VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum- 590014, Karnataka.

LAB REPORT

on

Machine Learning (23CS6PCMAL)

Submitted by

Govind Jairam Rathod (1BM23CS407)

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
February 2025 – July 25

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 **Govind Jairam Rathod (1BM23CS407),** 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 Laboratory report has been approved as it satisfies the academic requirements in respect of an Machine Learning (23CS6PCMAL) work prescribed for the said degree.

Radhika A D
Assistant Professor
Department of CSE, BMSCE

Dr. Kavitha Sooda
Professor & HOD
Department of CSE, BMSCE

INDEX

Github Link: https://github.com/govindrd/ML-LAB

LABORATORY PROGRAM – 1

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

OBSERVATION BOOK

Lab-1 03-03-25

1

1> Car price dataset

-> problem statement:

The goal is to develop a model that accurately predict the price of a car based on it's features such as brand, model, year, fuel type, transmission and other relevant attributes.

-> Attributes:-

Brand, Model, year, Engine Size, fuel type, Transmission, mileage, Doors, Owner-count and price.

-> Number of rows: 10,000

Number of columns: 10

2- Laptop price dataset

2

problem statement:

predict laptop price based on specifications like brand, processor, RAM, and storage. Based on this specification laptop price will be predicted

Attributes:

Brand, processor, RAM(GB), Storage, GPU, Screen size, Resolution, Battery life, weight, operating system, price

Number of columns: 11

Number of rows: 11,769

3> Student Grading Dataset

3

~~predict stu~~

problem statement:

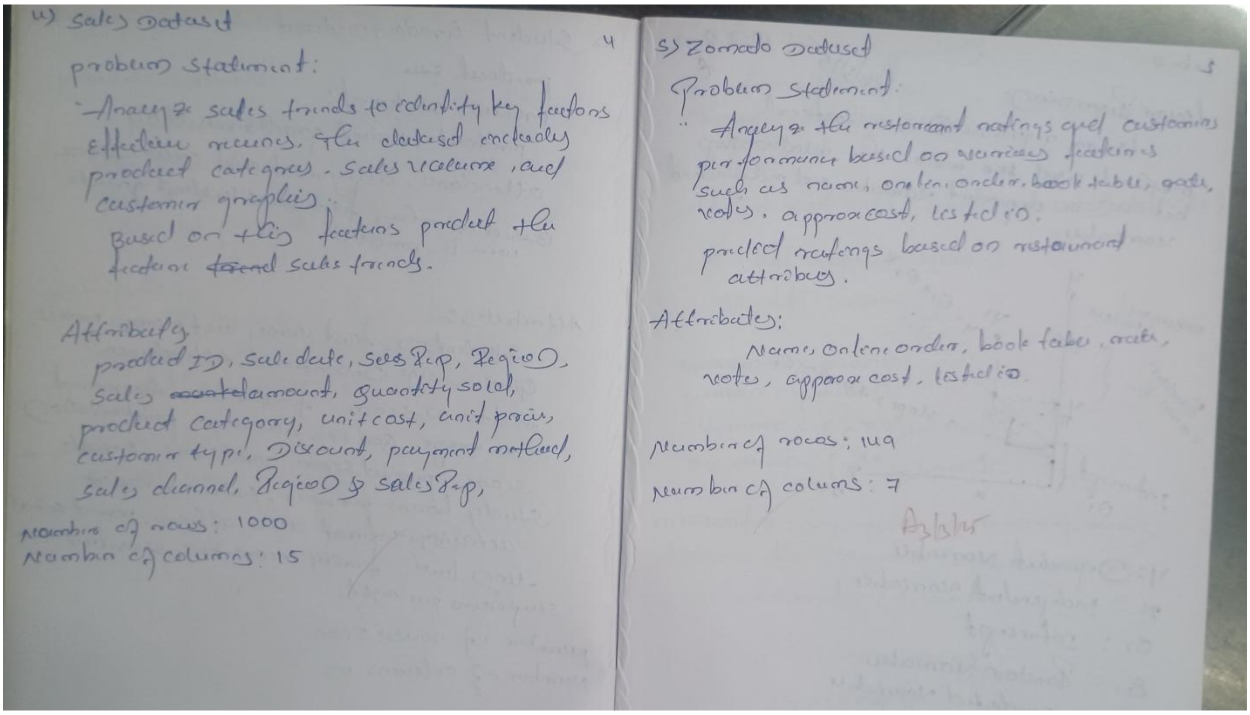
predict student grades based on attendance, assignment and exam score. Based on this features student grade will be predicted.

Attributes:-

StudentID, first Name, last Name, email, Gender, age, Department, Attendance, midterm score, final score, Assignment average, Quizzes average, participation score, project score, total score, Grade, Study hours per week, Extracurricular activity, Internet access at home, parent education level, family income level, stress level, sleep hours per night,

Number of rows: 5000

Number of columns: 23



4) Sales Dataset 4

Problem Statement:

- Analyze sales trends to identify key factors effecting revenue. The dataset includes product categories, sales volume, and customer graphics.

Based on this factors predict the future ~~trend~~ sales trends.

Attributes:

product ID, sale date, sales Rep, Region, sales ~~total~~ amount, quantity sold, product category, unit cost, unit price, customer type, Discount, payment method, sales channel, Region & sales Rep,

Number of rows: 1000

Number of columns: 15

5) Zomato Dataset 5

Problem Statement:

Analyze the restaurant ratings and customers performance based on various features such as name, online order, book table, rate, votes, approx cost, listed in.

predict ratings based on restaurant attributes.

Attributes:

Name, Online order, book table, rate, votes, approx cost, listed in

Number of rows: 149

Number of columns: 7

CODE WITH OUTPUT

Code:

```
import pandas as pd
# ----------- Step 1: Import Data from CSV -----------
# If you already have a CSV file, replace 'data.csv' with the filename.
# For this demo, let's create a DataFrame and save it to CSV first.
data = {
    'Name': ['Alice', 'Bob', 'Charlie'],
    'Age': [24, 30, 22],
    'City': ['New York', 'Los Angeles', 'Chicago']
}
# Create DataFrame
df = pd.DataFrame(data)
# Exporting DataFrame to CSV
df.to_csv('people.csv', index=False)
print("Data exported to 'people.csv'.")
# ----------- Step 2: Import Data from CSV -----------
imported_df = pd.read_csv('people.csv')
```

```
print("\nData imported from 'people.csv':")

print(imported_df)

# ----------- Step 3: Export Data to Excel -----------

df.to_excel('people.xlsx', index=False)

print("\nData exported to 'people.xlsx'.")

# ----------- Step 4: Import Data from Excel -----------

imported_excel_df = pd.read_excel('people.xlsx')

print("\nData imported from 'people.xlsx':")

print(imported_excel_df)
```

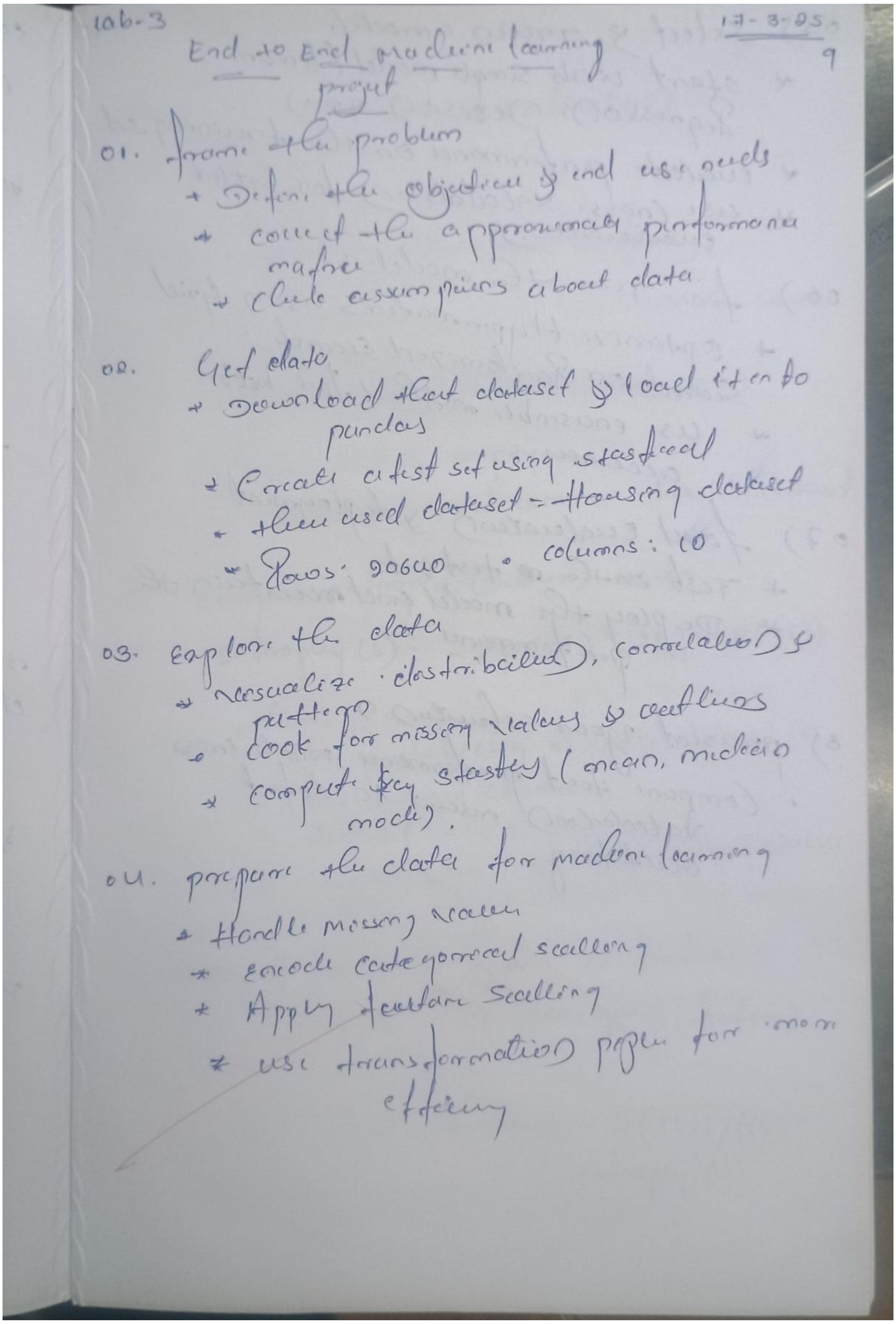
OUTPUT:

```
Data exported to 'people.csv'.

Data imported from 'people.csv':
      Name  Age         City
0    Alice   24     New York
1      Bob   30  Los Angeles
2  Charlie   22      Chicago

Data exported to 'people.xlsx'.

Data imported from 'people.xlsx':
      Name  Age         City
0    Alice   24     New York
1      Bob   30  Los Angeles
2  Charlie   22      Chicago
```

LABORATORY PROGRAM – 2

Demonstrate various data pre-processing techniques for a given dataset

OBSERVATION BOOK



lab-3 17-3-25

9

End to End machine learning project

01. Frame the problem
* Define the objective & end use needs
* collect the approximate performance matrix
* Check assumptions about data

02. Get data
* Download that dataset & load it into pandas
* Create a test set using stratified
* then used dataset - Housing dataset
* Rows: 20640 • columns: 10

03. Explore the data
* visualize distribution, correlations & pattern
* look for missing values & outliers
* compute key stats (mean, median mode).

04. prepare the data for machine learning
* Handle missing value
* encode categorical scaling
* Apply feature scaling
* use transformation pipeline for more efficiency

10

05) Select & Train a model.
* Start with simple model (Linear Regression, Decision Tree)
* evaluate performance on the training set
* use cross-validation for better evaluation

06) Fine-Tune the model
* Optimize Hyperparameters using Grid Search or Randomized Search.
* use ensemble method for better accuracy

07) Final evaluation & deployment
* Test on the test set
* Deploy the model and maintain the performance

8) present your solution?
* Compare test performance with cross validation result to detect overfitting.

