VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT

on

Machine Learning (23CS6PCMAL)

Submitted by

Bhanu Prakash M (1BM22CS067)

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING

(Autonomous Institution under VTU)

BENGALURU-560019

Sep-2024 to Jan-2025

B.M.S. College of Engineering,
Bull Temple Road, Bangalore 560019
(Affiliated To Visvesvaraya Technological University, Belgaum)
Department of Computer Science and Engineering

CERTIFICATE

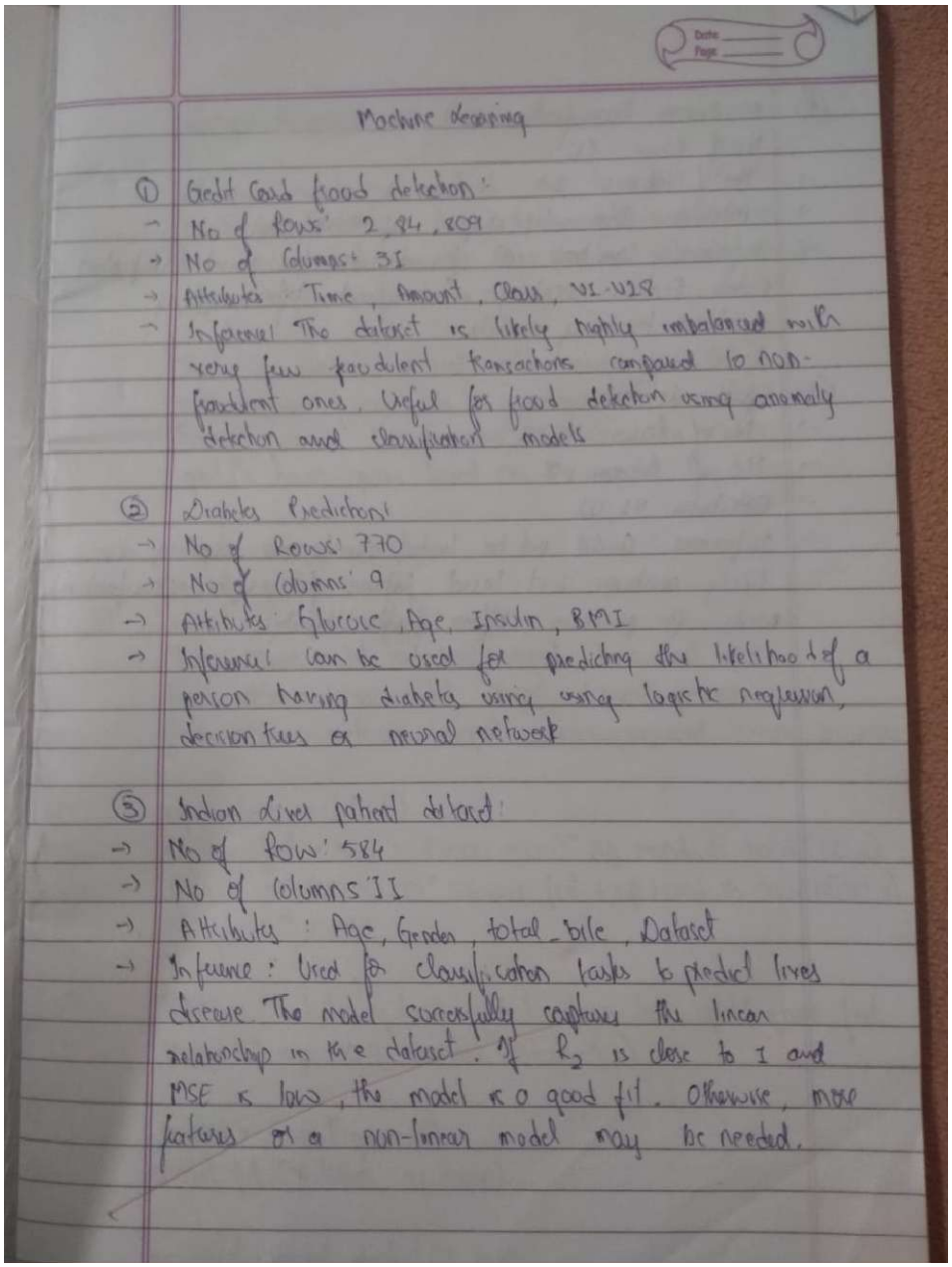
This is to certify that the Lab work entitled "Machine Learning (23CS6PCMAL)" carried out by Bhanu Prakash M(1BM22CS067), who is bonafide student of B.M.S. College of Engineering. It is in partial fulfillment for the award of Bachelor of Engineering in Computer Science and Engineering of the Visvesvaraya Technological University, Belgaum. The Lab report has been approved as it satisfies the academic requirements in respect of an Machine Learning (23CS6PCMAL) work prescribed for the said degree.

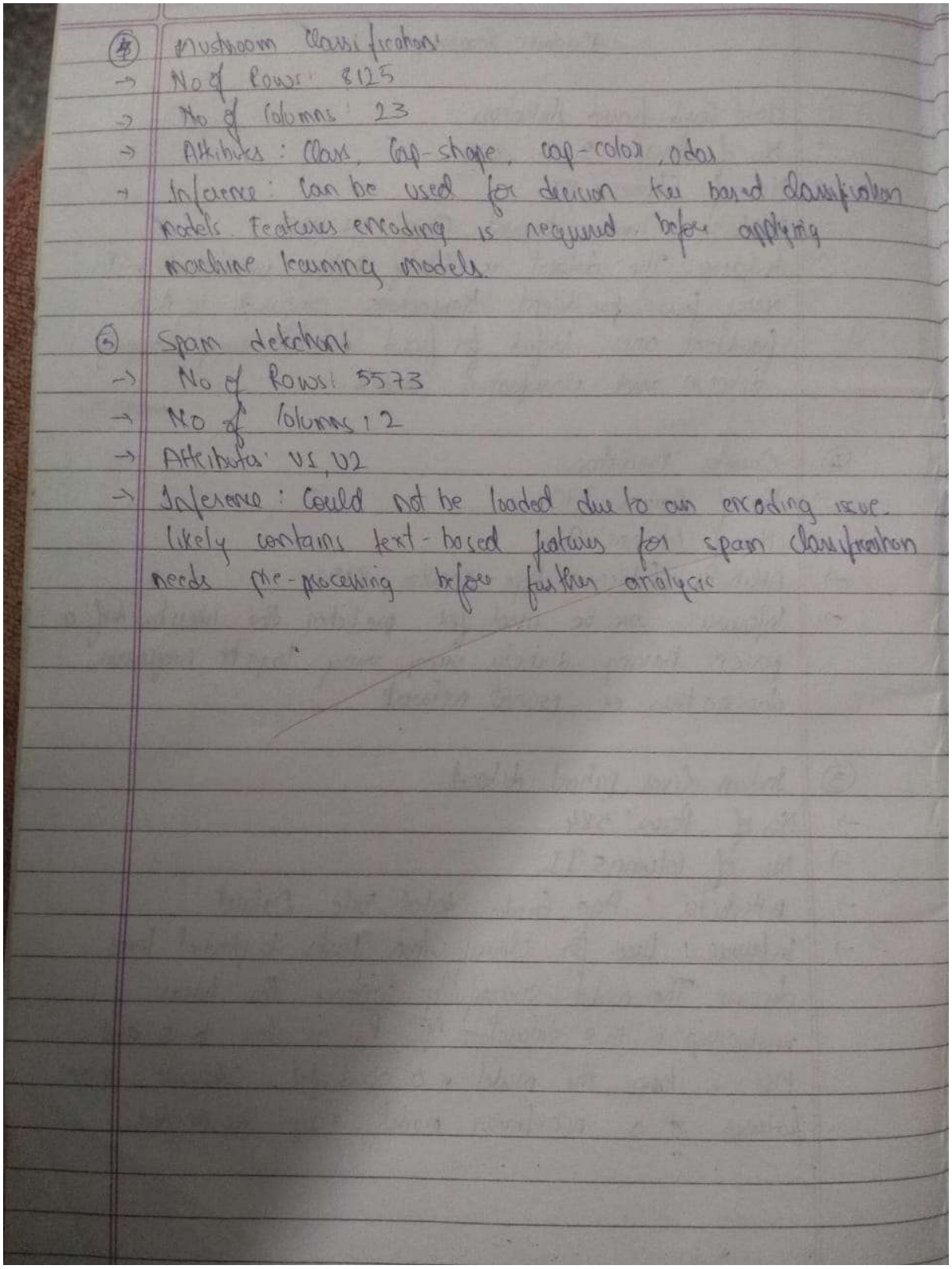
Lab faculty Incharge Name : Megha J Assistant Professor Department of CSE, BMSCE	Dr. Kavitha Sooda Professor & HOD Department of CSE, BMSCE

Index

Github Link: https://github.com/bhanu87777/MachineLearningLab.git

Program 1: Write a python program to import and export data using Pandas library functions





