MACHINE LEARNING

A PRACTICAL REPORT ON

MACHINE LEARNING

Submitted By Mr.Abusufiyan Kazi

Seat No :

UNDER THE GUIDANCE OF

Prof. Sufiya Ahmed

Submitted in fulfillment of the requirements for qualifying
M.Sc.I.T. Part II Semester – III Examination 2025-2026

University of Mumbai
Department of Information Technology

R.D. & S.H. National College of Arts, Commerce & S.W.A.
Science College Bandra (West), Mumbai – 400 050

R. D. & S. H. National & S. W. A. Science College

Department of Information Technology M.Sc.IT (Semester – III)

Certificate

This is to certify that ***on Machine Learning Practical's*** *performed at R.D. & S.H. National & S.W.A. Science College by* ***Mr.Abusufiyan KAzi*** *holding Seat No. studying Master of Science in Information Technology Semester – III has been satisfactorily completed as prescribed by the University of Mumbai, during the year 2025 – 2026.*

Subject In-Charge Coordinator In-Charge External Examiner

College Stamp

INDEX

Practical No: - 1

Aim: Data Pre-processing and Exploration.

Writeups:

1A) Aim: Load a CSV dataset. Handle missing values, inconsistent formatting, and outliers.

Code:

```
import pandas as pd
import numpy as np
import requests
from io import StringIO
from scipy import stats
url = 'https://archive.ics.uci.edu/ml/machine-learning-
databases/iris/iris.data'
columns = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width',
'species']
response = requests.get(url)
csv_data = StringIO(response.text)
df = pd.read_csv(csv_data, header=None, names=columns)
print("Before Data Cleaning:")
print(df.info())
print(df.head())
df_imputed = df.copy()
numeric_cols = df_imputed.select_dtypes(include=np.number).columns
df_imputed[numeric_cols] =
df_imputed[numeric_cols].fillna(df_imputed[numeric_cols].mean())
categorical_cols = df_imputed.select_dtypes(include='object').columns
for col in categorical_cols:
    df_imputed[col].fillna(df_imputed[col].mode()[0], inplace=True)
df_imputed.columns = df_imputed.columns.str.strip().str.lower().str.replace('
', '_')
z_scores = np.abs(stats.zscore(df_imputed.select_dtypes(include=np.number)))
outliers = (z_scores > 3)
df_cleaned = df_imputed[~np.any(outliers, axis=1)]
print("\nAfter Data Cleaning:")
print(df_cleaned.info())
print(df_cleaned.head())
print("\nSummary Statistics Before Cleaning:")
print(df.describe())
print("\nSummary Statistics After Cleaning:")
print(df_cleaned.describe())
```

Output:

```
PS C:\Users\Shivang Singh> python -u "c:\Users\Shivang Singh\prac1a.py"
Before Data Cleaning:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   species       150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
None
   sepal_length  sepal_width  petal_length  petal_width      species
0           5.1          3.5           1.4          0.2  Iris-setosa
1           4.9          3.0           1.4          0.2  Iris-setosa
2           4.7          3.2           1.3          0.2  Iris-setosa
3           4.6          3.1           1.5          0.2  Iris-setosa
4           5.0          3.6           1.4          0.2  Iris-setosa
```

```
After Data Cleaning:
<class 'pandas.core.frame.DataFrame'>
Index: 149 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   sepal_length  149 non-null    float64
 1   sepal_width   149 non-null    float64
 2   petal_length  149 non-null    float64
 3   petal_width   149 non-null    float64
 4   species       149 non-null    object
dtypes: float64(4), object(1)
memory usage: 7.0+ KB
None
   sepal_length  sepal_width  petal_length  petal_width      species
0           5.1          3.5           1.4          0.2  Iris-setosa
1           4.9          3.0           1.4          0.2  Iris-setosa
2           4.7          3.2           1.3          0.2  Iris-setosa
3           4.6          3.1           1.5          0.2  Iris-setosa
4           5.0          3.6           1.4          0.2  Iris-setosa
```

```
Summary Statistics Before Cleaning:
       sepal_length  sepal_width  petal_length  petal_width
count    150.000000   150.000000    150.000000   150.000000
mean       5.843333     3.054000      3.758667     1.198667
std        0.828066     0.433594      1.764420     0.763161
min        4.300000     2.000000      1.000000     0.100000
25%        5.100000     2.800000      1.600000     0.300000
50%        5.800000     3.000000      4.350000     1.300000
75%        6.400000     3.300000      5.100000     1.800000
max        7.900000     4.400000      6.900000     2.500000

Summary Statistics After Cleaning:
       sepal_length  sepal_width  petal_length  petal_width
count    149.000000   149.000000    149.000000   149.000000
mean       5.844295     3.044966      3.773826     1.204027
std        0.830775     0.420655      1.760543     0.762896
min        4.300000     2.000000      1.000000     0.100000
25%        5.100000     2.800000      1.600000     0.300000
50%        5.800000     3.000000      4.400000     1.300000
75%        6.400000     3.300000      5.100000     1.800000
max        7.900000     4.200000      6.900000     2.500000
```

1B) Aim: Load a dataset, calculate descriptive summary statistics, create visualizations using different graphs, and identify potential features and target variables Note: Explore Univariate and Bivariate graphs (Matplotlib) and Seaborn for visualization.

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

def load_dataset(file_url):
    try:
        data = pd.read_csv(file_url)
        print("Dataset loaded successfully!")
        return data
    except Exception as e:
        print(f"Error loading dataset: {e}")
        return None

def descriptive_statistics(data):
    print("\nDescriptive Statistics:")
    print(data.describe(include='all'))
    print("\nMissing Values in Each Column:")
    print(data.isnull().sum())

def create_visualizations(data):
    sns.set(style="whitegrid")

    # Filter only numeric columns for plotting
    numeric_data = data.select_dtypes(include='number')

    # Histogram
    numeric_data.hist(bins=30, edgecolor='black', figsize=(12, 10))
    plt.suptitle('Histogram of Numeric Features')
    plt.show()

    # Box Plot
    plt.figure(figsize=(12, 6))
    sns.boxplot(data=numeric_data)
    plt.title('Box Plot of Numeric Features')
    plt.xticks(rotation=45)
    plt.show()

    # Correlation Heatmap
    plt.figure(figsize=(10, 8))
    corr = numeric_data.corr()
    sns.heatmap(corr, annot=True, cmap='coolwarm', linewidths=0.5)
    plt.title('Correlation Heatmap of Numeric Features')
```

```
    plt.show()

    # Pairplot (use only selected numeric columns to avoid long processing or errors)
    selected_cols = ['Age', 'Fare', 'Pclass'] # Safe numeric columns from Titanic dataset
    sns.pairplot(data[selected_cols].dropna()) # Drop rows with NaN
    plt.suptitle('🧬 Pairplot of Selected Numeric Features', y=1.02)
    plt.show()

def main():
    file_url = 'https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.csv'

    data = load_dataset(file_url)

    if data is not None:
        descriptive_statistics(data)
        create_visualizations(data)

if _name_== "_main_":
    main()
```

Output:

```
Dataset loaded successfully!

Descriptive Statistics:
        PassengerId    Survived      Pclass                 Name   Sex         Age       SibSp       Parch  Ticket        Fare Cabin Embarked
count    891.000000  891.000000  891.000000                  891   891  714.000000  891.000000  891.000000     891  891.000000   204      889
unique          NaN         NaN         NaN                  891     2         NaN         NaN         NaN     681         NaN   147        3
top             NaN         NaN         NaN  Dooley, Mr. Patrick  male         NaN         NaN         NaN  347082         NaN    G6        S
freq            NaN         NaN         NaN                    1   577         NaN         NaN         NaN       7         NaN     4      644
mean     446.000000    0.383838    2.308642                  NaN   NaN   29.699118    0.523008    0.381594     NaN   32.204208   NaN      NaN
std      257.353842    0.486592    0.836071                  NaN   NaN   14.526497    1.102743    0.806057     NaN   49.693429   NaN      NaN
min        1.000000    0.000000    1.000000                  NaN   NaN    0.420000    0.000000    0.000000     NaN    0.000000   NaN      NaN
25%      223.500000    0.000000    2.000000                  NaN   NaN   20.125000    0.000000    0.000000     NaN    7.910400   NaN      NaN
50%      446.000000    0.000000    3.000000                  NaN   NaN   28.000000    0.000000    0.000000     NaN   14.454200   NaN      NaN
75%      668.500000    1.000000    3.000000                  NaN   NaN   38.000000    1.000000    0.000000     NaN   31.000000   NaN      NaN
max      891.000000    1.000000    3.000000                  NaN   NaN   80.000000    8.000000    6.000000     NaN  512.329200   NaN      NaN
```

```
Missing Values in Each Column:
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

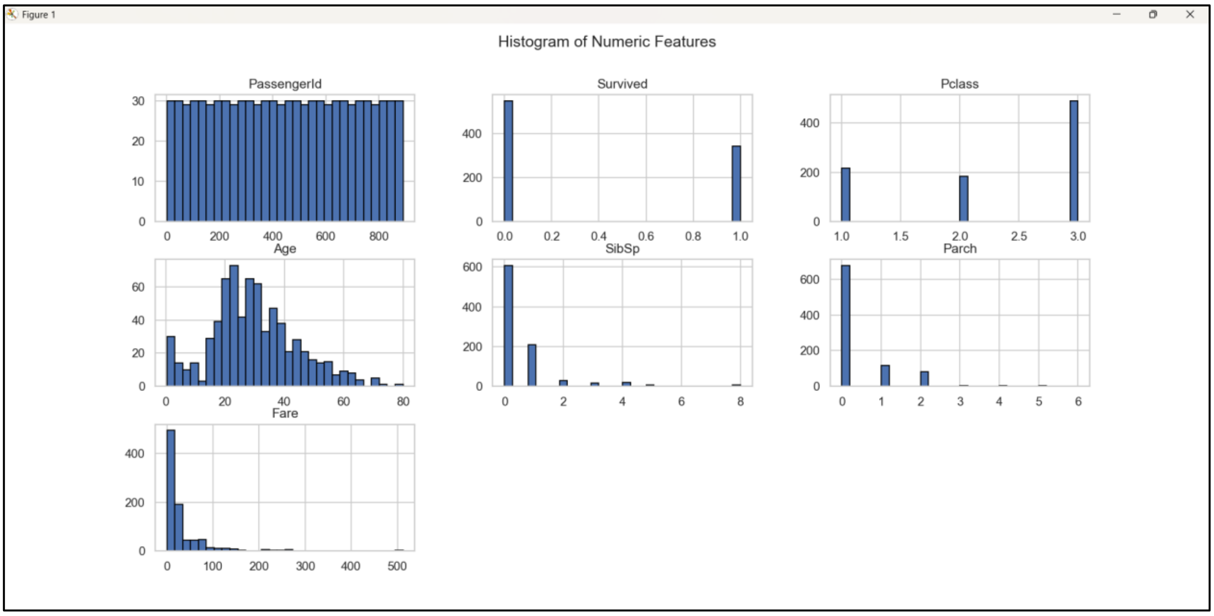


Figure 1
Histogram of Numeric Features
PassengerId
Survived
Pclass
Age
SibSp
Parch
Fare