CODE WITH OUTPUT

Code:

```
from google.colab import files
diabetes=files.upload()

from google.colab import files
adult_income=files.upload()

df1=pd.read_csv("Dataset of Diabetes .csv")
df1.head()

df2=pd.read_csv("adult.csv")
df2.head()

df1.info()
df2.info()
df1.describe()
df2.describe()

missing_values1 = df1.isnull().sum()
print(missing_values1)
missing_values2 = df2.isnull().sum()
print(missing_values2)

df1['Gender'] = df1['Gender'].replace('f', 'F')
ordinal_encoder = OrdinalEncoder(categories=[["F", M"]])
df1["Gender_Encoded"] =
ordinal_encoder.fit_transform(df1[["Gender"]]) onehot_encoder =
OneHotEncoder()
encoded_data =
onehot_encoder.fit_transform(df1[["CLASS"]]) encoded_array
= encoded_data.toarray()
encoded_df = pd.DataFrame(encoded_array,
```

```
columns=onehot_encoder.get_feature_names_out(["CLASS"])) df_encoded =
pd.concat([df1, encoded_df],  axis=1)

df1 = pd.concat([df1, encoded_df], axis=1)

df1.drop("CLASS", axis=1, inplace=True)

df1.drop("Gender", axis=1, inplace=True)

print(df2.head())

from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder

df_copy2 = df2

ordinal_encoder = OrdinalEncoder(categories=[["Male","Female"]])

df_copy2["Gender_Encoded"] =

ordinal_encoder.fit_transform(df_copy2[["gender"]])

print(df_copy2[["gender","Gender_Encoded"]])

onehot_encoder = OneHotEncoder()

encoded_data =

onehot_encoder.fit_transform(df2[["occupation","workclass
","education","marital status","relationship","race","n
ative-country","income"]])

encoded_array = encoded_data.toarray()

encoded_df =

pd.DataFrame(encoded_array,

columns=onehot_encoder.get_feature_names_out(["occupation","workclas
s","education","marital status","relatio nship","race","native-
country","income"]))

df_encoded = pd.concat([df_copy2, encoded_df], axis=1)

df_encoded.drop("gender", axis=1, inplace=True)

df_encoded.drop("occupation", axis=1, inplace=True)

df_encoded.drop("workclass", axis=1, inplace=True)

df_encoded.drop("education", axis=1, inplace=True)

df_encoded.drop("marital-status", axis=1, inplace=True)

df_encoded.drop("relationship", axis=1, inplace=True)

df_encoded.drop("race", axis=1, inplace=True)

df_encoded.drop("native-country", axis=1, nplace=True)

df_encoded.drop("income", axis=1, inplace=True)
```

```
print(df_encoded. head())

normalizer = MinMaxScaler()

df_encoded[["fnlwgt","educational-num","capital-gain","capital-loss","hours-per-
week"]] =  normalizer.fit_transform(df_encoded[["fnlwgt","educational-
num","capital-gain","capital-loss","hours-per week"]

])

df_encoded.head()

normalizer = MinMaxScaler()

df1[["No_Pation","AGE","Urea","Cr" , "HbA1c" ,
"Chol","TG","HDL","LDL","VLDL","BMI"]] =
normalizer.fit_transform(df1[["No_Pation","AGE","Urea","Cr" , "HbA1c" ,

"Chol","TG","HDL","LDL","VLDL","BMI"]])

df1.head()
```

```
import os

import tarfile

import urllib

import pandas as pd

DOWNLOAD_ROOT = "https://raw.githubusercontent.com/ageron/handson-ml2/master/"

HOUSING_PATH = os.path.join("datasets", "housing")

HOUSING_URL = DOWNLOAD_ROOT + "datasets/housing/housing.tgz"

def fetch_housing_data(housing_url=HOUSING_URL, housing_path=HOUSING_PATH):

   os.makedirs(housing_path, exist_ok=True)

   tgz_path = os.path.join(housing_path, "housing.tgz")

   urllib.request.urlretrieve(housing_url, tgz_path)

   housing_tgz = tarfile.open(tgz_path)

   housing_tgz.extractall(path=housing_path)

   housing_tgz.close()

fetch_housing_data()

def load_housing_data(housing_path=HOUSING_PATH):

   csv_path = os.path.join(housing_path, "housing.csv")

   return pd.read_csv(csv_path)
```

housing = load_housing_data()

housing.head()

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	NEAR BAY
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	NEAR BAY
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	NEAR BAY
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	NEAR BAY
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	NEAR BAY

housing.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  float64
 3   total_rooms         20640 non-null  float64
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  float64
 6   households          20640 non-null  float64
 7   median_income       20640 non-null  float64
 8   median_house_value  20640 non-null  float64
 9   ocean_proximity     20640 non-null  object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

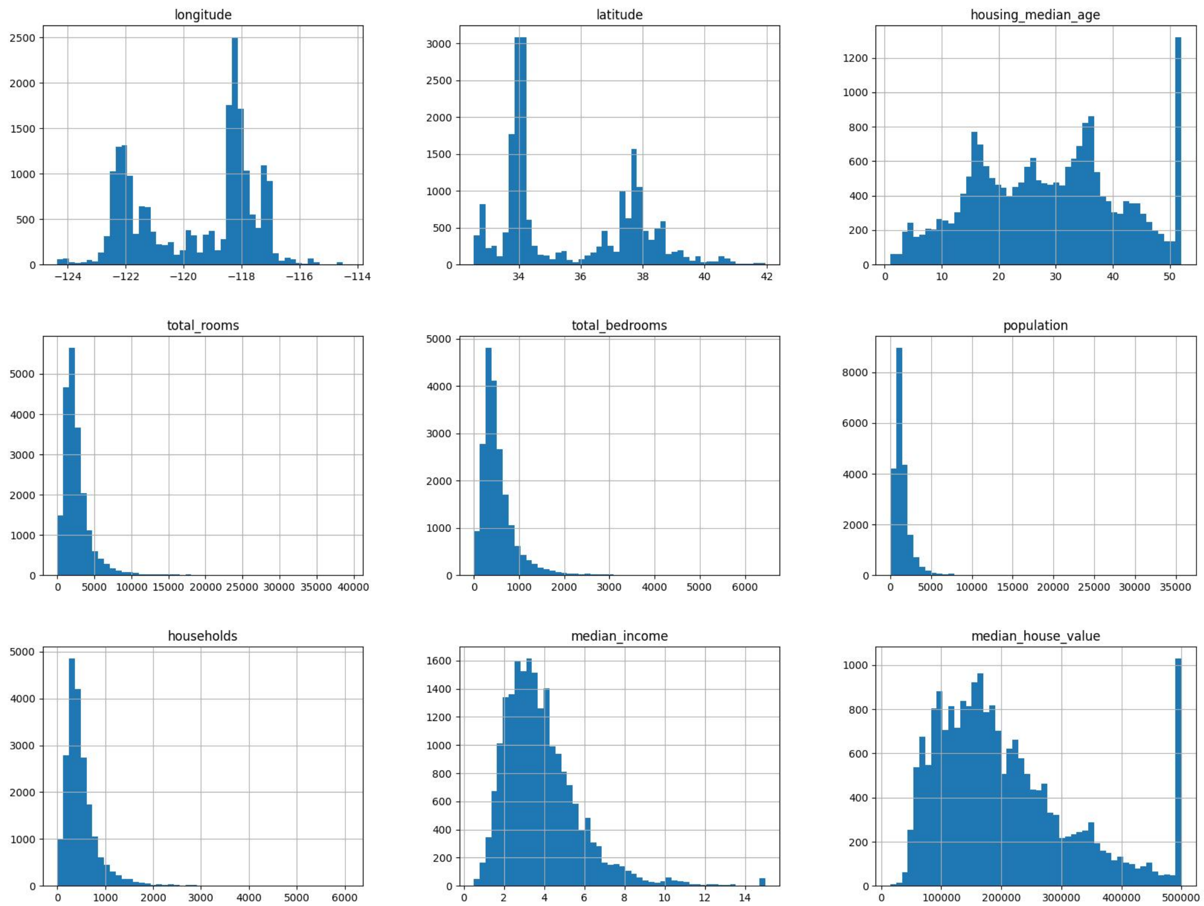
housing['ocean_proximity'].value_counts()

ocean_proximity	count
<1H OCEAN	9136
INLAND	6551
NEAR OCEAN	2658
NEAR BAY	2290
ISLAND	5

dtype: int64

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000	20640.000000	20640.000000	20640.000000
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744	499.539680	3.870671	206855.816909
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.462122	382.329753	1.899822	115395.615874
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	1.000000	0.499900	14999.000000
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.000000	280.000000	2.563400	119600.000000
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000	409.000000	3.534800	179700.000000
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.000000	605.000000	4.743250	264725.000000
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000	6082.000000	15.000100	500001.000000

```
%matplotlib inline

import matplotlib.pyplot as plt

housing.hist(bins=50, figsize=(20,15))

plt.show()
```

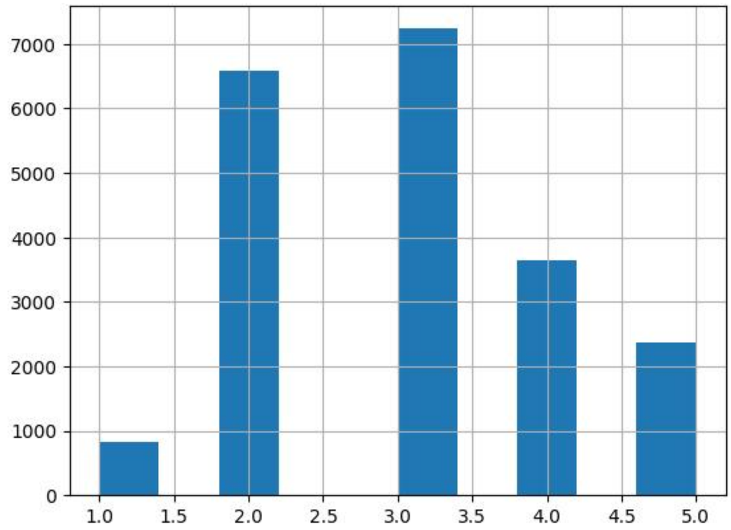



```
import numpy as np

# def split_train_test(data, test_ratio):

#   np.random.seed(42)

#   shuffled_indices = np.random.permutation(len(data))

#   test_set_size = int(len(data) * test_ratio)

#   test_indices = shuffled_indices[:test_set_size]

#   train_indices = shuffled_indices[test_set_size:]

#   return data.iloc[train_indices], data.iloc[test_indices]
```

```
from sklearn.model_selection import train_test_split, StratifiedShuffleSplit


train_set, test_set = train_test_split(housing, test_size=0.2, random_state=42)
len(test_set)
```

```
4128
```

```
housing['income_cat'] = pd.cut(housing['median_income'],
                               bins=[0., 1.5, 3.0, 4.5, 6., np.inf],
                               labels=[1, 2, 3, 4, 5])
housing['income_cat'].hist()
```

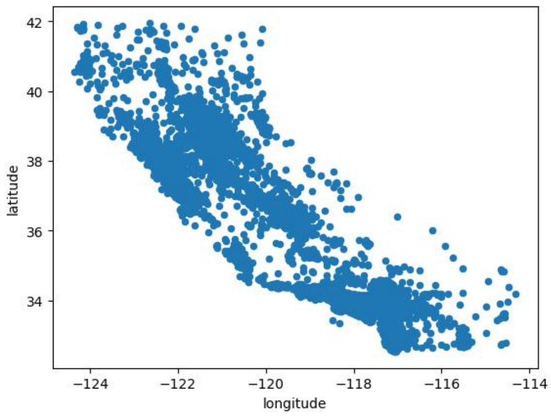



```
split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
for train_index, test_index in split.split(housing, housing['income_cat']):
   strat_train_set = housing.loc[train_index]
   strat_test_set = housing.loc[test_index]


strat_test_set['income_cat'].value_counts() / len(strat_test_set)
```