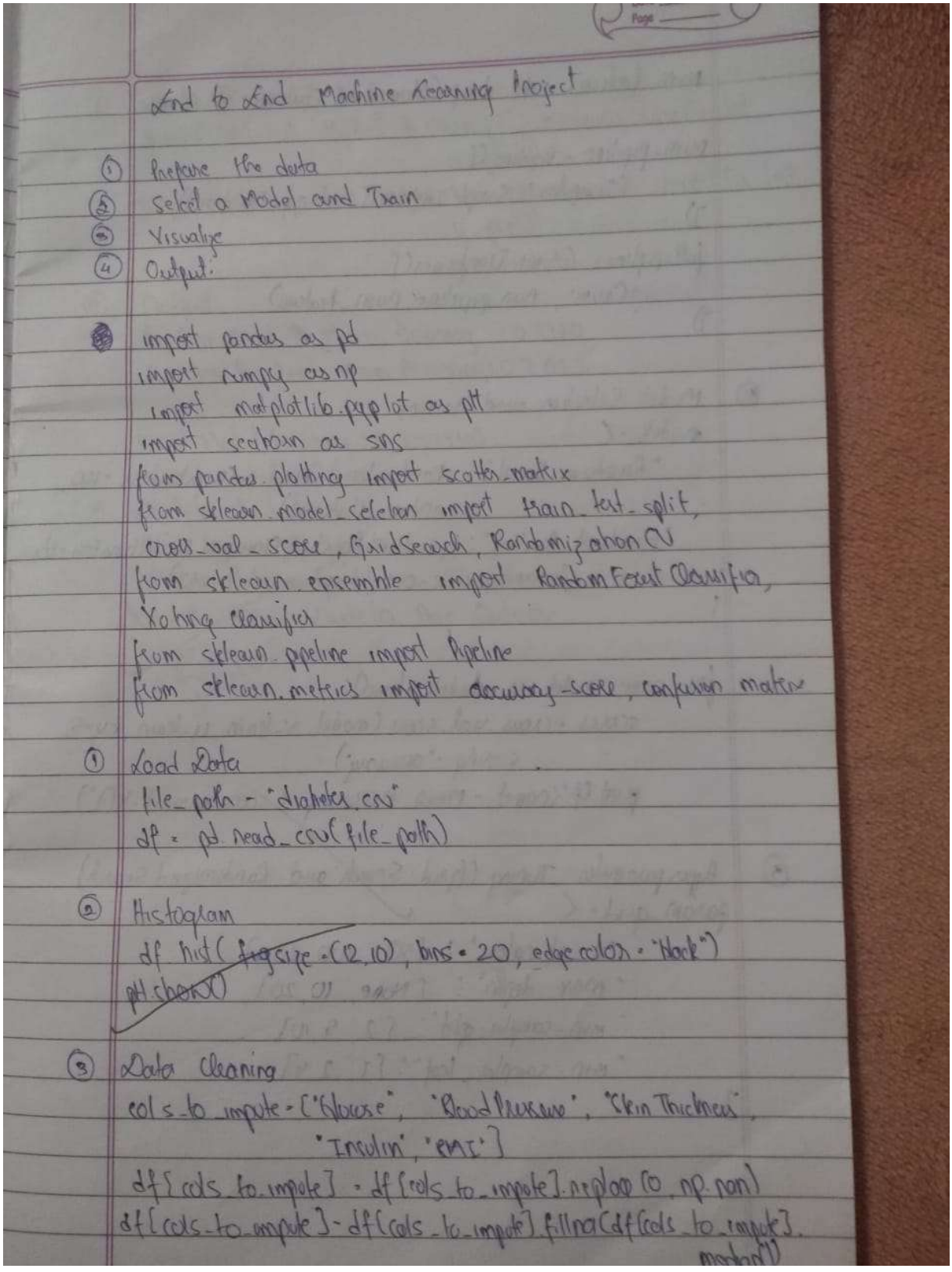
④ Mushroom Classification:
→ No of Rows: 8125
→ No of Columns: 23
→ Attributes: Class, Cap-shape, cap-color, odor
→ Inference: Can be used for decision tree based classification models. Features encoding is required before applying machine learning models.

⑤ Spam detection:
→ No of Rows: 5573
→ No of Columns: 2
→ Attributes: V1, V2
→ Inference: Could not be loaded due to an encoding issue. Likely contains text-based features for spam classification needs pre-processing before further analysis

Python Code:

```
import pandas as pd
# Import
iris_df = pd.read_csv('Iris.csv')
print("First 5 rows of the Iris dataset:")
print(iris_df.head())

# Export
output_path = "iris_exported.csv"
iris_df.to_csv(output_path, index=False)
print(f"\nIris dataset has been exported successfully to '{output_path}'")

df = pd.read_csv('iris_exported.csv')
df.head()
```

Program 2: Demonstrate various data pre-processing techniques for a given dataset



End to End Machine Learning Project

1. Prepare the data
2. Select a Model and Train
3. Visualize
4. Output:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from pandas.plotting import scatter_matrix
from sklearn.model_selection import train_test_split,
cross_val_score, GridSearch, RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier,
VotingClassifier
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score, confusion_matrix
```

1. Load Data

```
file_path = "diabetes.csv"
df = pd.read_csv(file_path)
```

2. Histogram

```
df.hist(figsize=(12,10), bins=20, edgecolor="black")
plt.show()
```

3. Data Cleaning

```
cols_to_impute = ["Glucose", "BloodPressure", "SkinThickness",
                  "Insulin", "BMI"]
df[cols_to_impute] = df[cols_to_impute].replace(0, np.nan)
df[cols_to_impute] = df[cols_to_impute].fillna(df[cols_to_impute].
                                                mean())
```

```
num_features = df.drop(columns=['Outcome']).columns

num_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='median')),
])
full_pipeline = ColumnTransformer([
    ('num', num_pipeline, num_features)
])
```

④ Model Selection and Training

```
models = {
    "Random Forest": RandomForestClassifier(n_estimators=100,
                                            random_state=42),
    "Gradient Boosting": GradientBoostingClassifier(n_estimators=100,
                                                    learning_rate=0.1, random_state=42)
}

for name, model in models.items():
    scores = cross_val_score(model, X_train, y_train, cv=5,
                             scoring='accuracy')
    print(f"{name} - Mean Accuracy: {scores.mean():.4f}")
```

⑤ Hyperparameter Tuning (Grid Search and Randomized Search)

```
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
```

⑥ Evaluate Model Performance

```
print("Final Model Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

⑦ Output:

Random Forest Mean Accuracy : 0.7720

Gradient Boosting Mean Accuracy: 0.7612

Final Model Accuracy : 0.739740

* Diabetes Prediction

770 Rows, 9 columns

Attributes: Pregnancy, Glucose, Blood Pressure, Skin Thickness, Insulin, BMI, Diabetes, Age, Outcome

12/3/25

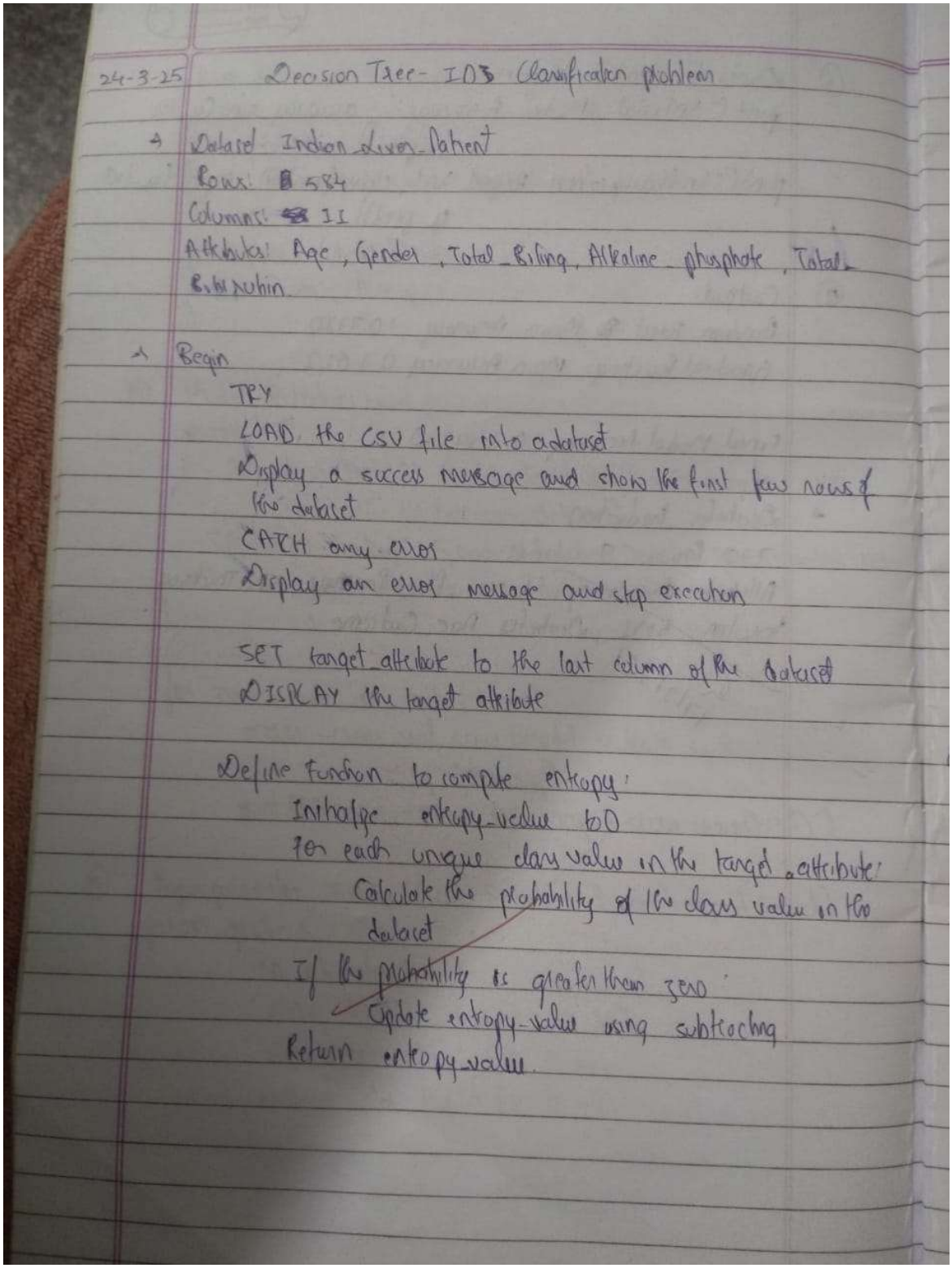
Python Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from scipy import stats

# Diabetes Dataset
df = pd.read_csv('diabetes.csv')

# Loading and Inspecting the Dataset
df.head()
df.shape
print(df.info())
print(df.describe())

# Checking for Missing Values
missing_values = df.isnull().sum()
print(missing_values[missing_values > 0])

# Identifying and Encoding Categorical Columns
categorical_cols = df.select_dtypes(include=['object']).columns
print("Categorical columns identified:", categorical_cols)

if len(categorical_cols) > 0:
    df = pd.get_dummies(df, columns=categorical_cols, drop_first=True)
    print("\nDataFrame after one-hot encoding:")
    print(df.head())
else:
    print("\nNo categorical columns found in the dataset.")

# Feature Scaling
numerical_cols = df.select_dtypes(include=['number']).columns

# a. Min-Max Scaling
scaler = MinMaxScaler()
df_minmax = df.copy()
df_minmax[numerical_cols] = scaler.fit_transform(df[numerical_cols])

# b. Standardization (Z-score Scaling)
```

```
scaler = StandardScaler()
df_standard = df.copy()
df_standard[numerical_cols] = scaler.fit_transform(df[numerical_cols])

# Printing Scaled Data
print("\nDataFrame after Min-Max Scaling:")
print(df_minmax.head())

print("\nDataFrame after Standardization:")
print(df_standard.head())

# Adult Income Dataset
df1 = pd.read_csv('adult.csv')

# Loading and Inspecting the Dataset
df1.head()
df1.shape
print(df1.info())
print(df1.describe())

# Missing Value Detection
missing_values = df1.isnull().sum()
print(missing_values[missing_values > 0])

# Detecting and Encoding Categorical Columns
categorical_cols = df1.select_dtypes(include=['object']).columns
print("Categorical columns identified:", categorical_cols)

if len(categorical_cols) > 0:
    df1 = pd.get_dummies(df1, columns=categorical_cols, drop_first=True)
    print("\nDataFrame after one-hot encoding:")
    print(df.head())
else:
    print("\nNo categorical columns found in the dataset.")

