DATA MINING

ASSIGNMENT-1

MD. SADMAN HAQUE
Id: 011221592
Section: B

Table of Contents

1. Error Detection & Fixing

Block-1 (Load dataset)

```
import pandas as pd

# Load dataset
file_path = "D:/dm_assignment/datasets/error_dataset.csv"
df = pd.read_csv(file_path)

print("Original Dataset:")
print(df)
```

Output

```
Original Dataset:
   ID     Name      Age    Salary Department
0   1    Alice       25   50000.0         HR
1   2      Bob  thirty   60000.0    Finance
2   3  Charlie       28       NaN         IT
3   4    David       42   75000.0    Finance
4   5      Eva       35  -45000.0         HR
5   6    Frank       29   65000.0         IT
6   7    Grace       28   62000.0    Finance
7   8     Hank      NaN   58000.0         HR
8   9      Ivy       31   68000.0  Marketing
9  10    Grace       28   62000.0    Finance
```

Block-2 (Detect errors)

```
# Convert Age to numeric (invalid values become NaN)
df['Age'] = pd.to_numeric(df['Age'], errors='coerce')

# Convert Salary to numeric (invalid values become NaN)
df['Salary'] = pd.to_numeric(df['Salary'], errors='coerce')

# Detect missing values
print("\nMissing Values per Column:")
print(df.isnull().sum())

# Detect negative salaries
print("\nNegative Salaries:")
```

```
print(df[df['Salary'] < 0])

# Detect duplicate rows
print("\nDuplicate Rows:")
print(df[df.duplicated()])
```

Output

```
Missing Values per Column:
ID            0
Name          0
Age           2
Salary        1
Department    0
dtype: int64

Negative Salaries:
   ID Name   Age   Salary Department
4   5  Eva  35.0 -45000.0         HR

Duplicate Rows:
Empty DataFrame
Columns: [ID, Name, Age, Salary, Department]
Index: []
```

Block-3 (Fix errors)

```
# Fill missing ages with mean
df['Age'].fillna(df['Age'].mean(), inplace=True)

# Replace negative salaries with absolute values
df['Salary'] = df['Salary'].apply(lambda x: abs(x) if pd.notnull(x) and x
< 0 else x)

# Fill missing salaries with median
df['Salary'].fillna(df['Salary'].median(), inplace=True)

# Drop duplicate rows
df = df.drop_duplicates()

print("\nCleaned Dataset:")
print(df)
```

Output

```
Cleaned Dataset:
   ID     Name    Age   Salary Department
0   1    Alice  25.00  50000.0         HR
1   2      Bob  30.75  60000.0    Finance
2   3  Charlie  28.00  62000.0         IT
3   4    David  42.00  75000.0    Finance
4   5      Eva  35.00  45000.0         HR
5   6    Frank  29.00  65000.0         IT
6   7    Grace  28.00  62000.0    Finance
7   8     Hank  30.75  58000.0         HR
8   9      Ivy  31.00  68000.0  Marketing
9  10    Grace  28.00  62000.0    Finance
```

2. Noise/ Outlier Detection

Block-1 (Load dataset)

```
import pandas as pd

# Load dataset
file_path = "D:/dm_assignment/datasets/noisy_dataset.csv"
df = pd.read_csv(file_path)

print("Original Dataset:")
print(df)
```

Output

```
Original Dataset:
   ID  Age   Salary
0   1   25    50000
1   2   26    52000
2   3   27    51000
3   4   26    50500
4   5  200    51500
5   6   28    49000
6   7   29    49500
7   8   28  1000000
8   9   30    51000
9  10   27      abc
```

Block-2 (Detect noise)

```
# Convert Age and Salary to numeric, invalid values -> NaN
df['Age'] = pd.to_numeric(df['Age'], errors='coerce')
df['Salary'] = pd.to_numeric(df['Salary'], errors='coerce')

# Detect missing / invalid entries
print("\nNoise (Missing or Non-Numeric Values):")
print(df[df.isnull().any(axis=1)])
```

Output

```
Noise (Missing or Non-Numeric Values):
   ID  Age  Salary
9  10   27     NaN
```

Block-3 (Detect outliers)

```
# Z-score method for outlier detection
from scipy.stats import zscore

# Drop NaN for calculation
df_clean = df.dropna()

# Compute z-scores
z_scores = df_clean[['Age', 'Salary']].apply(zscore)

# Mark rows where z-score > 3 as outliers
outliers = df_clean[(abs(z_scores) > 3).any(axis=1)]

print("\nOutliers Detected (Z-score > 3):")
print(outliers)
```

Output

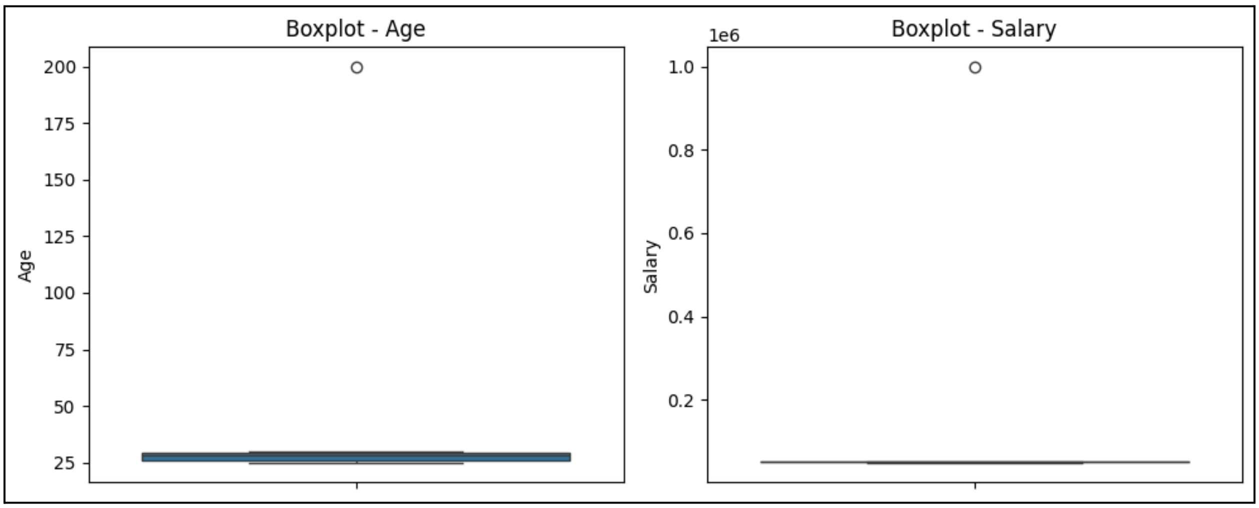
```
Outliers Detected (Z-score > 3):
Empty DataFrame
Columns: [ID, Age, Salary]
Index: []
```

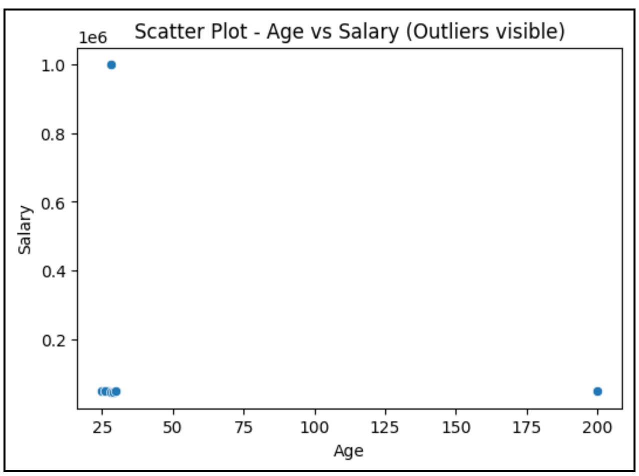
Block-4 (Plots)

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

# Convert Salary to numeric (invalid -> NaN)
df['Salary'] = pd.to_numeric(df['Salary'], errors='coerce')

# Drop NaN for plotting
df_plot = df.dropna()

# Boxplots
plt.figure(figsize=(10,4))

plt.subplot(1,2,1)
sns.boxplot(y=df_plot['Age'])
plt.title("Boxplot - Age")
```

```
plt.subplot(1,2,2)
sns.boxplot(y=df_plot['Salary'])
plt.title("Boxplot - Salary")

plt.tight_layout()
plt.show()