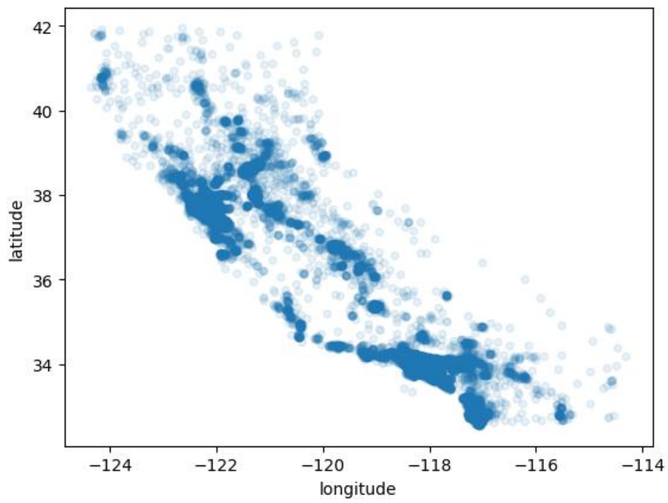
income_cat	count
3	0.350533
2	0.318798
4	0.176357
5	0.114341
1	0.039971

dtype: float64

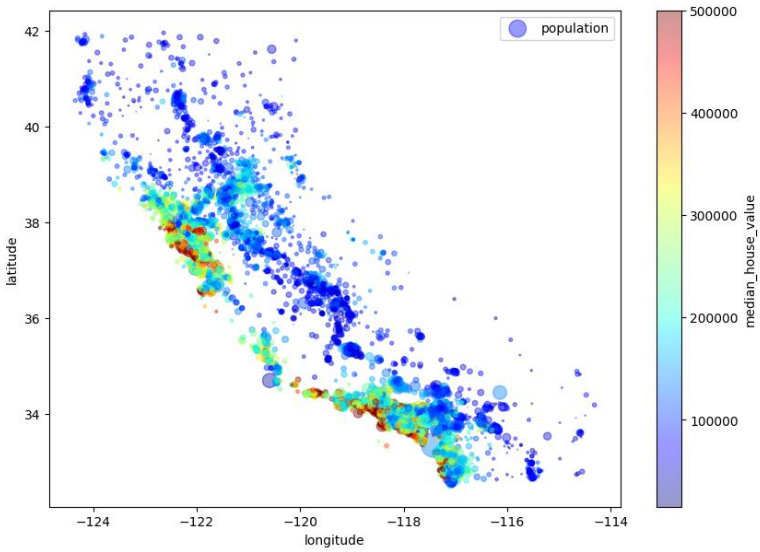
```
for set_ in (strat_train_set, strat_test_set):
    set_.drop('income_cat', axis=1, inplace=True)
housing = strat_train_set.copy()
housing.plot(kind='scatter', x='longitude', y='latitude')
```




```
housing.plot(kind='scatter', x='longitude', y='latitude', alpha=0.1)
```




```
housing.plot(kind='scatter', x='longitude', y='latitude', alpha=0.4,
         s=housing['population']/100, label='population', figsize=(10, 7),
         c='median_house_value', cmap=plt.get_cmap('jet'), colorbar=True)
plt.legend()
```



Select only numerical features for correlation calculation

numerical_features = housing.select_dtypes(include=['number']) # Exclude non-numeric columns

corr_matrix = numerical_features.corr() # Calculate correlation for numerical features only

corr_matrix['median_house_value'].sort_values(ascending=False)

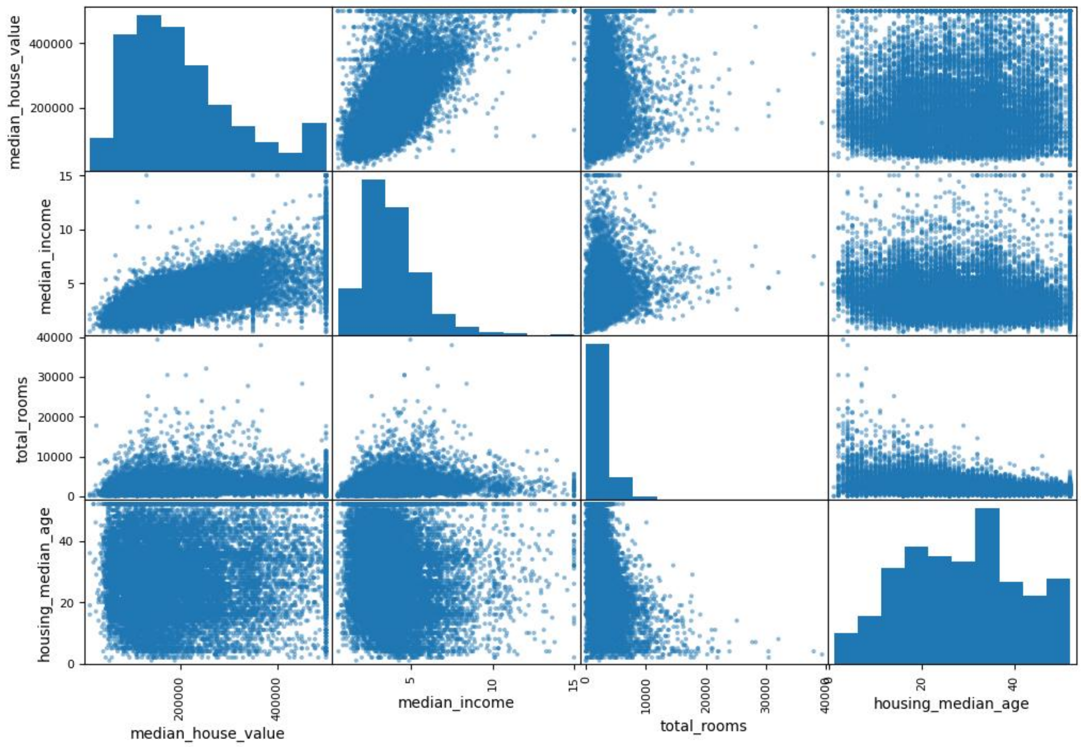
	median_house_value
median_house_value	1.000000
median_income	0.687151
total_rooms	0.135140
housing_median_age	0.114146
households	0.064590
total_bedrooms	0.047781
population	-0.026882
longitude	-0.047466
latitude	-0.142673

dtype: float64

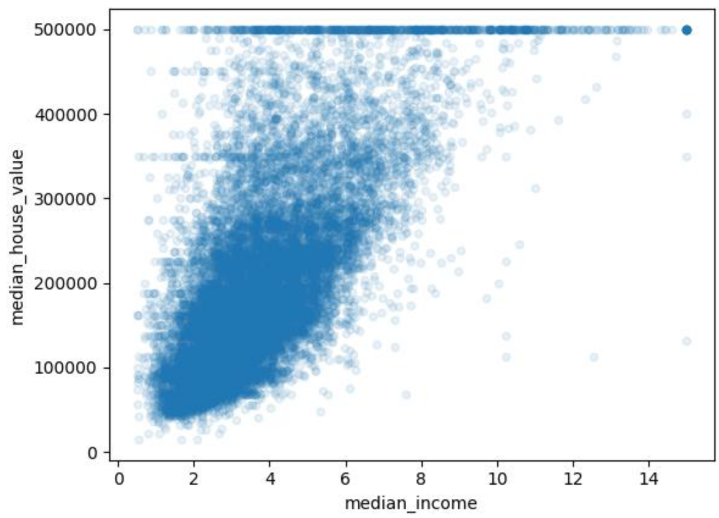
from pandas.plotting import scatter_matrix

attributes = ['median_house_value', 'median_income', 'total_rooms', 'housing_median_age']

scatter_matrix(housing[attributes], figsize=(12, 8))



housing.plot(kind='scatter', x='median_income', y='median_house_value', alpha=0.1)



housing['rooms_per_household'] = housing['total_rooms']/housing['households']

housing['bedrooms_per_room'] = housing['total_bedrooms']/housing['total_rooms']

housing['population_per_household'] = housing['population']/housing['households']

Select only numerical features for correlation calculation

numerical_features = housing.select_dtypes(include=['number']) # Exclude non-numeric columns

corr_matrix = numerical_features.corr() # Calculate correlation for numerical features only

corr_matrix['median_house_value'].sort_values(ascending=False)

	median_house_value
median_house_value	1.000000
median_income	0.687151
rooms_per_household	0.146255
total_rooms	0.135140
housing_median_age	0.114146
households	0.064590
total_bedrooms	0.047781
population_per_household	-0.021991
population	-0.026882
longitude	-0.047466
latitude	-0.142673
bedrooms_per_room	-0.259952

dtype: float64

LABORATORY PROGRAM – 3

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

OBSERVATION BOOK

lab-4

Decision Tree - ID3 classification

Algorithm:

ID3(Dataset, Target-column, features):

1. If all instance in Dataset belong to the same class:

   Return that class (as a leaf node)

2. If feature list is empty

   Return the most common class in dataset

3. Compute entrophy(dataset) using.

   $$entrophy(S) = -\Sigma p(c_i) \times \log_2(p(c_i))$$

   where,

   $p(c_i)$ = probability of each class c_i in the target column

   S = dataset

   Σ = summation over all unique classes in the target column

4. ~~Compute~~ for each feature in features

   - compute information gain (feature) using:

   $$IG(feature) = Entrophy - \Sigma(|S_v|/|S|) \times entrophy(S)$$

5. select Best-feature with the highest information gain

6. Create a new decision node for best-feature

7. for each unique value in best-feature
   - create a branch for value
   - split dataset where best-feature = value

   else:
   - Recursively call ID3()

8. Return the constructed decision Tree.

Dataset (simple weather forecast)

Attributes

- outlook
- Temperature
- Humidity
- wind
- play

Attributes / features.

-> outlook ∈ [sunny, overcast, Rainy]

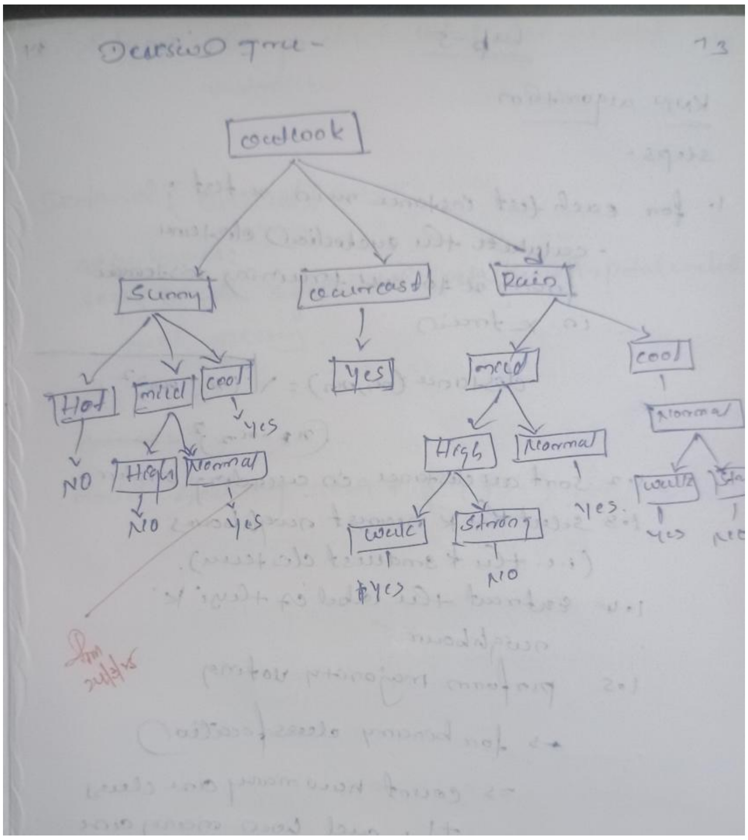
-> Temperature ∈ [Hot, mild, cool]

-> Humidity ∈ [High, Normal]

-> windy ∈ [weak, strong]

output/target variable

-> play ∈ [yes, No]

