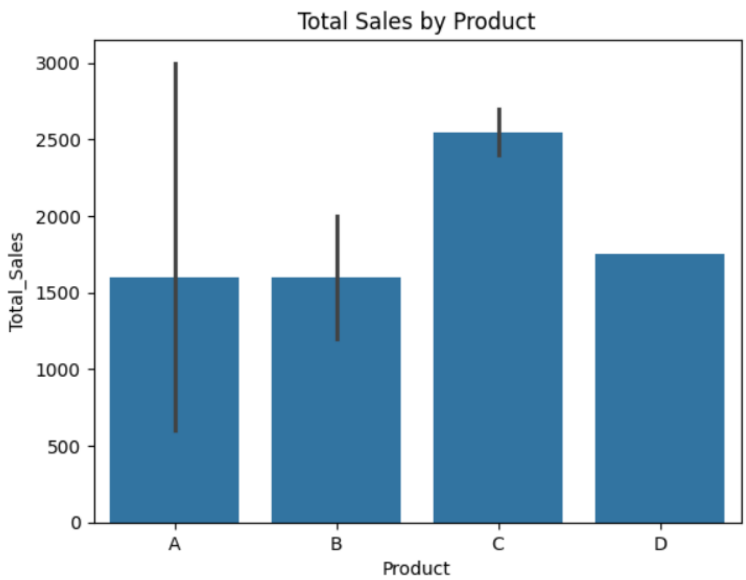
sns.heatmap(sales_data.corr(numeric_only=True), annot=True, cmap='coolwarm')

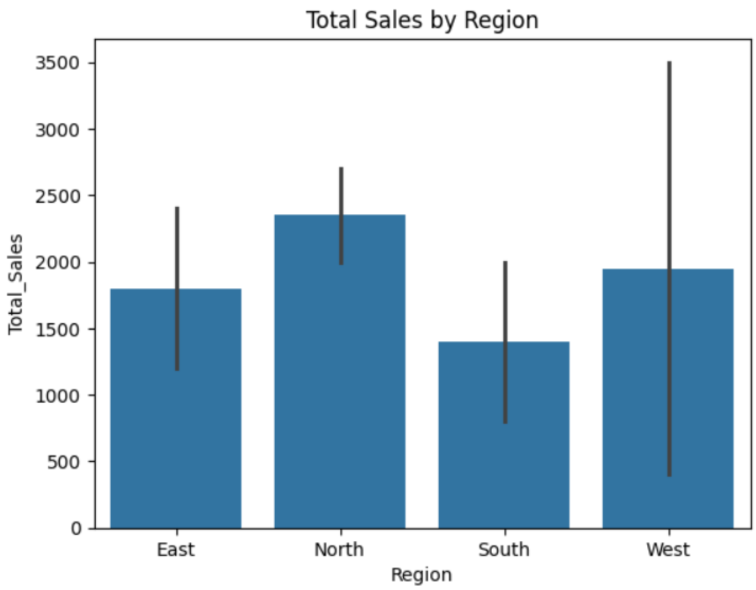
plt.title("Correlation Matrix")

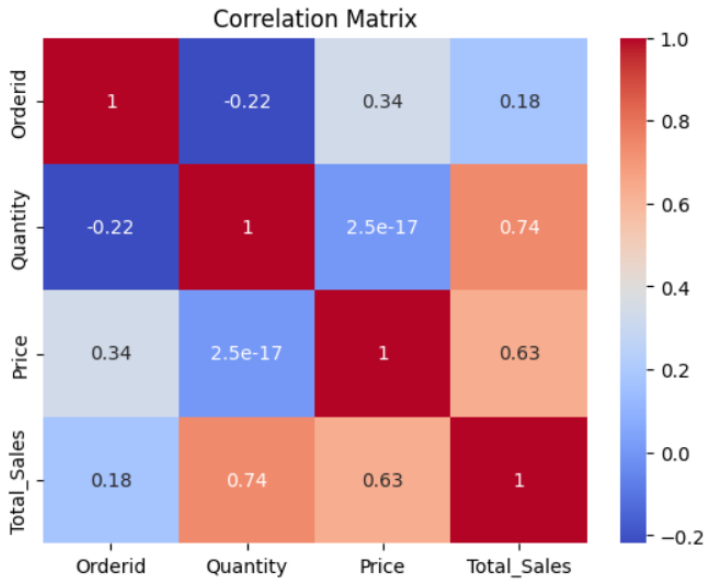
plt.show()

Output:

```
        Orderid Product  Quantity       Price Region  Total_Sales
count   8.00000       8    8.0000    8.000000      8     8.000000
unique      NaN       4       NaN         NaN      4          NaN
top         NaN       A       NaN         NaN   East          NaN
freq        NaN       3       NaN         NaN      2          NaN
mean    4.50000     NaN    6.0000  156.250000    NaN   937.500000
std     2.44949     NaN    2.9277   56.299581    NaN   514.608034
min     1.00000     NaN    2.0000  100.000000    NaN   200.000000
25%     2.75000     NaN    3.7500  100.000000    NaN   550.000000
50%     4.50000     NaN    6.0000  150.000000    NaN  1000.000000
75%     6.25000     NaN    8.2500  200.000000    NaN  1237.500000
max     8.00000     NaN   10.0000  250.000000    NaN  1750.000000
Product
A    1600
B    1600
C    2550
D    1750
Name: Total_Sales, dtype: int64
Region
East     1800
North    2350
South    1400
West     1950
Name: Total_Sales, dtype: int64
```




Total Sales by Region
Total_Sales
3500
3000
2500
2000
1500
1000
500
0
East
North
South
West
Region

