KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

(AN AUTONOMOUS INSTITUTION)

Accredited by NBA & NAAC, Approved by AICTE, Affiliated to JNTUH,Narayanguda, Hyderabad – 500029

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

LAB RECORD
MACHINE LEARNING USING PYTHON LAB

B.Tech. II YEAR I SEM (RKR21)
ACADEMIC YEAR 2023-24

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY
(AN AUTONOMOUS INSTITUTE)
Accredited by NBA & NAAC, Approved by AICTE, Affiliated to JNTUH, Hyderabad

Certificate

This is to certify that following is a Bonafide Record of the workbook task done by

_____________________________bearing Roll No_______________________of

__________________Branch of______________year B.Tech Course in the

__________________Subject during the Academic year______________&_________ under our supervision.

Number of experiments completed:_________________________________

Signature of Staff Member Incharge   Signature of Head of the Dept.

Signature of Internal Examiner   Signature of External Examiner

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY
(AN AUTONOMOUS INSTITUTE)
Accredited by NBA & NAAC, Approved by AICTE, Affiliated to JNTUH, Hyderabad

Daily Laboratory Assessment Sheet

Name of the Lab: Name of the Student:

Class: HT.No:

S.No.	Name of the Experiment	Date	Observation Marks (3M)	Record Marks (4M)	Viva Voice Marks (3M)	Total Marks (10M)	Signature of Faculty
	TOTAL						

INDEX

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY
(AN AUTONOMOUS INSTITUTE)
Accredited by NBA & NAAC, Approved by AICTE, Affiliated to JNTUH, Narayangud
Hyderabad – 500029

Department of Computer Science & Engineering

Vision of the Institution:

To be the fountain head of latest technologies, producing highly skilled, globally competent engineers.

Mission of the Institution:

- To provide a learning environment that inculcates problem solving skills, professional, ethical responsibilities, lifelong learning through multi modal platforms and prepare students to become successful professionals.
- To establish Industry Institute Interaction to make students ready for the industry.
- To provide exposure to students on latest hardware and software tools.
- To promote research based projects/activities in the emerging areas of technology convergence.
- To encourage and enable students to not merely seek jobs from the industry but also to create new enterprises
- To induce a spirit of nationalism which will enable the student to develop, understand India's challenges and to encourage them to develop effective solutions.
- To support the faculty to accelerate their learning curve to deliver excellent service to students

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY
(AN AUTONOMOUS INSTITUTE)
Accredited by NBA & NAAC, Approved by AICTE, Affiliated to JNTUH, Narayanguda,
Hyderabad – 500029

Department of Computer Science & Engineering

Vision of the Department:

To be among the region's premier teaching and research Computer Science and Engineering departments producing globally competent and socially responsible graduates in the most conducive academic environment.

Mission of the Department:

- To provide faculty with state of the art facilities for continuous professional development and research, both in foundational aspects and of relevance to emerging computing trends.
- To impart skills that transform students to develop technical solutions for societal needs and inculcate entrepreneurial talents.
- To inculcate an ability in students to pursue the advancement of knowledge in various specializations of Computer Science and Engineering and make them industry-ready.
- To engage in collaborative research with academia and industry and generate adequate resources for research activities for seamless transfer of knowledge resulting in sponsored projects and consultancy.
- To cultivate responsibility through sharing of knowledge and innovative computing solutions that benefits the society-at-large.
- To collaborate with academia, industry and community to set high standards in academic excellence and in fulfilling societal responsibilities.

PROGRAM OUTCOMES (POs)

PO1: Engineering Knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2: Problem Analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3: Design/Development of Solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4: Conduct Investigations of Complex Problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5: Modern Tool Usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.

PO6: The Engineer and Society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7: Environment and Sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9: Individual and Team Work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10: Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11: Project Management and Finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12: Life-long Learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Department of Computer Science & Engineering

PROGRAM SPECIFIC OUTCOMES (PSOs)

PSO1: An ability to analyze the common business functions to design and develop appropriate Computer Science solutions for social upliftment.

PSO2: Shall have expertise on the evolving technologies like Python, Machine Learning, Deep Learning, Internet of Things (IOT), Data Science, Full stack development, Social Networks, Cyber Security, Big Data, Mobile Apps, CRM, ERP etc.

PROGRAM EDUCATIONAL OBJECTIVES (PEOs)

PEO1: Graduates will have successful careers in computer related engineering fields or will be able to successfully pursue advanced higher education degrees.

PEO2: Graduates will try and provide solutions to challenging problems in their profession by applying computer engineering principles.

PEO3: Graduates will engage in life-long learning and professional development by rapidly adapting changing work environment.

PEO4: Graduates will communicate effectively, work collaboratively and exhibit high levels of professionalism and ethical responsibility.

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY
(AN AUTONOMOUS INSTITUTE)
Accredited by NBA & NAAC, Approved by AICTE, Affiliated to JNTUH, Narayangud
Hyderabad – 500029

Department of Computer Science & Engineering
B.Tech. II Year I Semester Course Syllabus(RKR21)
MACHINE LEARNING USING PYTHON LAB

Prerequisites/ Corequisites:

L	T	P	C
0	0	2	1

1. PP204ES - Python Programming Course
2. 21CS205PC - Introduction to Machine Learning Course

Course Objectives: The course will help to

1. To perform exploratory data analysis on the given data sets.
2. To provide hands on Descriptive Statistics and data analysis along with visualization.
3. To implement Regression models on given datasets.
4. To build Classification models.
5. To implement models on SVMs.

Course Outcomes: After learning the concepts of this course, the student is able to

1. Execute the basic concepts of Probability and Machine Learning.
2. Apply the Statistics and data analysis along with visualization.
3. Implement the different types of regression models.
4. Explore the classification model for categorical data.
5. Design and develop the non-Parametric models.

List of Exercise:

Exercise 1:

Apply central tendency and variability on given dataset

Excercise 2:

Perform EDA on given dataset and prepare dataset to train and test ML model

Exercise 3:

Build a linear regression model using python on given data set by

a. Prepare the data for ML model
b. Splitting Training data and Test data.
c. Evaluate the model (intercept and slope).
d. Visualize the training set and testing set using Matplotlib, Seaborn.
e. predicting the test set result f. compare actual output values with predicted values

Exercise 4:

Implement regression model without using ML libraries.

Exercise 5:

Apply various regression models on given dataset and find a proper model for prediction with minimal errors.

Exercise 6:

Implement a logistic regression model on given dataset and check the accuracy for test dataset

Exercise 7:

Build a decision tree model for given dataset to predict the target with best accuracy.

Exercise 8:

Implement KNN model to classify the target in given dataset.

Exercise 9:

Demonstrate regression and classification metrics using sample data.

Exercise 10:

Build SVM model with various kernels and select best kernel for given dataset.

Exercise 11:

Build Random Forest model and apply on given dataset. Evaluate the model with suitable metrics

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

(AN AUTONOMOUS INSTITUTE)
Accredited by NBA & NAAC, Approved by AICTE, Affiliated to JNTUH, Narayanguda, Hyderabad – 500029

Department of Computer Science & Engineering Course Outcomes and CO-PO-PSO Mapping

Course Outcomes:

After learning the contents of this course, the student is able to

CO1	Execute the basic concepts of Probability and Machine Learning.
CO2	Apply the Statistics and data analysis along with visualization.
CO3	Implement the different types of regression models.
CO4	Explore the classification model for categorical data.
CO5	Design and develop the non-Parametric models.

CO-PO-PSO MAPPING:

	CO	**PO1**	**PO2**	**PO3**	**PO4**	**PO5**	**PO6**	**PO7**	**PO8**	**PO9**	**PO10**	**PO11**	**PO12**	**PSO-1**	**PSO-2**
Machine Learning Lab	**CO1**	3				1							1		1
	CO2	3	3	3	2	1							1	1	
	CO3	3	3	3		1							2	1	
	CO4	2	3	3	1	1							1	1	
	CO5	3	2	1		1							1		1

Exercise 1:

APPLY CENTRAL TENDENCY AND VARIABILITY ON GIVEN DATASET

Aim: Apply central tendency and variability on given dataset

Software Required: Google Co Lab, Jupyter notebook, Kaggle .

Theory:
descriptive statistics. These are calculated values that summarize aspects of a data set. First we will focus on measures of central tendency, which as the name suggests, are statistics that aim to estimate the center of the data set around which (usually) most of the data values occur. Then, we will focus on measures of variability. These statistics reflect how spread out the data is in relation to other data points or the center of the data set. Measures of central tendency and variability require the data to have interval or ratio scales. After this notebook you will know:

- how to calculate sample mean, median, and mode and what each of these measures of central tendency tells you.
- how to calculate sample variance and standard deviation.
- how to calculate the z scores and determine the probability of a given z-score using a z-table.
- how to calculate standard error of a sample mean.
- how to calculate confidence intervals using a z-score for a given confidence level (α level).

Measures of Central Tendency

The following definitions of mean, median, and mode state in words how each is calculated.

Mean: The mean of a data set is the average value. Is that circular? Let me state this in the form of an algorithm. To find the mean of a data set first sum up all of the values in the data set then divide by the number of data points or examples in the data set. When we find or know the mean of an entire population we call that mean a parameter of the population and in is assigned the symbol μ. When the mean is of a sample it is a statistic of the sample ans is assigned the symbol $\bar{x}$.

Median: The median of a data set is simply the data point that occurs exactly in the middle of the data set. To find the median first the data needs to be sorted and listed in ascending or descending order. Then if there is an odd number of data points in the set the median is in the exact middle of this list. If there is an even number of data points in the list then the median is the average of the two middle values in the list.

Mode: The mode is the value in a data set that is repeated the most often. Below the mean, median, and mode are calculated for a dummy data set.

Measures of Variability

The following definitions of variance and standard deviation state in words how each is calculated.

Variance: The variance of a data set is found by first finding the deviation of each element in the data set from the mean. These deviations are squared and then added together. (Why are they squared?) Finally, the sum of squared deviations is normalized by the number of elements in the

population, N, for a population variance or the number of element in the sample minus one, N-1, for the sample variance. (Why is a sample variance normalized by N-1?)
Standard Deviation: Once the variance is in hand, standard deviation is eacy to find. It is simply the square root of the variance. The symbol for population standard deviation is σ while the simple for sample standard deviation is s or sd.