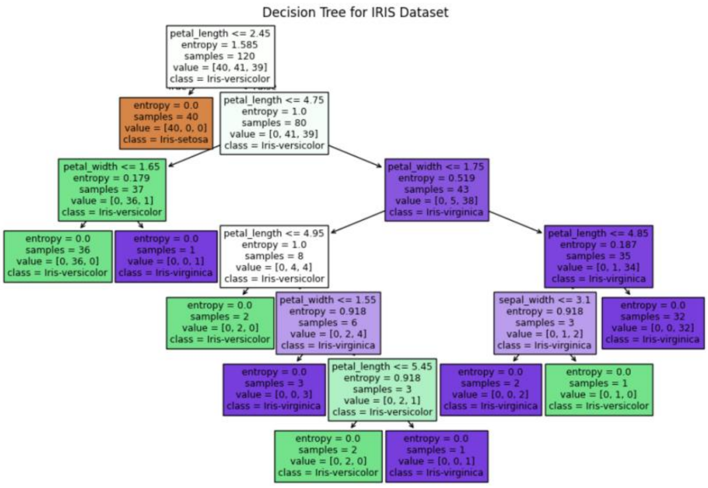


CODE WITH OUTPUT

```
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

# Load the iris dataset (make sure iris.csv is in the working directory)
iris = pd.read_csv("iris.csv")
# Assuming the last column is the target (species) and the rest are features.
X = iris.iloc[:, :-1]
y = iris.iloc[:, -1]

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

# Initialize and train the Decision Tree classifier
clf_iris = DecisionTreeClassifier(criterion='entropy', random_state=42)
clf_iris.fit(X_train, y_train)

# Make predictions and evaluate the model
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)

print("IRIS Dataset Decision Tree Classifier")
print("Accuracy:", accuracy_iris)
print("Confusion Matrix:\n", conf_matrix_iris)
print("Classification Report:\n", classification_report(y_test, y_pred_iris))

# Visualize the decision tree
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()
```

```
IRIS Dataset Decision Tree Classifier
Accuracy: 1.0
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
```

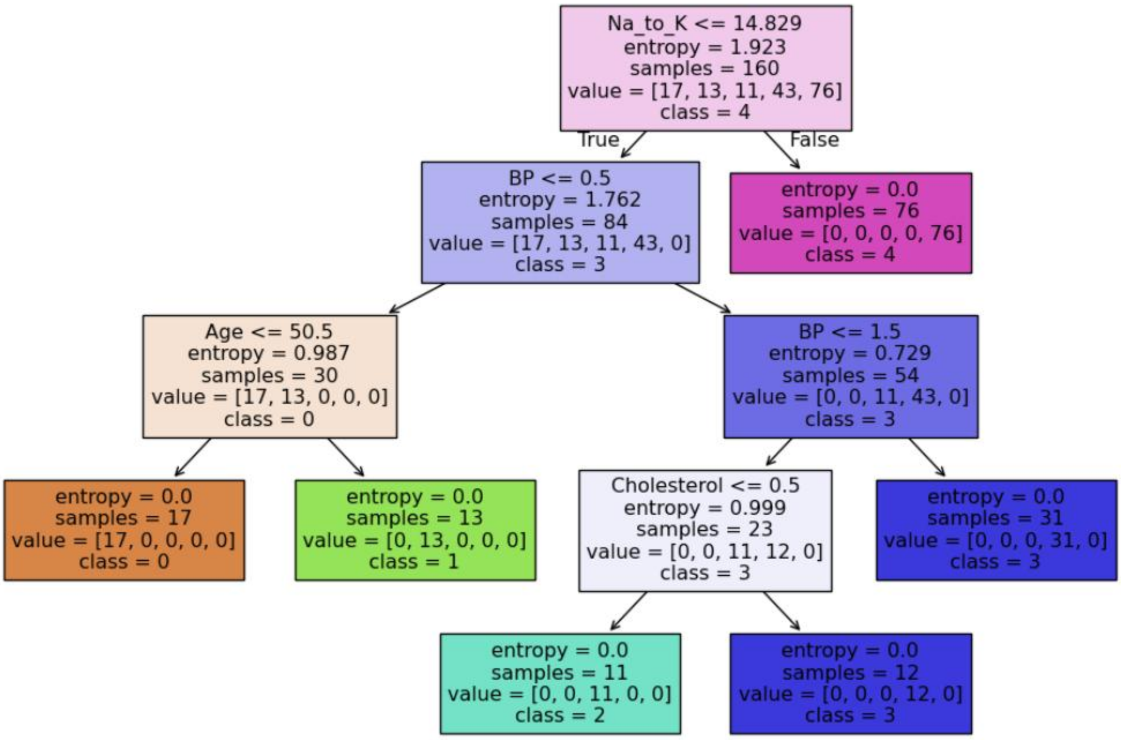



```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree

# Load the drug dataset (make sure drug.csv is in the working directory)
drug = pd.read_csv("drug.csv")

# Since the target column is 'Drug', drop it from the features
X_drug = drug.drop('Drug', axis=1)
y_drug = drug['Drug']

# If there are categorical features, perform necessary encoding
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
# Encode features that are categorical
for col in X_drug.select_dtypes(include='object').columns:
    X_drug[col] = le.fit_transform(X_drug[col])
# Also encode the target variable if necessary
y_drug = le.fit_transform(y_drug)

# Split the data (80% training, 20% testing)
X_train_d, X_test_d, y_train_d, y_test_d = train_test_split(X_drug, y_drug, test_size=0.2, random_state=42)

# Initialize and train the Decision Tree classifier using entropy criterion
clf_drug = DecisionTreeClassifier(criterion='entropy', random_state=42)
clf_drug.fit(X_train_d, y_train_d)

# Make predictions and evaluate the model
y_pred_drug = clf_drug.predict(X_test_d)
accuracy_drug = accuracy_score(y_test_d, y_pred_drug)
conf_matrix_drug = confusion_matrix(y_test_d, y_pred_drug)
```

```python
print("Drug Dataset Decision Tree Classifier")
print("Accuracy:", accuracy_drug)
print("Confusion Matrix:\n", conf_matrix_drug)
print("Classification Report:\n", classification_report(y_test_d, y_pred_drug))

# Visualize the decision tree
plt.figure(figsize=(12, 8))
plot_tree(clf_drug, filled=True, feature_names=X_drug.columns,
          class_names=[str(cls) for cls in clf_drug.classes_])
plt.title("Decision Tree for Drug Dataset")
plt.show()
```

```
Drug Dataset Decision Tree Classifier
Accuracy: 1.0
Confusion Matrix:
 [[ 6  0  0  0  0]
 [ 0  3  0  0  0]
 [ 0  0  5  0  0]
 [ 0  0  0 11  0]
 [ 0  0  0  0 15]]
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00         6
           1       1.00      1.00      1.00         3
           2       1.00      1.00      1.00         5
           3       1.00      1.00      1.00        11
           4       1.00      1.00      1.00        15

    accuracy                           1.00        40
   macro avg       1.00      1.00      1.00        40
weighted avg       1.00      1.00      1.00        40
```

Decision Tree for Drug Dataset



LABORATORY PROGRAM – 4

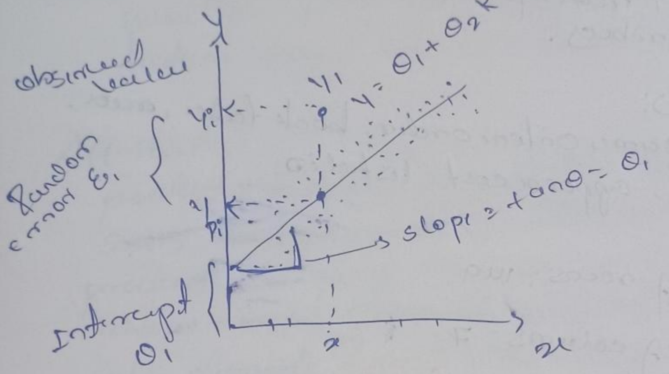
Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

OBSERVATION BOOK

lab-2 10-3-25

Linear Regression:

Linear regression is a statistical method used for model the relationship between a dependent variable and independent variable.



y = Dependent variable

x = Independent variable

θ_1 = Intercept

ε_i = Random variable

p_i = predicted variable

y_i = observed value.

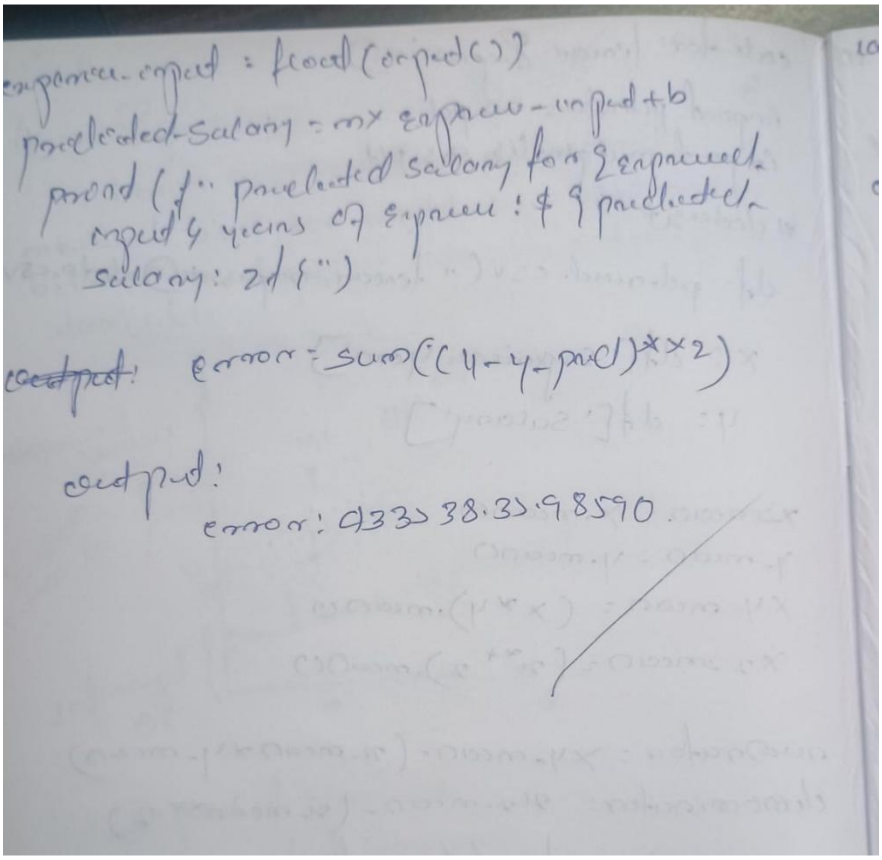
$\hat{y} = \theta_1 + \theta_2 x + \varepsilon$ → error

target variable

predicted

code for linear Regression

```
import pandas as pd
import matplotlib as plt

# dataset
df = pd.read_csv("linear-regression-data.csv")

x = df['Experience years']
y = df['salary']

x_mean = x.mean()
y_mean = y.mean()
xy_mean = (x*y).mean()
x2_mean = (x**2).mean()()

numerator = xy_mean - (x_mean * y_mean)
denominator = x2_mean - (x_mean** 2)
m = numerator / denomator

b = y_mean - (m * x_mean)
y_pred = m * x * b

plt.scatter(x, y, color, blue, label = 'Actual')
plt.plot(x, y_pred, color = red', label = 'Fitted line')
plt.legend()
plt.show()
plt.close().

print(f"Equation of line : y = {m:.2f}x +
{b:.2f}")
```



CODE WITH OUTPUT

import pandas as pd

import matplotlib.pyplot as plt

import numpy as np

df = pd.read_csv("/content/sample_data/linear_regression_dataset.csv")

df1 = pd.read_csv("/content/sample_data/data_for_lr.csv")

X = df[['Experience (Years)']].values.flatten()

y = df['Salary ($1000s)'].values

X = df1['x'].values.flatten()

y = df1['y'].values.flatten()

X, y

```
(array([ 7,  4, 13, 15, 11,  8, 13,  5,  7, 10,  3,  7, 11, 11,  8,  5,  4,
         8,  8,  3,  6,  5,  2,  8, 12, 14,  6,  2, 12,  5]),
 array([20184, 10459, 40385, 43021, 33300, 22747, 39904, 16632, 19474,
        29082,  9558, 22753, 33047, 34547, 24747, 13975, 11806, 22189,
        25005,  9734, 19005, 13562,  5899, 25638, 35267, 42879, 17528,
         7202, 37556, 16890]))
```

mean_x = np.mean(X)

mean_y = np.mean(y)