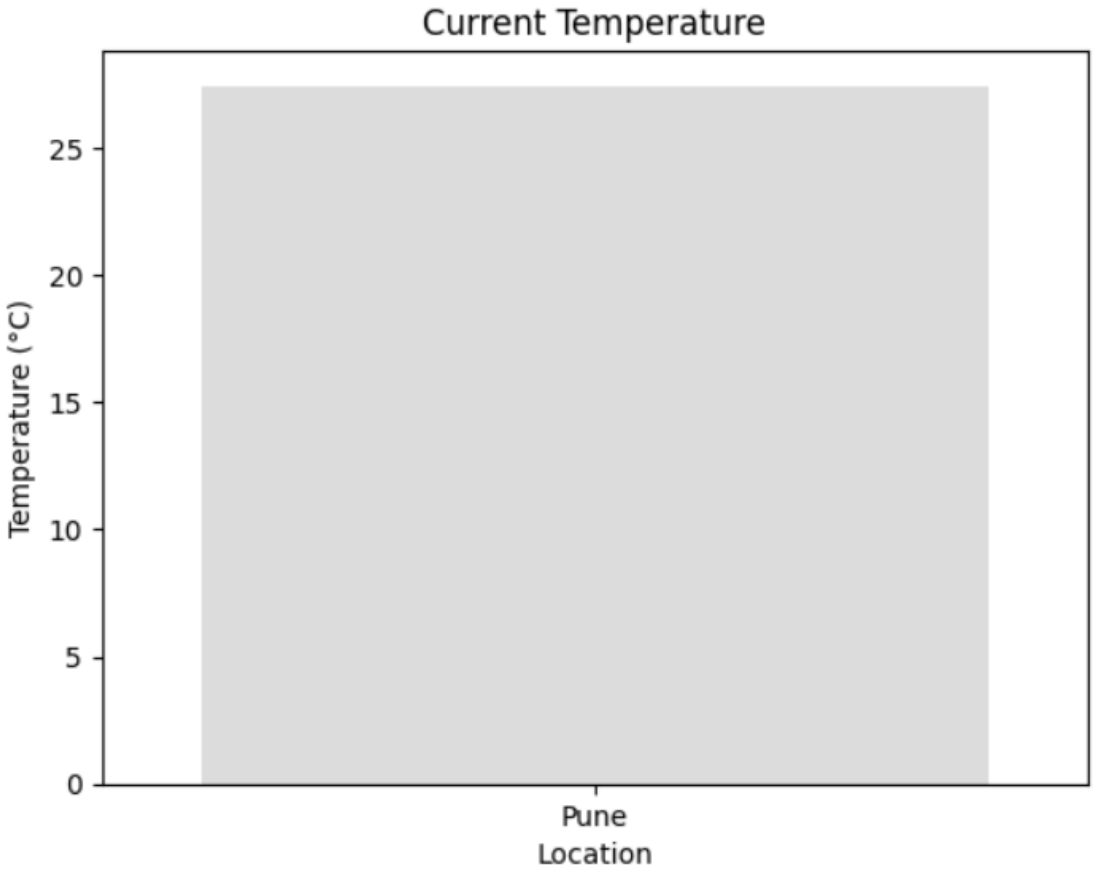

Correlation Matrix
Orderid
Quantity
Price
Total_Sales
1
-0.22
0.34
0.18
-0.22
1
2.5e-17
0.74
0.34
2.5e-17
1
0.63
0.18
0.74
0.63
1
1.0
0.8
0.6
0.4
0.2
0.0
−0.2

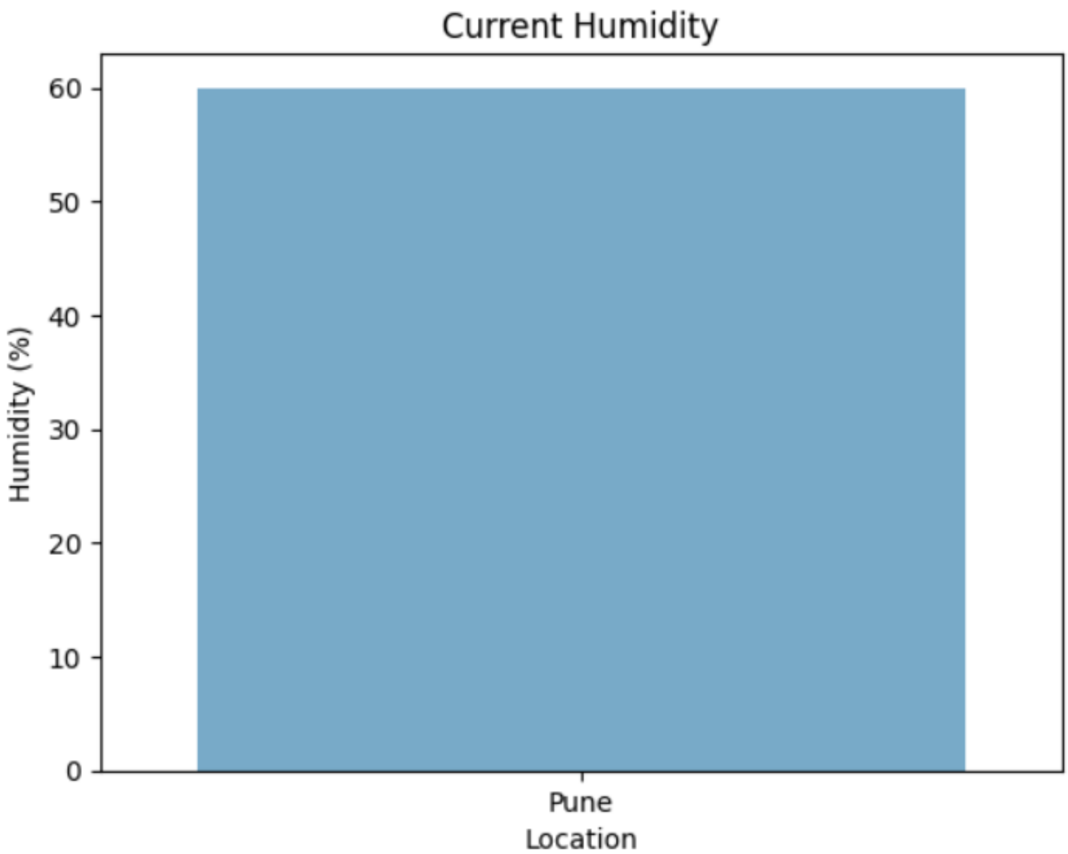
Practical No.: 08
Name: Anuj Shailendra Naikodi
Roll No:41

```
import requests

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import folium


api_key = "YOUR_API_KEY"

location = "Pune"

url =
f"http://api.openweathermap.org/data/2.5/weather?q={location}&appid={api_key}&units=m
etric"


response = requests.get(url)

data = response.json()


if data.get("cod") != 200:

   print(f"Error fetching location: {data.get('message')}")

else:

   weather = {

      "Location": location,

      "Temperature_C": data["main"]["temp"],

      "Humidity_%": data["main"]["humidity"],

      "Pressure_hPa": data["main"]["pressure"],

      "Wind_Speed_mps": data["wind"]["speed"],

      "Weather": data["weather"][0]["main"]

   }


df = pd.DataFrame([weather])
```

```
df.fillna(method="ffill", inplace=True)

avg_temp = df["Temperature_C"].mean()
max_temp = df["Temperature_C"].max()
min_temp = df["Temperature_C"].min()

sns.barplot(x=df["Location"], y=df["Temperature_C"], palette="coolwarm")
plt.title("Current Temperature")
plt.ylabel("Temperature (°C)")
plt.show()

sns.barplot(x=df["Location"], y=df["Humidity_%"], palette="Blues")
plt.title("Current Humidity")
plt.ylabel("Humidity (%)")
plt.show()

lat = data["coord"]["lat"]
lon = data["coord"]["lon"]

map_weather = folium.Map(location=[lat, lon], zoom_start=10)
folium.Marker(
    [lat, lon],
    popup=f"{location}: {df['Temperature_C'][0]}°C, {df['Weather'][0]}"
).add_to(map_weather)
map_weather
```