<u>Program :</u>

```
import pandas as pd
d={'Age':pd.Series([25,26,25,23,30,29,23,34,40,30,51,46]),
'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8,3.78,2.98,4.80,4.10,3.65])
}
df=pd.DataFrame(d)
df=pd.DataFrame(d)
print("Mean Values in the Distribution")
print(df.mean(axis=0))
print("Median Values in the Distribution")
print(df.median(axis=0))
#Calculate the standard deviation
print("standard deviation is")
print(df.std(axis=0))
```

output

```
Mean Values in the Distribution
Age
Rating
dtype: float64
Median Values in the Distribution
Age
Rating
dtype: float64
standard deviation is
Age
Rating
```

```
import pandas as pd
from scipy import stats
d={'Age':pd.Series([25,26,25,23,30,29,23,34,40,30,51,23])}
df=pd.DataFrame(d)
print("mode Values in the Distribution")
x=stats.mode(df)
print(x)
```

output

```
#Create the dataframe of student's marks
df=pd.DataFrame({"John - Marks ":[98,87,76,88,96],
"Adam- Marks ":[88,52,69,79,80],
"David- Marks ":[90,92,71,60,64],
"Rahul - Marks":[88,85,79,81,91]})
print(df.mean(axis=0))
print(df.median())
print(df.mode())
```

Output:

```
John - Marks
Adam- Marks
David- Marks
Rahul - Marks
dtype: float64
John - Marks
Adam- Marks
David- Marks
Rahul - Marks
dtype: float64
```

```
import numpy as np
#define data
data=np.array([18,22,32,38,41,46,53,58,67,71,78,84,91,98])
#find quarter-3 and quarter-1
q3,q1=np.percentile(data,[75,25])
#calculate the interquartile range
iqr=q3-q1
print("Interquartile Range: ",iqr)
```

Output:

```
Interquartile Range:
```

```
#Interquartile Range of a single column in a DataFrame
import numpy as np
import pandas as pd
df=pd.DataFrame([[32,24,30,40],[17,24,21,28],[50,25,28,32],[25,34,21,48],[17,31,18,28],[35,24,19,42,]],
columns=['Physics','Chemistry','Biology','Maths'],
index=['Student-1','Stedent-2','Student-3','Student-4','Student-5','Student-6'])
#find quarter-3 and quarter-1
q3,q1=np.percentile(df['Chemistry'],[75,25])
print("Quartile-1 is", q1)
print("Quartile-3 is", q3)
#calculate the interquartile range
iqr=q3-q1
print("Interquartile Range: ",iqr)
```

Output:

```
Quartile-1 is
Quartile-3 is
Interquartile Range:
```

```
#define function to calculate interquartile range of a single column
def single_iqr(x):
    return np.subtract(*np.percentile(x,[75,25]))
#calculate IQR for 'Physics' and 'Chemistry' columns
df[['Physics','Chemistry']].apply(single_iqr)
```

Output:

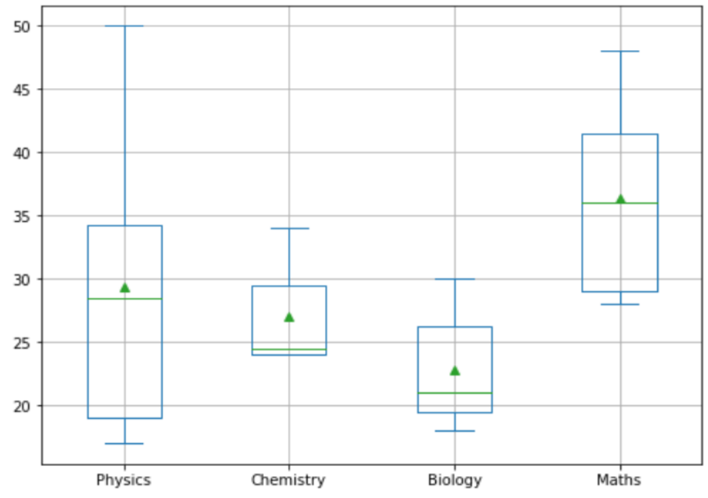
```
Physics
Chemistry
dtype: float64
```

```
#claculate IQR for all columns
df.apply(single_iqr)
```

Output:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.cbook import boxplot_stats
#plot the dataframe as needed
ax=df.plot.box(figsize=(8,6),showmeans=True)
ax.grid()
```

Output :



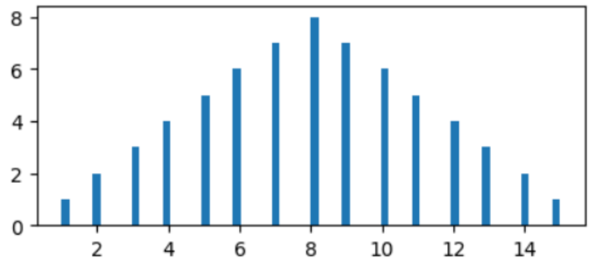
```
import warnings
warnings.filterwarnings('ignore')
import numpy as np
from scipy import stats
Dataset1 =
[1,2,2,3,3,3,4,4,4,4,5,5,5,5,5,6,6,6,6,6,6,7,7,7,7,7,7,7,8,8,8,8,8,8,8,8,9,9,9,9,9,9,9,10,10,10,10,10,10,1
1,11,11,11,11,12,12,12,12,13,13,13,14,14,15]
x1=np.mean(Dataset1)
print(" The Average of Dataset is:",x1)
x2=np.median(Dataset1)
print(" The middle value of Dataset is:",x2)
x3 = stats.mode(Dataset1)
```

```
print(" Most frequently used value of Dataset is ",x3)
```

Output :

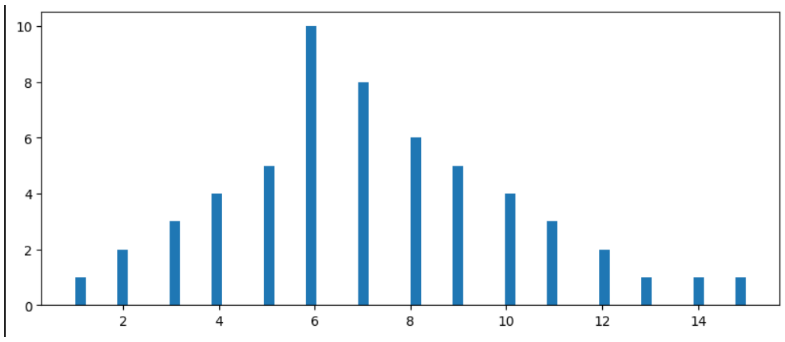
```
import numpy as np
import matplotlib.pyplot as plt
Dataset1 =
[1,2,2,3,3,3,4,4,4,4,5,5,5,5,5,6,6,6,6,6,6,7,7,7,7,7,7,7,8,8,8,8,8,8,8,8,9,9,9,9,9,9,9,10,10,10,10,10,10,1
1,11,11,11,11,12,12,12,12,13,13,13,14,14,15]
y=Dataset1
plt.figure(figsize=(5, 2))
plt1 = plt.hist(Dataset1,bins=64)
plt.show()
```

Output :



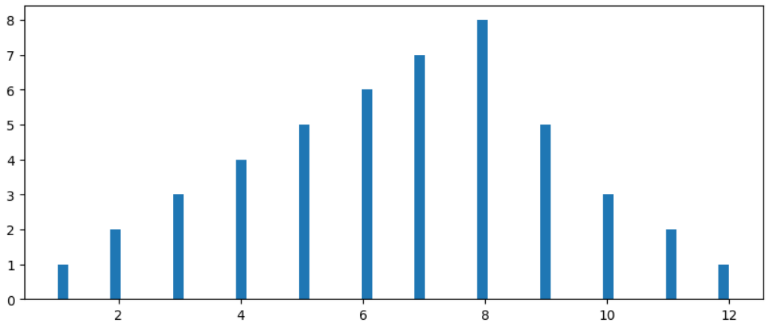
```
import warnings
warnings.filterwarnings('ignore')
import numpy as np
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
Dataset1 =
[1,2,2,3,3,3,4,4,4,4,5,5,5,5,5,6,6,6,6,6,6,6,6,6,6,7,7,7,7,7,7,7,7,8,8,8,8,8,8,9,9,9,9,9,10,10,10,10,11,11
,11,12,12,13,14,15]
x1=np.mean(Dataset1)
print(" The Average of Dataset is:",x1)
x2=np.median(Dataset1)
print(" The middle value of Dataset is:",x2)
x3 = stats.mode(Dataset1)
print(" Most frequently used value of Dataset is ",x3)
y=Dataset1
plt.figure(figsize=(10, 4))
plt1 = plt.hist(Dataset1,bins=64)
plt.show()
```

Output :



```
import warnings
warnings.filterwarnings('ignore')
import numpy as np
from scipy import stats
import numpy as np
import matplotlib.pyplot as plt
Dataset1 =
[1,2,2,3,3,3,4,4,4,4,5,5,5,5,5,6,6,6,6,6,6,7,7,7,7,7,7,7,8,8,8,8,8,8,8,8,9,9,9,9,9,10,10,10,11,11,12]
x1=np.mean(Dataset1)
print(" The Average of Dataset is:",x1)
x2=np.median(Dataset1)
print(" The middle value of Dataset is:",x2)
x3 = stats.mode(Dataset1)
print(" Most frequently used value of Dataset is ",x3)
y=Dataset1
plt.figure(figsize=(10, 4))
plt1 = plt.hist(Dataset1,bins=64)
plt.show()
```

Output :



```
import numpy as np
from scipy import stats
import numpy as np
list = [2,4,4,4,5,5,7,25]
x=list
x1=np.var(list)
print("variance of list is",x1)
range=np.max(x)-np.min(x)
print("range is:", range)
```

```
sd=np.std(x)
print("standard deviation is",sd)
```

Output :

Result: The probability parameters are
Mean =
Median=
Mode=
Variance=
Standard Deviation=

Exercise 2:

PERFORM EXPLORATORY DATA ANALYSIS ON GIVEN DATASET

Aim: Perform EDA on given dataset and prepare dataset to train and test ML model

Software Required: Google Co Lab,Jupyter notebook, Kaggle.

Dataset:Toyota.csv

Theory:

Exploratory Data Analysis (EDA) is exploring the data and discovering the insights Understand data, getting to know the context of data Getting to know the variables and their relationships Data Preprocessing (Handling missing values) Data visualization

Data Pre-processing includes the following:

- Detecting and Handling Missing values
- Type conversion
- Detecting and Treating Outliers
- Feature and Target attribute Selection
- Scaling
- Dimensionality Reduction
- Training & Test set splitting

Program :

```
import pandas as pd
import io
toyotadf=pd.read_csv('/kaggle/input/toyota-
dataset/Toyota.csv',index_col=0,na_values=["??","????"])

df = toyotadf.copy()
toyotadf.shape
```

Detecting and handling Missing Values

```
toyotadf.info()

toyotadf.isnull().sum()

toyotadf['HP'].unique()



import numpy as np
np.unique(toyotadf['KM'])
```

```
toyotadf['Doors'].unique()

toyotadf['Automatic'].unique()

toyotadf['MetColor'].unique()

toyotadf.info()
```

Now KM and HP columns are identified as float

```
toyotadf['MetColor'] = toyotadf['MetColor'].astype('O')
toyotadf['Automatic'] = toyotadf['Automatic'].astype('object')