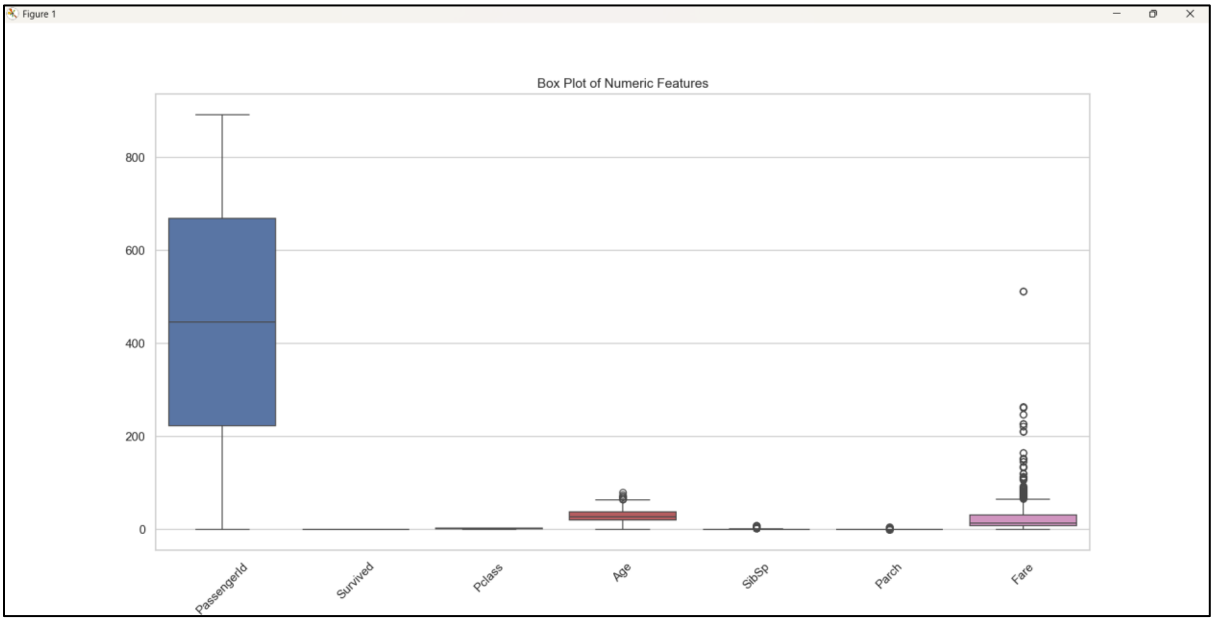


Figure 1
Box Plot of Numeric Features
PassengerId
Survived
Pclass
Age
SibSp
Parch
Fare

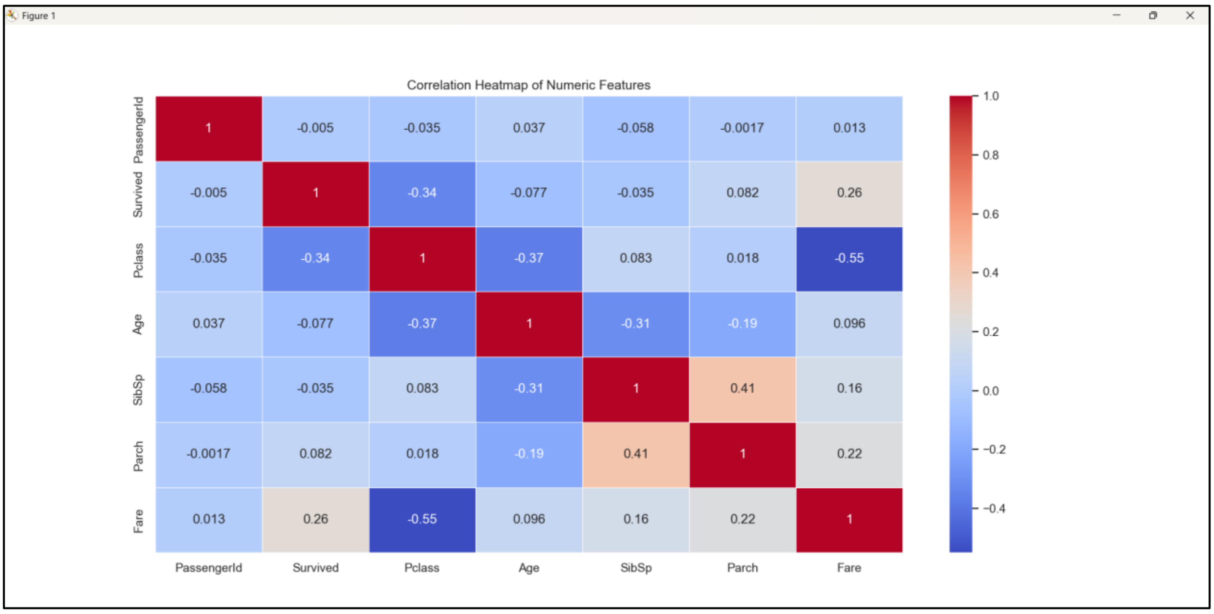


Figure 1
Correlation Heatmap of Numeric Features
PassengerId
Survived
Pclass
Age
SibSp
Parch
Fare
1 -0.005 -0.035 0.037 -0.058 -0.0017 0.013
-0.005 1 -0.34 -0.077 -0.035 0.082 0.26
-0.035 -0.34 1 -0.37 0.083 0.018 -0.55
0.037 -0.077 -0.37 1 -0.31 -0.19 0.096
-0.058 -0.035 0.083 -0.31 1 0.41 0.16
-0.0017 0.082 0.018 -0.19 0.41 1 0.22
0.013 0.26 -0.55 0.096 0.16 0.22 1
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4

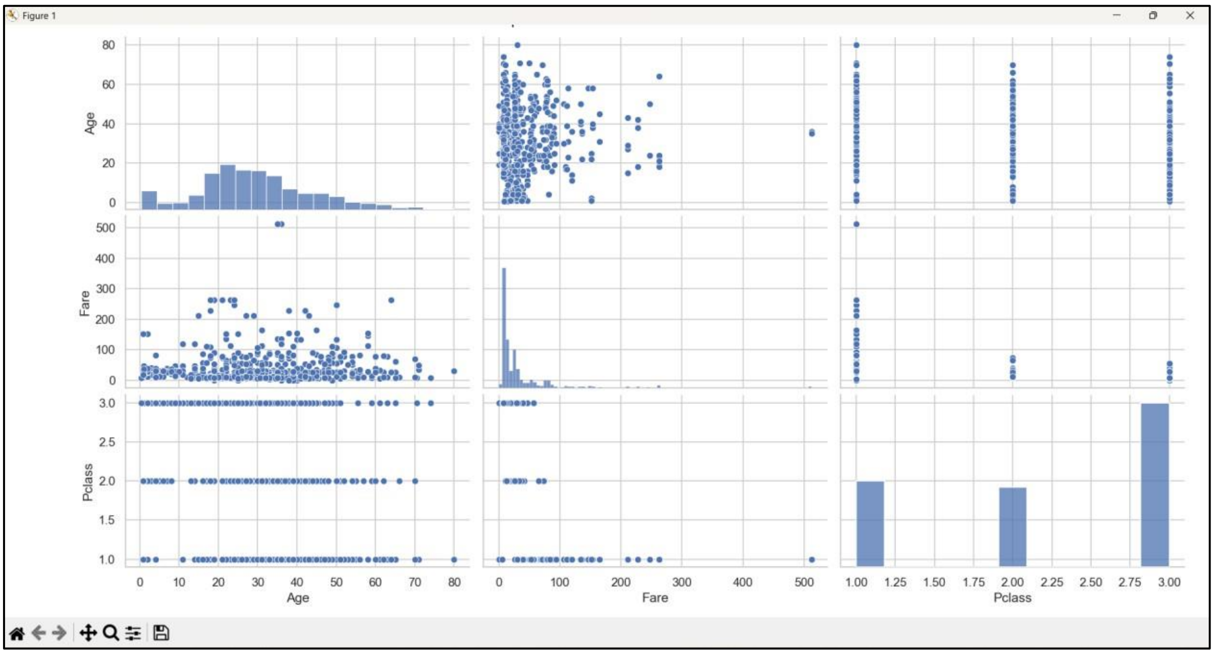


Figure 1
Age
Fare
Pclass
0 10 20 30 40 50 60 70 80
0 100 200 300 400 500
1.00 1.25 1.50 1.75 2.00 2.25 2.50 2.75 3.00

1C) Aim: Create or Explore datasets to use all pre-processing routines like label encoding, scaling, and binarization**.**