# Scatter plot
plt.figure(figsize=(6,4))
sns.scatterplot(x='Age', y='Salary', data=df_plot)
plt.title("Scatter Plot - Age vs Salary (Outliers visible)")
plt.show()
```

Output





3. Linear Regression

Block-1 (Load dataset)

```
import pandas as pd

# Load dataset
file_path = "D:/dm_assignment/datasets/linear_regression_dataset.csv"
df = pd.read_csv(file_path)

print("Dataset:")
print(df)
```

Output

```
Dataset:
   x    y
0  1  1.2
1  2  1.8
2  3  2.6
3  4  3.2
4  5  3.8
```

Block-2 (Build equation)

```
import numpy as np

# Variables
x = df['x']
y = df['y']

# Calculate coefficients manually
n = len(df)
sum_x = df['x'].sum()
sum_y = df['y'].sum()
sum_xy = (df['x'] * df['y']).sum()
sum_x2 = (df['x'] ** 2).sum()

# Slope (a1)
a1 = (sum_xy - (sum_x * sum_y) / n) / (sum_x2 - (sum_x**2) / n)
```

```python
# Intercept (a0)
x_mean = df['x'].mean()
y_mean = df['y'].mean()
a0 = y_mean - a1 * x_mean

print("Slope (a1):", a1)
print("Intercept (a0):", a0)
print(f"Equation: y = {a0:.2f} + {a1:.2f}x")
```

Output

```
Slope (a1): 0.6600000000000001
Intercept (a0): 0.54
Equation: y = 0.54 + 0.66x
```

Block-3 (Prediction)

```python
# Predict y for new x value
new_x = 6
predicted_y = a0 + a1 * new_x

print(f"Prediction: For x = {new_x}, y = {predicted_y:.2f}")
```

Output

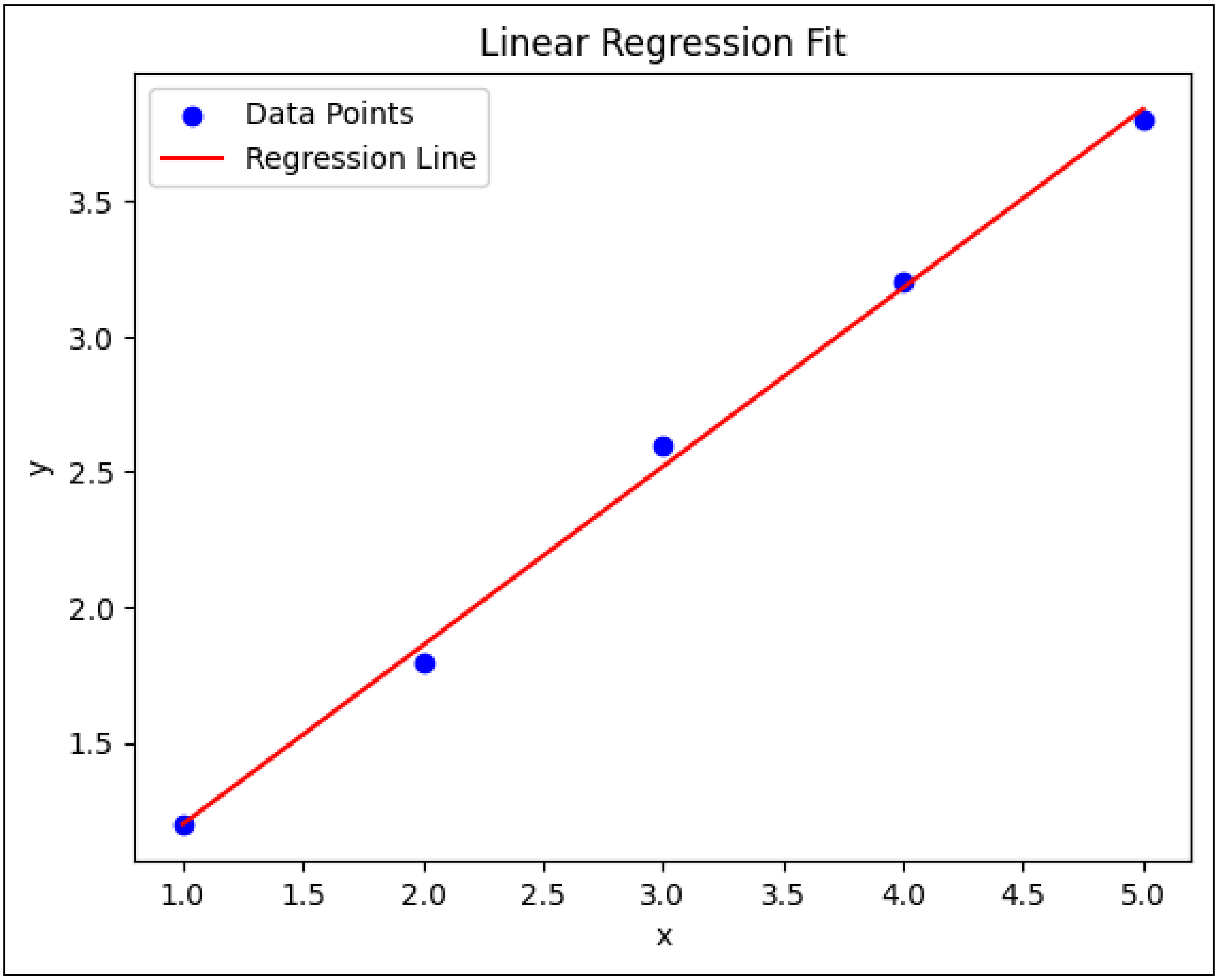
```
Prediction: For x = 6, y = 4.50
```

Block-4 (Linear plot)

```python
import matplotlib.pyplot as plt

# Scatter plot of actual data
plt.scatter(df['x'], df['y'], color='blue', label='Data Points')

# Regression line: y = a0 + a1x
x_vals = df['x']
y_pred = a0 + a1 * x_vals

plt.plot(x_vals, y_pred, color='red', label='Regression Line')

# Labels and title
plt.xlabel("x")
```

```
plt.ylabel("y")
plt.title("Linear Regression Fit")
plt.legend()
plt.show()
```

Output



4. Multiple Linear Regression

Block-1 (Load dataset)

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

# Load dataset
file_path =
"D:/dm_assignment/datasets/multiple_linear_regression_dataset.csv"
df = pd.read_csv(file_path)

# Features and target
X = df[['x1', 'x2']]
y = df['y']
```

Block-2 (Build equation)

```
# Fit the model
model = LinearRegression()
model.fit(X, y)

# Coefficients
a0 = model.intercept_
a1, a2 = model.coef_

print("Intercept (a0):", a0)
print("Coefficient for x1 (a1):", a1)
print("Coefficient for x2 (a2):", a2)

# Equation
print(f"Equation: y = {a0:.2f} + {a1:.2f}*x1 + {a2:.2f}*x2")
```

Output

```
Intercept (a0): 0.5000000000000071
Coefficient for x1 (a1): 6.499999999999999
Coefficient for x2 (a2): -1.5000000000000009
Equation: y = 0.50 + 6.50*x1 + -1.50*x2
```

Block-3 (Prediction)

```
# Prediction with new values
new_data = pd.DataFrame({'x1': [4], 'x2': [7]})
predicted_y = model.predict(new_data)[0]

print(f"Prediction: For x1=4, x2=7 → y = {predicted_y:.2f}")
```