# Alternate....
toyotadf[['MetColor','Automatic']] = toyotadf[['MetColor','Automatic']].astype
('object')

toyotadf['Doors'].unique()


toyotadf['Doors'].replace('three',3,inplace=True)
toyotadf['Doors'].replace('four',4,inplace=True)
toyotadf['Doors'].replace('five',5,inplace=True)


toyotadf['Doors'] = toyotadf['Doors'].astype('int')

toyotadf['Doors'].unique()

toyotadf.info()
```

Let's focus on NULL values....

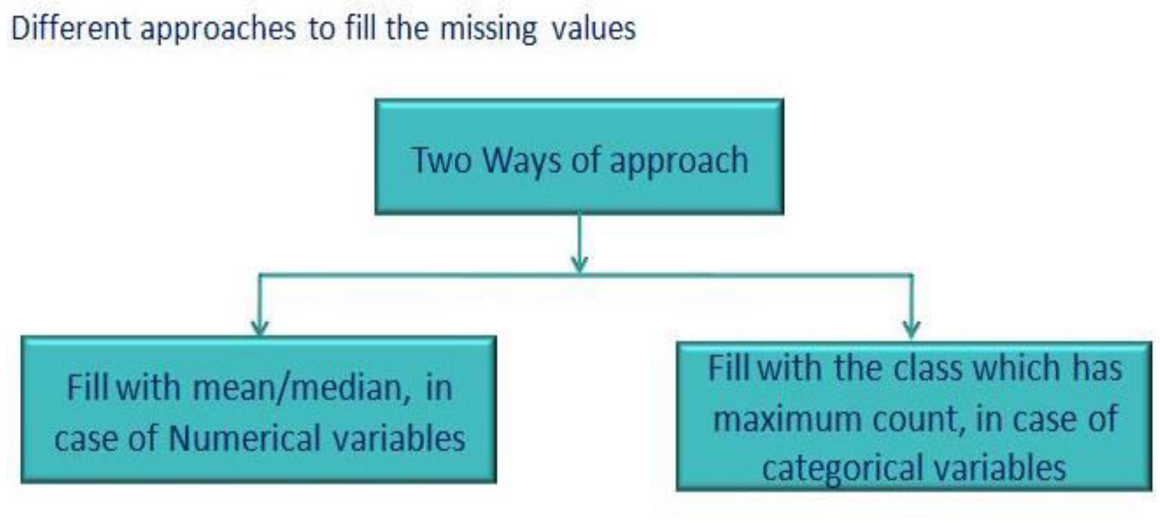
```
# Total no.of NULL values
toyotadf.isnull().sum().sum()
```

Output(Missing values):

B. Whether to Handle or Remove Missing values????

Imputing Missing Values

List item
List item


Different approaches to fill the missing values
Two Ways of approach
Fill with mean/median, in case of Numerical variables
Fill with the class which has maximum count, in case of categorical variables

'Age', 'KM', and 'HP' are numeric... Whehter to fill with mean of median value??

```
toyotadf.describe()
```

Thumb Rule: for normal distribution use mean value for imputing; median for skewed distribution

```
toyotadf.describe(exclude=['int','float'])

toyotadf.describe(include='O')

toyotadf['Age'].tail(10)

toyotadf['Age'].mean()

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


toyotadf['Age'].tail(10)

toyotadf['KM'].head(10)

toyotadf['KM'].fillna(toyotadf['KM'].median(), inplace=True)
toyotadf['KM'].median()
```

```
toyotadf['KM'].head(10)
```

```
toyotadf['FuelType'].value_counts()


toyotadf['FuelType'].mode() # most frequently occured value

toyotadf['MetColor'].value_counts()

toyotadf['MetColor'].mode()
```

MetColor of category-1 is most frequently occuring

```
toyotadf['FuelType'].fillna(toyotadf['FuelType'].value_counts().index[0], inplace=True)
toyotadf['MetColor'].fillna(toyotadf['MetColor'].mode().index[0], inplace=True)

toyotadf.isnull().sum()
```

Output(Missing values):

Result:

Exercise 3:

BUILDING A LINEAR REGRESSION MODEL

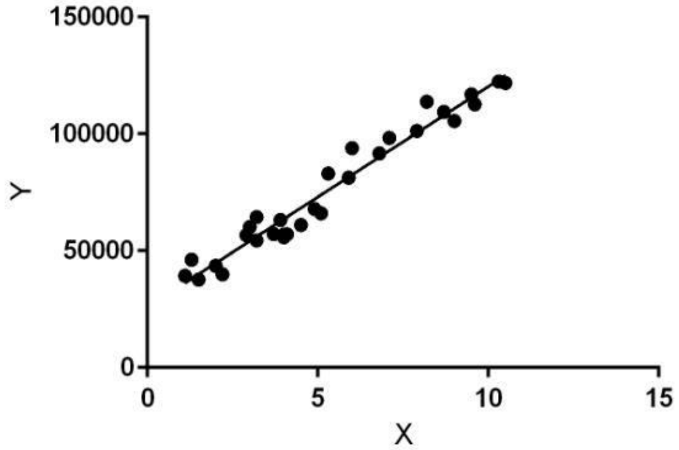
Aim: Calculate the MSE by Building a linear regression model using python on given dataset with and without data handling

a. Prepare the data for ML model
b. Splitting Training data and Test data.
c. Evaluate the model (intercept and slope).
d. Visualize the training set and testing set using Matplotlib, Seaborn.
e. predicting the test set result f. compare actual output values with predicted values

Dataset: https://www.kaggle.com/camnugent/california-housing-prices

Software Required: Google Co Lab, Jupyter notebook, Kaggle .

Theory:
Linear Regression is a machine learning algorithm based on supervised learning. It performs a regression task. Regression models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting. Different regression models differ based on – the kind of relationship between dependent and independent variables they are considering, and the number of independent variables getting used.



Linear regression performs the task to predict a dependent variable value (y) based on a given independent variable (x). So, this regression technique finds out a linear relationship between x (input) and y(output). Hence, the name is Linear Regression.
In the figure above, X (input) is the work experience and Y (output) is the salary of a person. The regression line is the best fit line for our model.

Hypothesis function for Linear Regression :

$$y = \theta_1 + \theta_2.x$$

While training the model we are given :
x: input training data (univariate – one input variable(parameter))
y: labels to data (supervised learning)

When training the model – it fits the best line to predict the value of y for a given value of x. The model gets the best regression fit line by finding the best θ1 and θ2 values.
θ1: intercept
θ2: coefficient of x

Once we find the best θ1 and θ2 values, we get the best fit line. So when we are finally using our model for prediction, it will predict the value of y for the input value of x.

Program:

```
# Import libraries
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import os

#Read dataset into pandas dataframe
data = pd.read_csv('/kaggle/input/housing/housing.csv')
data
```

a.Prepare the data for ML model

```
data.shape

data.columns

data.describe()[['median_income','median_house_value']]
```

Finding Null Values & Outliers

```
data.isnull().sum()


#  A. Data Visualizations

data.median_income.hist()

data.median_house_value.hist()

import matplotlib.pyplot as plt
plt.boxplot(data.median_income)



plt.boxplot(data.median_house_value)

plt.scatter(data.median_income,data.median_house_value)

data.corr()[['median_income','median_house_value']]
```

```
#data.median_income.values.reshape(-1,1)
x=data.median_income.values.reshape(-1,1)
x

y=data.median_house_value.values.reshape(-1,1)
y
```

# b. Splitting train and test datasets

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2)
x_train.shape,x_test.shape,y_train.shape,y_test.shape
```

i) Building model using sklearn

```
from sklearn.linear_model import LinearRegression
lm=LinearRegression()

lm.fit(x_train,y_train)
```

# c. Evaluate the model (intercept and slope).

```
lm.coef_,lm.intercept_

y_pred= lm.intercept_+lm.coef_*x_train
y_pred
```

# e. Predicting the test set result

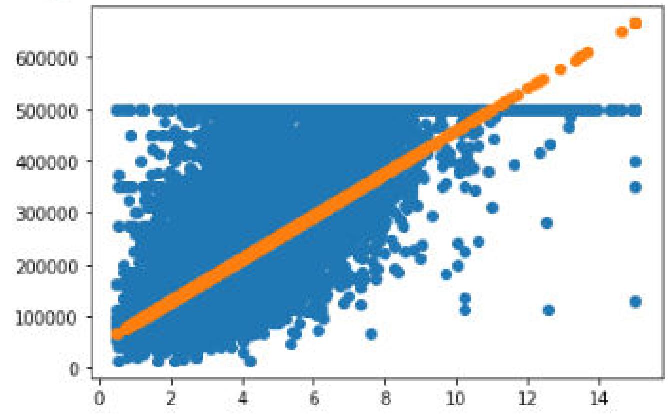
```
#Prediction for test dataset
y_pred=lm.predict(x_test)
y_pred


from sklearn.metrics import mean_squared_error, mean_absolute_error
mean_absolute_error(y_test,y_pred),np.sqrt(mean_squared_error(y_test,y_pred))
```

d.Visualize the training set and testing set using Matplotlib, Seaborn.

```
plt.scatter(x,y)
plt.scatter(x_test,y_pred)
```

```
<matplotlib.collections.PathCollection at 0x7f86cbfc42d0>
```



ii) Data Handling- Visualization of Data without outliers

plt.hist(x[x<14])

data1=data[data.median_income<data.median_income.quantile(0.99)]

data1.median_income.hist()

data1=data[data.median_income<data.median_income.quantile(0.92)]
data1=data1[data1.median_house_value<data1.median_house_value.quantile(0.92)]

data1.median_income.hist()

data.shape,data1.shape

plt.boxplot(data1.median_income)

plt.boxplot(data1.median_house_value)

plt.scatter(data1.median_income,data1.median_house_value)

data1.columns

x=data1[['median_income','households','housing_median_age']]
y=data1.median_house_value.values.reshape(-1,1)

x.max(),x.min(),y.max(),y.min()

iii)Building model after Data Handling for outliers

```
from sklearn.model_selection import train_test_split
x_train,x_test, y_train,y_test = train_test_split(x,y,test_size=0.2)
x_train.shape,y_train.shape, x_test.shape,y_test.shape

x_train.max(),y_train.max(), x_test.max(),y_test.max()
```

Building model using sklearn

```
from sklearn.linear_model import LinearRegression
lm=LinearRegression()
lm.fit(x_train,y_train)

# Display the parameters of model
lm.get_params()

# c. Evaluate the model (intercept and slope) after Data handling
for outliers

# Display the coefficients of input features
lm.coef_

# Display the intercept
lm.intercept_
```

e. Predicting the test set result

```
#prediction for test dataset
y_test_pred=lm.predict(x_test)
mean_absolute_error(y_test,y_test_pred)
```

f. compare actual output values with predicted values

```
#Predict the house value for given data of median_income, households and median_age of house
newval=pd.DataFrame({'income':[4,6,3],'households':[5,4,2],'houseage':[20,12,30]})
lm.predict(newval)
```

Result: The Mean square error for a given dataset is

MSE before Data Handling:

MSE after Data Handling:

Exercise 4:

IMPLEMENT REGRESSION MODEL WITHOUT USING ML LIBRARIES

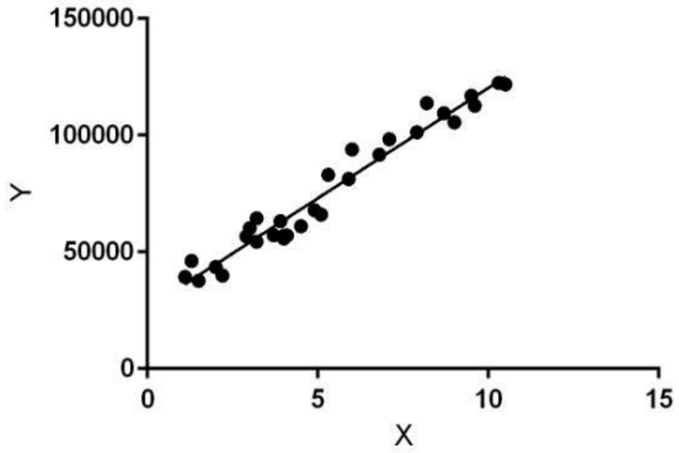
Aim: Implement regression model without using ML libraries.