Source Code

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder, MinMaxScaler, Binarizer

def load_dataset(url):
    try:
        return pd.read_csv(url)
    except Exception as e:
        print(f"Error loading dataset: {e}")
        return None

def preprocess_data(data):
    numeric_cols = data.select_dtypes(include=['number']).columns
    data[numeric_cols] = data[numeric_cols].fillna(data[numeric_cols].mean())
    for col in data.select_dtypes(include='object').columns:
        data[col] = LabelEncoder().fit_transform(data[col].astype(str))
    scaler = MinMaxScaler()
    data[numeric_cols] = scaler.fit_transform(data[numeric_cols])
    binarizer = Binarizer(threshold=0.5)
    data[numeric_cols] = binarizer.fit_transform(data[numeric_cols])
    return data
def main():
    url = 
'https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.csv
'
    data = load_dataset(url)
    if data is not None:
        print("Original Data:")
        print(data.head())
        data = preprocess_data(data)
        print("\nPreprocessed Data:")
        print(data.head())

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

Output:

```
Original Data:
   PassengerId  Survived  Pclass                                               Name     Sex   Age  SibSp  Parch            Ticket     Fare Cabin Embarked
0            1         0       3                            Braund, Mr. Owen Harris    male  22.0      1      0         A/5 21171   7.2500   NaN        S
1            2         1       1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0      1      0          PC 17599  71.2833   C85        C
2            3         1       3                             Heikkinen, Miss. Laina  female  26.0      0      0  STON/O2. 3101282   7.9250   NaN        S
3            4         1       1       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1      0            113803  53.1000  C123        S
4            5         0       3                           Allen, Mr. William Henry    male  35.0      0      0            373450   8.0500   NaN        S

Preprocessed Data:
   PassengerId  Survived  Pclass  Name  Sex  Age  SibSp  Parch  Ticket  Fare  Cabin  Embarked
0          0.0       0.0     1.0   108    1  0.0    0.0    0.0     523   0.0    147         2
1          0.0       1.0     0.0   190    0  0.0    0.0    0.0     596   0.0     81         0
2          0.0       1.0     1.0   353    0  0.0    0.0    0.0     669   0.0    147         2
3          0.0       1.0     0.0   272    0  0.0    0.0    0.0      49   0.0     55         2
4          0.0       0.0     1.0    15    1  0.0    0.0    0.0     472   0.0    147         2
```

Practical No: - 2

Aim: Hypothesis Testing.

Writeups:

Aim: Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a. CSV file and generate the final specific hypothesis.

Code:

```
import pandas as pd

def find_s_algorithm(data, target_column):
    specific_hypothesis = ['Ø'] * (len(data.columns) - 1)
    for _, row in data.iterrows():
        if row[target_column] == "Yes":
            for i in range(len(specific_hypothesis)):
                if specific_hypothesis[i] == 'Ø':
                    specific_hypothesis[i] = row.iloc[i]
                elif specific_hypothesis[i] != row.iloc[i]:
                    specific_hypothesis[i] = '?'
    return specific_hypothesis

def main():
    dataset = {
        'Sky': ['Sunny', 'Sunny', 'Rainy', 'Sunny', 'Sunny'],
        'Temp': ['Warm', 'Warm', 'Cold', 'Warm', 'Warm'],
        'Humidity': ['Normal', 'High', 'High', 'High', 'Normal'],
        'Wind': ['Strong', 'Strong', 'Strong', 'Strong', 'Strong'],
        'Water': ['Warm', 'Warm', 'Cold', 'Warm', 'Warm'],
        'Forecast': ['Same', 'Same', 'Change', 'Same', 'Same'],
        'EnjoySport': ['Yes', 'Yes', 'No', 'Yes', 'Yes']
    }
    df = pd.DataFrame(dataset)
    df.to_csv("training_data.csv", index=False)
    print("Dataset saved to training_data.csv")

    data = pd.read_csv("training_data.csv")
    print("\nTraining Data:")
    print(data)

    specific_hypothesis = find_s_algorithm(data, 'EnjoySport')
    print("\nFinal Specific Hypothesis:")
    print(specific_hypothesis)

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

Output:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\Shivang Singh> python -u "C:\Users\SHIVAN~1\AppData\Local\Temp\tempCodeRunnerFile.python"
Dataset saved to training_data.csv

Training Data:
     Sky  Temp Humidity    Wind Water Forecast EnjoySport
0  Sunny  Warm   Normal  Strong  Warm     Same        Yes
1  Sunny  Warm     High  Strong  Warm     Same        Yes
2  Rainy  Cold     High  Strong  Cold   Change         No
3  Sunny  Warm     High  Strong  Warm     Same        Yes
4  Sunny  Warm   Normal  Strong  Warm     Same        Yes

Final Specific Hypothesis:
['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
```

Practical No: - 3

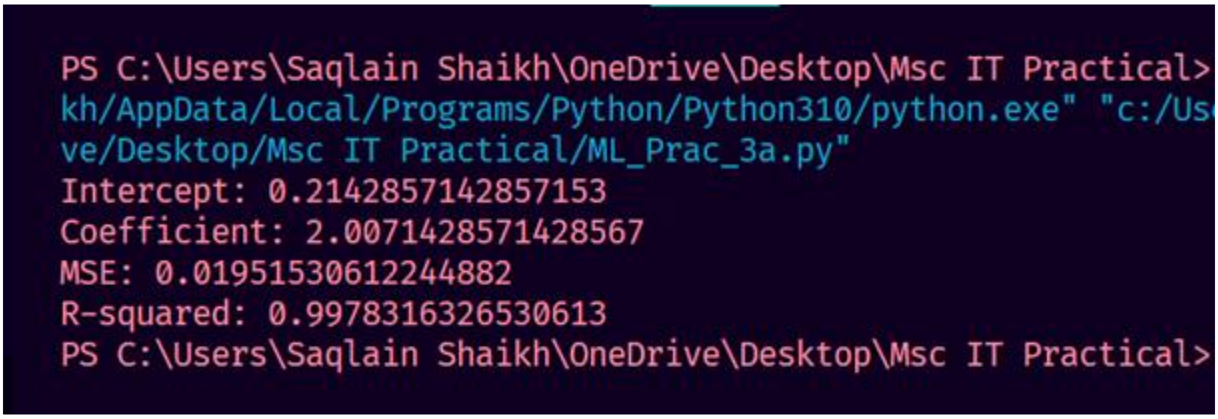
Aim: Linear Models

Writeups:

Practical 3A:

Code:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
data = {'X': [1, 2, 3, 4, 5], 'Y': [2.2, 4.1, 6.3, 8.2, 10.1]}
df = pd.DataFrame(data)
X = df[['X']]
Y = df['Y']
X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
test_size=0.4,
random_state=42)
model = LinearRegression()
model.fit(X_train, Y_train)
intercept = model.intercept_
coefficient = model.coef_[0]
Y_pred = model.predict(X_test)
mse = mean_squared_error(Y_test, Y_pred)

if len(Y_test) > 1:
    r_squared = r2_score(Y_test, Y_pred)
else:
    r_squared = float('nan')  # Set R-squared to NaN if only one
test sample

print(f"Intercept: {intercept}")
print(f"Coefficient: {coefficient}")
print(f"MSE: {mse}")
print(f"R-squared: {r_squared}")
```

OUTPUT:



Practical 3B
AIM:

- **Multiple Linear Regression Extend linear regression to multiple features. Handle feature selection and potential multicollinearity.**

CODE:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from statsmodels.stats.outliers_influence import
variance_inflation_factor

data = {
    'Feature1': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    'Feature2': [2, 4, 6, 8, 10, 12, 14, 16, 18, 20],
    'Feature3': [1, 3, 2, 4, 3, 5, 6, 7, 8, 9],
    'Target': [3, 7, 5, 9, 11, 15, 17, 21, 23, 27]
}
df = pd.DataFrame(data)

X = df[['Feature1', 'Feature2', 'Feature3']]
Y = df['Target']

vif_data = pd.DataFrame()
vif_data['Feature'] = X.columns
vif_data['VIF'] = [variance_inflation_factor(X.values, i) for i in
range(X.shape[1])]

print("\nVIF for Features:")
print(vif_data)

X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
test_size=0.2,
random_state=42)

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

print("\nIntercept:", model.intercept_)
print("Coefficients:", model.coef_)

Y_pred = model.predict(X_test)

print("MSE:", mean_squared_error(Y_test, Y_pred))
print("R-squared:", r2_score(Y_test, Y_pred))
```

```
  vif = 1. / (1. - r_squared_i)

VIF for Features:
    Feature        VIF
0  Feature1        inf
1  Feature2        inf
2  Feature3  69.271726

Intercept: -1.4322344322344343
Coefficients: [0.33846154 0.67692308 1.21611722]
MSE: 1.1180882609454033
R-squared: 0.982529870922728
PS C:\Users\Saqlain Shaikh\OneDrive\Desktop\Msc IT Practical>
```

OUTPUT:

```
Ridge: [ 2.5807138  -1.77414379  0.90149729] 0.04458190346459912 0.9584098704060822
Lasso: [ 1.60397718 -0.98933401  0.        ] 0.31614295332261333 0.705072565837569
ElasticNet: [ 1.38765543 -0.98031813  0.22377714] 0.34657204388252716 0.67668549122966
67
PS C:\Users\Saqlain Shaikh\OneDrive\Desktop\Msc IT Practical>
```

PRACTICAL3C
AIM:

- **Potential multicell linearity. Regularized Linear Models (Ridge, Lasso, Elastic Net)**
- **Implement regression variants like LASSO and Ridge on any generated dataset. 4 Discriminative Models OC2,O**

CODE:

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge, Lasso, ElasticNet
from sklearn.metrics import mean_squared_error, r2_score
np.random.seed(42)
X = np.random.rand(100, 3)
Y = 3 * X[:, 0] - 2 * X[:, 1] + X[:, 2] + np.random.normal(0, 0.1,
100)
X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
test_size=0.2,
random_state=42)
ridge_model = Ridge(alpha=1.0)
ridge_model.fit(X_train, Y_train)
ridge_pred = ridge_model.predict(X_test)
print("Ridge:", ridge_model.coef_, mean_squared_error(Y_test,
ridge_pred),
r2_score(Y_test, ridge_pred))
lasso_model = Lasso(alpha=0.1)
lasso_model.fit(X_train, Y_train)
lasso_pred = lasso_model.predict(X_test)
print("Lasso:", lasso_model.coef_, mean_squared_error(Y_test,
lasso_pred),
r2_score(Y_test, lasso_pred))
elasticnet_model = ElasticNet(alpha=0.1, l1_ratio=0.5)
elasticnet_model.fit(X_train, Y_train)
elasticnet_pred = elasticnet_model.predict(X_test)
print("ElasticNet:", elasticnet_model.coef_,
mean_squared_error(Y_test,
elasticnet_pred), r2_score(Y_test, elasticnet_pred))
```

OUTPUT:

Practical No: - 4

Aim: Discriminative Models

Writeups:

PRACTICAL 4a:

- Perform binary classification using logistic regression. Calculate accuracy, precision, recall, and understand the ROC curve.

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score,
    roc_curve, auc
)
import matplotlib.pyplot as plt

np.random.seed(42)
X = np.random.rand(100, 2)
Y = (X[:, 0] + X[:, 1] > 1).astype(int)

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3,
random_state=42)

model = LogisticRegression()
model.fit(X_train, Y_train)

Y_pred = model.predict(X_test)
Y_pred_prob = model.predict_proba(X_test)[:, 1]

print("Accuracy:", accuracy_score(Y_test, Y_pred))
print("Precision:", precision_score(Y_test, Y_pred))
print("Recall:", recall_score(Y_test, Y_pred))

fpr, tpr, _ = roc_curve(Y_test, Y_pred_prob)
plt.plot(fpr, tpr, label=f"AUC = {auc(fpr, tpr):.2f}")
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend(loc="lower right")
plt.show()
```

Output:

```
Accuracy: 0.8666666666666667
Precision: 1.0
Recall: 0.7333333333333333
```

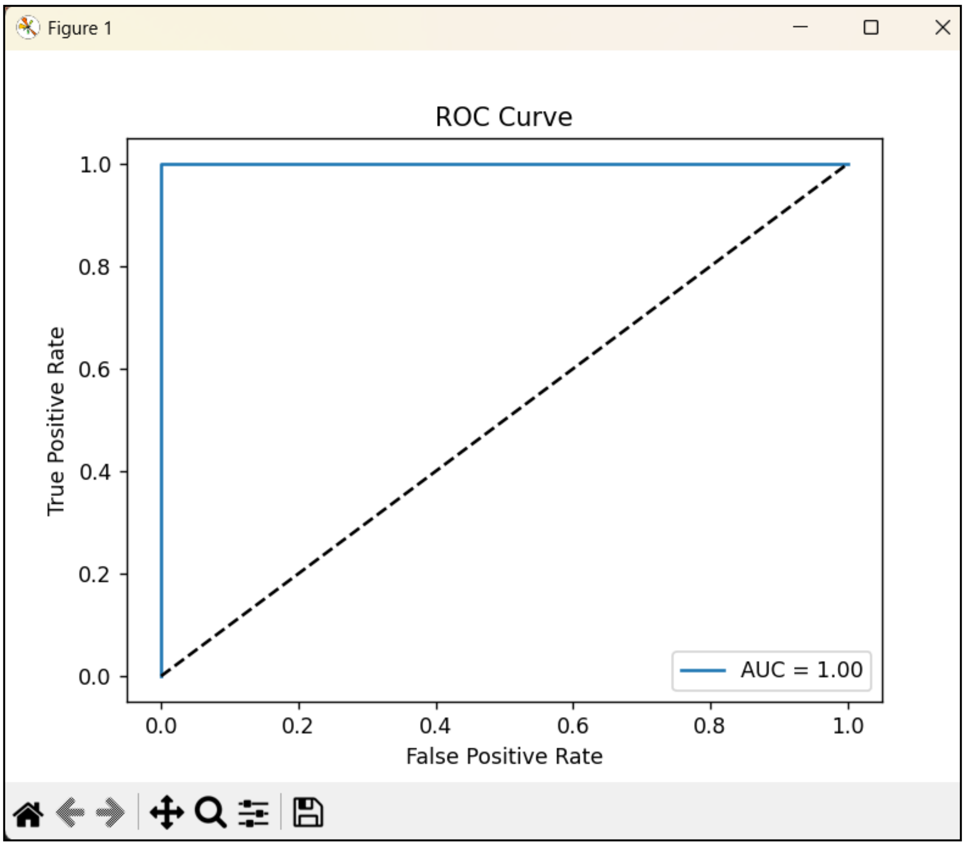


Figure 1
ROC Curve
True Positive Rate
False Positive Rate
AUC = 1.00
0.0
0.2
0.4
0.6
0.8
1.0

PRACTICAL 4b:

- Implement and demonstrate k-nearest Neighbor algorithm. Read the training data from a CSV file and build the model to classify a test sample. Print both correct and wrong predictions.

Code:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
import urllib.request
import os
url = "https://archive.ics.uci.edu/ml/machine-learning-
databases/iris/iris.data"
dataset_file = "iris.data"
if not os.path.exists(dataset_file):
    urllib.request.urlretrieve(url, dataset_file)
columns = ["sepal_length", "sepal_width", "petal_length", "petal_width",
"class"]
data = pd.read_csv(dataset_file, header=None, names=columns)
X = data.iloc[:, :-1].values
y = data.iloc[:, -1].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
k = 3
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
conf_matrix = confusion_matrix(y_test, y_pred)
report = classification_report(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)
print("\nClassification Report:")
print(report)
correct_predictions = np.sum(y_test == y_pred)
incorrect_predictions = np.sum(y_test != y_pred)
print(f"\nCorrect Predictions: {correct_predictions}")
print(f"Incorrect Predictions: {incorrect_predictions}")
results = pd.DataFrame({"Actual": y_test, "Predicted": y_pred})
print("\nSample Results:")
print(results.head())
```

Output:

```
Confusion Matrix:
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]

Classification Report:
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        10
Iris-versicolor       1.00      1.00      1.00         9
 Iris-virginica       1.00      1.00      1.00        11

       accuracy                           1.00        30
      macro avg       1.00      1.00      1.00        30
   weighted avg       1.00      1.00      1.00        30


Correct Predictions: 30
Incorrect Predictions: 0

Sample Results:
            Actual        Predicted
0  Iris-versicolor  Iris-versicolor
1      Iris-setosa      Iris-setosa
2   Iris-virginica   Iris-virginica
3  Iris-versicolor  Iris-versicolor
4  Iris-versicolor  Iris-versicolor
```

Pratical4c:

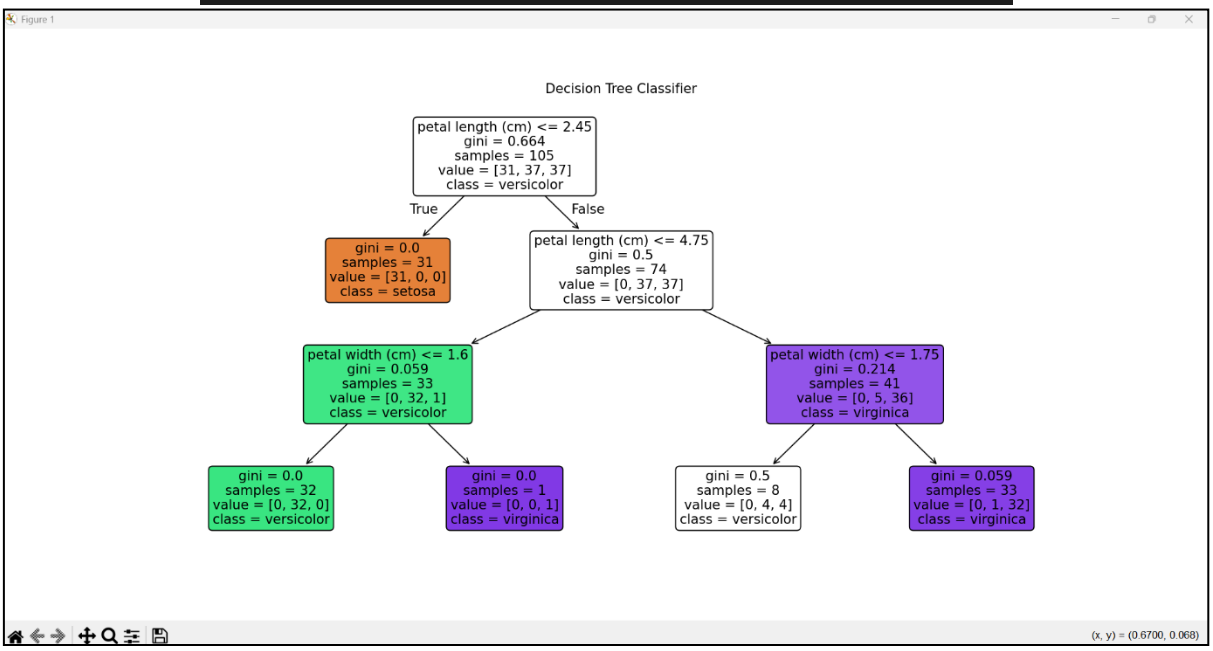
- Build a decision tree classifier or regressor. Control hyper parameters like tree depth to avoid overfitting. Visualize the tree.

Code:

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

data = load_iris()
X = data.data
y = data.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

clf = DecisionTreeClassifier(max_depth=3, random_state=42)
clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy of Decision Tree Classifier: {accuracy:.2f}')

plt.figure(figsize=(12,8))
plot_tree(clf, filled=True, feature_names=data.feature_names,
class_names=data.target_names, rounded=True)
plt.title("Decision Tree Classifier")
plt.show()
```

Output:

```
Accuracy of Decision Tree Classifier: 1.00
```



Practical 4d:

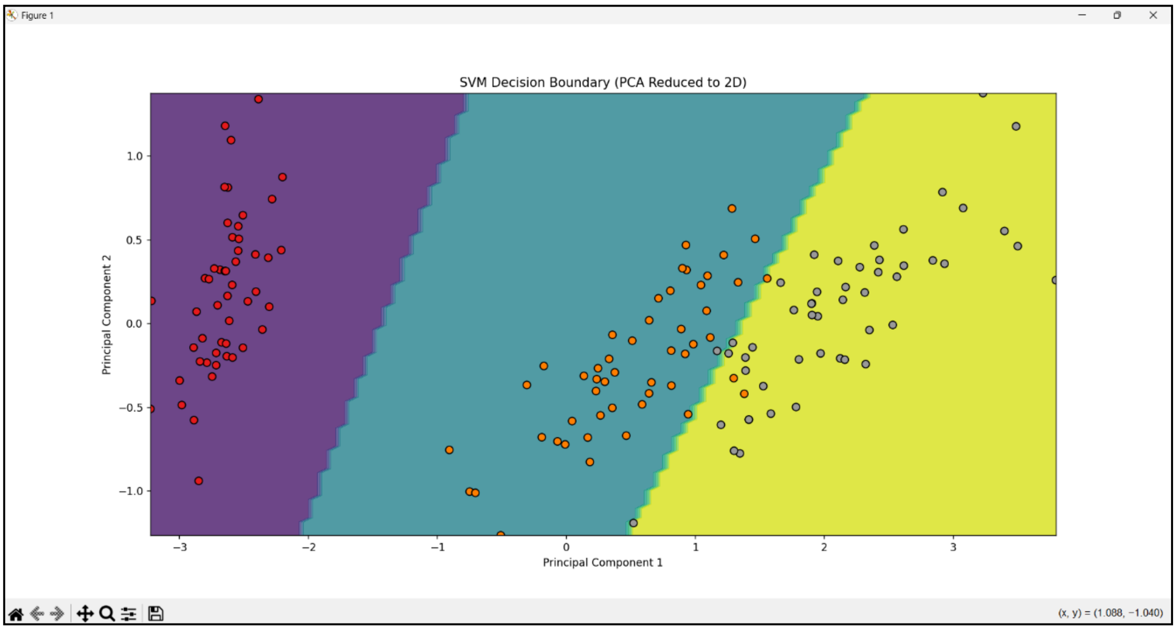
- Implement a Support Vector Machine for any relevant dataset.

Code:

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

data = load_iris()
X = data.data
y = data.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

svm_clf = SVC(kernel='linear', random_state=42)
svm_clf.fit(X_train, y_train)

y_pred = svm_clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy of SVM classifier: {accuracy:.2f}')

pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

svm_clf_2d = SVC(kernel='linear', random_state=42)
svm_clf_2d.fit(X_pca, y)

xx, yy = np.meshgrid(np.linspace(X_pca[:, 0].min(), X_pca[:, 0].max(),
100),
                     np.linspace(X_pca[:, 1].min(), X_pca[:, 1].max(),
100))
Z = svm_clf_2d.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

plt.contourf(xx, yy, Z, alpha=0.8)
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, edgecolors='k', marker='o',
s=50,
cmap=plt.cm.Set1)
plt.title("SVM Decision Boundary (PCA Reduced to 2D)")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.show()
```

Output:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\Shivang Singh> python -u "d:\6482_Shivang\MLPRACTICAL\4d prac.py"
Accuracy of SVM classifier: 1.00
```



Practical4e:

- Train a random forest ensemble. Experiment with the number of trees and feature sampling. Compare performance to a single decision tree.

Code:

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

data = load_iris()
X = data.data
y = data.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

dt_clf = DecisionTreeClassifier(random_state=42)
dt_clf.fit(X_train, y_train)
dt_y_pred = dt_clf.predict(X_test)
dt_accuracy = accuracy_score(y_test, dt_y_pred)

rf_clf = RandomForestClassifier(n_estimators=100, max_features='sqrt',
random_state=42)
rf_clf.fit(X_train, y_train)
rf_y_pred = rf_clf.predict(X_test)
rf_accuracy = accuracy_score(y_test, rf_y_pred)

rf_clf_50 = RandomForestClassifier(n_estimators=50, max_features='sqrt',
random_state=42)
rf_clf_50.fit(X_train, y_train)
rf_50_y_pred = rf_clf_50.predict(X_test)
rf_50_accuracy = accuracy_score(y_test, rf_50_y_pred)

print(f"Decision Tree Accuracy: {dt_accuracy:.2f}")
print(f"Random Forest (100 trees) Accuracy: {rf_accuracy:.2f}")
print(f"Random Forest (50 trees) Accuracy: {rf_50_accuracy:.2f}")
```

Output:

```
Decision Tree Accuracy: 1.00
Random Forest (100 trees) Accuracy: 1.00
Random Forest (50 trees) Accuracy: 1.00
PS C:\Users\Shivang Singh>
```

Practical4f

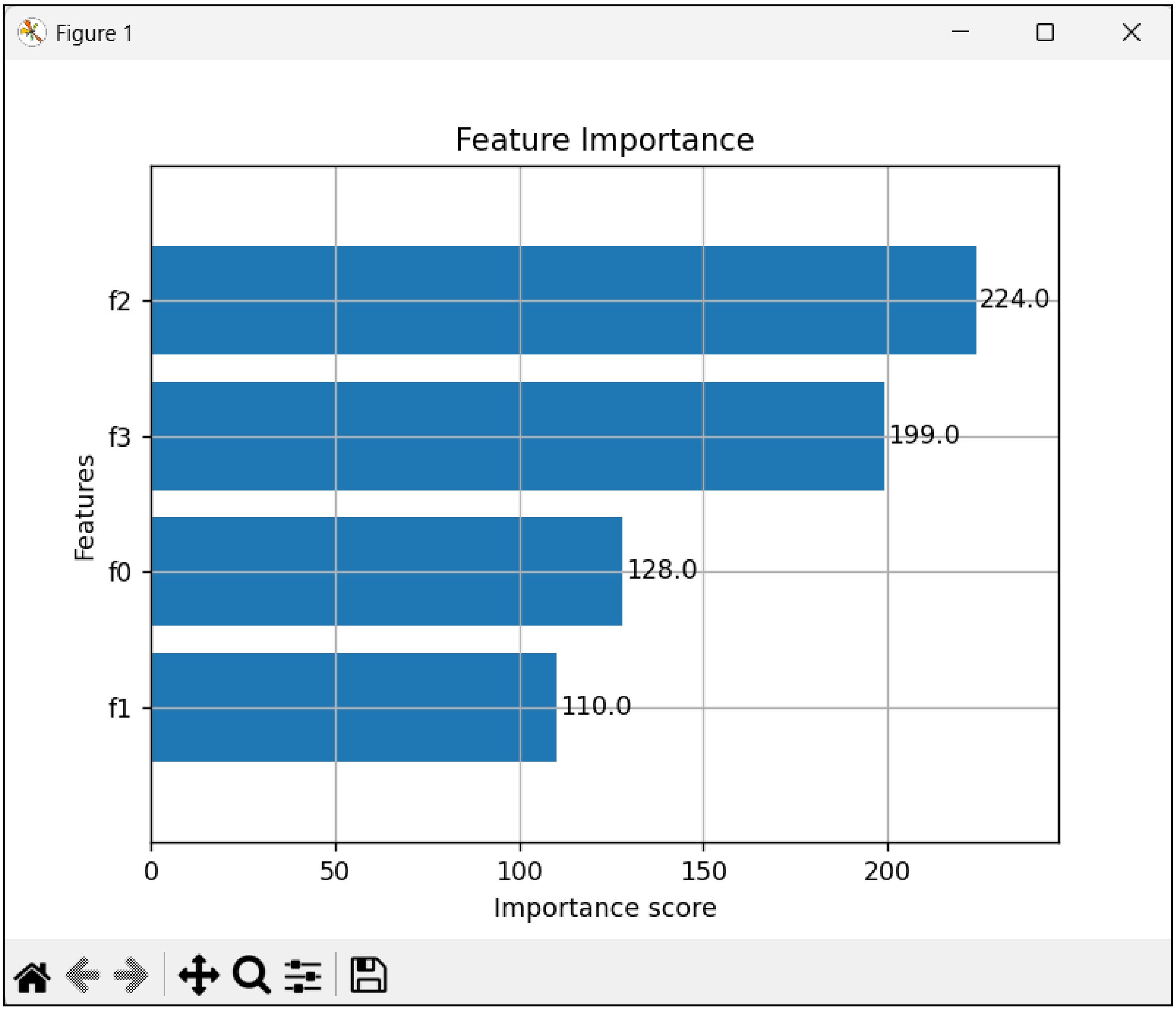
- Implement a gradient boosting machine (e.g., XGBoost). Tune hyperparameters and explore feature importance.

Code:

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
import xgboost as xgb
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

data = load_iris()
X = data.data
y = data.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

params = {
    'objective': 'multi:softmax',
    'num_class': 3,
    'max_depth': 3,
    'learning_rate': 0.1,
    'n_estimators': 100,
    'subsample': 0.8,
    'colsample_bytree': 0.8,
    'eval_metric': 'merror'
}

xgb_model = xgb.XGBClassifier(**params)
xgb_model.fit(X_train, y_train)

y_pred = xgb_model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy of XGBoost model: {accuracy:.2f}')

xgb.plot_importance(xgb_model, importance_type='weight',
max_num_features=4,
height=0.8)
plt.title('Feature Importance')
plt.show()
```

Output:

```
Accuracy of XGBoost model: 1.00
```



Practical No: - 5

Aim: Generative Models

Writeups:

Practical5a:

- **Implement and demonstrate the working of a Naive Bayesian classifier using a sample data set. Build the model to classify a test sample.**

Code:

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
data = load_iris()
X = data.data
y = data.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)
nb_classifier = GaussianNB()
nb_classifier.fit(X_train, y_train)
y_pred = nb_classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy of Naive Bayes classifier: {accuracy:.2f}')
test_sample = X_test[0].reshape(1, -1)
predicted_class = nb_classifier.predict(test_sample)
print(f'Predicted class for the test sample: {predicted_class[0]}')
```

Output:

```
================== RESTART: C:/Users/hp/Downloads/ML/MI
Accuracy of Naive Bayes classifier: 0.98
Predicted class for the test sample: 1
>
```

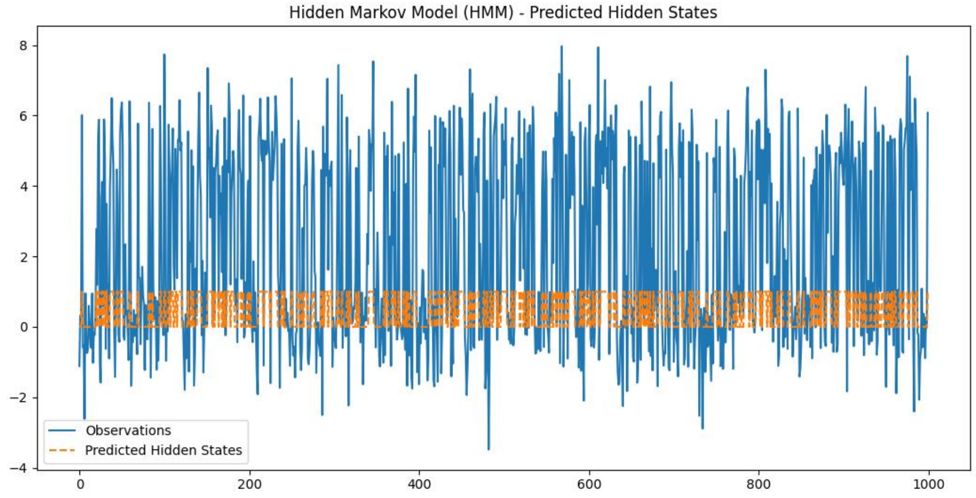
Practical 5b:

- **Implement Hidden Markov Models using hmmlearn**

Code:

```
import numpy as np
from hmmlearn.hmm import GaussianHMM
import matplotlib.pyplot as plt
np.random.seed(42)
hidden_states = 2
n_samples = 1000
trans_probs = np.array([[0.7, 0.3], [0.4, 0.6]])
means = np.array([[0.0], [5.0]])
# Covariance matrix needs to be 3D with shape (n_components, n_dim, n_dim)
covars = np.array([[[1.0]], [[1.0]]])  # Correct shape for 'full'
covariance
type
model = GaussianHMM(n_components=hidden_states, covariance_type="full",
n_iter=1000)
model.startprob_ = np.array([0.6, 0.4])
model.transmat_ = trans_probs
model.means_ = means
model.covars_ = covars
X, Z = model.sample(n_samples)

model.fit(X)
```

```
predicted_states = model.predict(X)
plt.figure(figsize=(12, 6))
plt.plot(X, label='Observations')
plt.plot(predicted_states, label='Predicted Hidden States', linestyle='--')
plt.legend()
plt.title('Hidden Markov Model (HMM) - Predicted Hidden States')
plt.show()
```



Practical No: - 6

Aim: Probabilistic Models

Writeups:

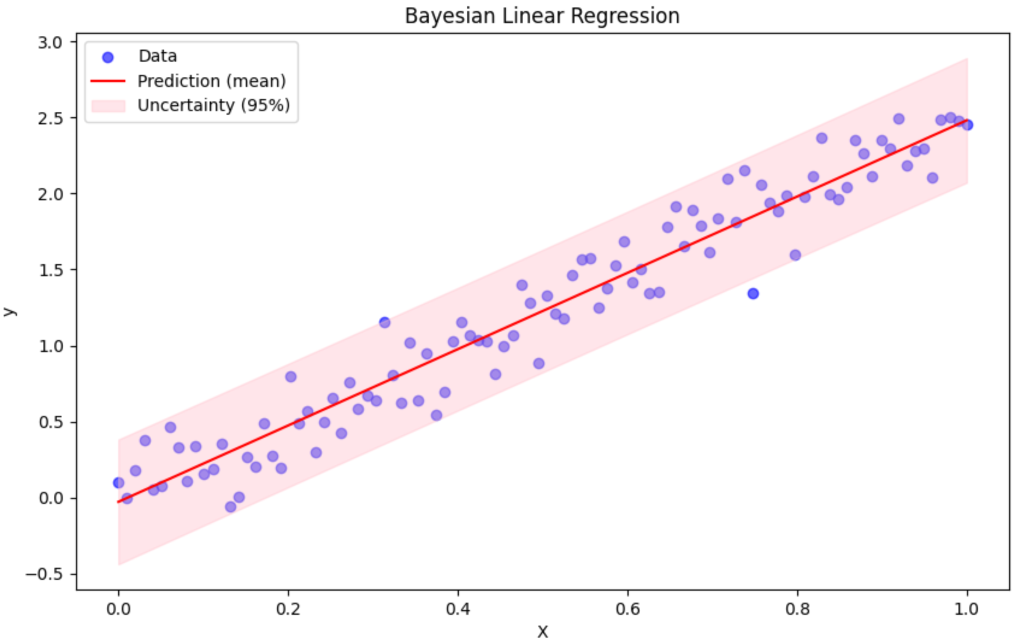
Practical6a

- **Implement Bayesian Linear Regression to explore prior and posterior distribution.**

Code:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
np.random.seed(42)
X = np.linspace(0, 1, 100).reshape(-1, 1)
y = 2.5 * X.squeeze() + np.random.normal(0, 0.2, X.shape[0])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
class BayesianLinearRegression:
    def __init__(self, alpha=1.0, beta=1.0):
        self.alpha = alpha
        self.beta = beta
        self.w_mean = None
        self.w_cov = None
    def fit(self, X, y):
        X = np.hstack((np.ones((X.shape[0], 1)), X))
        S0_inv = self.alpha * np.eye(X.shape[1])
        SN_inv = S0_inv + self.beta * X.T @ X
        SN = np.linalg.inv(SN_inv)
        mN = self.beta * SN @ X.T @ y
        self.w_mean = mN
        self.w_cov = SN

    def predict(self, X, return_std=False):
        X = np.hstack((np.ones((X.shape[0], 1)), X))
        mean = X @ self.w_mean
        if return_std:
            variance = 1 / self.beta + np.sum(X @ self.w_cov * X, axis=1)
            return mean, np.sqrt(variance)
        return mean
blr = BayesianLinearRegression(alpha=1.0, beta=25.0)
blr.fit(X_train, y_train)

X_pred = np.linspace(0, 1, 100).reshape(-1, 1)
y_pred, y_std = blr.predict(X_pred, return_std=True)
plt.figure(figsize=(10, 6))
plt.scatter(X, y, label="Data", color="blue", alpha=0.6)
plt.plot(X_pred, y_pred, label="Prediction (mean)", color="red")
plt.fill_between(
    X_pred.squeeze(),
    y_pred - 2 * y_std,
    y_pred + 2 * y_std,
    color="pink",
    alpha=0.4,
    label="Uncertainty (95%)", )
plt.title("Bayesian Linear Regression")
plt.xlabel("X")
plt.ylabel("y")
plt.legend()
plt.show()
```

Output:


Bayesian Linear Regression
Data
Prediction (mean)
Uncertainty (95%)
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
y
0.0
0.2
0.4
0.6
0.8
1.0
X

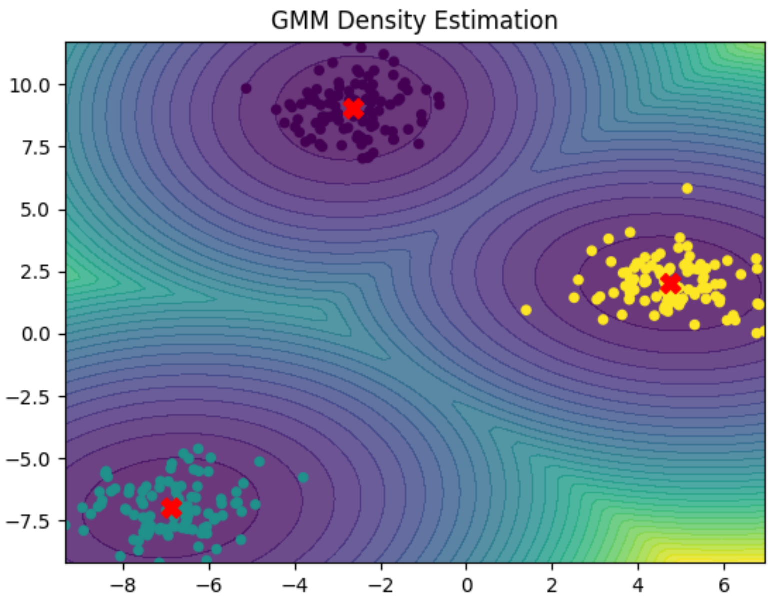
Practical6b

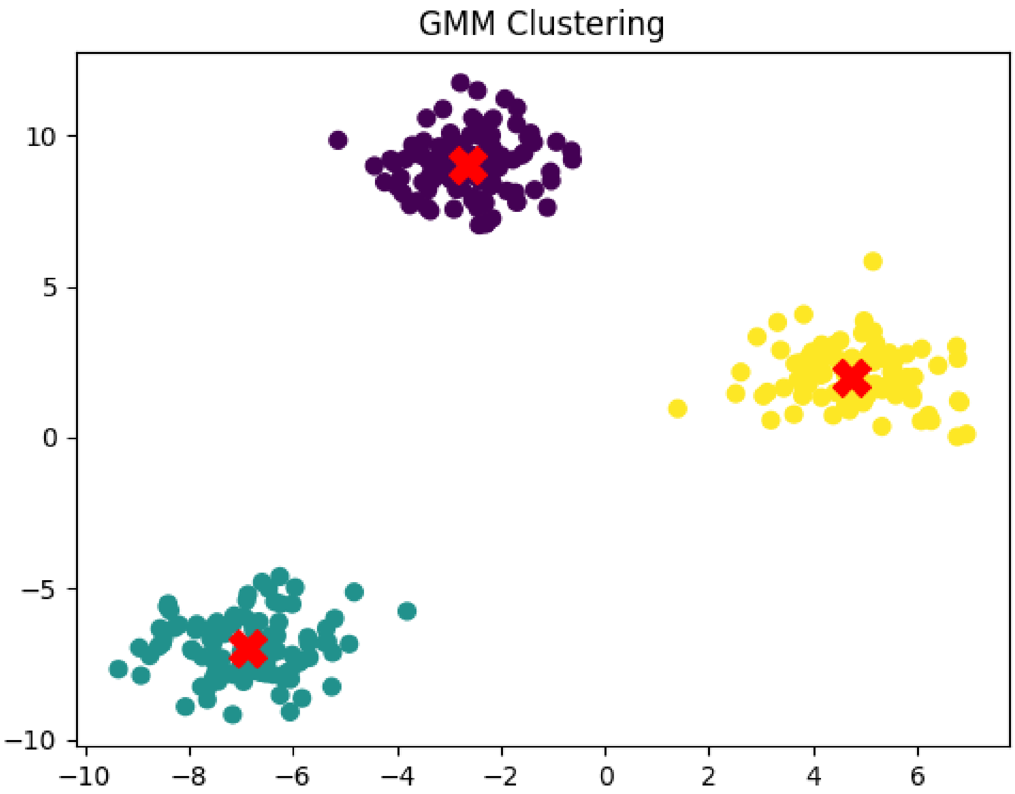
- **Implement Gaussian Mixture Models for density estimation and unsupervised clustering**