Output

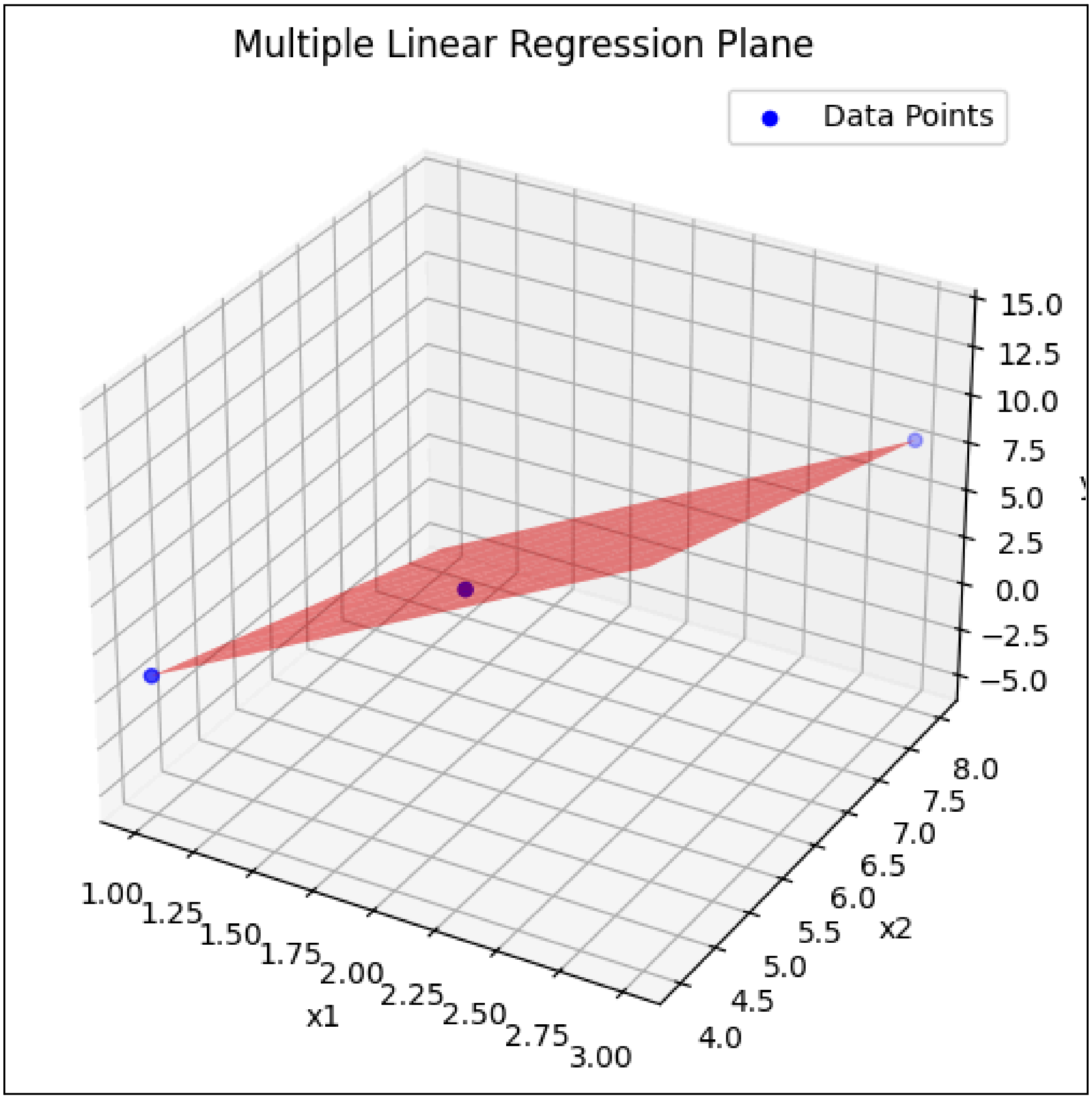
```
Prediction: For x1=4, x2=7 → y = 16.00
```

Block-4 (Plot)

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np

# Scatter plot of actual data
fig = plt.figure(figsize=(8,6))
ax = fig.add_subplot(111, projection='3d')

ax.scatter(df['x1'], df['x2'], y, color='blue', label='Data Points')

# Create meshgrid for regression plane
x1_range = np.linspace(df['x1'].min(), df['x1'].max(), 10)
x2_range = np.linspace(df['x2'].min(), df['x2'].max(), 10)
x1_grid, x2_grid = np.meshgrid(x1_range, x2_range)

# Predict y values on grid
y_pred_grid = a0 + a1 * x1_grid + a2 * x2_grid

# Plot regression plane
ax.plot_surface(x1_grid, x2_grid, y_pred_grid, color='red', alpha=0.5)

# Labels
ax.set_xlabel("x1")
ax.set_ylabel("x2")
ax.set_zlabel("y")
ax.set_title("Multiple Linear Regression Plane")
```

```
plt.legend()
plt.show()
```

Output



5. Polynomial Regression

Block-1 (Load dataset)

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression

# Load dataset
file_path = "D:/dm_assignment/datasets/polynomial_regression_dataset.csv"
df = pd.read_csv(file_path)

X = df[['x']]
y = df['y']
```

Block-2 (Build equation)

```
# Transform features for polynomial regression (degree 2)
poly = PolynomialFeatures(degree=2)
X_poly = poly.fit_transform(X)

# Fit polynomial regression model
model = LinearRegression()
model.fit(X_poly, y)

# Coefficients
a0 = model.intercept_
a1, a2 = model.coef_[1], model.coef_[2]

print("Intercept (a0):", a0)
print("Coefficient for x (a1):", a1)
print("Coefficient for x^2 (a2):", a2)

# Polynomial Equation
print(f"Equation: y = {a0:.2f} + {a1:.2f}x + {a2:.2f}x²")
```

Output

```
Intercept (a0): -0.7499999999999964
Coefficient for x (a1): 0.9500000000000011
Coefficient for x^2 (a2): 0.7499999999999992
Equation: y = -0.75 + 0.95x + 0.75x²
```

Block-3 (Prediction)

```
# Prediction for a new value
new_x = 5
new_x_poly = poly.transform([[new_x]])
predicted_y = model.predict(new_x_poly)[0]
print(f"Prediction: For x = {new_x}, y = {predicted_y:.2f}")
```

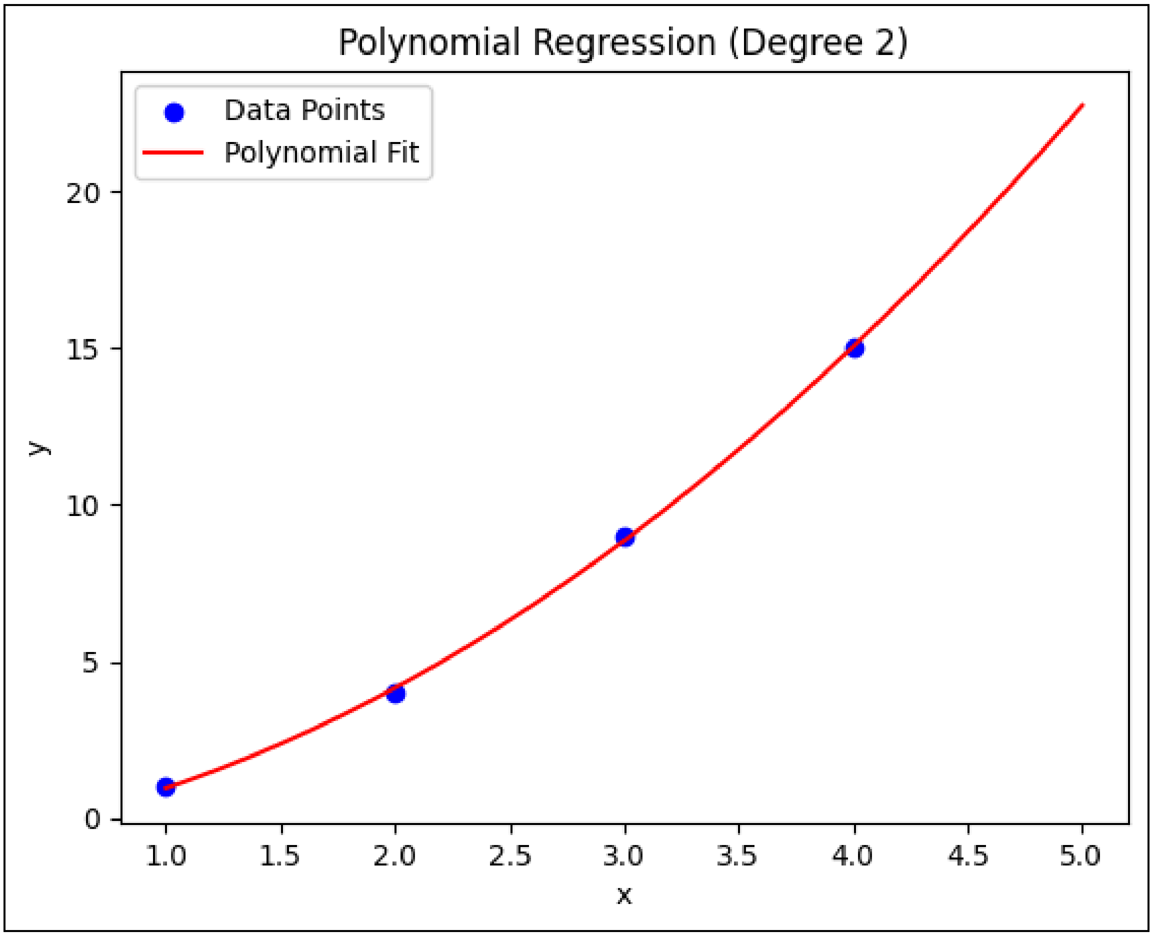
Output

```
Prediction: For x = 5, y = 22.75
```