Output:


Current Temperature
25
20
15
10
5
0
Temperature (°C)
Pune
Location


Current Humidity
60
50
40
30
20
10
0
Humidity (%)
Pune
Location

Practical No.: 09
Name: Anuj Shailendra Naikodi
Roll No:41

```
import pandas as pd

import numpy as np


data = {

    "CustomerID": [101,102,103,104,105,106,106],

    "Gender": ["Male","Female","Female","Male","fmale","Male","Male"],

    "Age": [25, np.nan, 35, 40, 30, 29, 29],

    "Tenure_Months": [12, 24, 36, 48, 60, 12, 12],

    "MonthlyCharges": [70, 80, 90, 1000, 85, 75, 75],

    "Churn": ["No","Yes","No","Yes","No","Yes","Yes"]

}


df = pd.DataFrame(data)

print("Step 1: Sample Dataset Created\n", df)


print("\nStep 2: Dataset Info")

print(df.info())

print("\nStep 2: Basic Stats")

print(df.describe())


df['Age'].fillna(df['Age'].mean(), inplace=True)

print("\nStep 3: Missing Values Handled\n", df)


df.drop_duplicates(inplace=True)

print("\nStep 4: Duplicates Removed\n", df)


df['Gender'] = df['Gender'].replace({'fmale':'Female'})
```

```
print("\nStep 5: Inconsistent Data Corrected\n", df)


df['CustomerID'] = df['CustomerID'].astype(str)

df['Churn'] = df['Churn'].astype('category')

print("\nStep 6: Data Types Corrected\n", df.dtypes)


Q1 = df['MonthlyCharges'].quantile(0.25)

Q3 = df['MonthlyCharges'].quantile(0.75)

IQR = Q3 - Q1

outliers = df[(df['MonthlyCharges'] < Q1 - 1.5*IQR) | (df['MonthlyCharges'] > Q3 +
1.5*IQR)]

print("\nStep 7: Outliers Detected\n", outliers)


df['MonthlyCharges'] = np.where(df['MonthlyCharges'] > Q3 + 1.5*IQR, Q3 + 1.5*IQR,
df['MonthlyCharges'])

print("\nStep 7: Outliers Handled\n", df)
```

Output:

```
Step 1: Sample Dataset Created
    CustomerID  Gender   Age  Tenure_Months  MonthlyCharges Churn
0          101    Male  25.0             12              70    No
1          102  Female   NaN             24              80   Yes
2          103  Female  35.0             36              90    No
3          104    Male  40.0             48            1000   Yes
4          105   fmale  30.0             60              85    No
5          106    Male  29.0             12              75   Yes
6          106    Male  29.0             12              75   Yes

Step 2: Dataset Info
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7 entries, 0 to 6
Data columns (total 6 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   CustomerID      7 non-null      int64
 1   Gender          7 non-null      object
 2   Age             6 non-null      float64
 3   Tenure_Months   7 non-null      int64
 4   MonthlyCharges  7 non-null      int64
 5   Churn           7 non-null      object
dtypes: float64(1), int64(3), object(2)
memory usage: 468.0+ bytes
None
```

```
Step 2: Basic Stats
       CustomerID        Age  Tenure_Months  MonthlyCharges
count    7.000000   6.000000       7.000000        7.000000
mean   103.857143  31.333333      29.142857      210.714286
std      1.951800   5.316641      19.420166      348.107126
min    101.000000  25.000000      12.000000       70.000000
25%    102.500000  29.000000      12.000000       75.000000
50%    104.000000  29.500000      24.000000       80.000000
75%    105.500000  33.750000      42.000000       87.500000
max    106.000000  40.000000      60.000000     1000.000000

Step 3: Missing Values Handled
   CustomerID  Gender        Age  Tenure_Months  MonthlyCharges Churn
0         101    Male  25.000000             12              70    No
1         102  Female  31.333333             24              80   Yes
2         103  Female  35.000000             36              90    No
3         104    Male  40.000000             48            1000   Yes
4         105   fmale  30.000000             60              85    No
5         106    Male  29.000000             12              75   Yes
6         106    Male  29.000000             12              75   Yes

Step 4: Duplicates Removed
   CustomerID  Gender        Age  Tenure_Months  MonthlyCharges Churn
0         101    Male  25.000000             12              70    No
1         102  Female  31.333333             24              80   Yes
2         103  Female  35.000000             36              90    No
3         104    Male  40.000000             48            1000   Yes
4         105   fmale  30.000000             60              85    No
5         106    Male  29.000000             12              75   Yes


Step 5: Inconsistent Data Corrected
   CustomerID  Gender        Age  Tenure_Months  MonthlyCharges Churn
0         101    Male  25.000000             12              70    No
1         102  Female  31.333333             24              80   Yes
2         103  Female  35.000000             36              90    No
3         104    Male  40.000000             48            1000   Yes
4         105  Female  30.000000             60              85    No
5         106    Male  29.000000             12              75   Yes

Step 6: Data Types Corrected
 CustomerID          object
Gender              object
Age                float64
Tenure_Months        int64
MonthlyCharges       int64
Churn             category
dtype: object

Step 7: Outliers Detected
   CustomerID Gender   Age  Tenure_Months  MonthlyCharges Churn
3         104   Male  40.0             48            1000   Yes

Step 7: Outliers Handled
   CustomerID  Gender        Age  Tenure_Months  MonthlyCharges Churn
0         101    Male  25.000000             12            70.0    No
1         102  Female  31.333333             24            80.0   Yes
2         103  Female  35.000000             36            90.0    No
3         104    Male  40.000000             48           107.5   Yes
4         105  Female  30.000000             60            85.0    No
5         106    Male  29.000000             12            75.0   Yes
```