Software Required: Google Co Lab, Jupyter notebook, Kaggle .

Dataset: https://www.kaggle.com/spscientist/students-performance-in-exams

Theory:

Linear Regression is a machine learning algorithm based on supervised learning. It performs a regression task. Regression models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting. Different regression models differ based on – the kind of relationship between dependent and independent variables they are considering, and the number of independent variables getting used.



Linear regression performs the task to predict a dependent variable value (y) based on a given independent variable (x). So, this regression technique finds out a linear relationship between x (input) and y(output). Hence, the name is Linear Regression.
In the figure above, X (input) is the work experience and Y (output) is the salary of a person. The regression line is the best fit line for our model.

Hypothesis function for Linear Regression :

$$y = \theta_1 + \theta_2 . x$$

While training the model we are given :
x: input training data (univariate – one input variable(parameter))
y: labels to data (supervised learning)

When training the model – it fits the best line to predict the value of y for a given value of x. The model gets the best regression fit line by finding the best θ1 and θ2 values.
θ1: intercept
θ2: coefficient of x

Once we find the best θ1 and θ2 values, we get the best fit line. So when we are finally using our model for prediction, it will predict the value of y for the input value of x.

Program :

```
# Import required libraries
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import os


# Read dataset into pandas dataframe
import pandas as pd
data=pd.read_csv('/kaggle/input/students-performance-in-exams/StudentsPerformance.csv')
data
```

Checking for Null/Missing Values

```
data.isnull().sum()
```

Histogram to check numerical data

```
data.hist()
```

Checking Covariance

```
data.cov()

data.cov()['math score']

data.cov()['math score']['writing score']

data.corr()

data.corr()['math score']['writing score']
```

Visualizing relationship between reading, writing and math scores

```
import matplotlib.pyplot as plt
plt.scatter(data['math score'],data['writing score'])
plt.xlabel('Maths Score')
plt.ylabel('Writing Score')
plt.title('Covariance='+str(data.cov()['math score']
                           ['writing score']))
plt.show()
plt.scatter(data['reading score'],data['writing score'])
plt.xlabel('Reading Score')
plt.ylabel('Writing Score')
plt.title('Covariance='+str(data.cov()['reading score']['writing score']))
plt.show()


plt.scatter(data['reading score'],data['writing score'])
plt.xlabel('Reading Score')
plt.ylabel('Writing Score')
plt.title('Covariance='+str(data.cov()['reading score']['writing score']))
plt.show()

plt.scatter(data['reading score'],data['writing score'])
plt.xlabel('Reading Score')
plt.ylabel('Writing Score')
plt.title('Corrlation='+str(data.corr()['reading score']['writing score']))
plt.show()
```

Selecting read score as predictor(x) and writing score(y) as target

```
x=data['reading score'].values
y=data['writing score'].values
```

Splitting Data into Train and Test sets

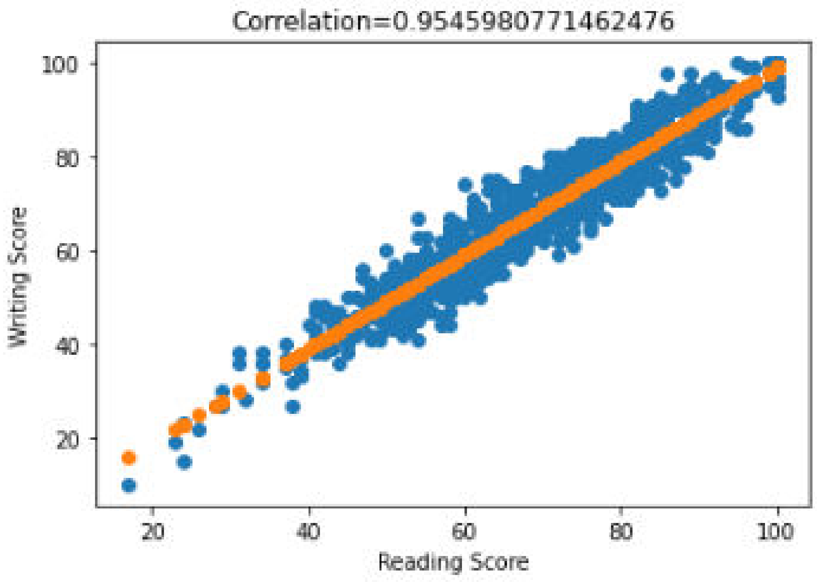
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2)
x_train.shape,x_test.shape,y_train.shape,y_test.shape
```

Finding the slope of the line and intercept using traditional statistical formula

```
m=np.cov(x_train,y_train)[0,1]/x_train.var()
c=y_train.mean()-(x_train.mean()*m)
print(m,c)

y_pred=(x_train*m)+c
```

```
plt.scatter(x,y)
plt.scatter(x_train,y_pred)
plt.xlabel('Reading Score')
plt.ylabel('Writing Score')
plt.title('Correlation='+str(data.corr()['reading score']
                                        ['writing score']))
plt.show()
```



```
#y_pred-y_train
#np.abs(y_pred-y_train)

np.sum(np.abs(y_pred-y_train))/(x_train.size)

# Prediction for Test data
y_pred=(x_test*m)+c
np.sum(np.abs(y_pred-y_test))/(x_test.size)
```

Using Least Squares method

```
l=x_train.size
x2=x_train**2
y2=y_train**2
xy=x_train*y_train

m= ((l * sum(xy)) - (sum(x_train)*sum(y_train))) / ((l * sum(x2)) - sum(x_train)**2)
m
c= y_train.mean()-m*x_train.mean()
```

```
c

y_pred=(x_train*m)+c

plt.scatter(x,y)
plt.scatter(x_train,y_pred)
plt.xlabel('Reading Score')
plt.ylabel('Writing Score')
plt.title('Covariance='+str(data.corr()['reading score']['writing score']))
plt.show()

np.sum(np.abs(y_pred-y_train))/(x_train.size)
```

Result: The Mean absolute error for a given dataset is

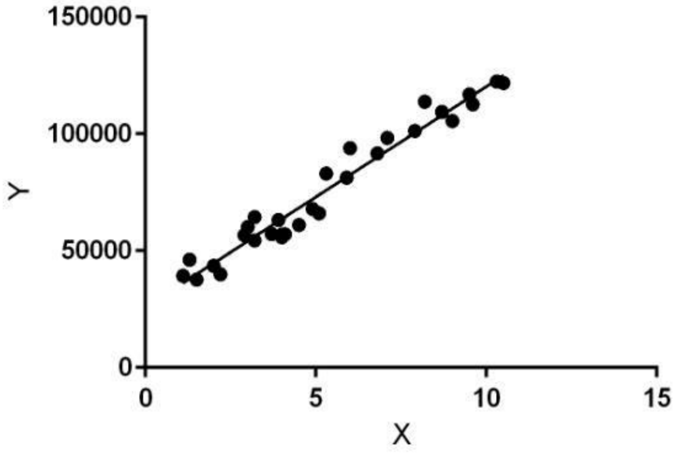
Exercise 5:

VARIOUS REGRESSION MODELS ON GIVEN DATASET

Aim: Apply various regression models on given dataset and find a proper model for prediction with minimal errors.

Software Required: Google Co Lab, Jupyter notebook, Kaggle .

Theory:
Linear Regression is a machine learning algorithm based on supervised learning. It performs a regression task. Regression models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting. Different regression models differ based on – the kind of relationship between dependent and independent variables they are considering, and the number of independent variables getting used.



Linear regression performs the task to predict a dependent variable value (y) based on a given independent variable (x). So, this regression technique finds out a linear relationship between x (input) and y(output). Hence, the name is Linear Regression.
In the figure above, X (input) is the work experience and Y (output) is the salary of a person. The regression line is the best fit line for our model.

Hypothesis function for Linear Regression :

$$y = \theta_1 + \theta_2 . x$$

While training the model we are given :
x: input training data (univariate – one input variable(parameter))
y: labels to data (supervised learning)

When training the model – it fits the best line to predict the value of y for a given value of x. The model gets the best regression fit line by finding the best θ1 and θ2 values.
θ1: intercept
θ2: coefficient of x

Once we find the best θ1 and θ2 values, we get the best fit line. So when we are finally using our model for prediction, it will predict the value of y for the input value of x.

Dataset: https://www.kaggle.com/camnugent/california-housing-prices

Program :

```
# Import libraries
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O
import os

#Read dataset into pandas dataframe

data = pd.read_csv('/kaggle/input/housing/housing.csv')
data

data.corr()

x=data.median_income.values.reshape(-1,1)
x

y=data.median_house_value.values.reshape(-1,1)
y

x.max(),x.min(),y.max(),y.min()
```

Splitting train and test datasets

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2)
x_train.shape,x_test.shape,y_train.shape,y_test.shape
```

Building model using sklearn

```
from sklearn.linear_model import LinearRegression
```

b. linear Regression

```
lm=LinearRegression()

lm.fit(x_train,y_train)
lm.coef_,lm.intercept_

#Prediction for train dataset
y_pred=lm.predict(x_test)
y_pred

print('MAE of regression is:',mean_absolute_error(y_test,y_pred))

import matplotlib.pyplot as plt
plt.scatter(x,y)
plt.scatter(x_test,y_pred)
```

b. Polynomial Regression

```
from sklearn.preprocessing import PolynomialFeatures

trans = PolynomialFeatures(degree=2)
x = trans.fit_transform(x)
x.shape

from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2)
x_train.shape,y_train.shape, x_test.shape,y_test.shape

from sklearn.linear_model import LinearRegression
lm=LinearRegression()

lm.fit(x_train,y_train)

# Prediction for train data
y_train_pred=lm.predict(x_train)
mean_absolute_error(y_train,y_train_pred)

# Prediction for test dataset
y_test_pred=lm.predict(x_test)
print('MAE of Polynomial regression is ',mean_absolute_error(
y_test,y_test_pred))
```

c. Multiple Regression

```
#Select features for multiple regression
x=data[['median_income','households','housing_median_age']]
y=data.median_house_value.values.reshape(-1,1)

x.max(),x.min(),y.max(),y.min()

from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2)
x_train.shape,y_train.shape, x_test.shape,y_test.shape

x_train.max(),y_train.max(), x_test.max(),y_test.max()

from sklearn.linear_model import LinearRegression
lm=LinearRegression()
lm.fit(x_train,y_train)

#Prediction for train data
y_train_pred=lm.predict(x_train)
mean_absolute_error(y_train,y_train_pred)

#prediction for test dataset
y_test_pred=lm.predict(x_test)
print('MAE of Multiple regressions is ',mean_absolute_error(
y_test,y_test_pred))
```

Result: The Mean absolute error for a given dataset using various regression are

MAE in Linear Regression:

MAE in Polynomial Regression:

MAE in Multiple Regression:

Exercise 6:

IMPLEMENTATION LOGISTIC REGRESSION MODEL

Aim: Implement a logistic regression model on given dataset and check the accuracy for test dataset.