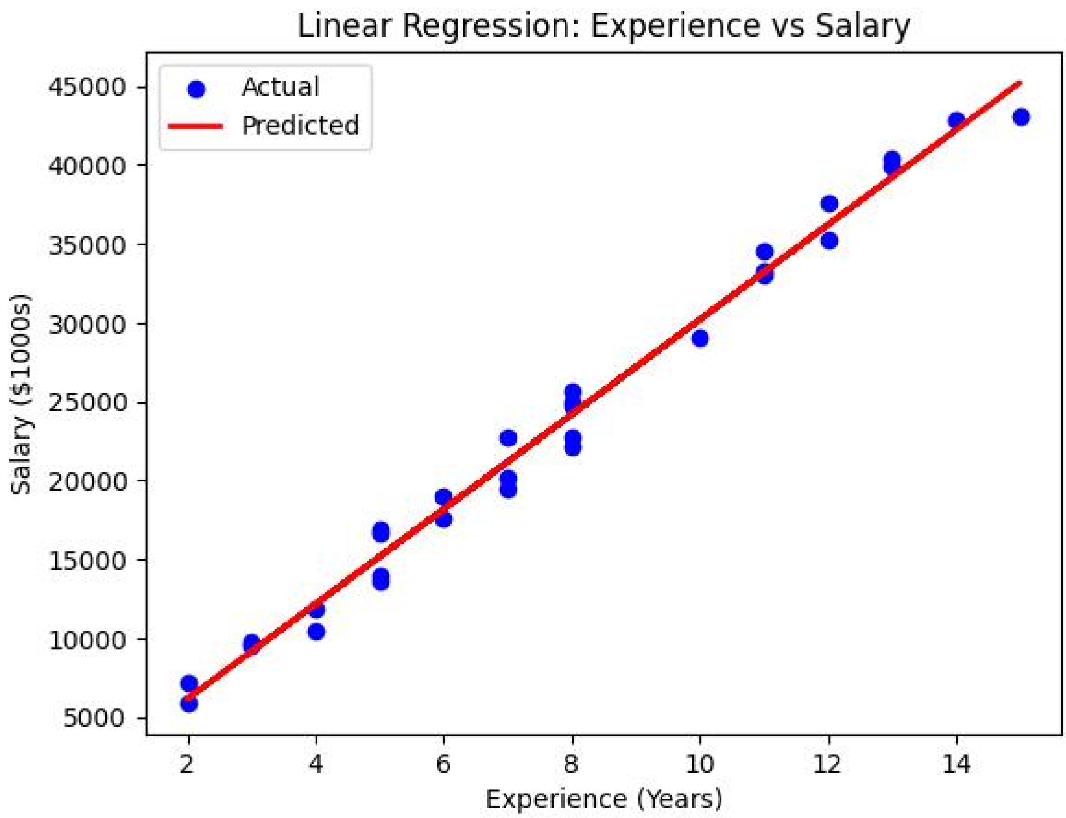
```
numerator = np.mean(X * y) - mean_x * mean_y
denominator = np.mean(X**2) - mean_x**2

beta1 = numerator / denominator
beta0 = mean_y - beta1 * mean_x

y_pred = beta0 + beta1 * X

error = np.sum((y - y_pred) ** 2)
mean_x, mean_y
```

```
(7.766666666666667, 23465.833333333332)
```

```
print(f"Intercept (beta0): {beta0:.2f}")
print(f"Slope (beta1): {beta1:.2f}")
print(f"Error: {error:.2f}")
```

```
Intercept (beta0): 138.77
Slope (beta1): 3003.49
Error: 43353835.99
```

```
plt.scatter(X, y, color='blue', label='Actual')
plt.plot(X, y_pred, color='red', linewidth=2, label='Predicted')
plt.xlabel('Experience (Years)')
plt.ylabel('Salary ($1000s)')
plt.title('Linear Regression: Experience vs Salary')
plt.legend()
plt.show()
```



result = f"Linear Equation: Y = {beta0:.2f} + {beta1:.2f}X + {error:.2f}"

result

```
'Linear Equation: Y = 138.77 + 3003.49X + 43353835.99'
```

LABORATORY PROGRAM – 5

Build Logistic Regression Model for a given dataset

OBSERVATION BOOK

Code:

```
from google.colab import files

hr=files.upload()

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

from sklearn import linear_model

import seaborn as sns

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score

df1=pd.read_csv("HR_comma_sep.csv")

df1.head()

df1.isnull().sum()

plt.figure(figsize=(12, 6))

sns.barplot(x='Department', y='left', data=df1)

plt.title('Employee Retention Rate by Department')

plt.xlabel('Department')

plt.ylabel('Proportion of Employees Left')
```

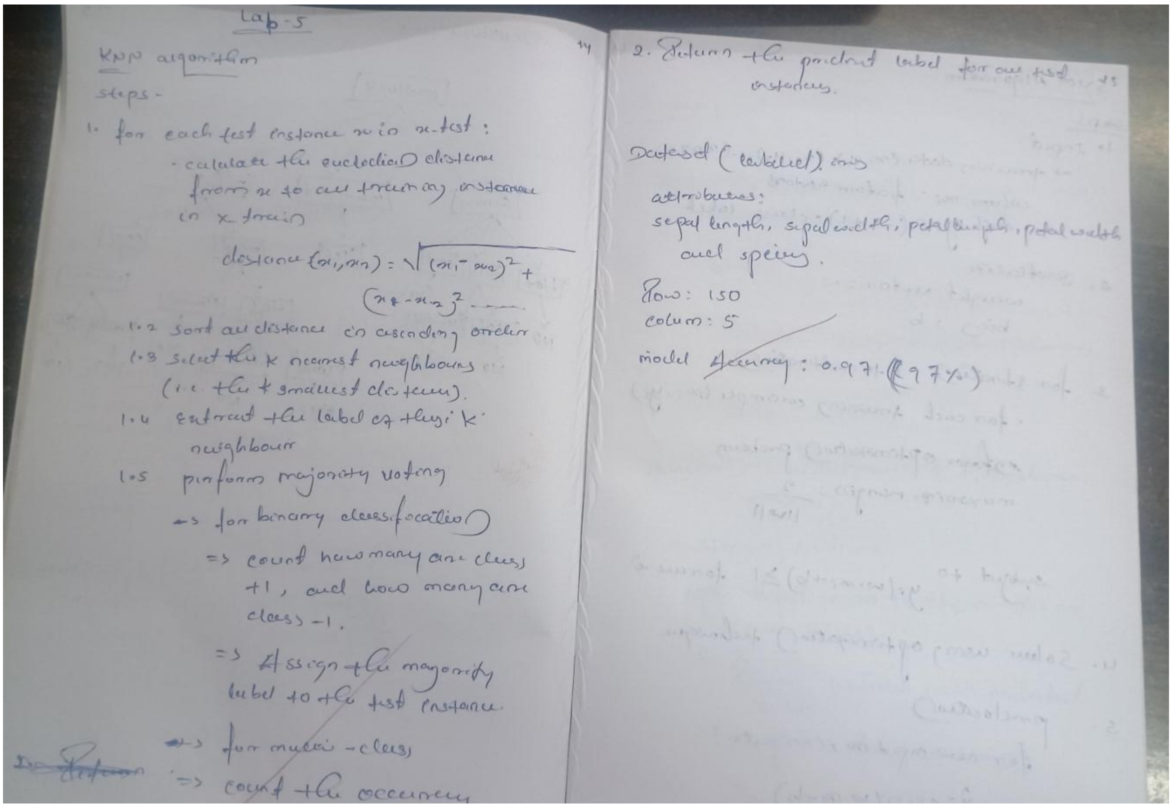
```
plt.xticks(rotation=45, ha='right')

plt.show()

ohe = OneHotEncoder(handle_unknown='ignore',
sparse_output=False)  department_encoded =
ohe.fit_transform(df1[['Department']])

department_encoded_df = pd.DataFrame(department_encoded,

columns=ohe.get_feature_names_out(['Department']))

df1 = pd.concat([df1, department_encoded_df], axis=1)

df1 = df1.drop('Department', axis=1)

ordinal_encoder = OrdinalEncoder(categories=[['low', 'medium',
'high']], dtype=np.int64)  salary_encoded =
ordinal_encoder.fit_transform(df1[['salary']])

df1['salary_encoded'] = salary_encoded

df1 = df1.drop('salary', axis=1)

df1.head()

correlation_matrix = df1.corr()

plt.figure(figsize=(12, 10))

sns.heatmap(correlation_matrix, annot=True,
cmap='coolwarm', fmt=".2f")  plt.title('Correlation
Matrix of Features')

plt.show()

plt.figure(figsize=(8, 6))

sns.barplot(x='salary_encoded', y='left', data=df1)

plt.title('Impact of Employee Salary on Retention')

plt.xlabel('Salary Level (Encoded)')

plt.ylabel('Proportion of Employees Left')

plt.show()

df_copy = df1[['number_project', 'average_montly_hours',
'time_spend_company', 'left','salary_encoded',
'satisfaction_level','Work_accident']]

df_copy.head()

X = df_copy.drop('left', axis=1)

y = df_copy['left']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)  model =
LogisticRegression(max_iter=1000)

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

print(f"Accuracy of the Logistic Regression model: {accuracy}")

from google.colab import files

zoodata=files.upload()

zootype=files.upload()

zoo_data = pd.read_csv('zoo-data.csv')

zoo_class = pd.read_csv('zoo-class-type.csv')

merged_data = pd.merge(zoo_data, zoo_class, left_on='class_type',
right_on='Class_Number') merged_data =
merged_data.drop(['Animal_Names',
'Number_Of_Animal_Species_In_Class',
'Class_Number','class_type','animal_name'], axis=1)

X = merged_data.drop('Class_Type', axis=1)

y = merged_data['Class_Type']

print(merged_data.head())

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)  model =
LogisticRegression(max_iter=1000)

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy}")

cm = confusion_matrix(y_test, y_pred)

disp = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=np.unique(y_test))  disp.plot(cmap="Blues",
values_format="d")

plt.title("Confusion Matrix")

plt.show()
```

LABORATORY PROGRAM – 6

Build KNN Classification model for a given dataset.

OBSERVATION BOOK



CODE WITH OUTPUT

Code:

```
import csv
import math
import random
from collections import Counter

# ---------- KNN Implementation ----------

def load_dataset(filename):
    with open(filename, 'r') as f:
        data = list(csv.reader(f))
```

```
    header = data[0]
    rows = data[1:]
    for row in rows:
        for i in range(4):
            row[i] = float(row[i])
    return rows

def compute_mean_std(dataset):
    means, stds = [], []
    for i in range(4):
        col = [row[i] for row in dataset]
        mean = sum(col) / len(col)
        std = (sum((x - mean) ** 2 for x in col) / len(col)) ** 0.5
        means.append(mean)
        stds.append(std)
    return means, stds

def normalize_dataset(dataset, means, stds):
    for row in dataset:
        for i in range(4):
            row[i] = (row[i] - means[i]) / stds[i]
    return dataset

def split_dataset(dataset, test_size=0.2):
    random.shuffle(dataset)
    split_index = int(len(dataset) * (1 - test_size))
    return dataset[:split_index], dataset[split_index:]

def euclidean_distance(row1, row2):
    return math.sqrt(sum((row1[i] - row2[i]) ** 2 for i in range(4)))
```

```
def knn_predict(train, test_row, k):
    distances = []
    for train_row in train:
        dist = euclidean_distance(test_row, train_row)
        distances.append((train_row, dist))
    distances.sort(key=lambda x: x[1])
    neighbors = distances[:k]
    labels = [neighbor[0][4] for neighbor in neighbors]
    prediction = Counter(labels).most_common(1)[0][0]
    return prediction

def evaluate_model(train, test, k):
    correct = 0
    predictions = []
    for row in test:
        prediction = knn_predict(train, row, k)
        predictions.append(prediction)
        if prediction == row[4]:
            correct += 1
    accuracy = correct / len(test)
    return accuracy, predictions

filename = '/content/sample_data/IRIS.csv'
dataset = load_dataset(filename)

means, stds = compute_mean_std(dataset)

dataset = normalize_dataset(dataset, means, stds)

train_data, test_data = split_dataset(dataset)
```

```
k = 3
accuracy, predictions = evaluate_model(train_data, test_data, k)


print(f"Model Accuracy: {accuracy:.2f}\n\n")


print("\n--- Predict Iris Species from Your Input ---")
try:
    user_input = []
    user_input.append(float(input("Enter Sepal Length (cm): ")))
    user_input.append(float(input("Enter Sepal Width (cm): ")))
    user_input.append(float(input("Enter Petal Length (cm): ")))
    user_input.append(float(input("Enter Petal Width (cm): ")))


    for i in range(4):
        user_input[i] = (user_input[i] - means[i]) / stds[i]


    predicted_species = knn_predict(train_data, user_input, k)
    print(f"\n Predicted Iris Species: {predicted_species}")


except ValueError:
    print("Invalid input. Please enter numeric values for all measurements.")
```

```
Model Accuracy: 0.97



--- Predict Iris Species from Your Input ---
Enter Sepal Length (cm): 5.1
Enter Sepal Width (cm): 3.5
Enter Petal Length (cm): 1.4
Enter Petal Width (cm): 0.3

 Predicted Iris Species: Iris-setosa
```