Practical No.: 10
Name: Anuj Shailendra Naikodi
Roll No:41

```
import pandas as pd

import numpy as np


data = {

    "Property_ID": [101,102,103,104,105],

    "Location": ["Downtown","Uptown","Suburb","Downtown","Suburb"],

    "Property_Type": ["Apartment","House","House","Apartment","House"],

    "Sale_Price ($)": [250000, 500000, 400000, np.nan, 1000000],

    "Size (sqft)": [850, 2000, 1500, 900, 3500],

    "Year_Built": [2005, 2010, 2000, 2015, 1995]

}


df = pd.DataFrame(data)

print("Step 1: Sample Dataset Created\n", df)


df.columns = ['Property_ID', 'Location', 'Property_Type', 'Sale_Price', 'Size_sqft',
'Year_Built']

print("\nStep 2: Column Names Cleaned\n", df.columns)


df['Sale_Price'].fillna(df['Sale_Price'].mean(), inplace=True)

print("\nStep 3: Missing Values Handled\n", df)


df_downtown = df[df['Location'] == "Downtown"]

print("\nStep 4: Filtered Data (Downtown Properties)\n", df_downtown)


df_encoded = pd.get_dummies(df, columns=['Location','Property_Type'])

print("\nStep 5: Categorical Variables Encoded\n", df_encoded.head())
```

```
avg_price_by_type = df.groupby('Property_Type')['Sale_Price'].mean()
print("\nStep 6: Average Sale Price by Property Type\n", avg_price_by_type)

Q1 = df['Sale_Price'].quantile(0.25)
Q3 = df['Sale_Price'].quantile(0.75)
IQR = Q3 - Q1
outliers = df[(df['Sale_Price'] < Q1 - 1.5*IQR) | (df['Sale_Price'] > Q3 + 1.5*IQR)]
print("\nStep 7: Outliers Detected\n", outliers)

df['Sale_Price'] = np.where(df['Sale_Price'] > Q3 + 1.5*IQR, Q3 + 1.5*IQR, df['Sale_Price'])
print("\nStep 7: Outliers Handled\n", df)
```

Output:

```
Step 1: Sample Dataset Created
    Property_ID  Location Property_Type  Sale_Price ($)  Size (sqft)  \
0          101  Downtown     Apartment        250000.0          850
1          102    Uptown         House        500000.0         2000
2          103    Suburb         House        400000.0         1500
3          104  Downtown     Apartment             NaN          900
4          105    Suburb         House       1000000.0         3500

   Year_Built
0        2005
1        2010
2        2000
3        2015
4        1995

Step 2: Column Names Cleaned
 Index(['Property_ID', 'Location', 'Property_Type', 'Sale_Price', 'Size_sqft',
       'Year_Built'],
      dtype='object')

Step 3: Missing Values Handled
    Property_ID  Location Property_Type  Sale_Price  Size_sqft  Year_Built
0          101  Downtown     Apartment    250000.0        850        2005
1          102    Uptown         House    500000.0       2000        2010
2          103    Suburb         House    400000.0       1500        2000
3          104  Downtown     Apartment    537500.0        900        2015
4          105    Suburb         House   1000000.0       3500        1995
```

```
Step 4: Filtered Data (Downtown Properties)
    Property_ID  Location Property_Type  Sale_Price  Size_sqft  Year_Built
0          101  Downtown     Apartment    250000.0        850        2005
3          104  Downtown     Apartment    537500.0        900        2015

Step 5: Categorical Variables Encoded
    Property_ID  Sale_Price  Size_sqft  Year_Built  Location_Downtown  \
0          101    250000.0        850        2005               True
1          102    500000.0       2000        2010              False
2          103    400000.0       1500        2000              False
3          104    537500.0        900        2015               True
4          105   1000000.0       3500        1995              False

   Location_Suburb  Location_Uptown  Property_Type_Apartment  \
0            False            False                     True
1            False             True                    False
2             True            False                    False
3            False            False                     True
4             True            False                    False

   Property_Type_House
0                False
1                 True
2                 True
3                False
4                 True

Step 6: Average Sale Price by Property Type
 Property_Type
Apartment    393750.000000
House        633333.333333
Name: Sale_Price, dtype: float64


Step 7: Outliers Detected
    Property_ID Location Property_Type  Sale_Price  Size_sqft  Year_Built
4          105   Suburb         House  1000000.0       3500        1995

Step 7: Outliers Handled
    Property_ID  Location Property_Type  Sale_Price  Size_sqft  Year_Built
0          101  Downtown     Apartment    250000.0        850        2005
1          102    Uptown         House    500000.0       2000        2010
2          103    Suburb         House    400000.0       1500        2000
3          104  Downtown     Apartment    537500.0        900        2015
4          105    Suburb         House    743750.0       3500        1995
```