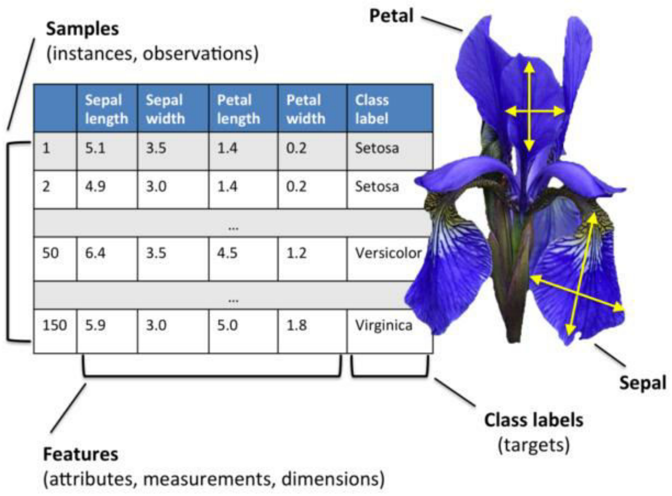
Software Required: Google Co Lab, Jupyter notebook, Kaggle .

Dataset:https://www.kaggle.com/datasets/uciml/iris

Iris Plants Dataset: Dataset contains 150 instances (50 in each of three classes)

Number of Attributes: 4 numeric, predictive attributes and the Class.

The Iris dataset was used in R.A. Fisher's classic 1936 paper, The Use of Multiple Measurements in Taxonomic Problems, and can also be found on the UCI Machine Learning Repository.
It includes three iris species with 50 samples each as well as some properties about each flower.
One flower species is linearly separable from the other two, but the other two are not linearly separable from each other.
The columns in this dataset are:
Id SepalLengthCmSepalWidthCmPetalLengthCmPetalWidthCm Species



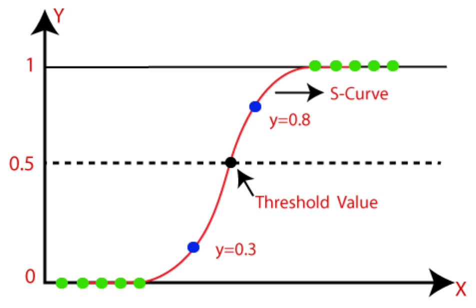
	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
			...		
50	6.4	3.5	4.5	1.2	Versicolor
			...		
150	5.9	3.0	5.0	1.8	Virginica

Samples
(instances, observations)
Petal
Sepal
Class labels
(targets)
Features
(attributes, measurements, dimensions)

Theory:

- Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique. It is used for predicting the categorical dependent variable using a given set of independent variables.
- Logistic regression predicts the output of a categorical dependent variable. Therefore the outcome must be a categorical or discrete value. It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, **it gives the probabilistic values which lie between 0 and 1**.
- Logistic Regression is much similar to the Linear Regression except that how they are used. Linear Regression is used for solving Regression problems, whereas **Logistic regression is used for solving the classification problems**.

- In Logistic regression, instead of fitting a regression line, we fit an "S" shaped logistic function, which predicts two maximum values (0 or 1).
- The curve from the logistic function indicates the likelihood of something such as whether the cells are cancerous or not, a mouse is obese or not based on its weight, etc.
- Logistic Regression is a significant machine learning algorithm because it has the ability to provide probabilities and classify new data using continuous and discrete datasets.
- Logistic Regression can be used to classify the observations using different types of data and can easily determine the most effective variables used for the classification. The below image is showing the logistic function:



Logistic Function (Sigmoid Function):

- The sigmoid function is a mathematical function used to map the predicted values to probabilities.
- It maps any real value into another value within a range of 0 and 1.
- The value of the logistic regression must be between 0 and 1, which cannot go beyond this limit, so it forms a curve like the "S" form. The S-form curve is called the Sigmoid function or the logistic function.
- In logistic regression, we use the concept of the threshold value, which defines the probability of either 0 or 1. Such as values above the threshold value tends to 1, and a value below the threshold values tends to 0.
- $y = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \cdots + b_nx_n$
- $\frac{y}{1-y}$; 0 for y= 0, and infinity for y=1
- $log\left[\frac{y}{1-y}\right] = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \cdots + b_nx_n$

Type of Logistic Regression:

On the basis of the categories, Logistic Regression can be classified into three types:

- **Binomial:** In binomial Logistic regression, there can be only two possible types of the dependent variables, such as 0 or 1, Pass or Fail, etc.
- **Multinomial:** In multinomial Logistic regression, there can be 3 or more possible unordered types of the dependent variable, such as "cat", "dogs", or "sheep"
- **Ordinal:** In ordinal Logistic regression, there can be 3 or more possible ordered types of dependent variables, such as "low", "Medium", or "High".

Program :

```
#Import required libraries
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O
import os

#Read dataset into pandas dataframe.
data=pd.read_csv('Iris.csv')

data.head()

# Display the datatype of each column
data.dtypes

# Create a column cat_code which represents the numerical values for Species column.
data['cat_code']=data.Species.astype('category').cat.codes

data.head()

data.columns

#Select the input features as x
x=data[['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm',
                              'PetalWidthCm']]

#Select the target
y=data['cat_code']

# Visualize the 3 categories of iris flower with respective to
Sepal Length and Sepal Width
```

```
import matplotlib.pyplot as plt
plt.scatter(x['SepalLengthCm'],x['SepalWidthCm'],c=y)


# Visualize the 3 categories of iris flower with respective to
Petal Length and Petal Width

plt.scatter(x['PetalLengthCm'],x['PetalWidthCm'],c=y)

x.shape,y.shape
# Splitting train and test datasets
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,shuffle=True)
x_train.shape,x_test.shape,y_train.shape,y_test.shape

# Fit the logistic regression model with SKlearn library
from sklearn.linear_model import LogisticRegression
lm=LogisticRegression()
lm.fit(x_train,y_train)

# Prediction for train set
y_pred = lm.predict(x_train)
y_train.values
y_pred

# Accuracy for train dataset
from sklearn.metrics import accuracy_score
# Prediction for test dataset and Accuracy
y_pred = lm.predict(x_test)
accuracy_score(y_test,y_pred)
```

Result: The accuracy score is

Exercise 7:

BUILD A DECISION TREE MODEL

Aim: Build a decision tree model for given dataset to predict the target with best accuracy.

Software Required: Google Co Lab, Jupyter notebook, Kaggle .

Dataset: https://www.kaggle.com/datasets/fredericobreno/play-tennis

Features*:* Outlook, Temperature, Humidity, Wind
Label*:* Play Tennis (The output feature that we want to predict)
Class*:* Yes, No (Unique values of the label)

Theory:

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. A tree can be seen as a piecewise constant approximation.

For instance, in the example below, decision trees learn from data to approximate a sine curve with a set of if-then-else decision rules. The deeper the tree, the more complex the decision rules and the fitter the model.

Some advantages of decision trees are:

- Simple to understand and to interpret. Trees can be visualized.
- Requires little data preparation. Other techniques often require data normalization, dummy variables need to be created and blank values to be removed. Note however that this module does not support missing values.
- The cost of using the tree (i.e., predicting data) is logarithmic in the number of data points used to train the tree.
- Able to handle both numerical and categorical data. However scikit-learn implementation does not support categorical variables for now. Other techniques are usually specialized in analyzing datasets that have only one type of variable. See algorithms for more information.
- Able to handle multi-output problems.
- Uses a white box model. If a given situation is observable in a model, the explanation for the condition is easily explained by boolean logic. By contrast, in a black box model (e.g., in an artificial neural network), results may be more difficult to interpret.
- Possible to validate a model using statistical tests. That makes it possible to account for the reliability of the model.
- Performs well even if its assumptions are somewhat violated by the true model from which the data were generated.
-

The disadvantages of decision trees include:

- Decision-tree learners can create over-complex trees that do not generalize the data well. This is called overfitting. Mechanisms such as pruning, setting the minimum number of samples required at a leaf node or setting the maximum depth of the tree are necessary to avoid this problem.

- Decision trees can be unstable because small variations in the data might result in a completely different tree being generated. This problem is mitigated by using decision trees within an ensemble.
- Predictions of decision trees are neither smooth nor continuous, but piecewise constant approximations as seen in the above figure. Therefore, they are not good at extrapolation.
- The problem of learning an optimal decision tree is known to be NP-complete under several aspects of optimality and even for simple concepts. Consequently, practical decision-tree learning algorithms are based on heuristic algorithms such as the greedy algorithm where locally optimal decisions are made at each node. Such algorithms cannot guarantee to return the globally optimal decision tree. This can be mitigated by training multiple trees in an ensemble learner, where the features and samples are randomly sampled with replacement.
- There are concepts that are hard to learn because decision trees do not express them easily, such as XOR, parity or multiplexer problems.
- Decision tree learners create biased trees if some classes dominate. It is therefore recommended to balance the dataset prior to fitting with the decision tree.

Program :

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
#numpy and pandas initialization
import numpy as np
import pandas as pd
PlayTennis = pd.read_csv("tennis.csv")
#numerical values into numerical values using LabelEncoder
from sklearn.preprocessing import LabelEncoder
Le = LabelEncoder()
PlayTennis['outlook'] = Le.fit_transform(PlayTennis['outlook'])
PlayTennis['temp'] = Le.fit_transform(PlayTennis['temp'])
PlayTennis['humidity'] = Le.fit_transform(PlayTennis['humidity'])
PlayTennis['windy'] = Le.fit_transform(PlayTennis['windy'])
PlayTennis['play'] = Le.fit_transform(PlayTennis['play'])



features_cols=['outlook','temp','humidity','windy']
x=PlayTennis[features_cols]
y=PlayTennis.play
from sklearn.model_selection import train_test_split
x_train, x_test, y_train,y_test=train_test_split(x,y,test_size=0.2)
from sklearn.tree import DecisionTreeClassifier
#classifier=DecisionTreeClassifier(criterion='gini')
classifier=DecisionTreeClassifier(criterion='entropy')
classifier.fit(x_train,y_train)
DecisionTreeClassifier(criterion='entropy')
classifier.predict(x_test)
classifier.score(x_test,y_test)
from sklearn import tree
clf = tree.DecisionTreeClassifier(criterion = 'entropy')
clf = clf.fit(x, y)
tree.plot_tree(clf)
```

Output:

Result:

Exercise 8:

KNN MODEL FOR CLASSIFICATION

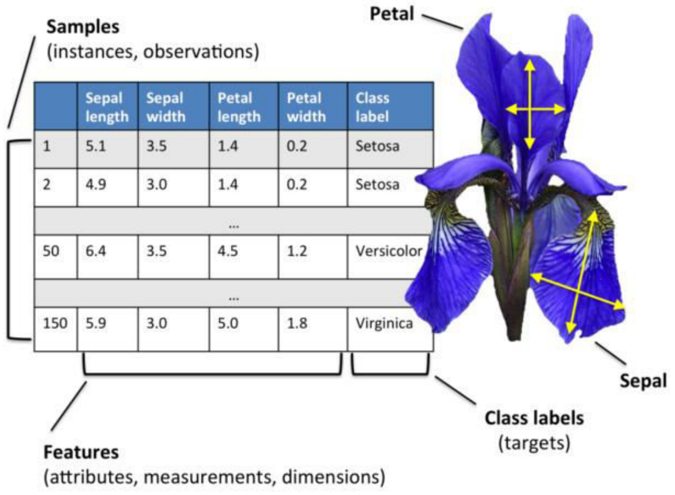
Aim: Implement KNN model to classify the target in given dataset. calculate the Accuracy & confusion matrix

Software Required: Google Co Lab, Jupyter notebook, Kaggle .

DataSet:https://www.kaggle.com/datasets/uciml/iris

Iris Plants Dataset: Dataset contains 150 instances (50 in each of three classes) Number of Attributes: 4 numeric, predictive attributes and the Class.

The Iris dataset was used in R.A. Fisher's classic 1936 paper, The Use of Multiple Measurements in Taxonomic Problems, and can also be found on the UCI Machine Learning Repository.
It includes three iris species with 50 samples each as well as some properties about each flower.
One flower species is linearly separable from the other two, but the other two are not linearly separable from each other.
The columns in this dataset are:
Id SepalLengthCmSepalWidthCmPetalLengthCmPetalWidthCm Species


Samples
(instances, observations)
Petal
Sepal
Class labels
(targets)
Features
(attributes, measurements, dimensions)

	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
			...		
50	6.4	3.5	4.5	1.2	Versicolor
			...		
150	5.9	3.0	5.0	1.8	Virginica

Theory:

Training algorithm:

For each training example (x, f (x)), add the example to the list training examples Classification algorithm:

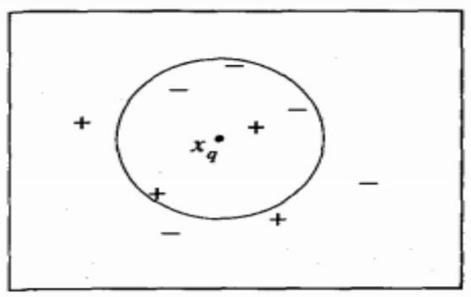
Given a query instance xq to be classified,

Let x1 . . .xk denote the k instances from training examples that are nearest to xq

Return Where, f(xi) function to calculate the mean value of the k nearest training examples.

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} f(x_i)}{k}$$

k-NEAREST NEIGHBOR.
A set of positive and negative training examples is shown on the right, along with a query instance x q, to be classified.



Program :