Block-4 (Plot)

```
# Plot
plt.scatter(X, y, color='blue', label="Data Points")

# Generate smooth curve
x_range = np.linspace(min(X['x']), max(X['x'])+1, 100).reshape(-1,1)
y_pred_curve = model.predict(poly.transform(x_range))

plt.plot(x_range, y_pred_curve, color='red', label="Polynomial Fit")
plt.xlabel("x")
plt.ylabel("y")
plt.title("Polynomial Regression (Degree 2)")
plt.legend()
plt.show()
```

Output


Polynomial Regression (Degree 2)
Data Points
Polynomial Fit
y
x
0
5
10
15
20
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0

6. Logistic Regression

Block-1 (Load dataset)

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
import numpy as np

# Load dataset
file_path = "D:/dm_assignment/datasets/logistic_regression_dataset.csv"
df = pd.read_csv(file_path)

X = df[['hrs']]
y = df['pass_fail']
```

Block-2 (Build equation)

```
# Fit logistic regression model
model = LogisticRegression()
model.fit(X, y)

# Coefficients
a0 = model.intercept_[0]
a1 = model.coef_[0][0]

print("Intercept (a0):", a0)
print("Coefficient for hrs (a1):", a1)
print(f"Equation: p = 1 / (1 + e^(-({a0:.2f} + {a1:.2f}*hrs)))")
```

Output

```
Intercept (a0): -9.971005141040912
Coefficient for hrs (a1): 0.35963478771919 69
Equation: p = 1 / (1 + e^(-(-9.97 + 0.36*hrs)))
```

Block-3 (Prediction)

```
# Prediction for a student with 20 hrs
new_x = np.array([[33]])
pred_prob = model.predict_proba(new_x)[0][1]  # probability of passing
pred_class = model.predict(new_x)[0]
```

```
print(f"\nPrediction: For 33 hours → Probability of pass = {pred_prob:.2f}, Class = {pred_class}")
```

Output

Prediction: For 33 hours → Probability of pass = 0.87, Class = 1

Block-4 (Log-odds for 95% probability)

```
import math

# log-odds for 95% probability
log_odds = np.log(0.95 / (1 - 0.95))

# Solve for hrs
required_hrs = (log_odds - a0) / a1
print(f"\nMinimum hours required for >95% probability of passing: {required_hrs:.2f}")
```

Output

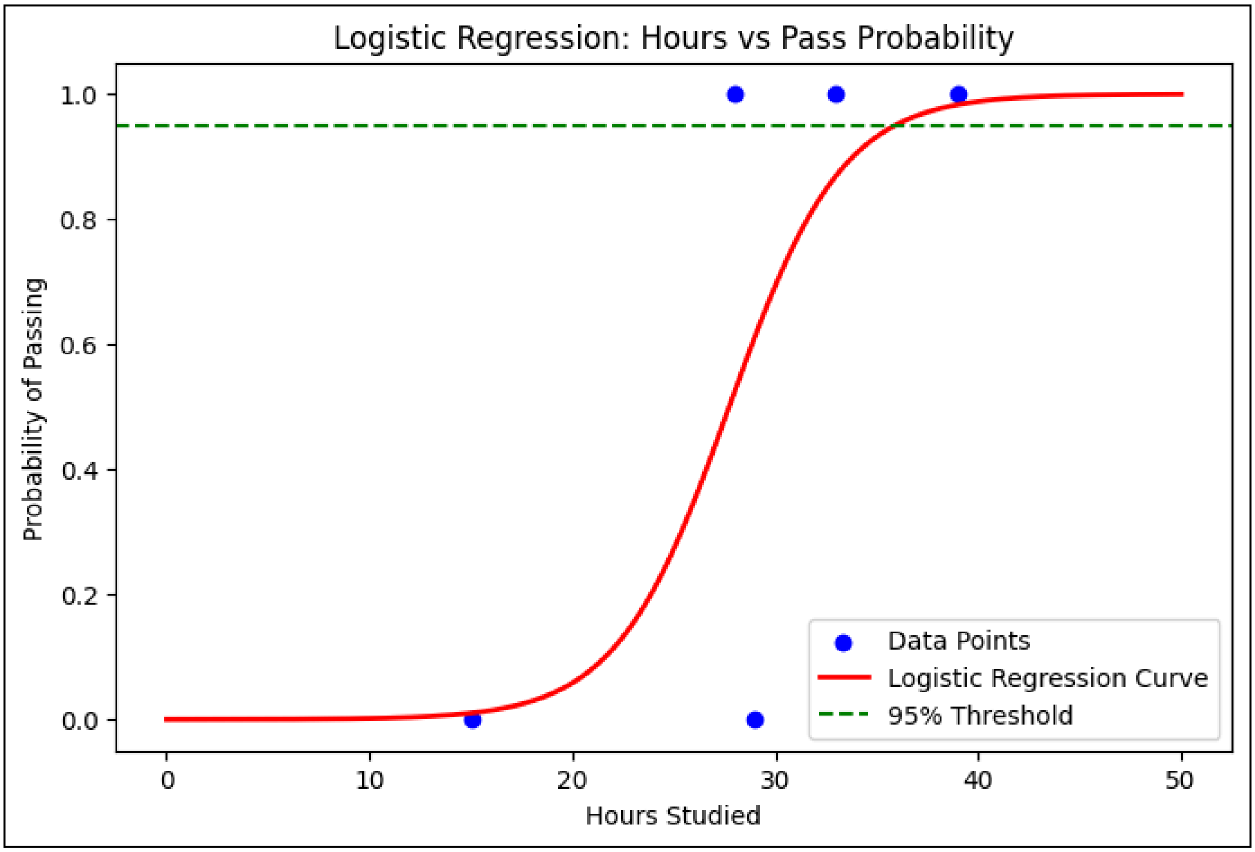
Minimum hours required for >95% probability of passing: 35.91

Block-5 (Plot)

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

# Range of study hours for plotting
x_range = np.linspace(0, 50, 200).reshape(-1, 1)
y_probs = model.predict_proba(x_range)[:, 1]  # probability of passing

# Plot
plt.figure(figsize=(8,5))
plt.scatter(df['hrs'], df['pass_fail'], color='blue', label='Data Points')
plt.plot(x_range, y_probs, color='red', linewidth=2, label='Logistic Regression Curve')

# Reference line at 0.95 probability
plt.axhline(y=0.95, color='green', linestyle='--', label='95% Threshold')
```

```
plt.xlabel("Hours Studied")
plt.ylabel("Probability of Passing")
plt.title("Logistic Regression: Hours vs Pass Probability")
plt.legend()
plt.show()
```

Output



7. Decision Tree (Entropy)

Block-1 (Load dataset)

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier, plot_tree
import matplotlib.pyplot as plt

# Load dataset
file_path = "D:/dm_assignment/datasets/decision_tree_dataset.csv"
df = pd.read_csv(file_path)

# Separate features and target
X = df[['Temp', 'Taste', 'Size']]
y = df['Appeal']
```

Block-2 (Train DT model)

```
# Encode categorical variables
encoders = {}
for column in X.columns:
    enc = LabelEncoder()
    X[column] = enc.fit_transform(X[column])
    encoders[column] = enc

y_enc = LabelEncoder().fit_transform(y)