Practical No.: 11
Name: Anuj Shailendra Naikodi
Roll No:41

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt


dates = pd.date_range(start="2025-01-01", periods=10, freq='D')

data = {

    "Date": dates,

    "PM2.5": np.random.randint(50, 150, size=10),

    "PM10": np.random.randint(60, 180, size=10),

    "CO": np.random.uniform(0.5, 2.0, size=10),

}

df = pd.DataFrame(data)

df["AQI"] = (0.5*df["PM2.5"] + 0.3*df["PM10"] + 50*df["CO"]).astype(int)

print("Step 1: Sample AQI Dataset Created\n", df)


print("\nStep 2: Dataset Info")

print(df.info())

print("\nStep 2: Dataset Description")

print(df.describe())


plt.figure(figsize=(8,4))

plt.plot(df["Date"], df["AQI"], marker='o', color='red', label='AQI')

plt.title("AQI Trend Over Time")

plt.xlabel("Date")

plt.ylabel("AQI")

plt.xticks(rotation=45)

plt.legend()
```

```
plt.tight_layout()
plt.show()

plt.figure(figsize=(8,4))
plt.plot(df["Date"], df["PM2.5"], marker='o', label='PM2.5')
plt.plot(df["Date"], df["PM10"], marker='s', label='PM10')
plt.plot(df["Date"], df["CO"], marker='^', label='CO')
plt.title("Pollutant Levels Over Time")
plt.xlabel("Date")
plt.ylabel("Concentration")
plt.xticks(rotation=45)
plt.legend()
plt.tight_layout()
plt.show()

plt.figure(figsize=(8,4))
plt.bar(df["Date"], df["AQI"], color='orange')
plt.title("AQI Values by Date")
plt.xlabel("Date")
plt.ylabel("AQI")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

plt.figure(figsize=(6,4))
plt.boxplot([df["PM2.5"], df["PM10"], df["CO"]], labels=["PM2.5","PM10","CO"])
plt.title("Pollutant Distribution")
plt.ylabel("Concentration")
plt.show()
```

plt.figure(figsize=(6,4))

plt.scatter(df["PM2.5"], df["AQI"], color='green', s=80)

plt.title("AQI vs PM2.5")

plt.xlabel("PM2.5")

plt.ylabel("AQI")

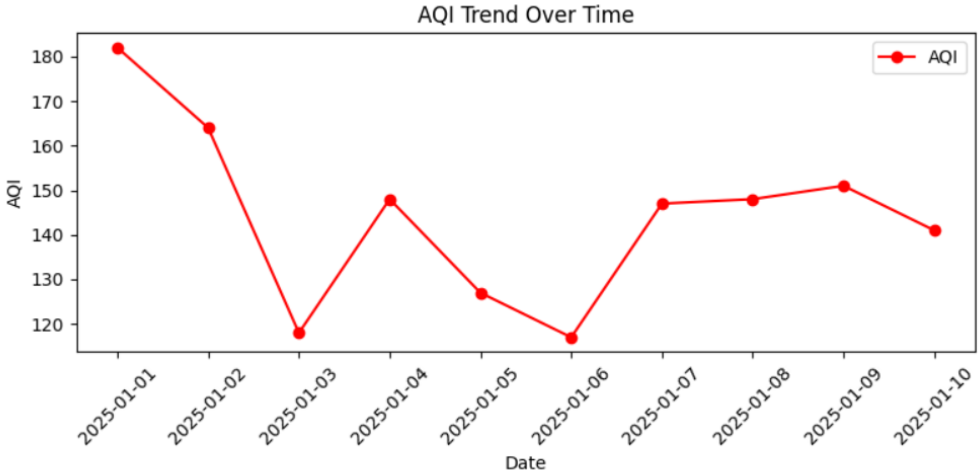
plt.show()

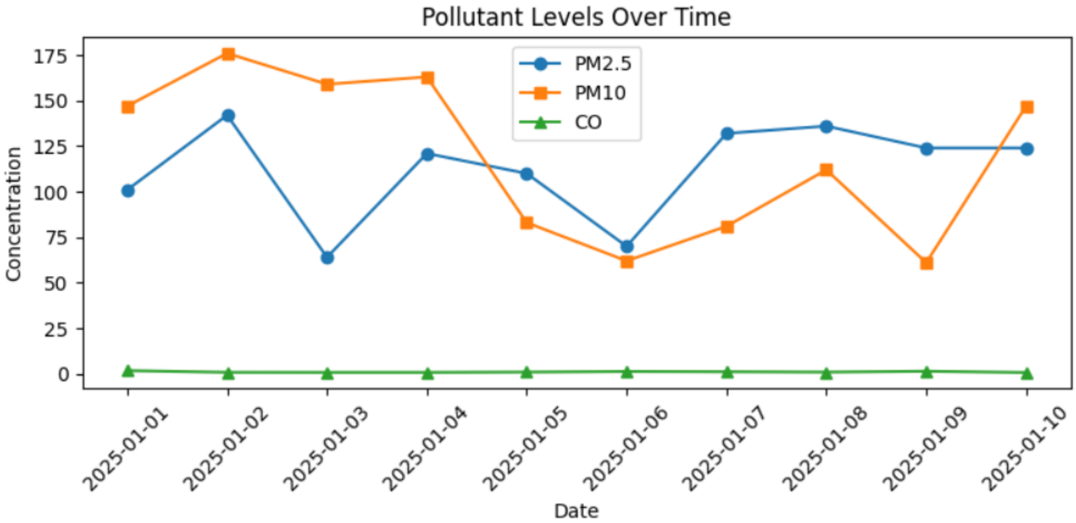
Output:

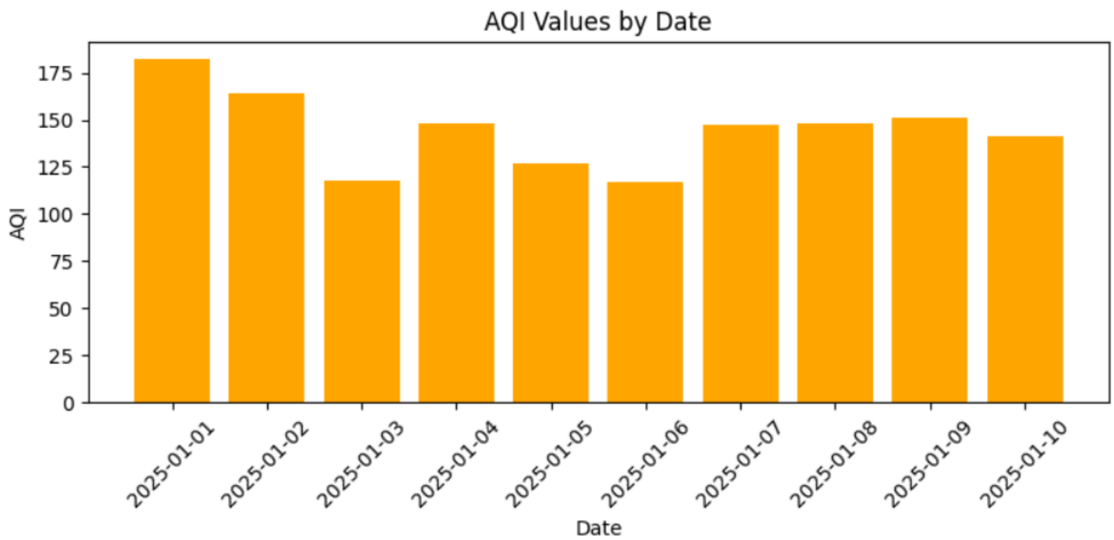
```
Step 1: Sample AQI Dataset Created
        Date  PM2.5  PM10        CO  AQI
0 2025-01-01    101   147  1.748664  182
1 2025-01-02    142   176  0.818509  164
2 2025-01-03     64   159  0.772737  118
3 2025-01-04    121   163  0.775107  148
4 2025-01-05    110    83  0.956363  127
5 2025-01-06     70    62  1.287135  117
6 2025-01-07    132    81  1.147918  147
7 2025-01-08    136   112  0.936844  148
8 2025-01-09    124    61  1.417779  151
9 2025-01-10    124   147  0.709241  141