Code:

```
import numpy as np
from sklearn.mixture import GaussianMixture
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt
X, _ = make_blobs(n_samples=300, centers=3, cluster_std=1.0,
random_state=42)
gmm = GaussianMixture(n_components=3, covariance_type='full',
random_state=42).fit(X)
labels = gmm.predict(X)
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', s=40)
plt.scatter(gmm.means_[:, 0], gmm.means_[:, 1], c='red', s=200, marker='X')
plt.title("GMM Clustering")
plt.show()
x, y = np.linspace(X[:, 0].min(), X[:, 0].max(), 100), np.linspace(X[:,
1].min(), X[:, 1].max(), 100)
X_grid, Y_grid = np.meshgrid(x, y)
grid_points = np.c_[X_grid.ravel(), Y_grid.ravel()]
Z = -gmm.score_samples(grid_points).reshape(X_grid.shape)
plt.contourf(X_grid, Y_grid, Z, levels=30, cmap='viridis', alpha=0.8)
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', s=20)
plt.scatter(gmm.means_[:, 0], gmm.means_[:, 1], c='red', s=100, marker='X')
plt.title("GMM Density Estimation")
plt.show()
```

Output:




GMM Clustering
10
5
0
−5
−10
−10
−8
−6
−4
−2
0
2
4
6

Practical No: - 7

Aim: Model Evaluation and Hyperparameter Tuning.

Writeups:

Practical 7a:

- **Implement cross-validation techniques (k-fold, stratified, etc.) for robust model evaluation.**

Code:

```
import numpy as np
from sklearn.datasets import make_classification
from sklearn.model_selection import KFold, StratifiedKFold, cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
X, y = make_classification(n_samples=1000, n_features=10, n_informative=5,
n_redundant=2, random_state=42)
model = RandomForestClassifier(random_state=42)
kf = KFold(n_splits=5, shuffle=True, random_state=42)
kfold_scores = cross_val_score(model, X, y, cv=kf, scoring='accuracy')
print("K-Fold Cross-Validation Scores:", kfold_scores)
print("K-Fold Average Accuracy:", np.mean(kfold_scores))
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
stratified_scores = cross_val_score(model, X, y, cv=skf,
scoring='accuracy')
print("Stratified K-Fold Cross-Validation Scores:", stratified_scores)
print("Stratified K-Fold Average Accuracy:", np.mean(stratified_scores))
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
holdout_accuracy = accuracy_score(y_test, y_pred)
print("Holdout Validation Accuracy:", holdout_accuracy)
```

Output:

```
= RESTART: C:/Users/hp/OneDrive/Documents/MSCIT/PART 2/SEMESTER 3/ML/ML p7a.py
K-Fold Cross-Validation Scores: [0.955 0.915 0.9   0.925 0.935]
K-Fold Average Accuracy: 0.9260000000000002
Stratified K-Fold Cross-Validation Scores: [0.925 0.955 0.94  0.925 0.94 ]
Stratified K-Fold Average Accuracy: 0.937
Holdout Validation Accuracy: 0.94
```

Practical 7b:

- **Systematically explore combinations of hyperparameters to optimize model performance. (use grid and randomized search)**

Code:

```
import numpy as np
from sklearn.datasets import make_classification
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
X, y = make_classification(n_samples=1000, n_features=10, n_informative=5,
n_redundant=2, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)
model = RandomForestClassifier(random_state=42)
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5,
n_jobs=-1, verbose=2, scoring='accuracy')
grid_search.fit(X_train, y_train)
print("Best parameters from Grid Search:", grid_search.best_params_)
best_model_grid = grid_search.best_estimator_
y_pred_grid = best_model_grid.predict(X_test)
grid_accuracy = accuracy_score(y_test, y_pred_grid)

print("Grid Search Model Accuracy:", grid_accuracy)
param_dist = {
    'n_estimators': [50, 100, 200, 300],
    'max_depth': [None, 10, 20, 30, 40],
    'min_samples_split': [2, 5, 10, 15],
    'min_samples_leaf': [1, 2, 4, 6]
}
random_search = RandomizedSearchCV(estimator=model,
param_distributions=param_dist, n_iter=10, cv=5, n_jobs=-1, verbose=2,
scoring='accuracy', random_state=42)
random_search.fit(X_train, y_train)
print("Best parameters from Randomized Search:", random_search.best_params_)
best_model_random = random_search.best_estimator_
y_pred_random = best_model_random.predict(X_test)
random_accuracy = accuracy_score(y_test, y_pred_random)
print("Randomized Search Model Accuracy:", random_accuracy)
```

Output:

```
= RESTART: C:/Users/hp/OneDrive/Documents/MSCIT/PART 2/SEMESTER 3/ML/ML p7b.py =
Fitting 5 folds for each of 108 candidates, totalling 540 fits
Best parameters from Grid Search: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_est
imators': 200}
Grid Search Model Accuracy: 0.94
Fitting 5 folds for each of 10 candidates, totalling 50 fits
Best parameters from Randomized Search: {'n_estimators': 100, 'min_samples_split': 10, 'min_samples_leaf':
1, 'max_depth': None}
Randomized Search Model Accuracy: 0.9333333333333333
>
```

Practical No: - 8

Aim: Bayesian Learning

Writeups:

Practical 8a:

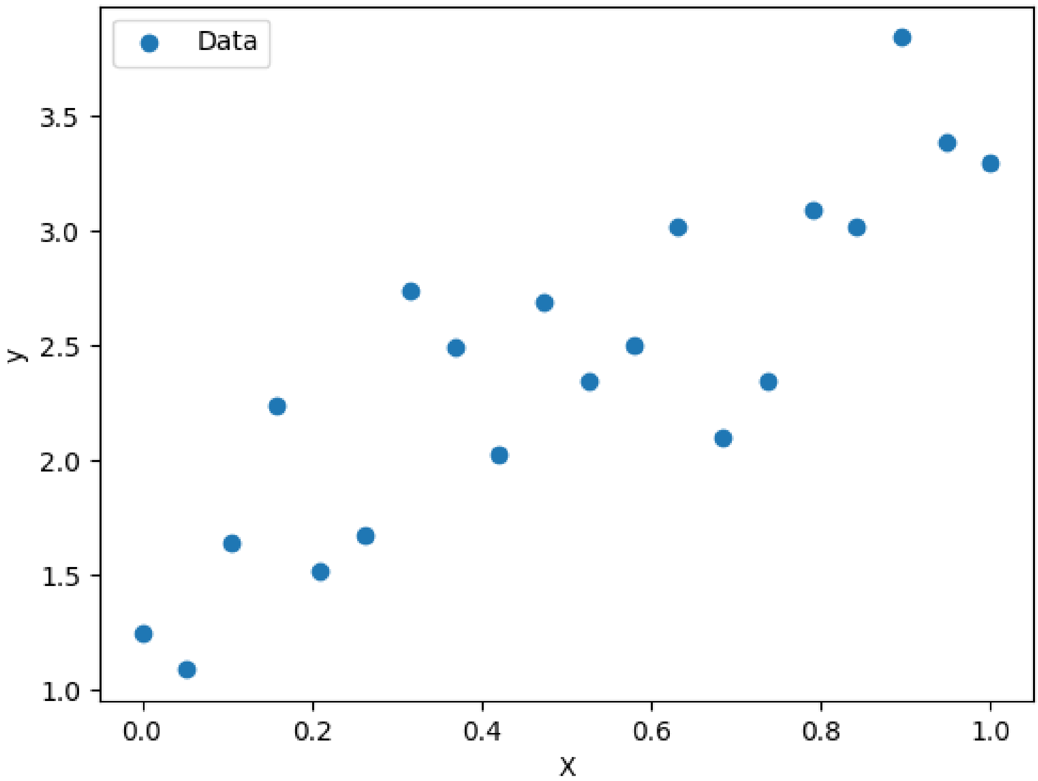
- Implement Bayesian Learning using inferences