# Scaling Numerical Features
numerical_cols = df1.select_dtypes(include=['number']).columns

# Feature Scaling
numerical_cols = df.select_dtypes(include=['number']).columns

# a. Min-Max Scaling
scaler = MinMaxScaler()
df_minmax = df.copy()
```

```
df_minmax[numerical_cols] = scaler.fit_transform(df[numerical_cols])

# b. Standardization (Z-score Scaling)
scaler = StandardScaler()
df_standard = df.copy()
df_standard[numerical_cols] = scaler.fit_transform(df[numerical_cols])

# Printing Scaled Data
print("\nDataFrame after Min-Max Scaling:")
print(df_minmax.head())

print("\nDataFrame after Standardization:")
print(df_standard.head())
```

Program 3: Use an appropriate data set for building the decision tree (ID3) and apply this knowledge to classify a new sample.



Define function to compute information gain for a given attribute:
Calculate total entropy using a entire dataset
Initialize weighted_entropy to 0
For each unique value in the attribute:
select the subset of the dataset when the attribute:
Select the subset of the dataset where the attribute equals that value
ADD (weight * subset entropy) to weighted_entropy
Return (total_entropy - weighted_entropy)

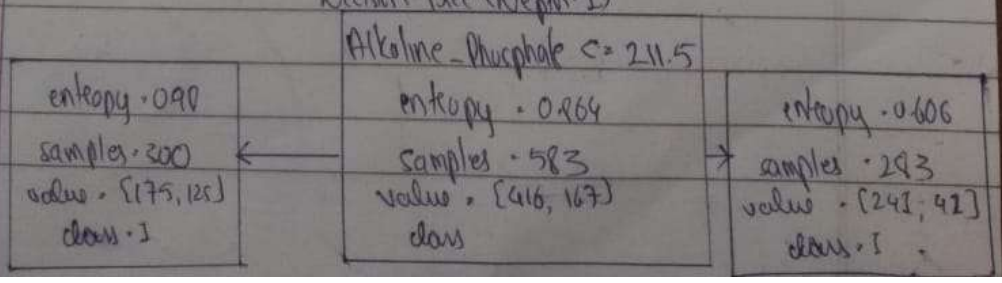
root_node to the attribute with the maximum information gain
Display the selected root node

selected_feature to a specific feature
IF the selected feature is not present in the dataset THEN
SET selected_feature to the previously determined root_node
END

Output: Target attribute for classification : Dataset
Calculating Information gain for each attribute:
Information Gain for Age : 0.18411
Information Gain for Gender : 0.048
Information Gain for Total_Bilirubin : 0.1529
Information Gain for Direct_Bilirubin : 0.1301
Information Gain for Alkaline_phosphate : 0.4071
Information Gain for Albumin : 0.0767

→ Root Node selected based on highest information gain:
Alkaline_phosphate

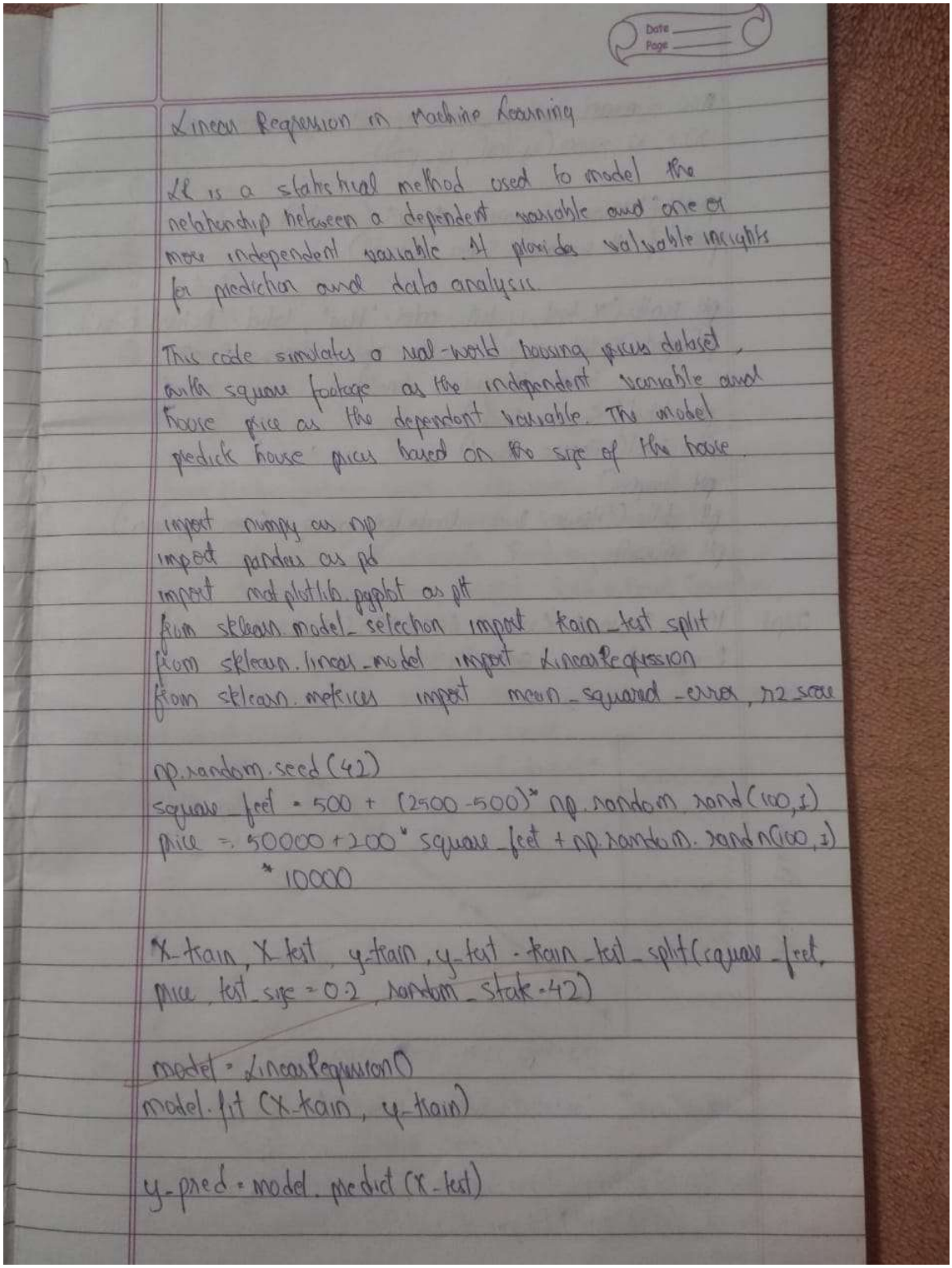
Decision Tree (Depth=1)



Python Code:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree
import numpy as np

iris = pd.read_csv('Iris.csv')

X = iris.iloc[:, 1:-1]
y = iris.iloc[:, -1]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

clf_iris = DecisionTreeClassifier(criterion='entropy', random_state=42)
clf_iris.fit(X_train, y_train)
y_pred_iris = clf_iris.predict(X_test)
accuracy_iris = accuracy_score(y_test, y_pred_iris)
conf_matrix_iris = confusion_matrix(y_test, y_pred_iris)

plt.figure(figsize=(12, 8))
plot_tree(clf_iris, filled=True, feature_names=X.columns, class_names=clf_iris.classes_)
plt.title("Decision Tree for IRIS Dataset")
plt.show()

# New Class
new_sample = pd.DataFrame([[5.1, 3.5, 1.4, 0.2]], columns=X.columns)
predicted_class = clf_iris.predict(new_sample)

print("Predicted class for the new sample:", predicted_class[0])
```

Program 4: Implement Linear and Multi-Linear Regression algorithm using appropriate dataset



Linear Regression in Machine Learning

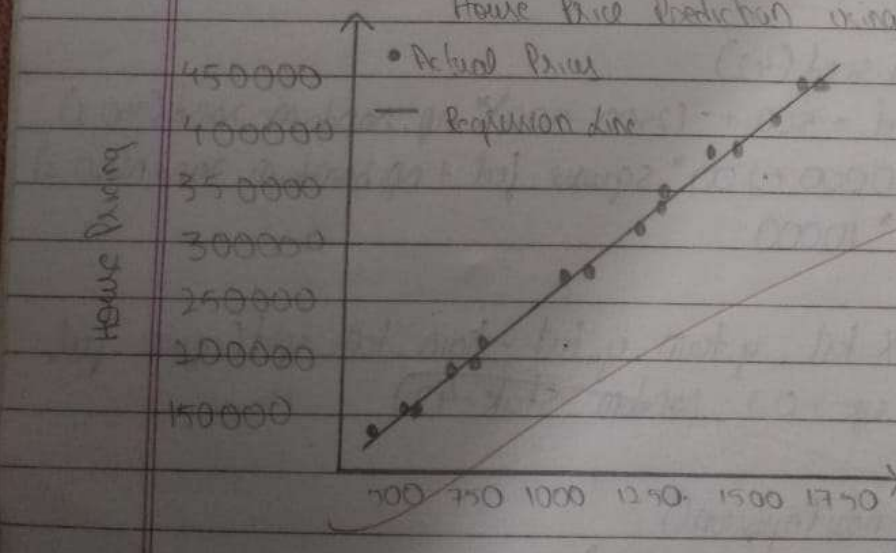
It is a statistical method used to model the relationship between a dependent variable and one or more independent variable. It provides valuable insights for prediction and data analysis.