Step 2: Dataset Info
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Date    10 non-null     datetime64[ns]
 1   PM2.5   10 non-null     int64
 2   PM10    10 non-null     int64
 3   CO      10 non-null     float64
 4   AQI     10 non-null     int64
dtypes: datetime64[ns](1), float64(1), int64(3)
memory usage: 532.0 bytes
None
```

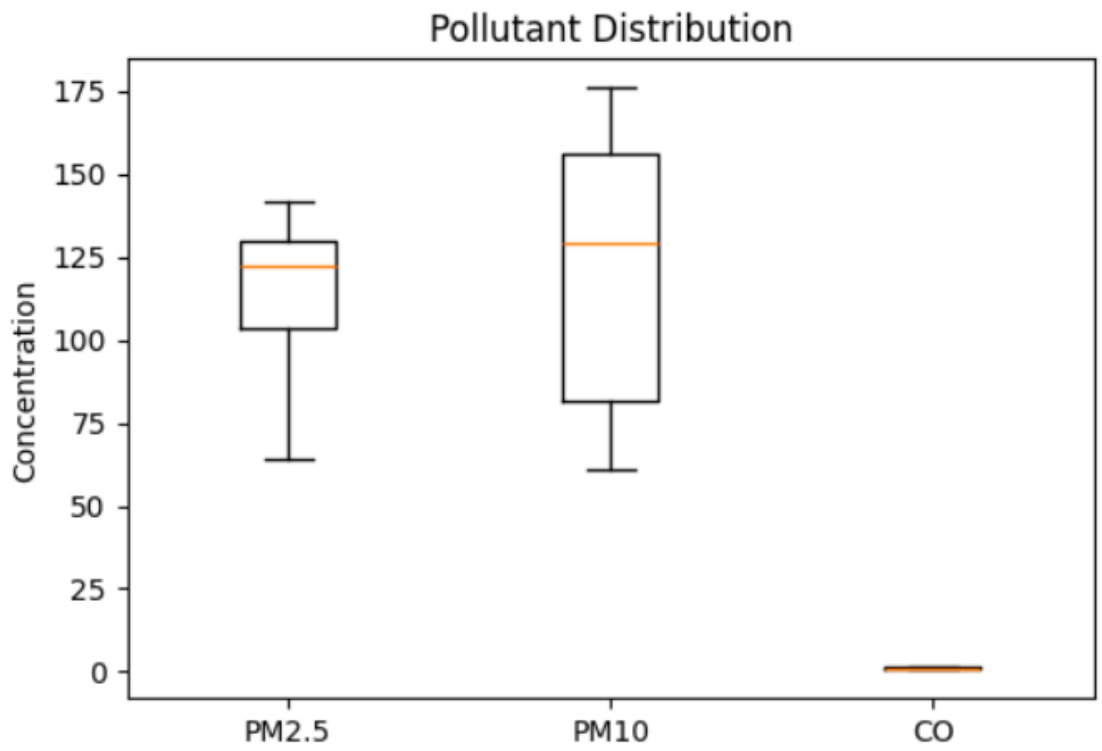
Step 2: Dataset Description

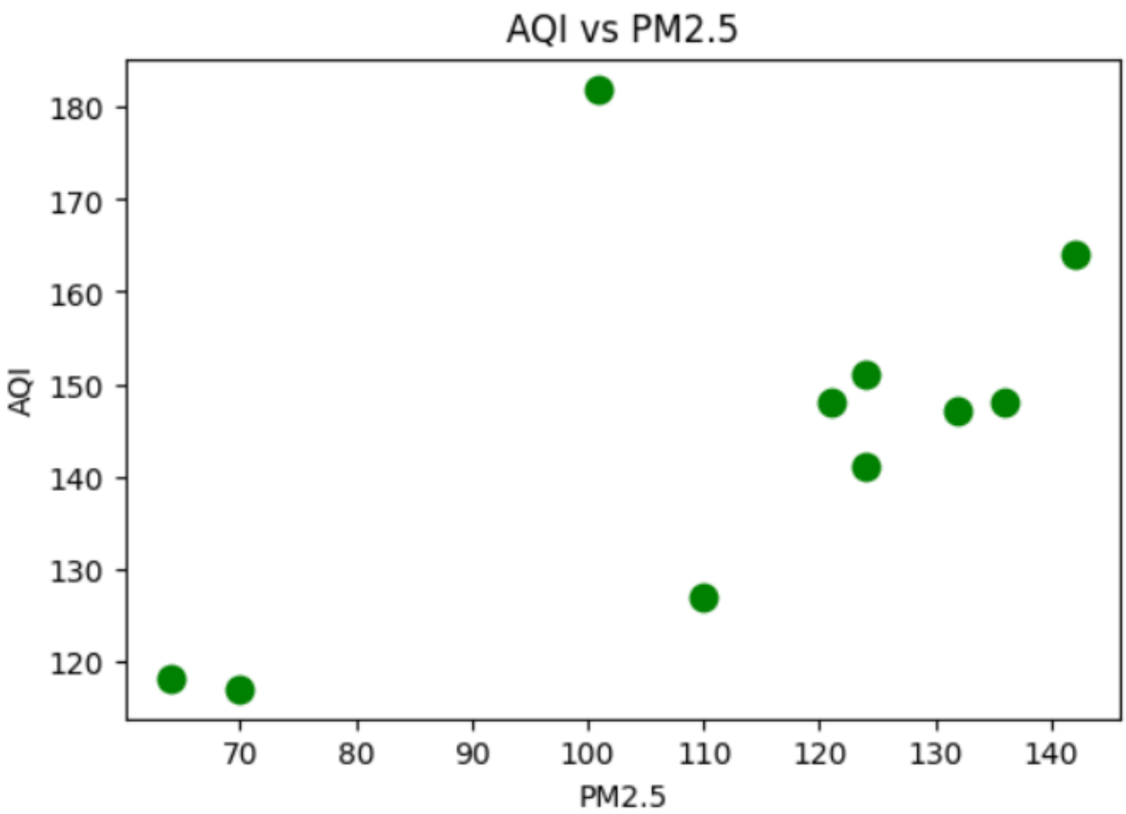
	Date	PM2.5	PM10	CO	AQI
count	10	10.000000	10.000000	10.000000	10.000000
mean	2025-01-05 12:00:00	112.400000	119.100000	1.057030	144.300000
min	2025-01-01 00:00:00	64.000000	61.000000	0.709241	117.000000