# Train Decision Tree
model = DecisionTreeClassifier(criterion="entropy", random_state=0)
model.fit(X, y_enc)
```

Output

Parameters

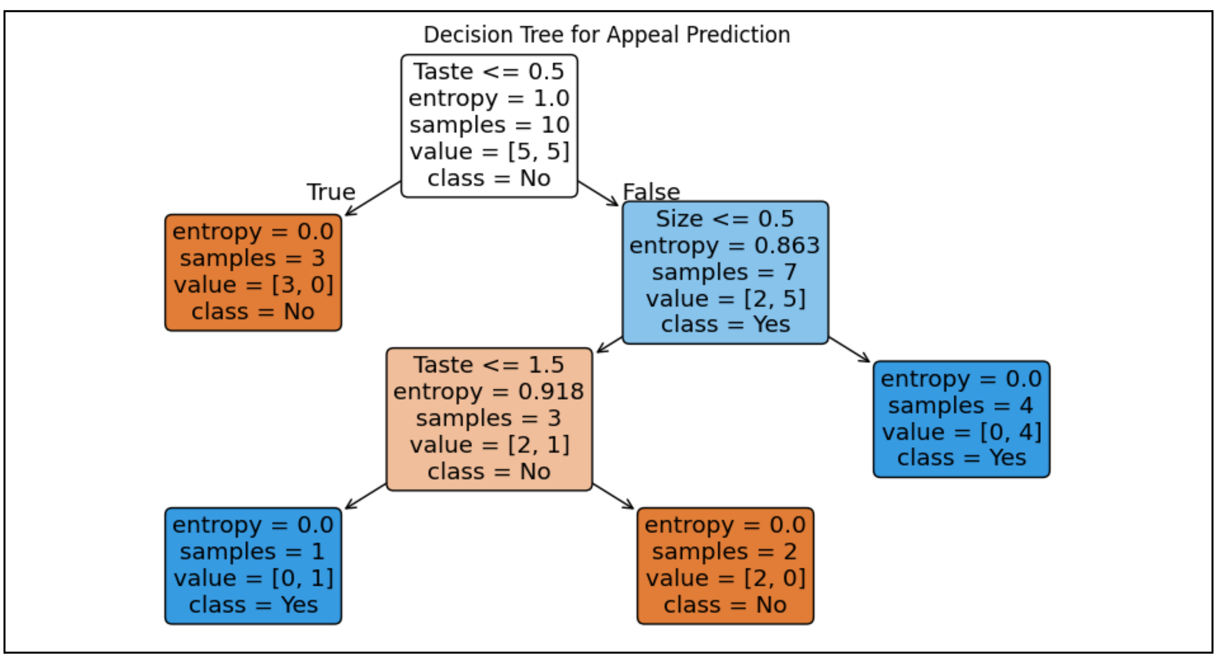
	criterion	'entropy'
	splitter	'best'
	max_depth	None
	min_samples_split	2

	min_samples_leaf	1
	min_weight_fraction_leaf	0.0
	max_features	None
	random_state	0
	max_leaf_nodes	None
	min_impurity_decrease	0.0
	class_weight	None
	ccp_alpha	0.0
	monotonic_cst	None

Block-3 (Plot tree)

```
# Plot Decision Tree
plt.figure(figsize=(12,6))
plot_tree(model, feature_names=X.columns, class_names=['No', 'Yes'],
filled=True, rounded=True)
plt.title("Decision Tree for Appeal Prediction")
plt.show()
```

Output



8. Aporiri Algorithm

Block-1 (Load dataset)

```
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, association_rules

# Load dataset
df = pd.read_csv("D:/dm_assignment/datasets/apriori_dataset.csv")
```

Block-2 (Generate rules)

```
# Convert items column into transactions (list of lists)
transactions = df['Items'].apply(lambda x: x.split(", ")).tolist()

# Transform into one-hot encoded dataframe
te = TransactionEncoder()
te_ary = te.fit(transactions).transform(transactions)
df_trans = pd.DataFrame(te_ary, columns=te.columns_)

# Apply Apriori with min support = 0.57 (57%)
frequent_itemsets = apriori(df_trans, min_support=0.57, use_colnames=True)

# Generate association rules with min confidence = 0.83 (83%)
rules = association_rules(frequent_itemsets, metric="confidence",
min_threshold=0.83)
```

Block-3 (Result)

```
# Show results
print("Frequent Itemsets:")
print(frequent_itemsets)

print("\nAssociation Rules:")
print(rules[['antecedents', 'consequents', 'support', 'confidence',
'lift']])
```

Output

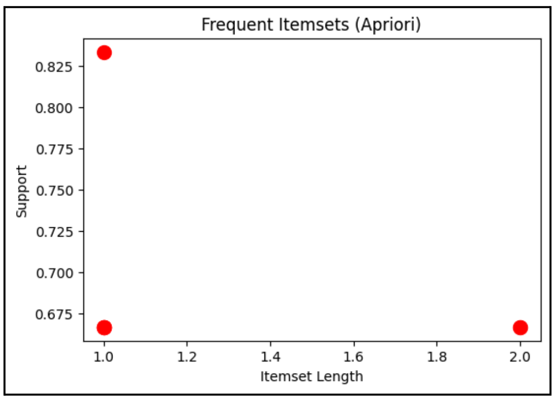
```
Frequent Itemsets:
    support         itemsets
0  0.666667          (bread)
1  0.666667           (cola)
2  0.666667         (diaper)
3  0.666667          (juice)
4  0.833333           (milk)
5  0.666667     (cola, milk)
6  0.666667   (milk, diaper)

Association Rules:
  antecedents consequents   support  confidence  lift
0      (cola)      (milk)  0.666667         1.0   1.2
1    (diaper)      (milk)  0.666667         1.0   1.2
```

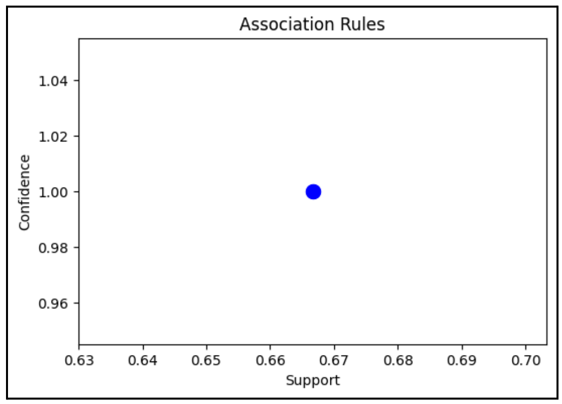
Block-4 (Plot)

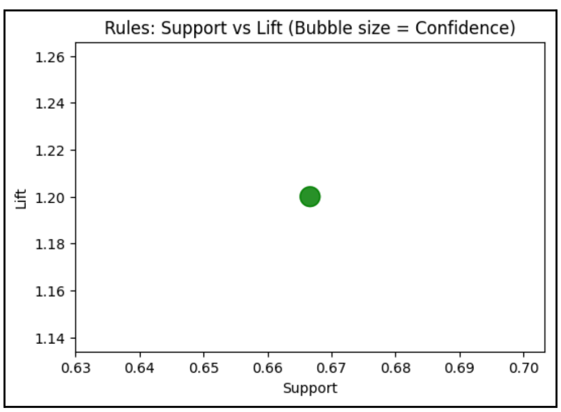
```
# Plot Frequent Itemsets (Support vs Itemset Length)

import matplotlib.pyplot as plt

# Add length column (number of items in each set)
frequent_itemsets['length'] = frequent_itemsets['itemsets'].apply(lambda
x: len(x))

plt.figure(figsize=(6,4))
plt.scatter(frequent_itemsets['length'], frequent_itemsets['support'],
s=100, c="red")
plt.xlabel("Itemset Length")
plt.ylabel("Support")
plt.title("Frequent Itemsets (Apriori)")
plt.show()

# Plot Association Rules (Support vs Confidence)
plt.figure(figsize=(6,4))
plt.scatter(rules['support'], rules['confidence'], s=100, c="blue")
plt.xlabel("Support")
plt.ylabel("Confidence")
plt.title("Association Rules")
plt.show()
```

```
# Support vs Lift (Bubble Plot)

plt.figure(figsize=(6,4))
plt.scatter(rules['support'], rules['lift'], s=rules['confidence']*200,
alpha=0.6, c="green")
plt.xlabel("Support")
plt.ylabel("Lift")
plt.title("Rules: Support vs Lift (Bubble size = Confidence)")
plt.show()
```

Output







9. Confusion Matrix (XGBoost)

Block-1 (Load dataset)

```
import pandas as pd

# Load dataset
file_path = "D:/dm_assignment/datasets/confusion_matrix.csv"
df = pd.read_csv(file_path)

print("Dataset shape:", df.shape)
print(df.head())