This code simulates a real-world housing prices dataset, with square footage as the independent variable and house price as the dependent variable. The model predicts house prices based on the size of the house.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

np.random.seed(42)
square_feet = 500 + (2500-500)* np.random.rand(100,1)
price = 50000 + 200* square_feet + np.random.randn(100,1)
        * 10000

X_train, X_test, y_train, y_test = train_test_split(square_feet,
price, test_size=0.2, random_state=42)

model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
```

```
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error: {mse}")
print(f"R-squared Score: {r2}")

plt.scatter(X_test, y_test, color='blue', label='Actual Prices')
plt.plot(X_test, y_pred, color='red', linewidth=2,
         label='Regression Line')
plt.xlabel('Square Feet')
plt.ylabel('House Price')
plt.legend()
plt.title('House Price Prediction using Linear Regression')
plt.show()
```

Output: Mean Squared Error: 65369951.3716
R-squared Score: 0.9957338020



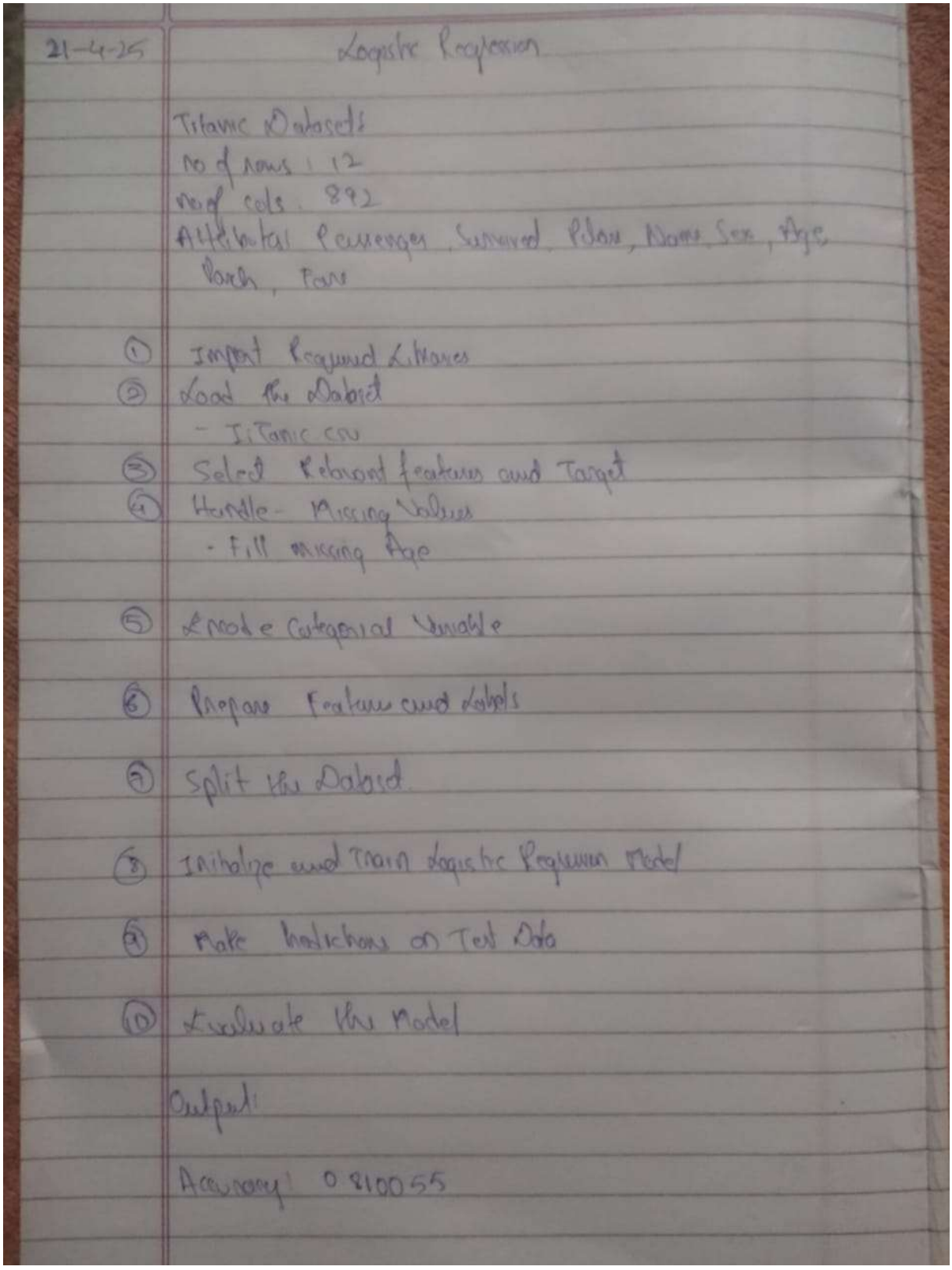
10/3/25

Python Code:

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

# --- Linear Regression (Single Feature) ---
df1 = pd.read_csv('Housing.csv')  # columns: area, price

X1 = df1[['area']]   # feature
y1 = df1['price']    # target

model1 = LinearRegression()
model1.fit(X1, y1)

# Predict for area = 3300 and 5000
pred_3300 = model1.predict(pd.DataFrame({'area':[3300]}))[0]
pred_5000 = model1.predict(pd.DataFrame({'area':[5000]}))[0]

# Plot for Linear Regression
plt.figure(figsize=(8, 6))
plt.scatter(X1, y1, color='blue', marker='o', label='Data Points')
plt.plot(X1, model1.predict(X1), color='red', label='Best Fit Line')
plt.xlabel('Area (sq ft)')
plt.ylabel('Price')
plt.title('Linear Regression (Area vs Price)')
plt.legend()
plt.grid(True)
plt.savefig('linear_regression_plot.png')  # Save plot as image
plt.show()


# --- Multiple Linear Regression ---
df2 = pd.read_csv('Housing.csv')  # columns: area, bedrooms, age, price

# Fill missing values in bedrooms if any
df2['bedrooms'] = df2['bedrooms'].fillna(df2['bedrooms'].median())

X2 = df2[['area', 'bedrooms']]  # features
y2 = df2['price']

model2 = LinearRegression()
```

```
model2.fit(X2, y2)

# Predict for multiple features
multi_pred = model2.predict(pd.DataFrame({'area':[3000], 'bedrooms':[3]}))[0]

# Plot for Multiple Linear Regression:
# Here we visualize predicted price for different bedroom counts with fixed area

bedroom_vals = np.arange(int(df2.bedrooms.min()), int(df2.bedrooms.max())+1)
pred_prices = model2.predict(pd.DataFrame({'area':[3000]*len(bedroom_vals),
'bedrooms':bedroom_vals}))

plt.figure(figsize=(8, 6))
plt.plot(bedroom_vals, pred_prices, marker='o', linestyle='-', color='green')
plt.xlabel('Number of Bedrooms')
plt.ylabel('Predicted Price')
plt.title('Multiple Linear Regression\n(Predicted Price vs Bedrooms for area=3000 sq ft)')
plt.grid(True)
plt.savefig('multiple_linear_regression_plot.png')  # Save plot as image
plt.show()

# Print prediction results for clarity
print(f"Linear Regression Predictions:")
print(f"Price for 3300 sq ft: ₹{pred_3300:,.0f}")
print(f"Price for 5000 sq ft: ₹{pred_5000:,.0f}\n")

print(f"Multiple Linear Regression Prediction:")
print(f"Price for 3000 sq ft and 3 bedrooms: ₹{multi_pred:,.0f}")
```

Program 5: Build Logistic Regression Model for a given dataset



21-4-25

Logistic Regression

Titanic Dataset:
no of rows : 12
no of cols : 892
Attributes: Passenger, Survived, Pclass, Name, Sex, Age, Parch, Fare

1. Import Required Libraries
2. Load the Dataset
   - Titanic csv
3. Select Relevant features and Target
4. Handle - Missing Values
   - Fill missing Age
5. Encode Categorical Variable
6. Prepare Features and Labels
7. Split the Dataset
8. Initialize and Train Logistic Regression Model
9. Make Predictions on Test Data
10. Evaluate the Model

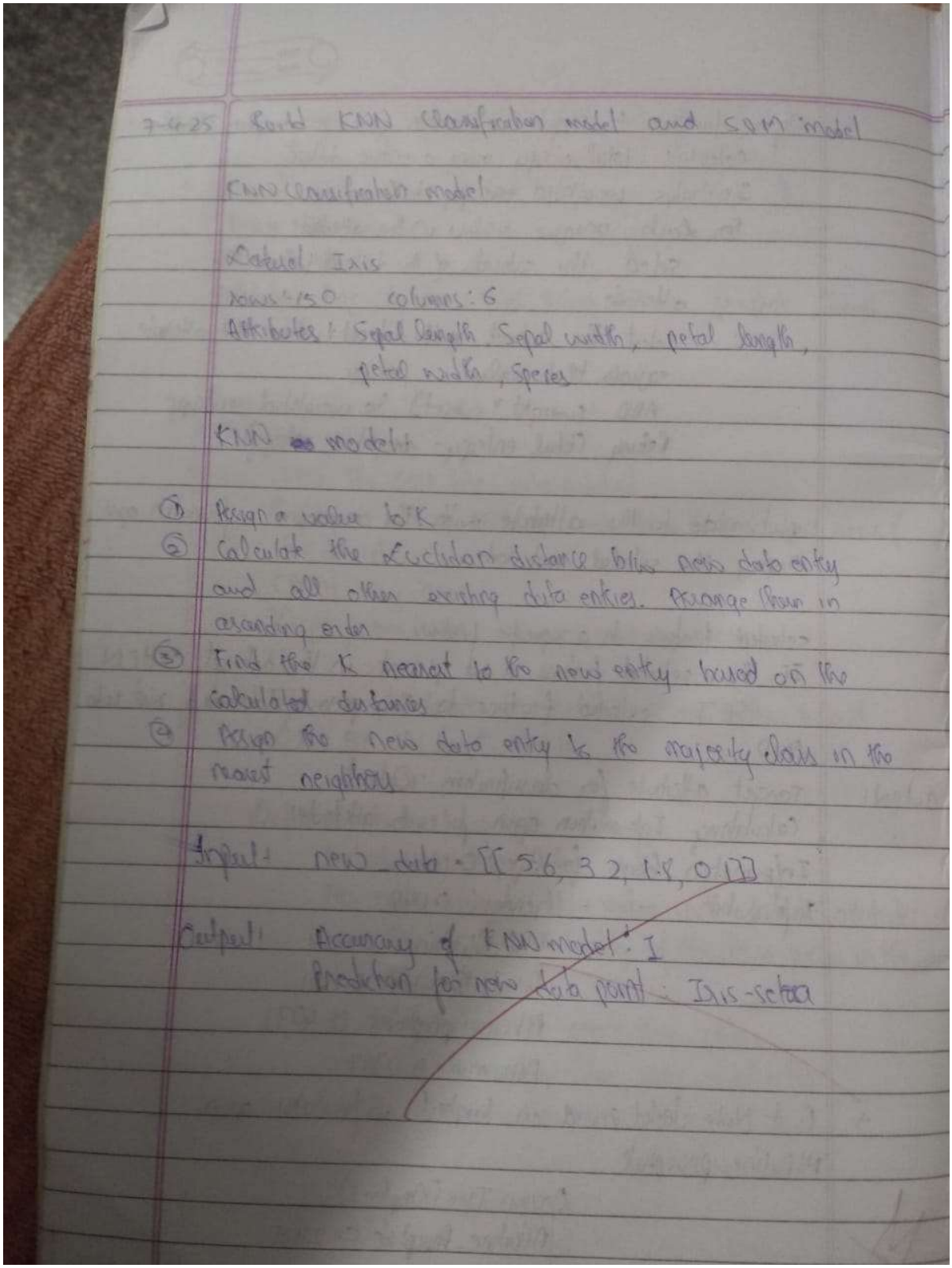
Output:

Accuracy: 0.810055

Python Code:

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Load Titanic dataset
df = pd.read_csv('Titanic.csv')

# Select features & target
# Common useful features: Pclass, Sex, Age, SibSp, Parch, Fare, Embarked
# Target: Survived

# Handle missing data
df['Age'] = df['Age'].fillna(df['Age'].median())
df['Embarked'] = df['Embarked'].fillna(df['Embarked'].mode()[0])

# Convert categorical variables to numeric
df['Sex'] = df['Sex'].map({'male': 0, 'female': 1})
df = pd.get_dummies(df, columns=['Embarked'], drop_first=True)

# Features and target
X = df[['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked_Q', 'Embarked_S']]
y = df['Survived']

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Build Logistic Regression model
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)

# Predict on test data
y_pred = model.predict(X_test)

# Evaluation
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

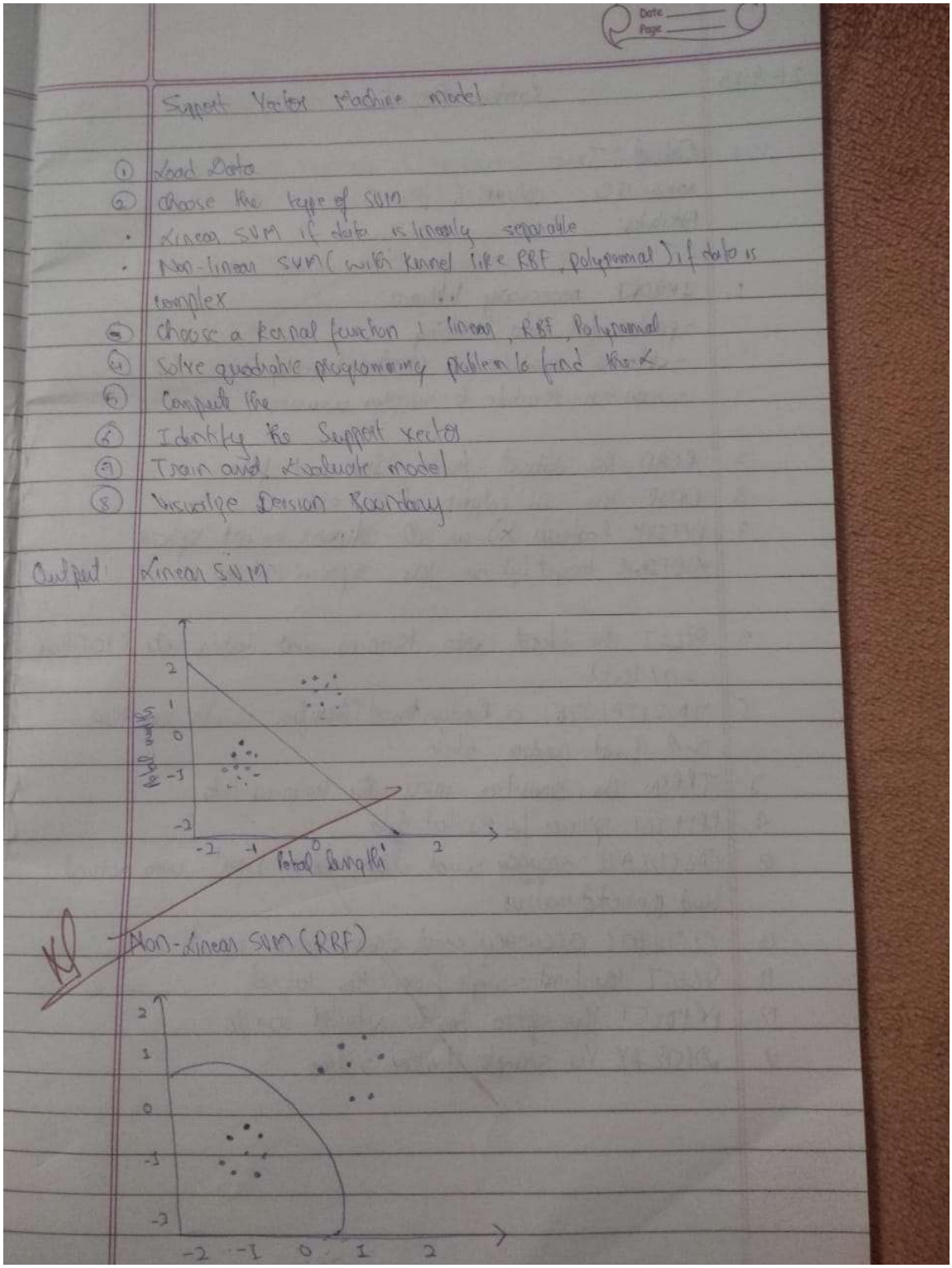
Program 6: Build KNN Classification model for a given dataset.



Python Code:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import seaborn as sns
import matplotlib.pyplot as plt

# Load iris dataset
iris_df = pd.read_csv("Iris.csv")

# Features and target
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Split data
X_train, X_test, y_train, y_test = train_test_split(X_iris, y_iris, test_size=0.2, random_state=42)

# Train KNN
knn_iris = KNeighborsClassifier(n_neighbors=3)
knn_iris.fit(X_train, y_train)

# Predict
y_pred = knn_iris.predict(X_test)

# Evaluate
print("Iris Dataset - Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))

sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, cmap="Blues", fmt='g')
plt.title("Confusion Matrix - Iris KNN")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