25%	2025-01-03 06:00:00	103.250000	81.500000	0.785957	130.500000
50%	2025-01-05 12:00:00	122.500000	129.500000	0.946604	147.500000
75%	2025-01-07 18:00:00	130.000000	156.000000	1.252330	150.250000
max	2025-01-10 00:00:00	142.000000	176.000000	1.748664	182.000000
std	NaN	26.742392	44.415838	0.338696	20.100028








Pollutant Distribution
Concentration
175
150
125
100
75
50
25
0
PM2.5
PM10
CO


AQI vs PM2.5
AQI
180
170
160
150
140
130
120
70
80
90
100
110
120
130
140
PM2.5

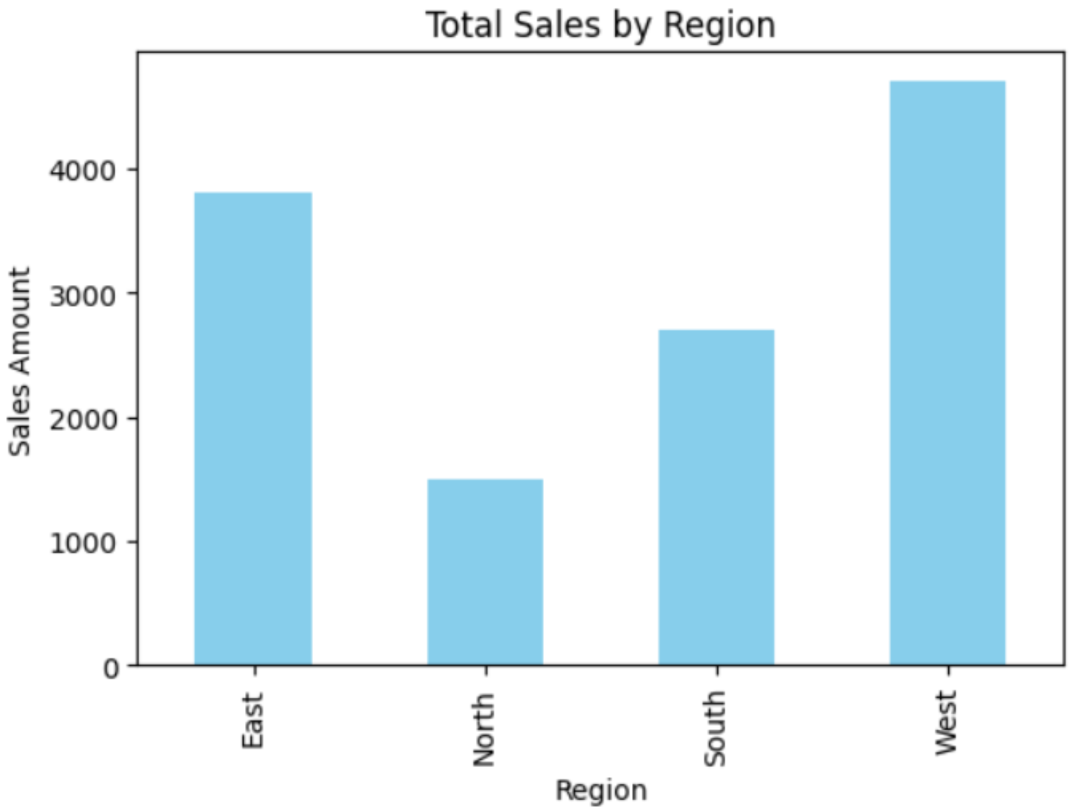
Practical No.: 12
Name: Anuj Shailendra Naikodi
Roll No:41

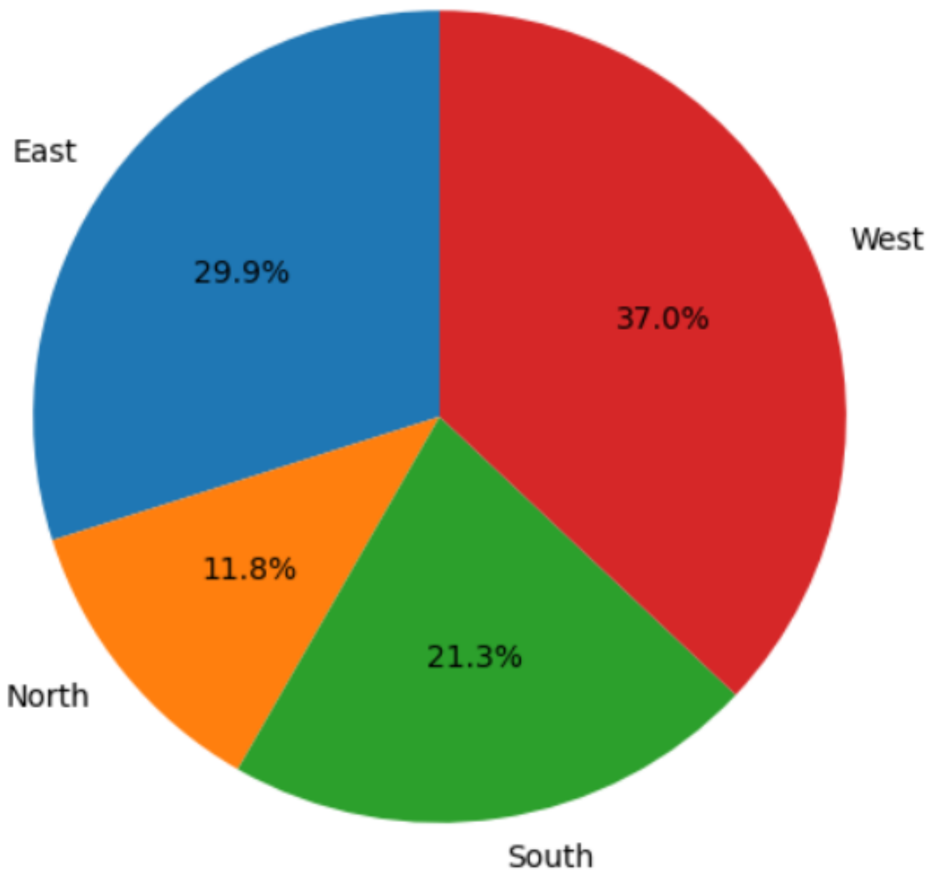
```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt


data = {

    "Region": ["North","South","East","West","North","South","East","West"],

    "Product_Category":
["Electronics","Electronics","Clothing","Clothing","Clothing","Electronics","Clothing","Elec
tronics"],

    "Quantity_Sold": [10,15,20,25,5,10,15,20],

    "Sales_Amount": [1000,1500,2000,2500,500,1200,1800,2200]

}


df = pd.DataFrame(data)

print("Step 1: Sample Retail Sales Dataset\n", df)


print("\nStep 2: Dataset Info")

print(df.info())

print("\nStep 2: Dataset Description")

print(df.describe())


sales_by_region = df.groupby('Region')['Sales_Amount'].sum()

print("\nStep 4: Total Sales by Region\n", sales_by_region)


plt.figure(figsize=(6,4))

sales_by_region.plot(kind='bar', color='skyblue')

plt.title("Total Sales by Region")

plt.ylabel("Sales Amount")
```

```
plt.show()


plt.figure(figsize=(6,6))
sales_by_region.plot(kind='pie', autopct='%1.1f%%', startangle=90)
plt.title("Sales Distribution by Region")
plt.ylabel("")
plt.show()


top_region = sales_by_region.idxmax()
top_sales = sales_by_region.max()
print(f"\nStep 6: Top-Performing Region: {top_region} with sales {top_sales}")


sales_region_category =
df.groupby(['Region','Product_Category'])['Sales_Amount'].sum().unstack()
print("\nStep 7: Sales by Region & Product Category\n", sales_region_category)


sales_region_category.plot(kind='bar', stacked=True, figsize=(8,5), colormap='viridis')
plt.title("Sales by Region and Product Category")
plt.ylabel("Sales Amount")
plt.show()
```

Output:

```
Step 1: Sample Retail Sales Dataset
  Region Product_Category  Quantity_Sold  Sales_Amount
0  North      Electronics             10          1000
1  South      Electronics             15          1500
2   East         Clothing             20          2000
3   West         Clothing             25          2500
4  North         Clothing              5           500
5  South      Electronics             10          1200
6   East         Clothing             15          1800
7   West      Electronics             20          2200

Step 2: Dataset Info
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8 entries, 0 to 7
Data columns (total 4 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Region            8 non-null      object
 1   Product_Category  8 non-null      object
 2   Quantity_Sold     8 non-null      int64
 3   Sales_Amount      8 non-null      int64
dtypes: int64(2), object(2)
memory usage: 388.0+ bytes
None

Step 2: Dataset Description
       Quantity_Sold  Sales_Amount
count       8.000000      8.000000
mean       15.000000   1587.500000
std         6.546537    666.413642
min         5.000000    500.000000
25%        10.000000   1150.000000
50%        15.000000   1650.000000
75%        20.000000   2050.000000
max        25.000000   2500.000000

Step 4: Total Sales by Region
 Region
East     3800
North    1500
South    2700
West     4700
Name: Sales_Amount, dtype: int64
```


Total Sales by Region
Sales Amount
4000
3000
2000
1000
0
East
North
South
West
Region


East
29.9%
West
37.0%
11.8%
21.3%
North
South

```
Region
East              3800.0        NaN
North              500.0     1000.0
South                NaN     2700.0
West              2500.0     2200.0
```