```
# Import libraries
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import os
import pandas as pd
import warnings
warnings.filterwarnings("ignore")
import seaborn as sns
import matplotlib.pyplot as plt
sns.set(style='white',color_codes=True)


df=pd.read_csv("/kaggle/input/iris/Iris.csv")
df.head()

df.info()
df.describe()
df.head()
df.tail()
df.isnull().sum()
df['Species'].value_counts()

df.plot(kind='scatter',x="SepalLengthCm", y="SepalWidthCm")
df.plot(kind='scatter',x="PetalLengthCm", y="PetalWidthCm")
sns.FacetGrid(df, size=5,hue="Species").map(plt.scatter,"SepalLengthCm","SepalWidthCm").add_l
egend()
sns.FacetGrid(df, size=5,hue="Species").map(plt.scatter,"PetalLengthCm","PetalWidthCm").add_le
gend()

sns.boxplot(x="Species", y="PetalLengthCm", data=df)

sns.boxplot(x="Species", y="PetalWidthCm", data=df)

sns.stripplot(x="Species", y="PetalLengthCm", data=df, jitter=True, edgecolor="gray")

# Distribution density plot KDE (kernel density estimate)
sns.FacetGrid(df, hue="Species", size=6) \
   .map(sns.kdeplot, "PetalLengthCm") \
   .add_legend()

sns.pairplot(df.drop("Id", axis=1), hue="Species", size=4)

from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics

train, test = train_test_split(df, test_size = 0.25,random_state=20)
print(train.shape)
print(test.shape)

train_X=train[['SepalLengthCm','SepalWidthCm','PetalLengthCm','PetalWidthCm']]
train_y=train.Species
test_X=test[['SepalLengthCm','SepalWidthCm','PetalLengthCm','PetalWidthCm']]
test_y=test.Species

train_X.head()

KNN=KNeighborsClassifier(n_neighbors=3)
KNN.fit(train_X,train_y)
prediction=KNN.predict(test_X)
print('The accuracy of the KNN is',metrics.accuracy_score(prediction,test_y)*100,
     'percent')

#Get accuracy. Note: In case of classification algorithms score method represents accuracy.
KNN.score(test_X,test_y)

#import confusion_matrix
from sklearn.metrics import confusion_matrix
prediction=KNN.predict(test_X)
confusion_matrix(test_y,prediction)

import seaborn as sns
cf_matrix=confusion_matrix(test_y,prediction)
sns.heatmap(cf_matrix, annot=True)

 #import classification_report
from sklearn.metrics import classification_report
print(classification_report(test_y,prediction))
```

Result:

The accuracy for KNN is

Exercise 9:

DEMONSTRATE REGRESSION AND CLASSIFICATION METRICS

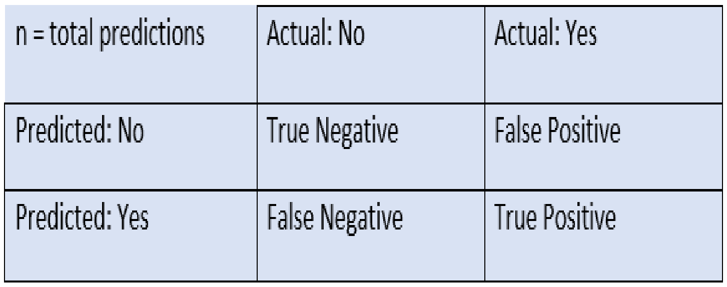
Aim: Demonstrate regression and classification metrics using sample data.

Software Required: Google Co Lab, Jupyter notebook, Kaggle .

Theory:

The confusion matrix is a matrix used to determine the performance of the classification models for a given set of test data. It can only be determined if the true values for test data are known. The matrix itself can be easily understood, but the related terminologies may be confusing. Since it shows the errors in the model performance in the form of a matrix, hence also known as an **error matrix**. Some features of Confusion matrix are given below:

- For the 2 prediction classes of classifiers, the matrix is of 2*2 table, for 3 classes, it is 3*3 table, and so on.
- The matrix is divided into two dimensions, that are **predicted values** and **actual values** along with the total number of predictions.
- Predicted values are those values, which are predicted by the model, and actual values are the true values for the given observations.
- It looks like the below table:



n = total predictions	Actual: No	Actual: Yes
Predicted: No	True Negative	False Positive
Predicted: Yes	False Negative	True Positive

The above table has the following cases:

- **True Negative:** Model has given prediction No, and the real or actual value was also No.
- **True Positive:** The model has predicted yes, and the actual value was also true.
- **False Negative:** The model has predicted no, but the actual value was Yes, it is also called as **Type-II error**.
- **False Positive:** The model has predicted Yes, but the actual value was No. It is also called a **Type-I error.**

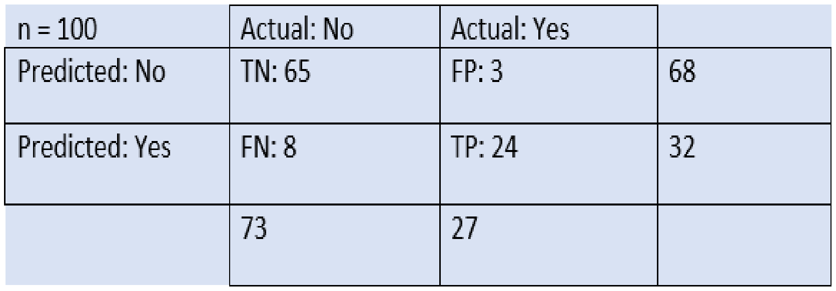
Need for Confusion Matrix in Machine learning

- It evaluates the performance of the classification models, when they make predictions on test data, and tells how good our classification model is.
- It not only tells the error made by the classifiers but also the type of errors such as it is either type-I or type-II error.
- With the help of the confusion matrix, we can calculate the different parameters for the model, such as accuracy, precision, etc.

Example: We can understand the confusion matrix using an example.

Suppose we are trying to create a model that can predict the result for the disease that is either a person has that disease or not. So, the confusion matrix for this is given as:

00:00/02:13



n = 100	Actual: No	Actual: Yes	
Predicted: No	TN: 65	FP: 3	68
Predicted: Yes	FN: 8	TP: 24	32
	73	27	

From the above example, we can conclude that:

- The table is given for the two-class classifier, which has two predictions "Yes" and "NO." Here, Yes defines that patient has the disease, and No defines that patient does not has that disease.
- The classifier has made a total of **100 predictions**. Out of 100 predictions, **89 are true predictions**, and **11 are incorrect predictions**.
- The model has given prediction "yes" for 32 times, and "No" for 68 times. Whereas the actual "Yes" was 27, and actual "No" was 73 times.

Calculations using Confusion Matrix:

We can perform various calculations for the model, such as the model's accuracy, using this matrix. These calculations are given below:

- **Classification Accuracy:** It is one of the important parameters to determine the accuracy of the classification problems. It defines how often the model predicts the correct output. It can be calculated as the ratio of the number of correct predictions made by the classifier to all number of predictions made by the classifiers. The formula is given below:

$$\text{Accuracy}=\frac{TP+TN}{TP+FP+FN+TN}$$

- **Misclassification rate:** It is also termed as Error rate, and it defines how often the model gives the wrong predictions. The value of error rate can be calculated as the number of incorrect predictions to all number of the predictions made by the classifier. The formula is given below:

$$\text{Error rate} = \frac{FP+FN}{TP+FP+FN+TN}$$

- **Precision:** It can be defined as the number of correct outputs provided by the model or out of all positive classes that have predicted correctly by the model, how many of them were actually true. It can be calculated using the below formula:

$$\text{Precision} = \frac{TP}{TP+FP}$$

- **Recall:** It is defined as the out of total positive classes, how our model predicted correctly. The recall must be as high as possible.

$$\text{Recall} = \frac{TP}{TP+FN}$$

- **F-measure:** If two models have low precision and high recall or vice versa, it is difficult to compare these models. So, for this purpose, we can use F-score. This score helps us to evaluate the recall and precision at the same time. The F-score is maximum if the recall is equal to the precision. It can be calculated using the below formula:

$$\text{F-measure} = \frac{2*Recall*Precision}{Recall+Precision}$$

Other important terms used in Confusion Matrix:

- **Null Error rate:** It defines how often our model would be incorrect if it always predicted the majority class. As per the accuracy paradox, it is said that "*the best classifier has a higher error rate than the null error rate.*"
- **ROC Curve:** The ROC is a graph displaying a classifier's performance for all possible thresholds. The graph is plotted between the true positive rate (on the Y-axis) and the false Positive rate (on the x-axis).

Program :

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score,precision_score,recall_score,precision_recall_curve,roc_curve,r
2_score,f1_score
import matplotlib.pyplot as plt

y=[1,1,0,0,1,1,1,1,0,0,1,0,0,0,1,1,0,1]
y1=[1,1,0,0,1,1,1,1,0,0,1,0,0,0,1,1,0,1]
confusion_matrix(y,y1)
print('Accuracy Score: ', accuracy_score(y,y1))
print('Precision: ', precision_score(y,y1))
print('Recall: ', recall_score(y,y1))
print('F1 Score: ',f1_score(y,y1))
print('R2 Score: ',r2_score(y,y1))
print('Precision-Recall Curve: ',precision_recall_curve(y,y1))
print('ROC Curve: ',roc_curve(y,y1))
roc=roc_curve(y,y1)
plt.plot(roc[0],roc[1])

pr=precision_recall_curve(y,y1)
plt.plot(pr[0],pr[1])

y=[1,1,0,0,1,1,1,1,0,0,1,0,0,0,1,1,0,1]
y1=[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
confusion_matrix(y,y1)
print('Accuracy Score: ', accuracy_score(y,y1))
print('Precision: ', precision_score(y,y1))
print('Recall: ', recall_score(y,y1))
print('F1 Score: ',f1_score(y,y1))
print('R2 Score: ',r2_score(y,y1))
print('Precision-Recall Curve: ',precision_recall_curve(y,y1))
print('ROC Curve: ',roc_curve(y,y1))
roc=roc_curve(y,y1)
plt.plot(roc[0],roc[1])

pr=precision_recall_curve(y,y1)
plt.plot(pr[0],pr[1])

y=[1,1,0,0,1,1,1,1,0,0,1,0,0,0,1,1,0,1]
y1=[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1]
confusion_matrix(y,y1)
print('Accuracy Score: ', accuracy_score(y,y1))
print('Precision: ', precision_score(y,y1))
print('Recall: ', recall_score(y,y1))
print('F1 Score: ',f1_score(y,y1))
print('R2 Score: ',r2_score(y,y1))
print('Precision-Recall Curve: ',precision_recall_curve(y,y1))
print('ROC Curve: ',roc_curve(y,y1))
roc=roc_curve(y,y1)
plt.plot(roc[0],roc[1])

pr=precision_recall_curve(y,y1)
plt.plot(pr[0],pr[1])

y=[1,1,0,0,1,1,1,1,0,0,1,0,0,0,1,1,0,1]
y1=[0,0,0,0,1,1,1,1,1,0,1,0,0,0,1,1,1,1]
confusion_matrix(y,y1)
print('Accuracy Score: ', accuracy_score(y,y1))
print('Precision: ', precision_score(y,y1))
print('Recall: ', recall_score(y,y1))
print('F1 Score: ',f1_score(y,y1))
print('R2 Score: ',r2_score(y,y1))
print('Precision-Recall Curve: ',precision_recall_curve(y,y1))
print('ROC Curve: ',roc_curve(y,y1))
roc=roc_curve(y,y1)
plt.plot(roc[0],roc[1])

pr=precision_recall_curve(y,y1)
plt.plot(pr[0],pr[1])

y=[1,1,0,0,1,1,1,1,0,0,1,0,0,0,1,1,0,1]
y1=[0,0,1,1,0,0,0,0,1,1,0,1,1,1,0,0,1,0]
confusion_matrix(y,y1)
print('Accuracy Score: ', accuracy_score(y,y1))
print('Precision: ', precision_score(y,y1))
print('Recall: ', recall_score(y,y1))
print('F1 Score: ',f1_score(y,y1))
print('R2 Score: ',r2_score(y,y1))
print('Precision-Recall Curve: ',precision_recall_curve(y,y1))
print('ROC Curve: ',roc_curve(y,y1))
roc=roc_curve(y,y1)
plt.plot(roc[0],roc[1])

pr=precision_recall_curve(y,y1)
plt.plot(pr[0],pr[1])
```