Program 7: Build Support vector machine model for a given dataset

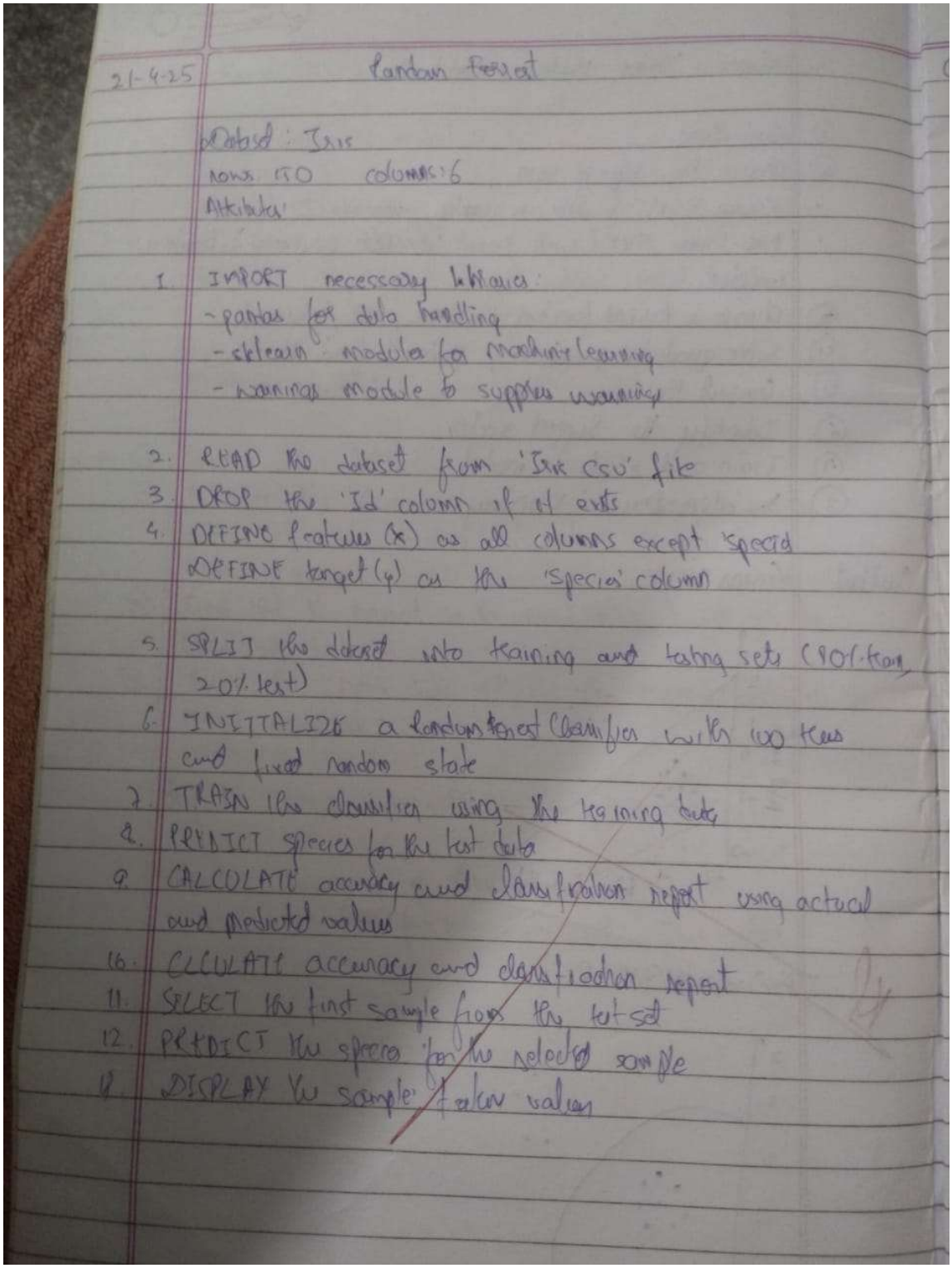


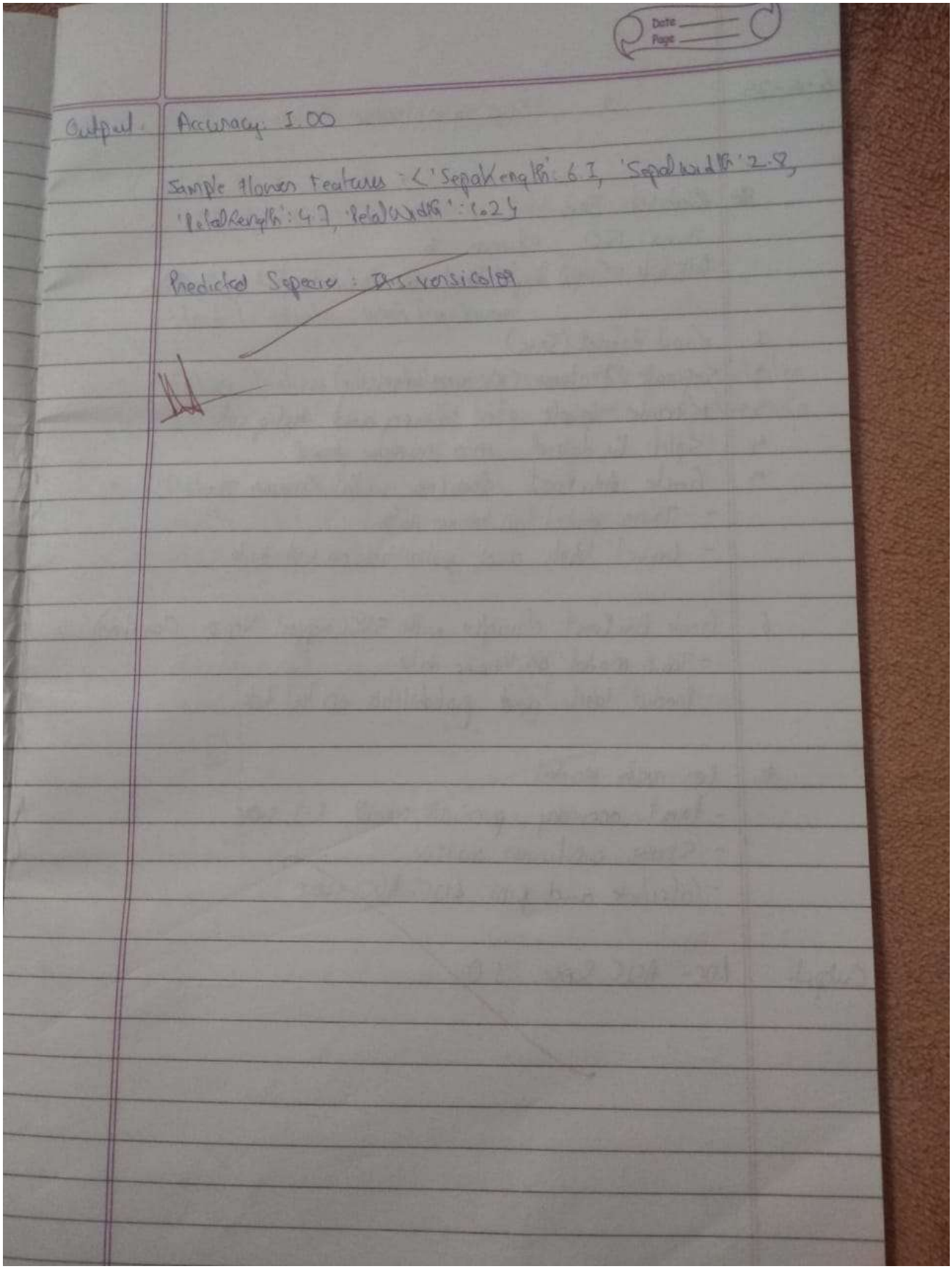
Python Code:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Load the IRIS dataset
iris_df = pd.read_csv("Iris.csv")

# Split into features and target
X = iris_df.drop("Species", axis=1)
y = iris_df["Species"]

# Split into training and testing (80/20)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# --- SVM with Linear Kernel ---
svm_linear = SVC(kernel='linear')
svm_linear.fit(X_train, y_train)
y_pred_linear = svm_linear.predict(X_test)

# Accuracy and Confusion Matrix
acc_linear = accuracy_score(y_test, y_pred_linear)
cm_linear = confusion_matrix(y_test, y_pred_linear)

print("Linear Kernel:")
print("Accuracy:", acc_linear)
print("Confusion Matrix:\n", cm_linear)

# Plot Confusion Matrix for Linear Kernel
plt.figure(figsize=(6, 4))
sns.heatmap(cm_linear, annot=True, fmt="d", cmap="Blues",
        xticklabels=svm_linear.classes_, yticklabels=svm_linear.classes_)
plt.title("Confusion Matrix - Linear Kernel")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.tight_layout()
plt.show()

# --- SVM with RBF Kernel ---
svm_rbf = SVC(kernel='rbf')
```

```
svm_rbf.fit(X_train, y_train)
y_pred_rbf = svm_rbf.predict(X_test)

# Accuracy and Confusion Matrix
acc_rbf = accuracy_score(y_test, y_pred_rbf)
cm_rbf = confusion_matrix(y_test, y_pred_rbf)

print("\nRBF Kernel:")
print("Accuracy:", acc_rbf)
print("Confusion Matrix:\n", cm_rbf)

# Plot Confusion Matrix for RBF Kernel
plt.figure(figsize=(6, 4))
sns.heatmap(cm_rbf, annot=True, fmt="d", cmap="Greens",
        xticklabels=svm_rbf.classes_, yticklabels=svm_rbf.classes_)
plt.title("Confusion Matrix - RBF Kernel")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.tight_layout()
plt.show()
```

Program 8: Implement Random forest ensemble method on a given dataset



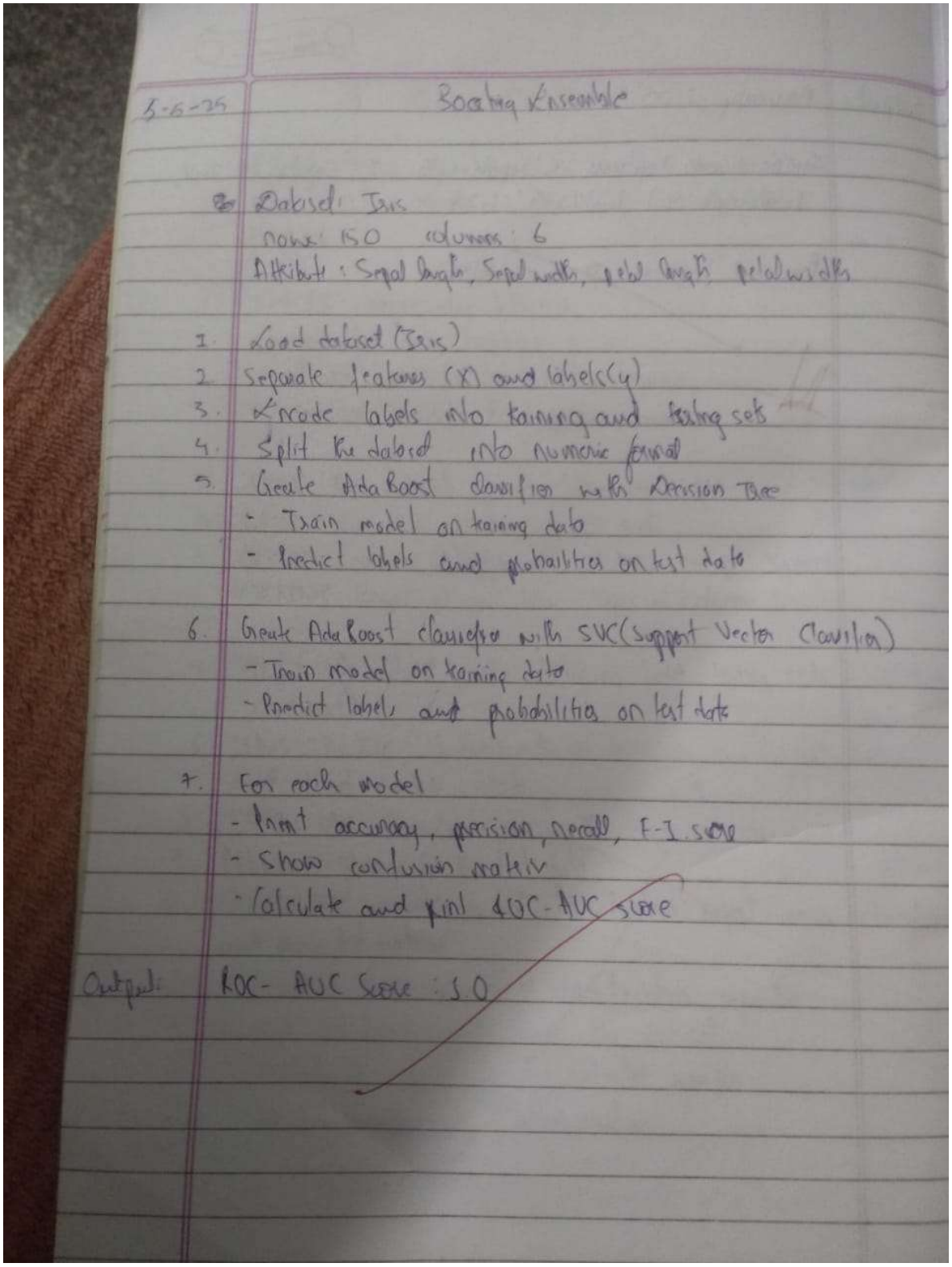

Output.
Accuracy: 1.00
Sample flower features : {'SepalLength': 6.7, 'SepalWidth': 2.8,
'PetalLength': 4.7, 'PetalWidth': 1.2}
Predicted Species : Iris-versicolor

Python Code:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
df = pd.read_csv("Iris.csv")  # Adjust filename if needed

# Prepare data
X = df.drop(columns=["Id", "Species"])  # Drop non-informative columns
y = df["Species"]

# Split dataset with stratified sampling
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# Default Random Forest with 10 trees
rf_default = RandomForestClassifier(n_estimators=10, random_state=42)
rf_default.fit(X_train, y_train)
y_pred_default = rf_default.predict(X_test)
acc_default = accuracy_score(y_test, y_pred_default)
conf_matrix_default = confusion_matrix(y_test, y_pred_default)

print(f"Default RF (10 trees) Accuracy: {acc_default:.4f}")
print("Confusion Matrix:\n", conf_matrix_default)
print("\nClassification Report for Default Model:")
print(classification_report(y_test, y_pred_default))

# Try different numbers of trees to find the best
best_acc = 0
best_n = 10
acc_list = []

for n in range(1, 101):
    rf = RandomForestClassifier(n_estimators=n, random_state=42)
    rf.fit(X_train, y_train)
    y_pred = rf.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    acc_list.append((n, acc))
```

```
    if acc > best_acc:
        best_acc = acc
        best_n = n
        best_conf_matrix = confusion_matrix(y_test, y_pred)
        best_model = rf  # Save the best model

print(f"\nBest Accuracy: {best_acc:.4f} using {best_n} trees")
print("Best Confusion Matrix:\n", best_conf_matrix)

# Plot accuracy vs number of trees
x_vals, y_vals = zip(*acc_list)
plt.plot(x_vals, y_vals, marker='o')
plt.title("Accuracy vs Number of Trees")
plt.xlabel("Number of Trees")
plt.ylabel("Accuracy")
plt.ylim(0.8, 1.05)  # Optional axis limit
plt.grid(True)
plt.axvline(best_n, color='r', linestyle='--', label=f'Best: {best_n} trees')
plt.legend()
plt.show()

# Evaluate best model
y_pred_best = best_model.predict(X_test)
print("\nClassification Report for Best Model:")
print(classification_report(y_test, y_pred_best))

# Plot feature importances
importances = best_model.feature_importances_
features = X.columns
sns.barplot(x=importances, y=features)
plt.title("Feature Importances from Best Random Forest")
plt.xlabel("Importance")
plt.ylabel("Feature")
plt.tight_layout()
plt.show()
```

Program 9: Implement Boosting ensemble method on a given dataset.