# Features and target
X = df.drop("price_range", axis=1)
y = df["price_range"]
```

Output

```
Dataset shape: (2000, 21)
   battery_power  blue  clock_speed  dual_sim  fc  four_g  int_memory
m_dep  \
0            842     0          2.2         0   1       0           7
0.6
1           1021     1          0.5         1   0       1          53
0.7
2            563     1          0.5         1   2       1          41
0.9
3            615     1          2.5         0   0       0          10
0.8
4           1821     1          1.2         0  13       1          44
0.6

   mobile_wt  n_cores  ...  px_height  px_width   ram  sc_h  sc_w
talk_time  \
0        188        2  ...         20       756  2549     9     7
19
1        136        3  ...        905      1988  2631    17     3
7
2        145        5  ...       1263      1716  2603    11     2
9
3        131        6  ...       1216      1786  2769    16     8
11
```

```
4         141         2  ...        1208       1212  1411      8      2
15

   three_g  touch_screen  wifi  price_range
0        0             0     1            1
1        1             1     0            2
2        1             1     0            2
3        1             0     0            2
4        1             1     0            1

[5 rows x 21 columns]
```

Block-2 (Train XGB model)

```python
from sklearn.model_selection import train_test_split, GridSearchCV
from xgboost import XGBClassifier

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)

# Define model
xgb = XGBClassifier(use_label_encoder=False, eval_metric="mlogloss")

# Hyperparameter grid
param_grid = {
    "n_estimators": [100, 200],
    "max_depth": [3, 5, 7],
    "learning_rate": [0.05, 0.1, 0.2],
    "subsample": [0.8, 1.0],
    "colsample_bytree": [0.8, 1.0]
}

grid = GridSearchCV(
    estimator=xgb, param_grid=param_grid,
    scoring="accuracy", cv=3, verbose=1, n_jobs=-1
)

# Train
grid.fit(X_train, y_train)
```

```
print("Best Parameters:", grid.best_params_)
best_model = grid.best_estimator_
```

Output

```
Fitting 3 folds for each of 72 candidates, totalling 216 fits
bst.update(dtrain, iteration=i, fobj=obj)
Best Parameters: {'colsample_bytree': 1.0, 'learning_rate': 0.2,
'max_depth': 3, 'n_estimators': 200, 'subsample': 0.8}
```

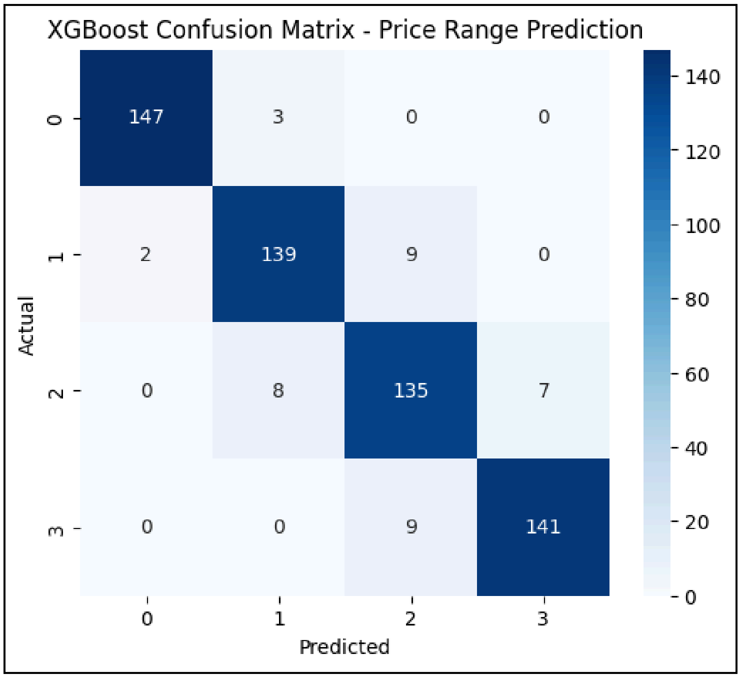
Block-3 (Results)

```
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Predictions
y_pred = best_model.predict(X_test)

# Classification Report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6,5))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("XGBoost Confusion Matrix - Price Range Prediction")
plt.show()
```

Output

```
Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.98      0.98       150
           1       0.93      0.93      0.93       150
           2       0.88      0.90      0.89       150
           3       0.95      0.94      0.95       150

    accuracy                           0.94       600
```

```
   macro avg       0.94      0.94      0.94       600
weighted avg       0.94      0.94      0.94       600
```

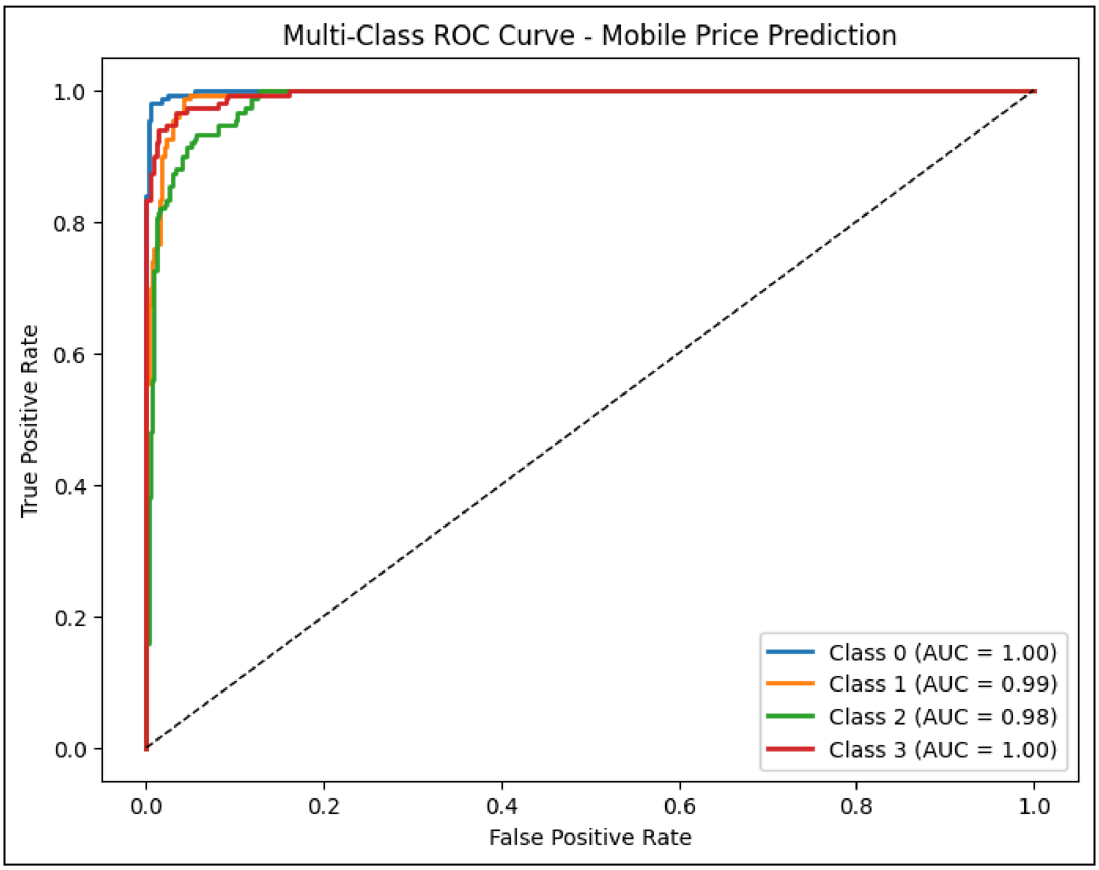


Block-4 (ROC-curve)

```
import numpy as np
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc

# Binarize labels for multi-class ROC
n_classes = len(y.unique())
y_test_bin = label_binarize(y_test, classes=list(range(n_classes)))
y_pred_prob = best_model.predict_proba(X_test)

# Compute ROC curve & AUC for each class
fpr, tpr, roc_auc = {}, {}, {}
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_pred_prob[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plot ROC Curves
plt.figure(figsize=(8,6))
for i in range(n_classes):
```

```
    plt.plot(fpr[i], tpr[i], lw=2, label=f"Class {i} (AUC =
{roc_auc[i]:.2f})")

plt.plot([0,1], [0,1], 'k--', lw=1)
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Multi-Class ROC Curve - Mobile Price Prediction")
plt.legend()
plt.show()
```

Output



10. KNN Classification

Block-1 (Load dataset)

```
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
import seaborn as sns
import matplotlib.pyplot as plt