LABORATORY PROGRAM – 7

Build Support vector machine model for a given dataset

OBSERVATION BOOK

SVM Algorithm

Steps

1. Input
   → training data $(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)$
   column x_i = feature vectors
   and $y_i \in \{-1, +1\}$ = class label

2. Initialize
   weight vector = w
   bias = b

3. for iteration = 1 to n-iteration
   · for each training example (x_i, y_i)
   define optimization problem
   maximize margin = $\frac{2}{\|w\|}$

   subject to $y_i(w \cdot x_i + b) \geq 1$ for all i

4. Solve using optimization technique

5. prediction
   for new input x, compute
   $\hat{y} = \text{sign}(w \cdot x + b)$

$$\boxed{f(x) = \text{sign}\left(\sum_i \alpha_i y_i x_i \cdot x + b\right)}$$

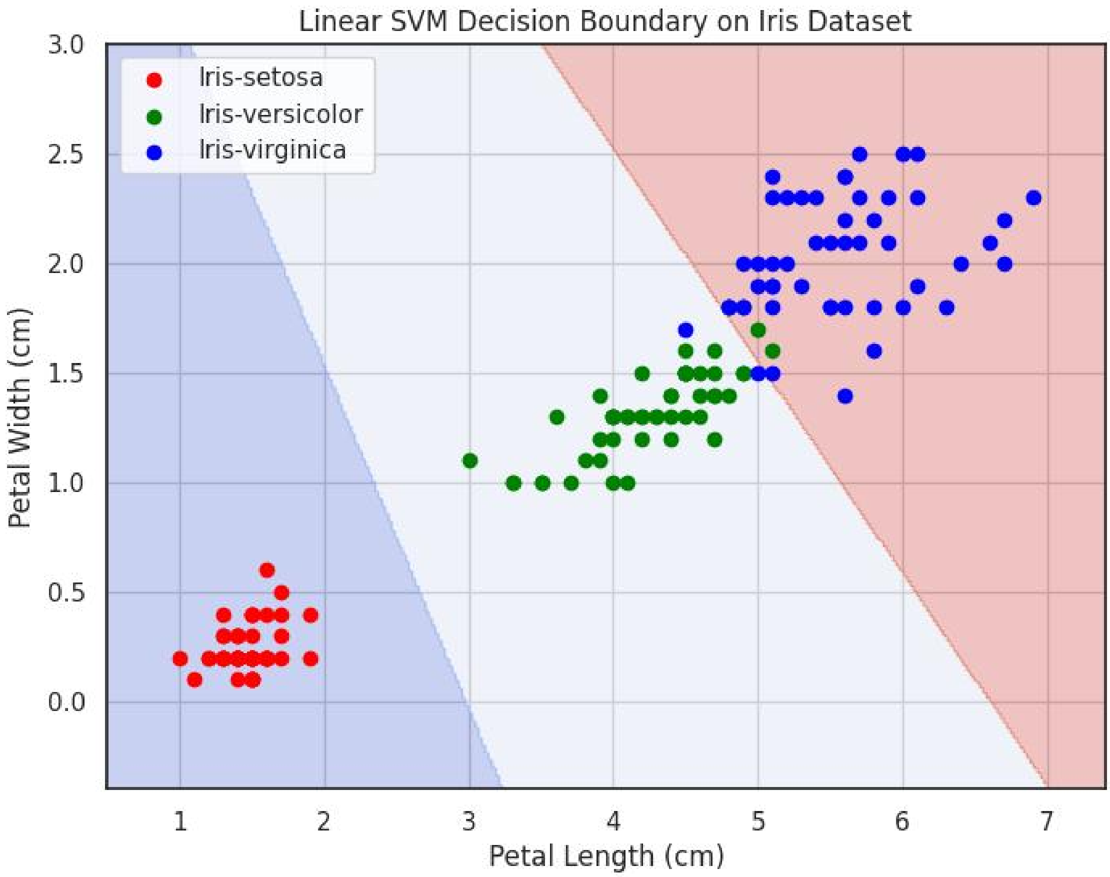
7/11/24

CODE WITH OUTPUT

Code:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
def PlotData(x,y):
    pos = np.argwhere(y == 1)
    neg = np.argwhere(y == 0)

    plt.plot(x[pos, 0], x[pos, 1], linestyle='', marker='+', color='k')
    plt.plot(x[neg, 0], x[neg, 1], linestyle='', marker='o', color='y')
    plt.xlabel('Exam 1 score')
    plt.ylabel('Exam 2 score')
    plt.legend(['Admitted', 'Not admitted'], loc='upper right', numpoints=1)
    plt.figure()
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.svm import SVC

df = pd.read_csv("/content/sample_data/IRIS.csv")

X = df[["petal_length", "petal_width"]].values
y = df["species"].values

label_map = {"Iris-setosa": 0, "Iris-versicolor": 1, "Iris-virginica": 2}
label_map_rev = {v: k for k, v in label_map.items()}
y_numeric = np.array([label_map[label] for label in y])

svm_model = SVC(kernel="linear")
```

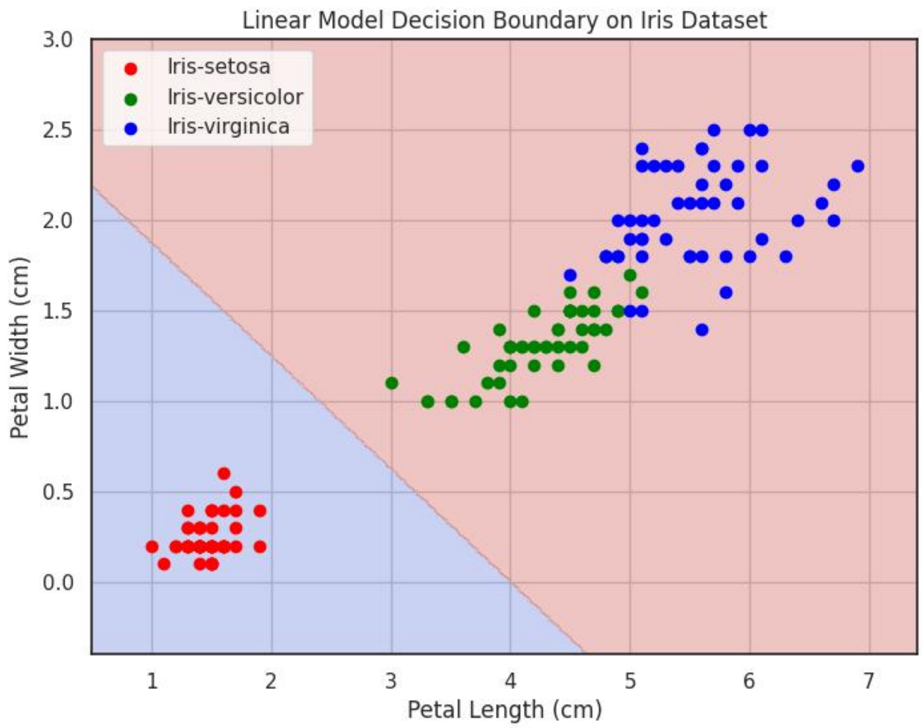
```python
svm_model.fit(X, y_numeric)

x_min, x_max = X[:, 0].min() - 0.5, X[:, 0].max() + 0.5
y_min, y_max = X[:, 1].min() - 0.5, X[:, 1].max() + 0.5
xx, yy = np.meshgrid(np.linspace(x_min, x_max, 300),
                     np.linspace(y_min, y_max, 300))

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

colors = ["red", "green", "blue"]

plt.figure(figsize=(8, 6))
plt.contourf(xx, yy, Z, alpha=0.3, cmap=plt.cm.coolwarm)

for label in np.unique(y_numeric):
    plt.scatter(X[y_numeric == label, 0], X[y_numeric == label, 1],
                label=label_map_rev[label], color=colors[label])

plt.xlabel("Petal Length (cm)")
plt.ylabel("Petal Width (cm)")
plt.title("Linear SVM Decision Boundary on Iris Dataset")
plt.legend()
plt.grid(True)
plt.show()
```


Linear SVM Decision Boundary on Iris Dataset
Iris-setosa
Iris-versicolor
Iris-virginica
Petal Width (cm)
Petal Length (cm)
3.0
2.5
2.0
1.5
1.0
0.5
0.0
1
2
3
4
5
6
7

```
import csv
import math
import random
from collections import Counter
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

def load_dataset(filename):
    df = pd.read_csv("/content/sample_data/IRIS.csv")
    X  = df[["petal_length", "petal_width"]].values
    y = df["species"].values

    label_map = {"Iris-setosa": 0, "Iris-versicolor": 1, "Iris-virginica": 2}
    label_map_rev = {v: k for k, v in label_map.items()}
    y_numeric = np.array([label_map[label] for label in y])

class SimpleLinearModel:
    def __init__(self, learning_rate=0.01, n_iters=1000):
        self.lr = learning_rate
        self.n_iters = n_iters
        self.activation = self._unit_step_func
        self.weights = None
        self.bias = None

    def fit(self, X, y):
        n_samples, n_features = X.shape
        self.weights = np.zeros(n_features)
        self.bias = 0

        y_ = np.array([1 if i > 0 else 0 for i in y])
```

```
        for _ in range(self.n_iters):
            for idx, x_i in enumerate(X):
                linear_output = np.dot(x_i, self.weights) + self.bias
                y_predicted = self.activation(linear_output)

                update = self.lr * (y_[idx] - y_predicted)
                self.weights += update * x_i
                self.bias += update

    def predict(self, X):
        linear_output = np.dot(X, self.weights) + self.bias
        y_predicted = self.activation(linear_output)
        return y_predicted

    def _unit_step_func(self, x):
        return np.where(x>=0, 1, 0)

svm_model = SimpleLinearModel()
svm_model.fit(X, y_numeric)

x_min, x_max = X[:, 0].min() - 0.5, X[:, 0].max() + 0.5
y_min, y_max = X[:, 1].min() - 0.5, X[:, 1].max() + 0.5
xx, yy = np.meshgrid(np.linspace(x_min, x_max, 300),
                     np.linspace(y_min, y_max, 300))