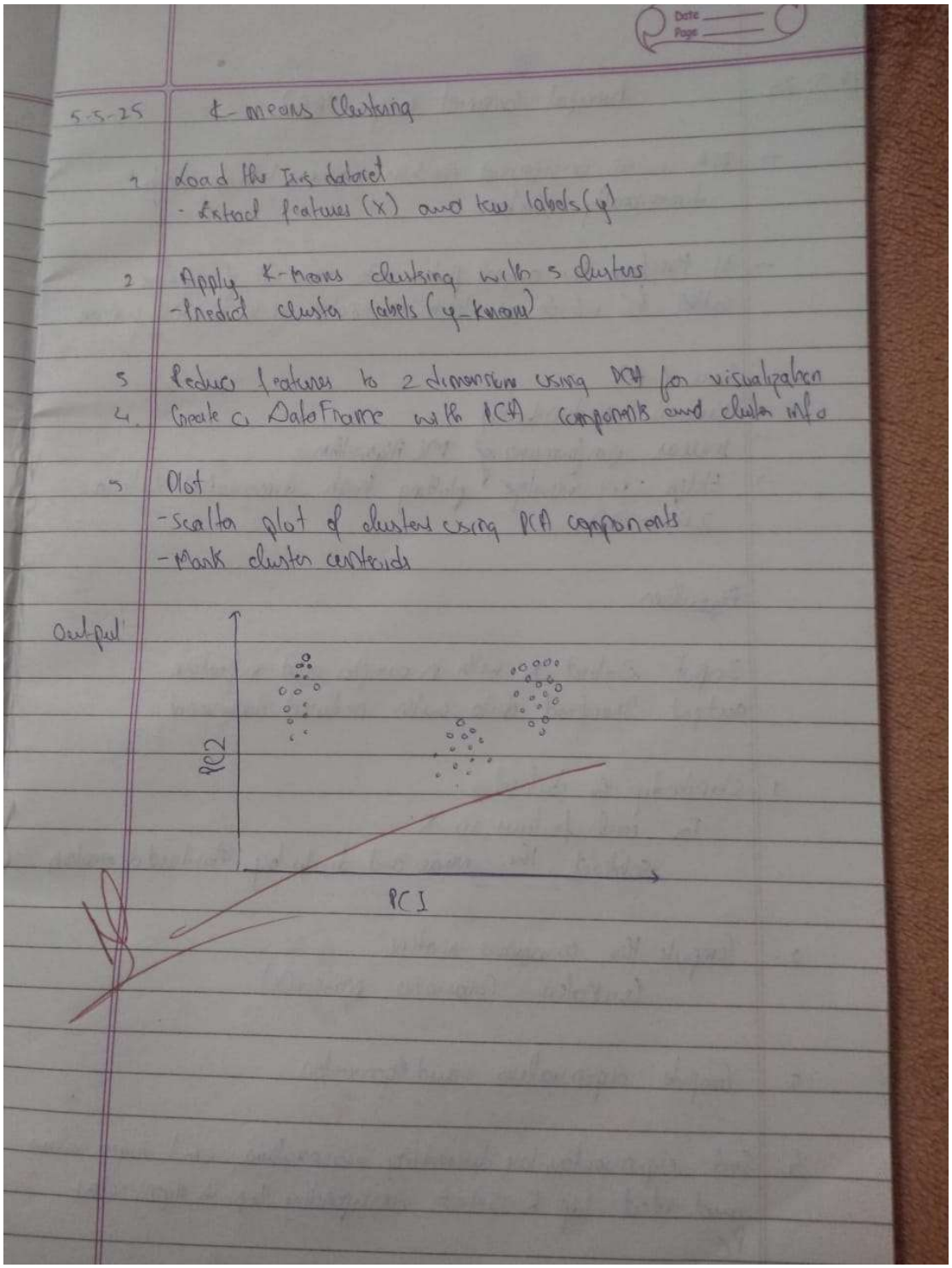
Python Code:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report

# Load dataset
df = pd.read_csv("adult.csv")

# Drop rows with missing values
df.dropna(inplace=True)

# Encode categorical columns
label_encoders = {}
for column in df.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    df[column] = le.fit_transform(df[column])
    label_encoders[column] = le

# Separate features and target
X = df.drop(columns=['income'], errors='ignore', axis=1)
y = df['income']

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Define base estimator
base_est = DecisionTreeClassifier(max_depth=3)

# AdaBoost with 10 estimators using SAMME algorithm
model_10 = AdaBoostClassifier(estimator=base_est, n_estimators=10, random_state=42,
algorithm='SAMME')
model_10.fit(X_train, y_train)
y_pred_10 = model_10.predict(X_test)
score_10 = accuracy_score(y_test, y_pred_10)
print(f"Accuracy with 10 estimators: {score_10:.4f}")
print("Classification Report (10 Estimators):\n", classification_report(y_test, y_pred_10))
```

```
# Initialize variables for fine-tuning
estimators_range = list(range(10, 201, 10))
scores = []
best_score = 0
best_n = 0

# Fine-tune number of estimators
for n in estimators_range:
    model = AdaBoostClassifier(estimator=base_est, n_estimators=n, random_state=42,
algorithm='SAMME')
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    score = accuracy_score(y_test, y_pred)
    scores.append(score)
    print(f"n_estimators={n}, Accuracy={score:.4f}")
    if score > best_score:
        best_score = score
        best_n = n
        best_model = model  # Save the best model
        best_y_pred = y_pred

print(f"\nBest Accuracy: {best_score:.4f} using {best_n} estimators")
print("Classification Report (Best Estimator):\n", classification_report(y_test, best_y_pred))

# Plot accuracy vs number of estimators
plt.figure(figsize=(7, 4))
plt.plot(estimators_range, scores, marker='o', linestyle='-', color='blue')
plt.title("Accuracy vs Number of Estimators (AdaBoost)")
plt.xlabel("Number of Estimators (Trees)")
plt.ylabel("Accuracy")
plt.grid(True)
plt.xticks(estimators_range)
plt.tight_layout()
plt.show()