RESULT: The Performance Metrics for a classifier are

Accuracy=
Precision
Recall=
F1 score=
R2 score=

Exercise 10:

SUPPORT VECTOR MACHINE MODEL

Aim: Build SVM model with various kernels and select best kernel for given dataset.

Software Required: Google Co Lab, Jupyter notebook, Kaggle .

Dataset: https://www.kaggle.com/datasets/uciml/human-activity-recognition-with-smartphones

About Dataset The Human Activity Recognition database was built from the recordings of 30 study participants performing activities of daily living (ADL) while carrying a waist-mounted smartphone with embedded inertial sensors. The objective is to classify activities into one of the six activities performed.

Description of experiment The experiments have been carried out with a group of 30 volunteers within an age bracket of 19-48 years. Each person performed six activities (WALKING, WALKINGUPSTAIRS, WALKINGDOWNSTAIRS, SITTING, STANDING, LAYING) wearing a smartphone (Samsung Galaxy S II) on the waist. Using its embedded accelerometer and gyroscope, we captured 3-axial linear acceleration and 3-axial angular velocity at a constant rate of 50Hz. The experiments have been video-recorded to label the data manually. The obtained dataset has been randomly partitioned into two sets, where 70% of the volunteers was selected for generating the training data and 30% the test data.

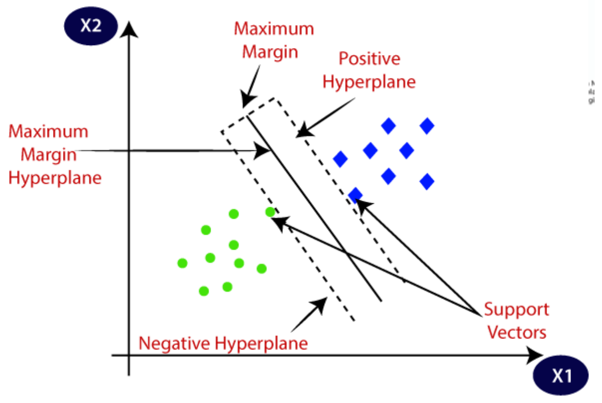
The sensor signals (accelerometer and gyroscope) were pre-processed by applying noise filters and then sampled in fixed-width sliding windows of 2.56 sec and 50% overlap (128 readings/window). The sensor acceleration signal, which has gravitational and body motion components, was separated using a Butterworth low-pass filter into body acceleration and gravity. The gravitational force is assumed to have only low frequency components, therefore a filter with 0.3 Hz cutoff frequency was used. From each window, a vector of features was obtained by calculating variables from the time and frequency domain.

Attribute information
For each record in the dataset the following is provided:
Triaxial acceleration from the accelerometer (total acceleration) and the estimated body acceleration.
Triaxial Angular velocity from the gyroscope.
A 561-feature vector with time and frequency domain variables.
Its activity label.
An identifier of the subject who carried out the experiment.

Theory:
Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine. Consider the below diagram in which there are two different categories that are classified using a decision boundary or hyperplane:



Example: SVM can be understood with the example that we have used in the KNN classifier. Suppose we see a strange cat that also has some features of dogs, so if we want a model that can accurately identify whether it is a cat or dog, so such a model can be created by using the SVM algorithm. We will first train our model with lots of images of cats and dogs so that it can learn about different features of cats and dogs, and then we test it with this strange creature. So as support vector creates a decision boundary between these two data (cat and dog) and choose extreme cases (support vectors), it will see the extreme case of cat and dog. On the basis of the support vectors, it will classify it as a cat.
Consider the below diagram:
Types of SVM
SVM can be of two types:

- Linear SVM: Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.
- Non-linear SVM: Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier.

Hyperplane and Support Vectors in the SVM algorithm:

Hyperplane: There can be multiple lines/decision boundaries to segregate the classes in n-dimensional space, but we need to find out the best decision boundary that helps to classify the data points. This best boundary is known as the hyperplane of SVM.

The dimensions of the hyperplane depend on the features present in the dataset, which means if there are 2 features (as shown in image), then hyperplane will be a straight line. And if there are 3 features, then hyperplane will be a 2-dimension plane.

We always create a hyperplane that has a maximum margin, which means the maximum distance between the data points.

Support Vectors:

The data points or vectors that are the closest to the hyperplane and which affect the position of the hyperplane are termed as Support Vector. Since these vectors support the hyperplane, hence called a Support vector.

Program :

```
#Import required basic libraries
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import os

# Read dataset into dataframe
data=pd.read_csv('/kaggle/input/human-activity-recognition-with-smartphones/train.csv')
data.describe()
data

data.info()

data.columns

data.Activity.value_counts()

data.shape

data['activity_code'] = data.Activity.astype('category').cat.codes

data.shape

data.activity_code

data1=data.drop('Activity',axis=1)
data1.shape

x_col=data1.columns.to_list()
x_col.pop(-1)
x_data=data1[x_col]
y_col='activity_code'
from sklearn.model_selection import train_test_split
train_x,test_x,train_y,test_y = train_test_split(data1[x_col],data1[y_col].values,test_size=0.1)

train_x.shape,test_x.shape,train_y.shape,test_y.shape

test_y

# Try to build Logistic regression
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
lg=LogisticRegression()#max_iter=800, solver='sag')
lg.fit(train_x,train_y)
train_y_pred=lg.predict(train_x)
test_y_pred= lg.predict(test_x)
#
print("Training Accuracy",accuracy_score(train_y,train_y_pred))
print("Testing Accuracy", accuracy_score(test_y,test_y_pred))
```

```
from sklearn.metrics import confusion_matrix
confusion_matrix(train_y,train_y_pred)

confusion_matrix(test_y,test_y_pred)
```

Support Vector Classifier:

SVC(*, C=1.0, kernel="rbf", degree=3, gamma="scale", coef0=0.0, shrinking=True, probability=False, tol=1e-3, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape="ovr", break_ties=False, random_state=None)

Parameters

C : float, default=1.0 Regularization parameter. The strength of the regularization is inversely proportional to C. Must be strictly positive. The penalty is a squared l2 penalty.

kernel : {'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'} or callable, default='rbf' Specifies the kernel type to be used in the algorithm. If none is given, 'rbf' will be used. If a callable is given it is used to pre-compute the kernel matrix from data matrices; that matrix should be an array of shape (n_samples, n_samples).

degree : int, default=3 Degree of the polynomial kernel function ('poly'). Ignored by all other kernels.

gamma : {'scale', 'auto'} or float, default='scale' Kernel coefficient for 'rbf', 'poly' and 'sigmoid'.

coef0 : float, default=0.0 Independent term in kernel function. It is only significant in 'poly' and 'sigmoid'.

shrinking : bool, default=True Whether to use the shrinking heuristic. See the :ref:User Guide <shrinking_svm>.

probability : bool, default=False Whether to enable probability estimates. This must be enabled prior to calling fit, will slow down that method as it internally uses 5-fold cross-validation, and predict_proba may be inconsistent with predict. Read more in the :ref:User Guide <scores_probabilities>.

tol : float, default=1e-3 Tolerance for stopping criterion.

cache_size : float, default=200 Specify the size of the kernel cache (in MB).

class_weight :dict or 'balanced', default=None Set the parameter C of class i to class_weight[i]*C for SVC. If not given, all classes are supposed to have weight one. The "balanced" mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data as n_samples / (n_classes * np.bincount(y)).

verbose : bool, default=False Enable verbose output. Note that this setting takes advantage of a per-process runtime setting in libsvm that, if enabled, may not work properly in a multithreaded context.

max_iter : int, default=-1 Hard limit on iterations within solver, or -1 for no limit.

decision_function_shape : {'ovo', 'ovr'}, default='ovr' Whether to return a one-vs-rest ('ovr') decision function of shape (n_samples, n_classes) as all other classifiers, or the original one-vs-one ('ovo') decision function of libsvm which has shape (n_samples, n_classes * (n_classes - 1) / 2). However, one-vs-one ('ovo') is always used as multi-class strategy. The parameter is ignored for binary classification.

break_ties : bool, default=False If true, decision_function_shape='ovr', and number of classes > 2, :term:predict will break ties according to the confidence values of :term:decision_function;

otherwise the first class among the tied classes is returned. Please note that breaking ties comes at a relatively high computational cost compared to a simple predict.