Code:

```
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(42)
X = np.linspace(0, 1, 20)
true_slope = 3
true_intercept = 1
y = true_slope * X + true_intercept + np.random.normal(scale=0.5,
size=X.shape)
plt.scatter(X, y, label="Data")
plt.xlabel("X")
plt.ylabel("y")
plt.legend()
plt.show()
X_ = np.vstack([X, np.ones_like(X)]).T
sigma_prior = 10
mu_prior = np.array([0, 0])
sigma_likelihood = 0.5
sigma_likelihood_inv = np.linalg.inv(sigma_likelihood**2 * np.eye(len(X)))
X_T = X_.T
covariance_post = np.linalg.inv(X_T @ X_ / sigma_likelihood**2 + np.eye(2)
/
sigma_prior**2)
mean_post = covariance_post @ (X_T @ y / sigma_likelihood**2 + mu_prior /
sigma_prior**2)


num_samples = 1000
posterior_samples = np.random.multivariate_normal(mean_post,
covariance_post,
num_samples)
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.hist(posterior_samples[:, 0], bins=30, color='skyblue',
edgecolor='black')
plt.title("Posterior Distribution of Slope")
plt.subplot(1, 2, 2)
plt.hist(posterior_samples[:, 1], bins=30, color='skyblue',
edgecolor='black')
plt.title("Posterior Distribution of Intercept")
plt.tight_layout()
plt.show()
plt.scatter(X, y, label="Data")
for sample in posterior_samples:
    plt.plot(X, sample[0] * X + sample[1], color='red', alpha=0.05)
plt.xlabel("X")
plt.ylabel("y")
plt.legend()
plt.show()
estimated_slope = mean_post[0]
estimated_intercept = mean_post[1]
```

```
print(f"Estimated Slope: {estimated_slope}")
print(f"Estimated Intercept: {estimated_intercept}")
```

Output:



Practical 9

Aim: Deep Generative Models

a) Set up a generator network to produce samples and a discriminator network to distinguish between real and generated data. (Use a simple small dataset)

Goal

We want to build a **GAN** (Generative Adversarial Network) with:

- a **Generator** → makes fake data (from random noise).
- a **Discriminator** → checks if data is real or fake.
- They train together until the generator produces data that looks

real. We'll do this on a **small dataset** (toy 2D data or MNIST digits).

Step-by-Step Instructions:

1. Prepare Python and VS Code

- Make sure Python is installed (type python --version in terminal).
- Install **VS Code** and the **Python extension**.

2. Create a project folder

- Open a terminal (cmd, PowerShell, or bash) and
  type: mkdir gan-practical

```
mkdir gan-practical
cd gan-practical
```

cd gan-practical

3. Create a virtual environment (to keep things clean)

```
python -m venv venv
```

python -m venv venv

Activate it:

- Windows:

venv\Scripts\activ

Now your terminal should show (venv) in front.

4. Install needed libraries

pip install torch torchvision matplotlib numpy tqdm

```
pip install torch torchvision matplotlib numpy tqdm
```

This installs PyTorch and helper tools.

5. Open folder in VS Code

- Open VS Code → **File → Open Folder** → select gan-practical.
- In VS Code, press Ctrl+Shift+P → type **Python: Select Interpreter** → choose the one inside venv.

6. Create Python file

- Make a file in VS Code called gan_2d_toy.py.
  Paste this small code (toy GAN):

```
import torch, torch.nn as nn
import matplotlib.pyplot
as plt import numpy as np,
os

device = torch.device("cuda" if torch.cuda.is_available() else
"cpu") os.makedirs("samples", exist_ok=True)

# Real data: points around (1,0), (-1,0), (0,1),
(0,-1) def sample_real(batch):
    centers = np.array([[1,0],[-1,0],[0,1],[0,-1]],
    dtype=np.float32) idx = np.random.randint(0,4,batch)
    pts = centers[idx] +
    0.1*np.random.randn(batch,2).astype(np.float32) return
    torch.tensor(pts, device=device)

# Generator: noise -> 2D
point class
Generator(nn.Module):
    def __init__(self):
        super().__init__()
        self.net = nn.Sequential(
```

```
# Discriminator: 2D point -> prob class
Discriminator(nn.Module):
    def __init__(self):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(2,64), nn.LeakyReLU(0.2), nn.Linear(64,64), nn.LeakyReLU(0.2),
            nn.Linear(64,1), nn.Sigmoid()
        )
    def forward(self,x): return self.net(x)

G, D = Generator().to(device), Discriminator().to(device)
optG = torch.optim.Adam(G.parameters(), lr=2e-4, betas=(0.5,0.999)) optD =
torch.optim.Adam(D.parameters(), lr=2e-4, betas=(0.5,0.999)) bce = nn.BCELoss()

for it in range(1000): # 1. Train D
    real = sample_real(128)
    fake = G(torch.randn(128,2,device=device)).detach()
    lossD = (bce(D(real), torch.ones(128,1,device=device)) +
              bce(D(fake), torch.zeros(128,1,device=device))) * 0.5 optD.zero_grad(); lossD.backward();
    optD.step()

    # 2. Train G
    z = torch.randn(128,2,device=device) fake = G(z)
    lossG = bce(D(fake), torch.ones(128,1,device=device)) optG.zero_grad();
    lossG.backward(); optG.step()

    # Save sample plot every 200 steps if (it+1)%200==0:
        with torch.no_grad():
            gen = G(torch.randn(1000,2,device=device)).cpu().numpy() real =
            sample_real(1000).cpu().numpy()
            plt.scatter(real[:,0],real[:,1],alpha=0.2)
            plt.scatter(gen[:,0],gen[:,1],alpha=0.6)
            plt.savefig(f"samples/iter{it+1}.png"); plt.close() print("Saved sampl at iter",
            it+1)
```

7. Run the code

In VS Code terminal:

☞ This will create a folder samples/ with pictures showing how fake points move closer to real ones.

What is happening?

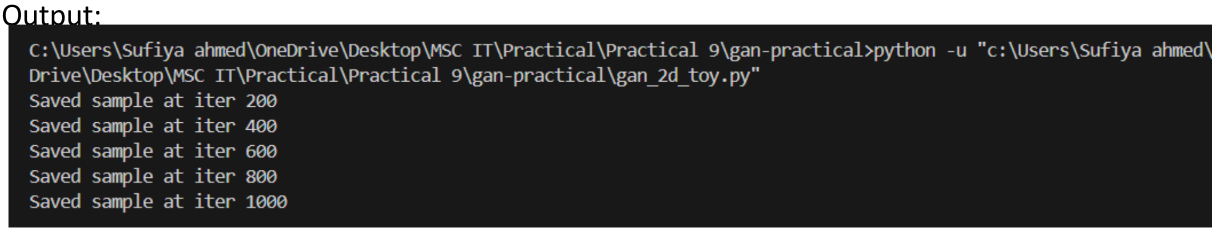
1. **Generator** starts with random noise → tries to make fake points.
2. **Discriminator** learns to tell apart real vs fake.

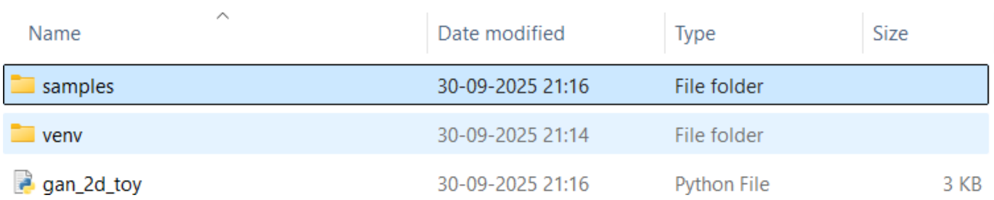
3. Both play a “game”:
    - o D gets better at spotting fakes.
    - o G gets better at fooling D.
4. After training, G can produce points similar to the real data.

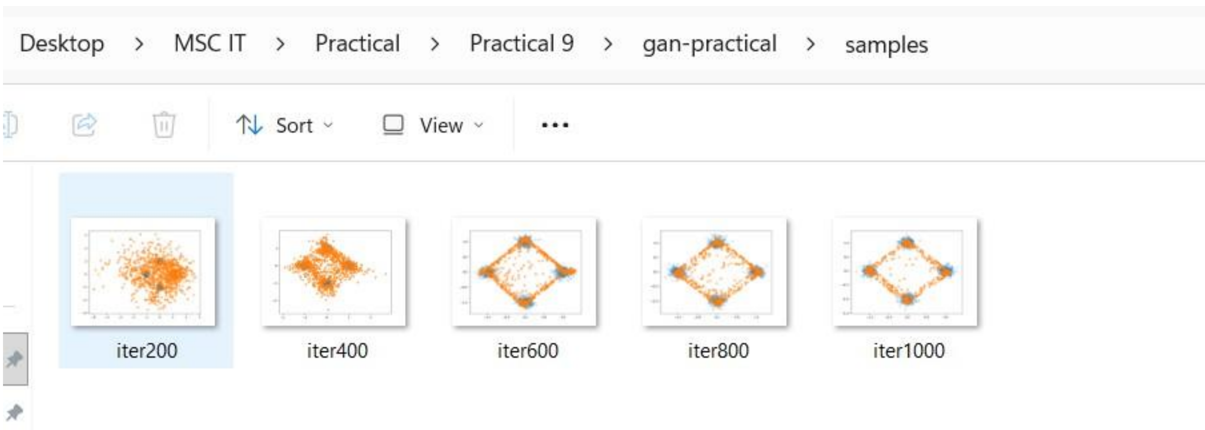
What to submit for practical

- The code (gan_2d_toy.py).
- The sample images (samples/iter200.png, iter400.png, etc.).
- A short explanation: *“Generator makes fake data from noise, Discriminator checks real vs fake, both train together until generator produces realistic data.”*

Output:







If you’ll get the below error how to fix it:

Error 1:

```
venv\Scripts\activate : File C:\Users\Sufiya
ahmed\OneDrive\Desktop\MSC IT\Practical\Practical 9\gan-
practical\venv\Scripts\Activate.ps1 cannot be loaded
because running scripts is disabled on this system. For more
information, see about_Execution_Policies at
https:/go.microsoft.com/fwlink/?LinkID=135170.
At line:1 char:1
+ venv\Scripts\activate
+ ~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : SecurityError: (:) [], PSSecurityException
    + FullyQualifiedErrorId : UnauthorizedAccess
```

Quick Fix (without changing policies)

Instead of PowerShell, open **Command Prompt**:

1. In VS Code → Ctrl+Shift+P → type **"Terminal: Select Default Profile"** → pick Command Prompt.
2. Open a new terminal (it should say cmd.exe at the top).
3. Run:

    venv\Scripts\activate.bat

Now your terminal will show (venv).