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

colors = ["red", "green", "blue"]
```

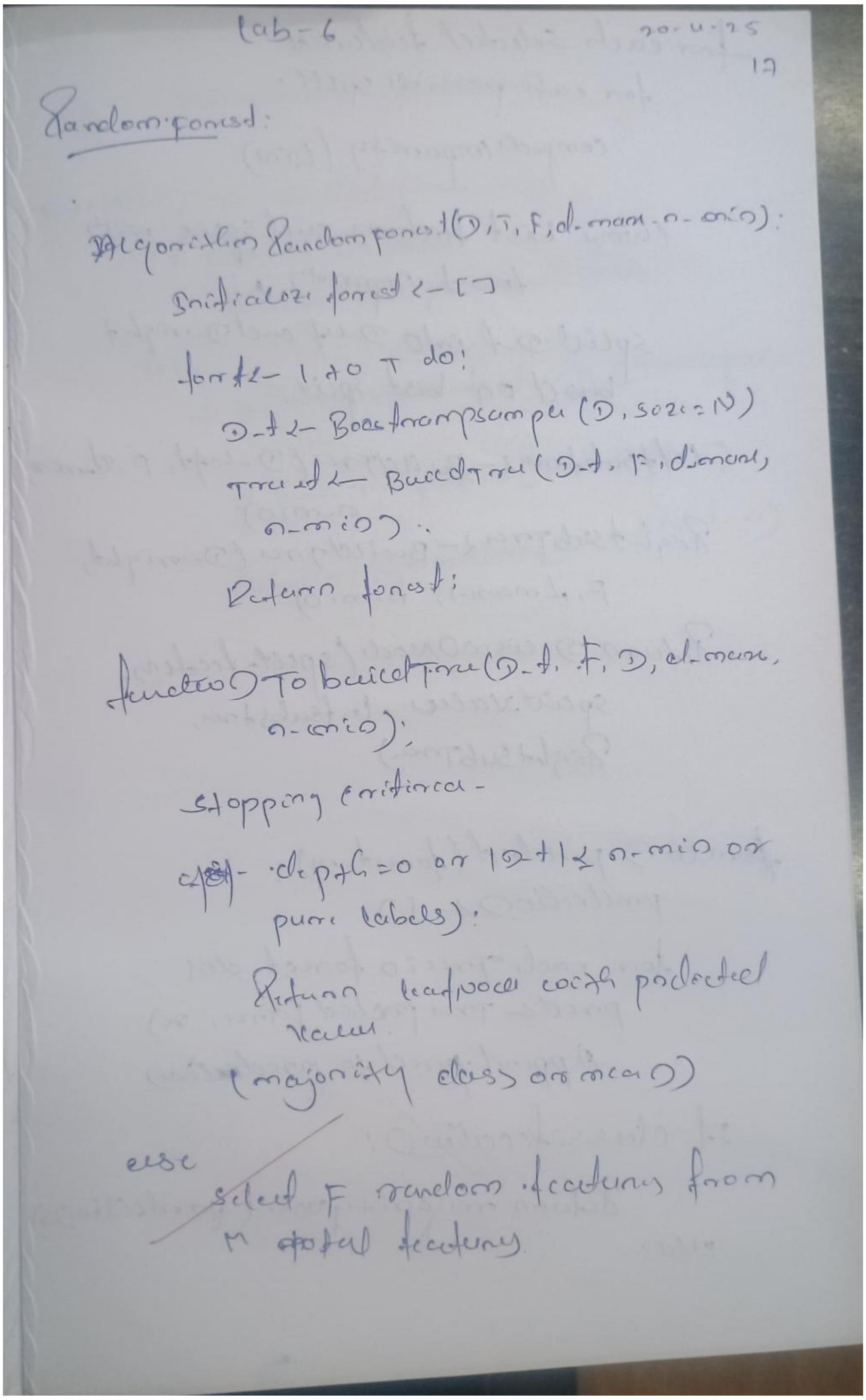
```
plt.figure(figsize=(8, 6))
plt.contourf(xx, yy, Z, alpha=0.3, cmap=plt.cm.coolwarm)

for label in np.unique(y_numeric):
    plt.scatter(X[y_numeric == label, 0], X[y_numeric == label, 1],
                label=label_map_rev[label], color=colors[label])

plt.xlabel("Petal Length (cm)")
plt.ylabel("Petal Width (cm)")
plt.title("Linear Model Decision Boundary on Iris Dataset")
plt.legend()
plt.grid(True)
plt.show()
```



LABORATORY PROGRAM – 8

Implement Random forest ensemble method on a given dataset.

OBSERVATION BOOK



lab-6 20-4-25

17

Random forest:

Algorithm Randomforest(D, T, F, d_max, n_min):

Initialize forest ← [ ]

for t ← 1 to T do:

D_t ← Bootstrapsample (D, size = N)

Tree_t ← BuildTree (D_t, F, d_max, n_min) .

Return forest;

function To buildTree(D_t, F, D, d_max, n_min);

Stopping criteria -

if - depth = 0 or |D_t| ≤ n_min or pure labels):

Return leafnode with predicted value

(majority class or mean))

else

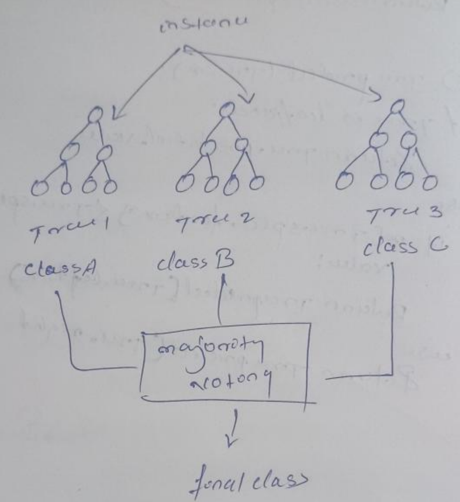
select F random features from M total features

for each selected features:
    for each possible split:
        compute impurity (Gini)
    choose best features and split with
        lowest impurity.
    split D-t into D-left and D-right
        based on best split
    LeftSubtree ← BuildTree(D-left, F, d-max-1,
        n-min)
    RightSubTree ← BuildTree(D-right,
        F, d-max-1, n-min)
    Return DecisionNode(split-features,
        split value, leftsubstree,
        RightSubTree)

function predict(forest, x):
    predictions ← []
    for each Tree in forest do:
        pred ← Treepredict(Tree, x)
        Append pred to prediction
If classification:
    Return majorityvote(predictions)
(if regression)
    Return mean(prediction)

Function Treepredict(Tree, x):
    If Tree is leafnode:
        Return Tree.predicted_value.
    else:
        if x[Tree.split_feature] ≤ Tree.split_
            value:
            Return Treepredict[Tree.left, x]
        else
            Return Treepredict[Tree.right, x)

Random forest



Dataset

filename: pumpkin seeds Dataset.xlsx

Accuracy : 0.801
classification report
feature importance

21/4/25

Code:

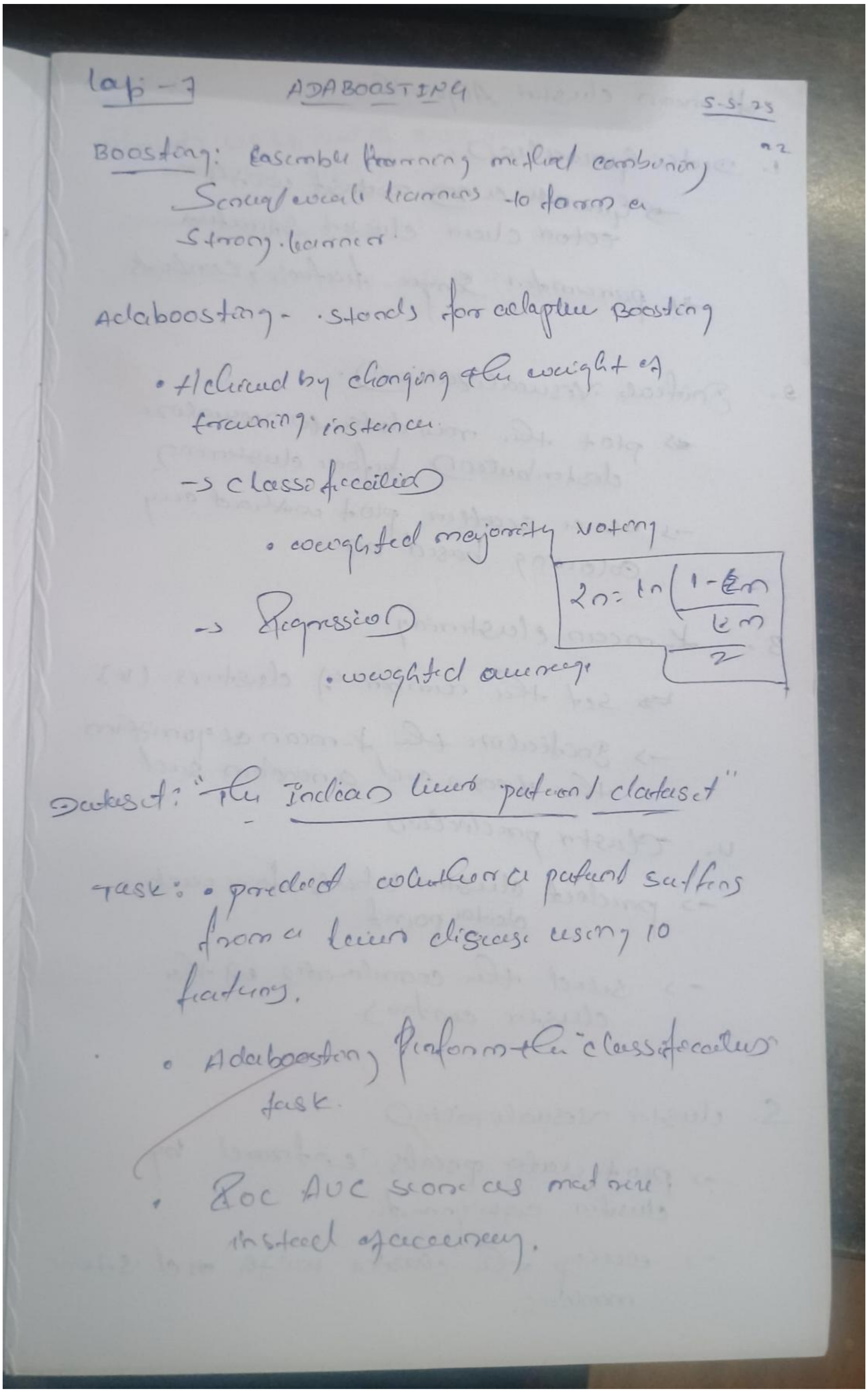
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
import seaborn as sns
import matplotlib.pyplot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

# Load the dataset
file_path = "/content/Pumpkin_Seeds_Dataset.xlsx"
df = pd.read_excel(file_path, sheet_name='Pumpkin_Seeds_Dataset')

# Separate features and target
X = df.drop(columns=['Class'])
y = df['Class']

# Encode the target labels
le = LabelEncoder()
y_encoded = le.fit_transform(y)

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

# Train the Random Forest classifier
rf_model = RandomForestClassifier(n_estimators=100, max_depth=None, random_state=42)
rf_model.fit(X_train, y_train)

# Predict on test data
```

```
y_pred = rf_model.predict(X_test)

# Evaluate the model
print(" Accuracy:", accuracy_score(y_test, y_pred))
print("\n Classification Report:\n")
print(classification_report(y_test, y_pred, target_names=le.classes_))

# Plot the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=le.classes_, yticklabels=le.classes_)
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.tight_layout()
plt.show()

# Plot top 10 feature importances
importances = rf_model.feature_importances_
features = X.columns
indices = importances.argsort()[::-1][:10]  # Top 10

plt.figure(figsize=(10, 6))
sns.barplot(x=importances[indices], y=features[indices], palette="viridis")
plt.title("Top 10 Feature Importances")
plt.xlabel("Importance")
plt.ylabel("Feature")
plt.tight_layout()
plt.show()
```

LABORATORY PROGRAM – 9

Implement Boosting ensemble method on a given dataset.

OBSERVATION BOOK



Lab-7 ADABOOSTING 5-5-23

Boosting: Ensemble learning method combining several weak learners to form a strong learner

Adaboosting - stands for adaptive Boosting

• Achieved by changing the weight of training instances

-> classification

• weighted majority voting

-> Regression

• weighted average

$$\boxed{\alpha_n = \frac{\ln\left(\frac{1-E_m}{E_m}\right)}{2}}$$

Dataset: "The Indian liver patient dataset"

Task: • predict whether a patient suffers from a liver disease using 10 features.

• Adaboosting perform the "classification" task.

• ROC AUC score as metric instead of accuracy.