random_state : int, RandomState instance or None, default=None Controls the pseudo random number generation for shuffling the data for probability estimates. Ignored when probability is False. Pass an int for reproducible output across multiple function calls. See :term:Glossary<random_state>.

```
# Build SVM model using svm.SVC with default parameters. Default kernel is rbl
from sklearn.svm import SVC
sv=SVC() #default kernel is 'rbl'
sv.fit(train_x,train_y)
train_y_pred=sv.predict(train_x)
test_y_pred= sv.predict(test_x)
#
print("Training Accuracy",accuracy_score(train_y,train_y_pred))
print("Testing Accuracy", accuracy_score(test_y,test_y_pred))


# Build SVM model using svm.SVC with Linear Kernel
from sklearn.svm import SVC
sv=SVC(kernel='linear')
sv.fit(train_x,train_y)
train_y_pred=sv.predict(train_x)
test_y_pred= sv.predict(test_x)
#
print("Training Accuracy",accuracy_score(train_y,train_y_pred))
print("Testing Accuracy", accuracy_score(test_y,test_y_pred))

# Build SVM model using svm.SVC with Polynomial kernel
from sklearn.svm import SVC
sv=SVC(kernel='poly')
sv.fit(train_x,train_y)
train_y_pred=sv.predict(train_x)
test_y_pred= sv.predict(test_x)
#
print("Training Accuracy",accuracy_score(train_y,train_y_pred))
print("Testing Accuracy", accuracy_score(test_y,test_y_pred))



# Build SVM model using svm.SVC with Sigmoid kernel
from sklearn.svm import SVC
sv=SVC(kernel='sigmoid')
sv.fit(train_x,train_y)
train_y_pred=sv.predict(train_x)
test_y_pred= sv.predict(test_x)
#
print("Training Accuracy",accuracy_score(train_y,train_y_pred))
print("Testing Accuracy", accuracy_score(test_y,test_y_pred))

# Build SVM model using svm.SVC with Sigmoid kernel and gamma as auto
from sklearn.svm import SVC
sv=SVC(kernel='sigmoid',gamma='auto')
sv.fit(train_x,train_y)
```

```
train_y_pred=sv.predict(train_x)
test_y_pred= sv.predict(test_x)
#
print("Training Accuracy",accuracy_score(train_y,train_y_pred))
print("Testing Accuracy", accuracy_score(test_y,test_y_pred))
```

RESULT:

Training Accuracy of SVC without Kernel is
Testing Accuracy of RFC without Kernel is

Training Accuracy of SVC with linear Kernel is
Testing Accuracy of SCV with linear Kernel is

Training Accuracy of SVC with poly Kernel
Testing Accuracy of SVC with Poly Kernel

Training Accuracy of SVC with sigmoid Kernel
Testing Accuracy of SVC with sigmoid Kernel
Training Accuracy of SVC with sigmoid Kernel and gamma auto is
Testing Accuracy of SVC with sigmoid Kernel and gamma auto is

Exercise 11:

RANDOM FOREST MODEL

Aim: Build Random Forest model and apply on given dataset. Evaluate the model with suitable metrics.

Software Required: Google Co Lab, Jupyter notebook, Kaggle .

Dataset: https://www.kaggle.com/datasets/uciml/human-activity-recognition-with-smartphones

About Dataset The Human Activity Recognition database was built from the recordings of 30 study participants performing activities of daily living (ADL) while carrying a waist-mounted smartphone with embedded inertial sensors. The objective is to classify activities into one of the six activities performed.

Description of experiment The experiments have been carried out with a group of 30 volunteers within an age bracket of 19-48 years. Each person performed six activities (WALKING, WALKINGUPSTAIRS, WALKINGDOWNSTAIRS, SITTING, STANDING, LAYING) wearing a smartphone (Samsung Galaxy S II) on the waist. Using its embedded accelerometer and gyroscope, we captured 3-axial linear acceleration and 3-axial angular velocity at a constant rate of 50Hz. The experiments have been video-recorded to label the data manually. The obtained dataset has been randomly partitioned into two sets, where 70% of the volunteers was selected for generating the training data and 30% the test data.

The sensor signals (accelerometer and gyroscope) were pre-processed by applying noise filters and then sampled in fixed-width sliding windows of 2.56 sec and 50% overlap (128 readings/window). The sensor acceleration signal, which has gravitational and body motion components, was separated using a Butterworth low-pass filter into body acceleration and gravity. The gravitational force is assumed to have only low frequency components, therefore a filter with 0.3 Hz cutoff frequency was used. From each window, a vector of features was obtained by calculating variables from the time and frequency domain.

Attribute information
For each record in the dataset the following is provided:
Triaxial acceleration from the accelerometer (total acceleration) and the estimated body acceleration.
Triaxial Angular velocity from the gyroscope.
A 561-feature vector with time and frequency domain variables.
Its activity label.
An identifier of the subject who carried out the experiment.

Theory:

Program :

```
#Import required basic libraries
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import os
```

```
# Read dataset into dataframe
data=pd.read_csv('/kaggle/input/human-activity-recognition-with-smartphones/train.csv')
```

data.describe()

data

data.info()

data.columns

data.Activity.value_counts()

```
data.shape

data['activity_code'] = data.Activity.astype('category').cat.codes


data.shape

data.activity_code

data1=data.drop('Activity',axis=1)

data.info()

data1.shape

x_col=data1.columns.to_list()
x_col.pop(-1)
x_data=data1[x_col]
y_col='activity_code'
from sklearn.model_selection import train_test_split
train_x,test_x,train_y,test_y=train_test_split(data1[x_col],data1[y_col].values,test_size=0.1)

train_x.shape,test_x.shape,train_y.shape,test_y.shape

test_y

# Try to build Logistic regression
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
lg=LogisticRegression()#max_iter=800, solver='sag')
lg.fit(train_x,train_y)
train_y_pred=lg.predict(train_x)
test_y_pred= lg.predict(test_x)
#
print("Training Accuracy",accuracy_score(train_y,train_y_pred))
print("Testing Accuracy", accuracy_score(test_y,test_y_pred))

from sklearn.metrics import confusion_matrix
confusion_matrix(train_y,train_y_pred)

confusion_matrix(test_y,test_y_pred)
```

Random Forest Classifier:

RandomForestClassifier(n_estimators=100, *, criterion="gini", max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features="auto", max_leaf_nodes=None, min_impurity_decrease=0.0, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, class_weight=None, ccp_alpha=0.0, max_samples=None)

A random forest classifier.

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is controlled with the max_samples parameter if bootstrap=True (default), otherwise the whole dataset is used to build each tree.

Parameters

n_estimators : int, default=100 The number of trees in the forest.

criterion : {"gini", "entropy"}, default="gini" The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain. Note: this parameter is tree-specific.

max_depth : int, default=None The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

min_samples_split : int or float, default=2 The minimum number of samples required to split an internal node:

min_samples_leaf : int or float, default=1 The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least min_samples_leaf training samples in each of the left and right branches. This may have the effect of smoothing the model, especially in regression.

min_weight_fraction_leaf : float, default=0.0 The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node. Samples have equal weight when sample_weight is not provided.

max_features : {"auto", "sqrt", "log2"}, int or float, default="auto" The number of features to consider when looking for the best split:

max_leaf_nodes : int, default=None Grow trees with max_leaf_nodes in best-first fashion. Best nodes are defined as relative reduction in impurity. If None then unlimited number of leaf nodes.

min_impurity_decrease : float, default=0.0 A node will be split if this split induces a decrease of the impurity greater than or equal to this value.

bootstrap : bool, default=True Whether bootstrap samples are used when building trees. If False, the whole dataset is used to build each tree.

oob_score : bool, default=False Whether to use out-of-bag samples to estimate the generalization score. Only available if bootstrap=True.

n_jobs : int, default=None The number of jobs to run in parallel. :meth:fit, :meth:predict, :meth:decision_path and :meth:apply are all parallelized over the trees. None means 1 unless in a :obj:joblib.parallel_backend context. -1 means using all processors. See :term:Glossary<n_jobs> for more details.

random_state : int, RandomState instance or None, default=None Controls both the randomness of the bootstrapping of the samples used when building trees (if bootstrap=True) and the sampling of the features to consider when looking for the best split at each node (if max_features<n_features). See :term:Glossary<random_state> for details.

verbose : int, default=0 Controls the verbosity when fitting and predicting.

warm_start : bool, default=False When set to True, reuse the solution of the previous call to fit and add more estimators to the ensemble, otherwise, just fit a whole new forest. See :term:the Glossary <warm_start>.

class_weight : {"balanced", "balanced_subsample"}, dict or list of dicts, default=None Weights associated with classes in the form {class_label: weight}. If not given, all classes are supposed to have weight one. For multi-output problems, a list of dicts can be provided in the same order as the columns of y.

ccp_alpha : non-negative float, default=0.0 Complexity parameter used for Minimal Cost-Complexity Pruning. The subtree with the largest cost complexity that is smaller than ccp_alpha will be chosen. By default, no pruning is performed.

max_samples : int or float, default=None If bootstrap is True, the number of samples to draw from X to train each base estimator.

Program:

```
# Build Random Forest model using ensemble.RandomForestClassifier
from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier()
rf.fit(train_x,train_y)
train_y_pred=rf.predict(train_x)
test_y_pred= rf.predict(test_x)
#
print("Training Accuracy",accuracy_score(train_y,train_y_pred))
print("Testing Accuracy", accuracy_score(test_y,test_y_pred))


# Build Random Forest model using ensemble.RandomForestClassifier
from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier(criterion='entropy')
rf.fit(train_x,train_y)
train_y_pred=rf.predict(train_x)
test_y_pred= rf.predict(test_x)
#
print("Training Accuracy",accuracy_score(train_y,train_y_pred))
print("Testing Accuracy", accuracy_score(test_y,test_y_pred))

# Build Random Forest model using ensemble.RandomForestClassifier
from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier(max_depth=5)
rf.fit(train_x,train_y)
train_y_pred=rf.predict(train_x)
test_y_pred= rf.predict(test_x)
#
print("Training Accuracy",accuracy_score(train_y,train_y_pred))
print("Testing Accuracy", accuracy_score(test_y,test_y_pred))

# Build Random Forest model using ensemble.RandomForestClassifier
from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier(max_depth=10)
rf.fit(train_x,train_y)


train_y_pred=rf.predict(train_x)
test_y_pred= rf.predict(test_x)
```

```
#
print("Training Accuracy",accuracy_score(train_y,train_y_pred))
print("Testing Accuracy", accuracy_score(test_y,test_y_pred))
# Build Random Forest model using ensemble.RandomForestClassifier
from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier(max_features='sqrt')
rf.fit(train_x,train_y)
train_y_pred=rf.predict(train_x)
test_y_pred= rf.predict(test_x)
#
print("Training Accuracy",accuracy_score(train_y,train_y_pred))
print("Testing Accuracy", accuracy_score(test_y,test_y_pred))
```

RESULT: The training and Testing accuracy are

Training Accuracy of RFC without entropy criteria is
Testing Accuracy of RFC without entropy criteria is
Training Accuracy of RFC with entropy criteria is
Testing Accuracy of RFC with entropy criteria is

Training Accuracy of RFC with max depth of 5is
Testing Accuracy of RFC with max depth of 5is

Training Accuracy of RFC with max depth of 10 is
Testing Accuracy of RFC with max depth of 10 is

Training Accuracy of RFC with max feature of sqrt is
Testing Accuracy of RFC with max feature of sqrt is