# Visualize feature importances for best model
importances = best_model.feature_importances_
features = X.columns
plt.figure(figsize=(8, 5))
sns.barplot(x=importances, y=features)
plt.title("Feature Importances (AdaBoost)")
plt.xlabel("Importance")
plt.ylabel("Feature")
```

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

Program 10: Build k-Means algorithm to cluster a set of data stored in a .CSV file.



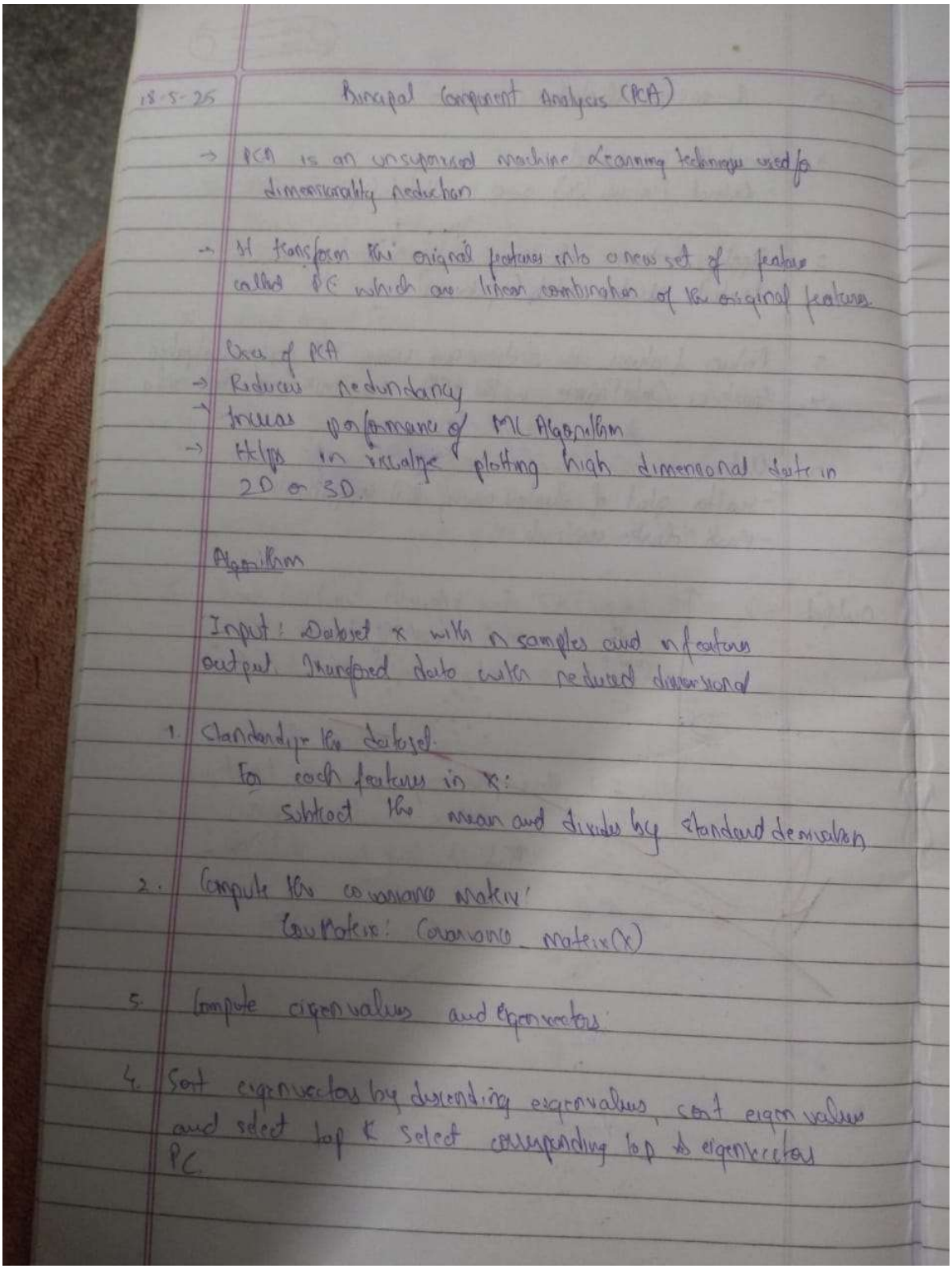
Python Code:

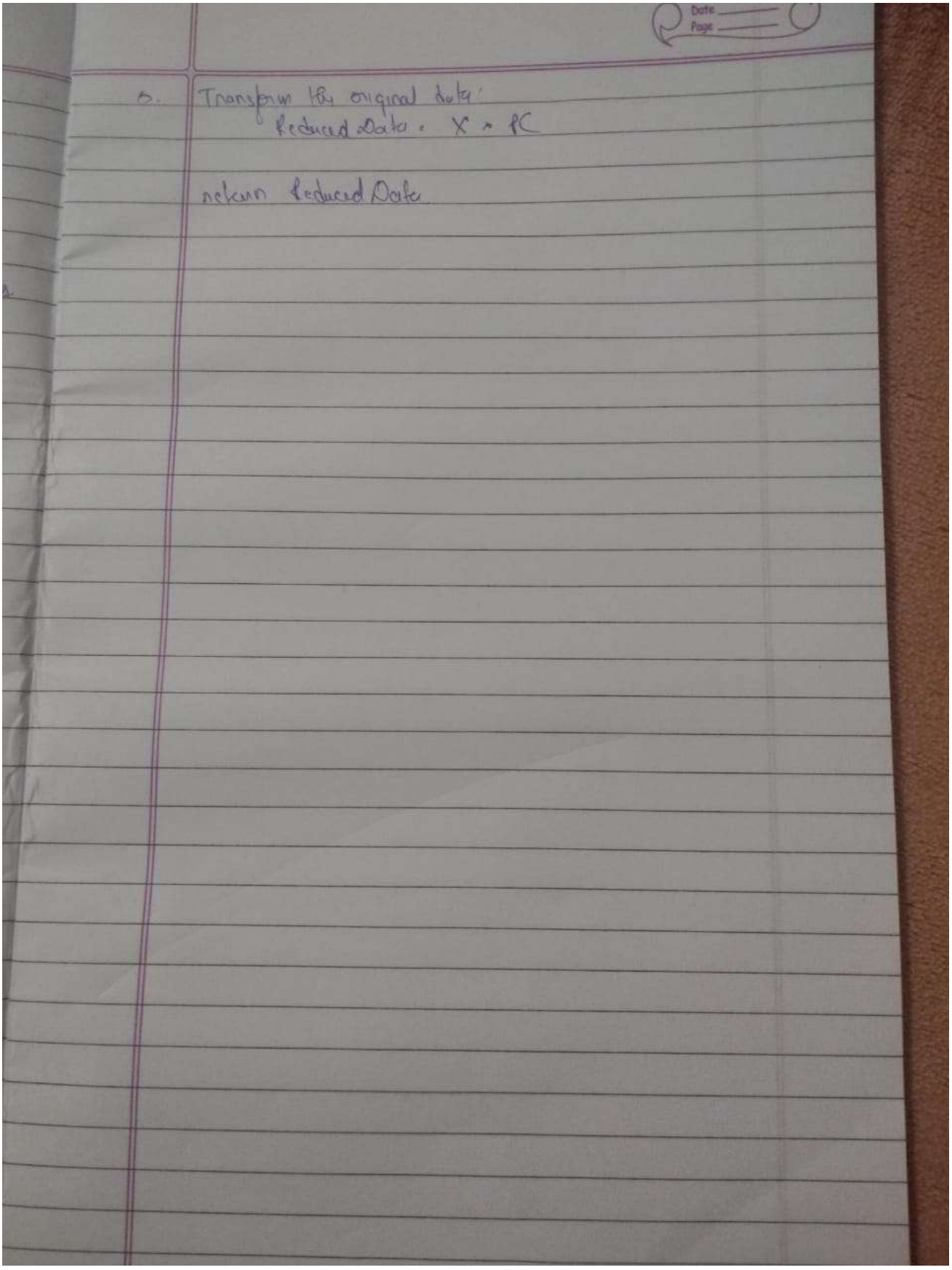
```
import os
os.environ["OMP_NUM_THREADS"] = "1"  # Avoid Windows MKL memory leak warnings

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

def load_data(csv_path='Iris.csv'):
    try:
        df = pd.read_csv(csv_path)
        df.columns = [c.strip().replace(' ', '_').replace('(', '').replace(')', '').replace('.', '') for c in
df.columns]
        if 'species' not in df.columns and 'Species' not in df.columns:
            # Try to find a species column case-insensitive
            species_cols = [c for c in df.columns if 'species' in c.lower()]
            if species_cols:
                df['Species'] = df[species_cols[0]]
            else:
                raise ValueError("Species column not found in CSV")
        if 'Species' not in df.columns and 'species' in df.columns:
            df['Species'] = df['species']
    except Exception:
        # fallback to sklearn iris dataset
        iris = load_iris()
        df = pd.DataFrame(
            data=np.c_[iris['data'], iris['target']],
            columns=iris['feature_names'] + ['target']
        )
        df.columns = [c.strip().replace(' (cm)', '').replace(' ', '_') for c in df.columns]
        df['Species'] = [iris['target_names'][int(t)] for t in df['target']]
    return df

def preprocess(df):
    # Use PetalLengthCm and PetalWidthCm if present, else fall back to sklearn column names
    if 'PetalLengthCm' in df.columns and 'PetalWidthCm' in df.columns:
        X = df[['PetalLengthCm', 'PetalWidthCm']].values
    else:
```

```
        # Fall back to iris feature names without (cm)
        col_pl = next((c for c in df.columns if 'petal_length' in c.lower()), None)
        col_pw = next((c for c in df.columns if 'petal_width' in c.lower()), None)
        if col_pl and col_pw:
            X = df[[col_pl, col_pw]].values
        else:
            raise ValueError("Cannot find petal length and width columns.")
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)
    return X_scaled, scaler

def plot_elbow(X_scaled, max_k=10):
    inertias = []
    ks = range(1, max_k + 1)
    for k in ks:
        km = KMeans(n_clusters=k, random_state=42)
        km.fit(X_scaled)
        inertias.append(km.inertia_)
    plt.figure(figsize=(6, 4))
    plt.plot(ks, inertias, 'o-', linewidth=2)
    plt.xlabel('Number of clusters (k)')
    plt.ylabel('Inertia')
    plt.title('Elbow Method for Optimal k')
    plt.xticks(ks)
    plt.grid(True, linestyle='--', alpha=0.5)
    plt.tight_layout()
    plt.show()
    return inertias

def run_kmeans(X_scaled, k):
    km = KMeans(n_clusters=k, random_state=42)
    labels = km.fit_predict(X_scaled)
    return km, labels

def plot_confusion(df, labels, k):
    species_names = df['Species'].unique()
    species_to_num = {name: idx for idx, name in enumerate(species_names)}
    true_nums = df['Species'].map(species_to_num)
    cm = confusion_matrix(true_nums, labels)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                                  display_labels=[f"Cluster {i}" for i in range(k)])
    fig, ax = plt.subplots(figsize=(6, 6))
    disp.plot(ax=ax, cmap='Blues', colorbar=True)
    ax.set_xlabel('Predicted Cluster')
```

```
    ax.set_ylabel('True Species')
    plt.title('K-Means Clustering Confusion Matrix')
    plt.tight_layout()
    plt.show()
    cm_df = pd.DataFrame(cm,
                         index=[f"True: {name}" for name in species_names],
                         columns=[f"Cluster {i}" for i in range(k)])
    print("\nConfusion Matrix (counts):")
    print(cm_df)

def main():
    df = load_data('Iris.csv')
    if 'Species' not in df.columns:
        print("Error: 'Species' column not found in the data.")
        return

    X_scaled, scaler = preprocess(df)

    print("Generating elbow plot to find optimal k...")
    plot_elbow(X_scaled, max_k=10)

    optimal_k = 3
    print(f"Choosing k = {optimal_k} based on elbow plot.")

    km_model, labels = run_kmeans(X_scaled, optimal_k)
    df['cluster'] = labels

    plt.figure(figsize=(6, 4))
    plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=labels, cmap='viridis', edgecolor='k', s=50)
    centroids = km_model.cluster_centers_
    plt.scatter(centroids[:, 0], centroids[:, 1], marker='X', c='red', s=200, label='Centroids')
    plt.xlabel('Scaled Petal Length')
    plt.ylabel('Scaled Petal Width')
    plt.title(f'K-Means Clusters (k={optimal_k})')
    plt.legend()
    plt.grid(True, linestyle='--', alpha=0.5)
    plt.tight_layout()
    plt.show()

    plot_confusion(df, labels, optimal_k)

if __name__ == "__main__":
    main()
```

Program 11: Implement Dimensionality reduction using Principal Component Analysis (PCA) method.




5.
Transform the original data:
Reduced Data = X * PC
return Reduced Data

Python Code:

```
import numpy as np
from sklearn.decomposition import PCA
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt

# 1. Load sample data (Iris dataset)
data = load_iris()
X = data.data  # features
y = data.target  # labels (optional, for visualization)

# 2. Initialize PCA and reduce to 2 components
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

# 3. Show explained variance ratio by each component
print("Explained variance ratio:", pca.explained_variance_ratio_)

# 4. Plot the 2D transformed data
plt.figure(figsize=(8,6))
for target in np.unique(y):
    plt.scatter(
        X_pca[y == target, 0],
        X_pca[y == target, 1],
        label=data.target_names[target]
    )
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.title("PCA - Iris dataset")
plt.legend()
plt.show()
```