CODE WITH OUTPUT

Code:

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

from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score

from sklearn.tree import DecisionTreeRegressor
from sklearn.tree import DecisionTreeClassifier

from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier as KNN

from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import mean_squared_error as MSE

from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import VotingClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingRegressor

SEED =1
# Dataset
liver = pd.read_csv('/content/sample_data/indian_liver_patient_preprocessed.csv', index_col = 0)
X = liver.drop('Liver_disease', axis = 1)
y = liver['Liver_disease']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=SEED)
```

Split data into 80% train and 20% test

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

stratify=y,

random_state=SEED)

liver.head()

	Age_std	Total_Bilirubin_std	Direct_Bilirubin_std	Alkaline_Phosphotase_std	Alamine_Aminotransferase_std	Aspartate_Aminotransferase_std	Total_Protiens_std	Albumin_std	Albumi_
0	1.247403	-0.420320	-0.495414	-0.428870	-0.355832	-0.319111	0.293722	0.203446	
1	1.062306	1.218936	1.423518	1.675083	-0.093573	-0.035962	0.939655	0.077462	
2	1.062306	0.640375	0.926017	0.816243	-0.115428	-0.146459	0.478274	0.203446	
3	0.815511	-0.372106	-0.388807	-0.449416	-0.366760	-0.312205	0.293722	0.329431	
4	1.679294	0.093956	0.179766	-0.395996	-0.295731	-0.177537	0.755102	-0.930414	

Import AdaBoostClassifier

from sklearn.ensemble import AdaBoostClassifier

Instantiate dt

dt = DecisionTreeClassifier(max_depth=2, random_state=1)

Instantiate ada

ada = AdaBoostClassifier(estimator=dt, n_estimators=180, random_state=1)

Fit ada to the training set

ada.fit(X_train, y_train)

Compute the probabilities of obtaining the positive class

y_pred_proba = ada.predict_proba(X_test)[:,1]

Import roc_auc_score

#from sklearn.metrics import roc_auc_score

Evaluate test-set roc_auc_score

```
ada_roc_auc = roc_auc_score(y_test, y_pred_proba)


# Print roc_auc_score
print('ROC AUC score: {:.2f}'.format(ada_roc_auc))
```

```
ROC AUC score: 0.70
```

LABORATORY PROGRAM – 10

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

OBSERVATION BOOK

K-mean cluster Algorithm

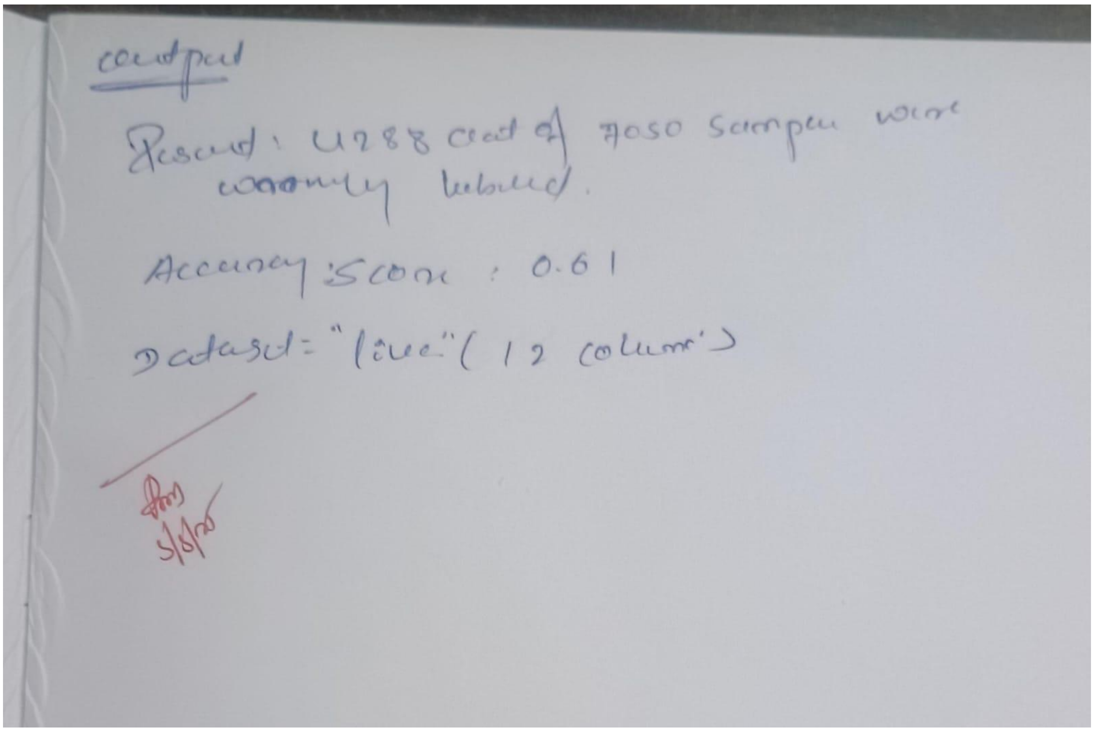
1. Data Generation:
   -> Generate a 2D dataset with ~~color~~ clear cluster structure
   -> parameter: Sample, features, centroids

2. Initial visualization:
   -> plot the raw data to visualize distribution before clustering.
   -> use a scatter plot without any coloring based on

3. k-mean clustering
   -> set the number of clusters (k).
   -> Initialize the k-mean algorithm the chosen and a random seed.

4. Cluster prediction
   -> predict cluster labels for each data point
   -> select the coordinates of the cluster centres

5. cluster visualization
   -> plot data points colored by cluster assignment.
   -> display the cluster with red star markers.



output

Result: 4288 out of 7050 samples were wrongly labelled.

Accuracy Score: 0.61

Dataset = "live" (12 columns)

CODE WITH OUTPUT

Code:

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt # for data visualization
import seaborn as sns # for statistical data visualization
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')

df = pd.read_csv('/content/sample_data/Live.csv')
df.drop(['Column1', 'Column2', 'Column3', 'Column4'], axis=1, inplace=True)
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=2, random_state=0)
kmeans.fit(X)
kmeans.cluster_centers_
kmeans.inertia_
```

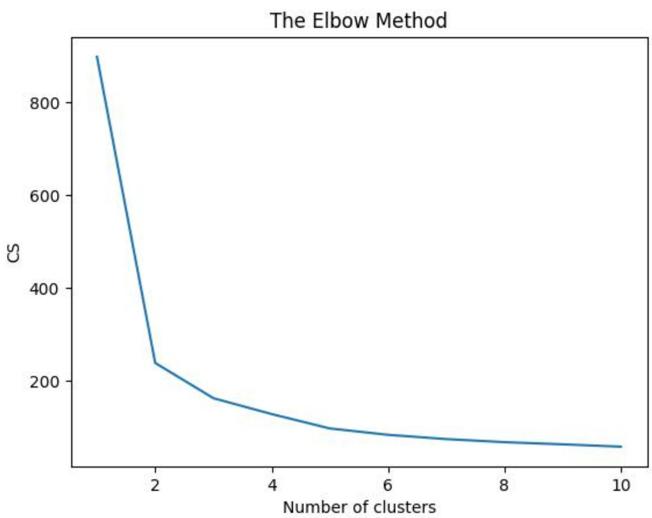
```
labels = kmeans.labels_

# check how many of the samples were correctly labeled

correct_labels = sum(y == labels)


print("Result: %d out of %d samples were correctly labeled." % (correct_labels, y.size))
```

```
Result: 4288 out of 7050 samples were correctly labeled.
```

```
print('Accuracy score: {0:0.2f}'. format(correct_labels/float(y.size)))
```

```
Accuracy score: 0.61
```

```
from sklearn.cluster import KMeans

cs = []

for i in range(1, 11):

   kmeans = KMeans(n_clusters = i, init = 'k-means++', max_iter = 300, n_init = 10,
random_state = 0)

   kmeans.fit(X)

   cs.append(kmeans.inertia_)

plt.plot(range(1, 11), cs)

plt.title('The Elbow Method')

plt.xlabel('Number of clusters')

plt.ylabel('CS')

plt.show()
```



```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=2,random_state=0)
kmeans.fit(X)
labels = kmeans.labels_
# check how many of the samples were correctly labeled
correct_labels = sum(y == labels)
print("Result: %d out of %d samples were correctly labeled." % (correct_labels, y.size))
print('Accuracy score: {0:0.2f}'. format(correct_labels/float(y.size)))
```

```
Result: 4288 out of 7050 samples were correctly labeled.
Accuracy score: 0.61
```

```
kmeans = KMeans(n_clusters=3, random_state=0)
kmeans.fit(X)
# check how many of the samples were correctly labeled
labels = kmeans.labels_
correct_labels = sum(y == labels)
print("Result: %d out of %d samples were correctly labeled." % (correct_labels, y.size))
print('Accuracy score: {0:0.2f}'. format(correct_labels/float(y.size)))
```

```
Result: 4066 out of 7050 samples were correctly labeled.
Accuracy score: 0.58
```

```
kmeans = KMeans(n_clusters=4, random_state=0)
kmeans.fit(X)
# check how many of the samples were correctly labeled
labels = kmeans.labels_
correct_labels = sum(y == labels)
print("Result: %d out of %d samples were correctly labeled." % (correct_labels, y.size))
print('Accuracy score: {0:0.2f}'. format(correct_labels/float(y.size)))
```

```
Result: 4112 out of 7050 samples were correctly labeled.
Accuracy score: 0.58
```

LABORATORY PROGRAM – 11

Implement Dimensionality reduction using Principle Component Analysis (PCA) method.

OBSERVATION BOOK

CODE WITH OUTPUT

Code:

```
from google.colab import files

heart=files.upload()

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from scipy import stats

import seaborn as sns

from sklearn.preprocessing import LabelEncoder, OneHotEncoder  from sklearn.model_selection
import train_test_split

from sklearn.metrics import accuracy_score

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score  from
sklearn.preprocessing import StandardScaler

from sklearn.svm import SVC

from sklearn.linear_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier

from sklearn.decomposition import PCA

df1=pd.read_csv("heart (1).csv")

df1.head()

text_cols = df1.select_dtypes(include=['object']).columns

label_encoder = LabelEncoder()

for col in text_cols:

df1[col] =
```

```
label_encoder.fit_transform(df1[col])

print(df1.head())

X = df1.drop('HeartDisease', axis=1)

y = df1['HeartDisease']

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)

# Support Vector Machine

svm_model = SVC(kernel='linear', random_state=42)

svm_model.fit(X_train, y_train)

svm_predictions = svm_model.predict(X_test)

svm_accuracy = accuracy_score(y_test, svm_predictions)

print(f"SVM Accuracy: {svm_accuracy}")

# Logistic Regression

lr_model = LogisticRegression(random_state=42)

lr_model.fit(X_train, y_train) lr_predictions = lr_model.predict(X_test) lr_accuracy =
accuracy_score(y_test, lr_predictions)

print(f"Logistic Regression Accuracy: {lr_accuracy}")

# Random Forest

rf_model = RandomForestClassifier(random_state=42)

rf_model.fit(X_train, y_train)

rf_predictions = rf_model.predict(X_test)

rf_accuracy = accuracy_score(y_test, rf_predictions)

print(f"Random Forest Accuracy: {rf_accuracy}")

models = {

"SVM": svm_accuracy,

"Logistic Regression":

lr_accuracy, "Random Forest":

rf_accuracy

}
```

```
best_model = max(models, key=models.get)
print(f"\nBest Model: {best_model} with accuracy {models[best_model]}")
pca = PCA(n_components=0.95)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)
svm_model_pca = SVC(kernel='linear', random_state=42)
svm_model_pca.fit(X_train_pca, y_train)
svm_predictions_pca = svm_model_pca.predict(X_test_pca)
svm_accuracy_pca = accuracy_score(y_test, svm_predictions_pca)
print(f"SVM Accuracy (with PCA): {svm_accuracy_pca}")
lr_model_pca = LogisticRegression(random_state=42)
lr_model_pca.fit(X_train_pca, y_train)
lr_predictions_pca = lr_model_pca.predict(X_test_pca)
lr_accuracy_pca = accuracy_score(y_test, lr_predictions_pca)
print(f"Logistic Regression Accuracy (with PCA): {lr_accuracy_pca}")
rf_model_pca = RandomForestClassifier(random_state=42)
rf_model_pca.fit(X_train_pca, y_train)
rf_predictions_pca = rf_model_pca.predict(X_test_pca)
rf_accuracy_pca = accuracy_score(y_test, rf_predictions_pca)
print(f"Random Forest Accuracy (with PCA): {rf_accuracy_pca}")
models_pca = {
"SVM": svm_accuracy_pca,
"Logistic Regression": lr_accuracy_pca,
"Random Forest": rf_accuracy_pca
}
best_model_pca = max(models_pca, key=models_pca.get)
print(f"\nBest Model (with PCA): {best_model_pca} with accuracy {models_pca[best_model_pca]}")
```