# 1. Load Dataset
file_path = "D:/dm_assignment/datasets/knn_dataset.csv"
df = pd.read_csv(file_path)

# Features and target
X = df[['x1', 'x2', 'x3']]
y = df['label']
```

Block-2 (Train KNN model)

```
# Train KNN with k=4
knn = KNeighborsClassifier(n_neighbors=4)
knn.fit(X, y)
```

Output

Parameters

	n_neighbors	4
	weights	'uniform'
	algorithm	'auto'
	leaf_size	30
	p	2
	metric	'minkowski'
	metric_params	None
	n_jobs	None

Block-3 (Prediction)

```
# Classify new point (2,3,4)
```

```
new_point = [[2, 3, 4]]
prediction = knn.predict(new_point)

print(f"Classification of (2,3,4): {prediction[0]}")
```

Output

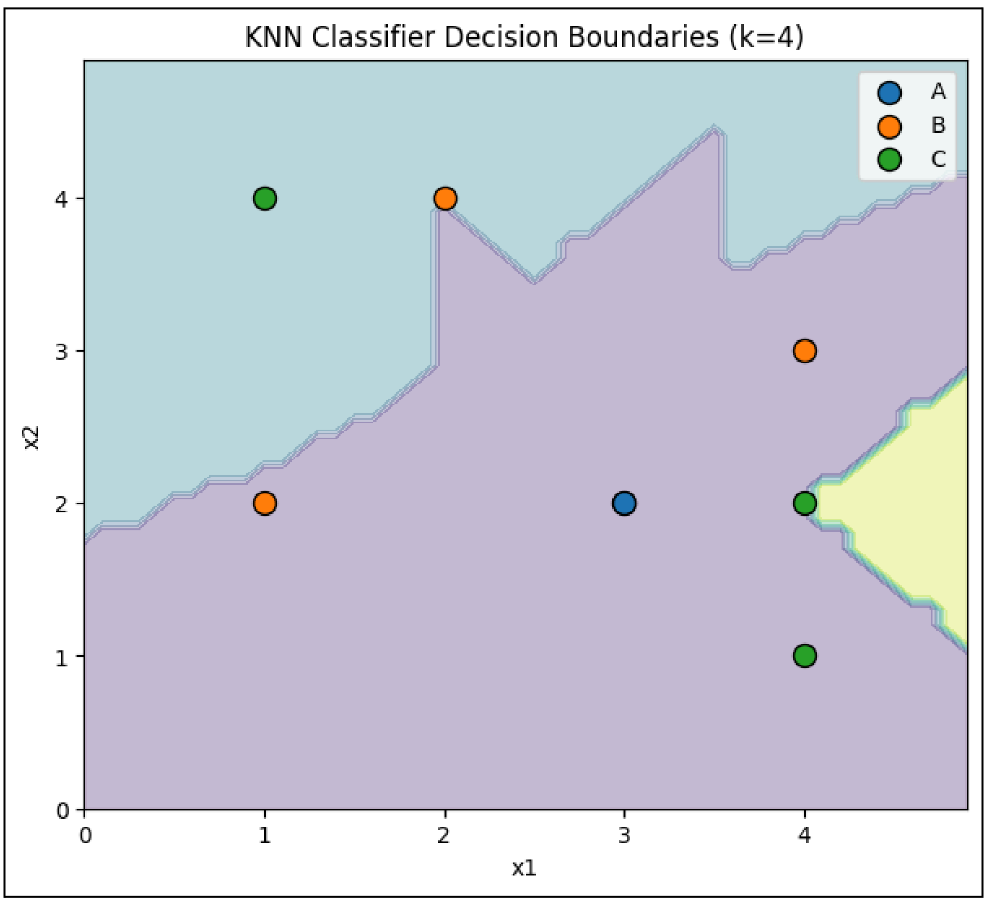
Classification of (2,3,4): A

Block-4 (Plot)

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import LabelEncoder

# 1. Load Dataset
df = pd.read_csv("D:/dm_assignment/datasets/knn_dataset.csv")

# Encode labels to numbers for plotting
le = LabelEncoder()
y = le.fit_transform(df['label'])    # convert A,B,C -> 0,1,2

X = df[['x1', 'x2']].values

# 2. Train KNN
knn = KNeighborsClassifier(n_neighbors=4)
knn.fit(X, y)

# 3. Decision Boundaries
h = 0.1
x_min, x_max = X[:,0].min() - 1, X[:,0].max() + 1
y_min, y_max = X[:,1].min() - 1, X[:,1].max() + 1

xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                     np.arange(y_min, y_max, h))

Z = knn.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
```

```
plt.figure(figsize=(7,6))
plt.contourf(xx, yy, Z, alpha=0.3, cmap='viridis')

# Plot Data Points
for class_index, class_label in enumerate(le.classes_):
    subset = df[df['label'] == class_label]
    plt.scatter(subset['x1'], subset['x2'], label=class_label, s=100,
edgecolors='k')

plt.xlabel("x1")
plt.ylabel("x2")
plt.title("KNN Classifier Decision Boundaries (k=4)")
plt.legend()
plt.show()
```

Output



11. K-Means Clustering

Block-1 (Load dataset)

```
import pandas as pd
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import numpy as np

# Load dataset
file_path = "D:/dm_assignment/datasets/kmeans.csv"
df = pd.read_csv(file_path)

# Features for clustering
X = df[['Area', 'Perimeter', 'Compactness']]
```

Block-2 (Train K-means cluster model & results)

```
# Features (2D visualization)
X = df[['Area', 'Perimeter', 'Compactness']].values

# Apply KMeans
kmeans = KMeans(n_clusters=3, random_state=42, n_init=10)
df['Cluster'] = kmeans.fit_predict(X)

print("Cluster Centers:")
print(kmeans.cluster_centers_)
print("\nClustered Data:")
print(df)
```

Output

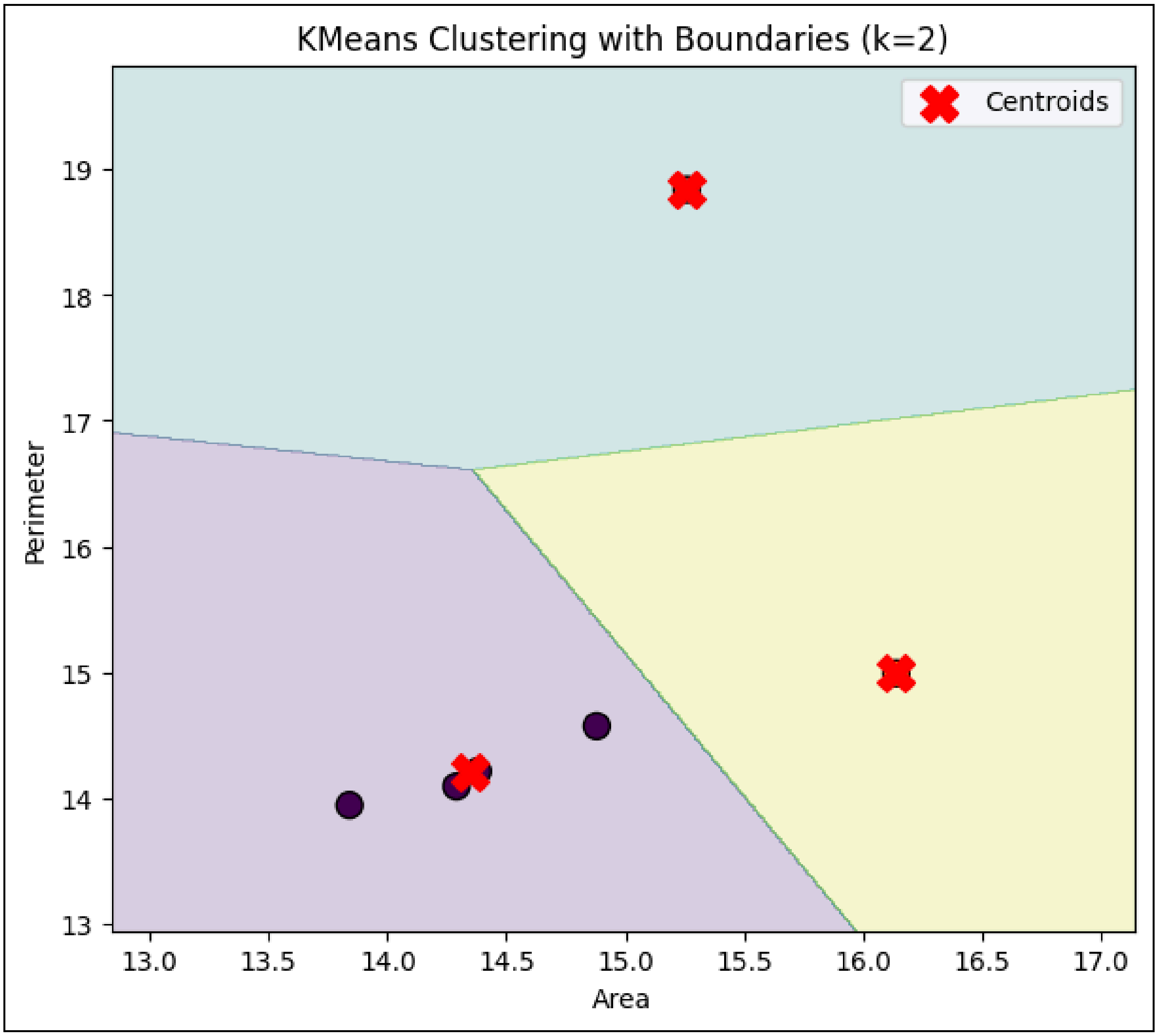
```
Cluster Centers:
[[14.3475   14.2025    0.895675]
 [15.26     18.84      0.871   ]
 [16.14     14.99      0.9034  ]]

Clustered Data:
    Area  Perimeter  Compactness  Cluster
0  15.26      18.84       0.8710        1
1  14.88      14.57       0.8871        0
2  14.29      14.09       0.9050        0
```

```
3   13.84      13.94      0.8955       0
4   16.14      14.99      0.9034       2
5   14.38      14.21      0.8951       0
```

Block-3 (Plot)

```
# Plot decision boundaries
h = 0.01  # step size in the mesh
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1

xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                     np.arange(y_min, y_max, h))

Z = kmeans.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

plt.figure(figsize=(7,6))
plt.contourf(xx, yy, Z, alpha=0.2, cmap='viridis')  # soft cluster regions

# Plot data points
plt.scatter(X[:, 0], X[:, 1], c=df['Cluster'], cmap='viridis', s=100,
edgecolors='k')

# Plot centroids
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
            marker='X', s=200, c='red', label='Centroids')

plt.xlabel("Area")
plt.ylabel("Perimeter")
plt.title("KMeans Clustering with Boundaries")
plt.legend()
plt.show()
```

Output


KMeans Clustering with Boundaries (k=2)
Centroids
Perimeter
19
18
17
16
15
14
13
13.0
13.5
14.0
14.5
15.0
15.5
16.0
16.5
17.0
Area

12. Bayes Classification

Block-1 (Load dataset)

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.naive_bayes import CategoricalNB

# 1. Load Dataset
file_path = "D:/dm_assignment/datasets/naive_bayes_dataset.csv"
df = pd.read_csv(file_path)

# Features and target
X = df[['Age', 'Income', 'Buyer', 'Credit_rating']]
y = df['Buys_laptop']
```

Block-2 (Train NB cluster model)

```
# Encode categorical features
encoders = {}
for col in X.columns:
    enc = LabelEncoder()
    X[col] = enc.fit_transform(X[col])
    encoders[col] = enc

y_enc = LabelEncoder().fit_transform(y)

# 2. Train Naive Bayes with Laplace smoothing (alpha=1)
nb = CategoricalNB(alpha=1.0)
nb.fit(X, y_enc)
```

Block-3 (Prediction)

```
# 3. Prediction on new data
new_data = pd.DataFrame({
    "Age": ["<=30"],
    "Income": ["Average"],
    "Buyer": ["Yes"],
    "Credit_rating": ["Fair"]
})
```

```
# Encode using same encoders
for col in new_data.columns:
    new_data[col] = encoders[col].transform(new_data[col])

# Predict
prediction = nb.predict(new_data)[0]
prediction_label = "Yes" if prediction == 1 else "No"

print("Prediction for new data:", prediction_label)
```

Output

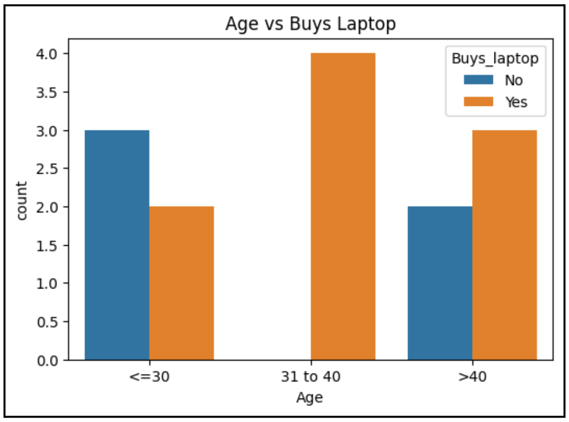
Prediction for new data: Yes

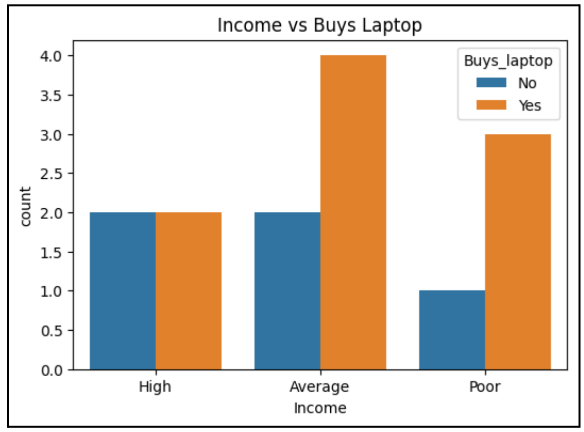
Block-4 (Data count probability plot)

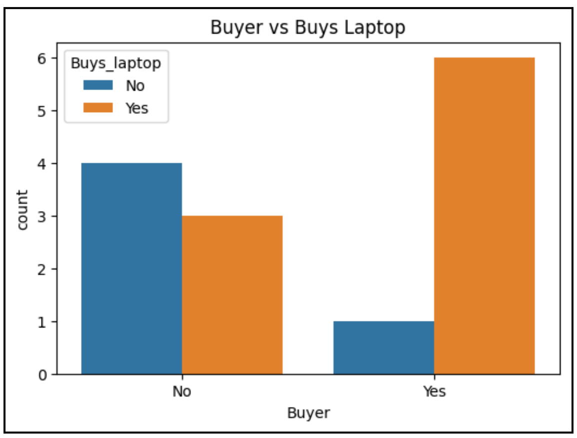
```
import seaborn as sns
import matplotlib.pyplot as plt

# Plot 1: Distribution of Age vs Buys Laptop
plt.figure(figsize=(6,4))
sns.countplot(data=df, x="Age", hue="Buys_laptop")
plt.title("Age vs Buys Laptop")
plt.show()

# Plot 2: Distribution of Income vs Buys Laptop
plt.figure(figsize=(6,4))
sns.countplot(data=df, x="Income", hue="Buys_laptop")
plt.title("Income vs Buys Laptop")
plt.show()

# Plot 3: Distribution of Buyer vs Buys Laptop
plt.figure(figsize=(6,4))
sns.countplot(data=df, x="Buyer", hue="Buys_laptop")
plt.title("Buyer vs Buys Laptop")
plt.show()

# Plot 4: Distribution of Credit Rating vs Buys Laptop
plt.figure(figsize=(6,4))
sns.countplot(data=df, x="Credit_rating", hue="Buys_laptop")
```

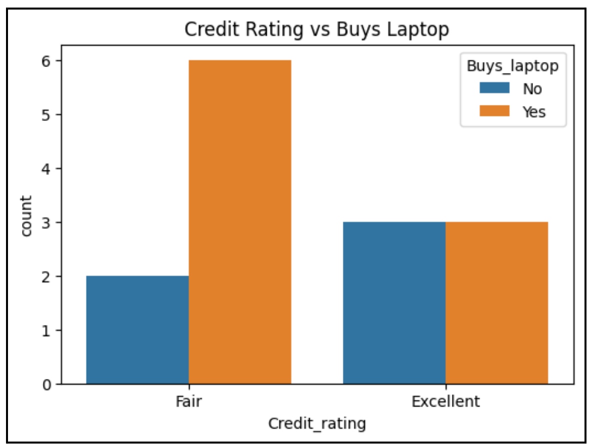
```
plt.title("Credit Rating vs Buys Laptop")
plt.show()
```

Output









All the codes and datasets are available at github:
https://github.com/sadman-adib/Data-Mining-Lab/tree/main/data_mining_assignment