MALLA REDDY COLLEGE OF ENGINEERING

R22 B.Tech. CSE (AI and ML) Syllabus JNTU Hyderabad

AM505PC: MACHINE LEARNING LAB MANUAL

B.Tech. III Year I Sem. L T P C
0 0 2 1

Course Objective:

 The objective of this lab is to get an overview of the various machine learning techniques and
can demonstrate them using python.

Course Outcomes:

 Understand modern notions in predictive data analysis
 Select data, model selection, model complexity and identify the trends
 Understand a range of machine learning algorithms along with their strengths and
weaknesses
 Build predictive models from data and analyze their performance

List of Experiments

1. Write a python program to compute Central Tendency Measures: Mean, Median,
Mode Measure of Dispersion: Variance, Standard Deviation
2. Study of Python Basic Libraries such as Statistics, Math, Numpy and Scipy
3. Study of Python Libraries for ML application such as Pandas and Matplotlib
4. Write a Python program to implement Simple Linear Regression
5. Implementation of Multiple Linear Regression for House Price Prediction using sklearn

AIM: a python program to compute Central Tendency Measures: Mean, Median

PROCEDURE: to compute central tendency measures mean median mode measure of dispersion variance, standard deviation

PROGRAM:

```
import statistics
def calculate_mean(data):
return sum(data) / len(data)
def calculate_median(data):
sorted_data = sorted(data)
  n = len(sorted_data)
  if n % 2 == 0:
 middle1 = sorted_data[n // 2 - 1]
 middle2 = sorted_data[n // 2]
  return (middle1 + middle2) / 2
  else:
   return sorted_data[n // 2]
def calculate_mode(data):
  return statistics.mode(data)
def calculate_variance(data):
 mean_value = calculate_mean(data)
  squared_diff_sum = sum((x - mean_value) ** 2 for x in data)
 return squared_diff_sum / (len(data) - 1)
def calculate_standard_deviation(data):
variance_value = calculate_variance(data)
  return variance_value ** 0.5
# Example dataset
dataset = [10, 20, 30, 40, 50]
mean_value = calculate_mean(dataset)
```

```
median_value = calculate_median(dataset)
mode_value = calculate_mode(dataset)
variance_value = calculate_variance(dataset)
std_deviation_value = calculate_standard_deviation(dataset)
print(f"Dataset: {dataset}")
print(f"Mean: {mean_value:.2f}")
print(f"Median: {median_value:.2f}")
print(f"Mode: {mode_value}")
print(f"Variance: {variance_value:.2f}")
print(f"Standard Deviation: {std_deviation_value:.2f}")
```

output : Dataset: [10, 20, 30, 40, 50]

Mean: 30.00

Median: 30.00

Mode: 10

Variance: 250.00

Standard Deviation: 15.81

OUTPUT:

Dataset: [10, 20, 30, 40, 50]

Mean: 30.00

Median: 30.00

Mode: 10

Variance: 250.00

Standard Deviation: 15.81

RESULT: calculating the values of mean mode and variance and standard deviation for sample data set

2. Study of Python Basic Libraries such as Statistics, Math, Numpy and Scipy

AIM: Study of Python Basic Libraries such as Statistics, Math, Numpy and Scipy

PROCEDURE: for python basic libraries by import command use the necessary libraries to develop various designs which we can solve complex mathematical problems.

PROGRAM:

Study of library such as statistics and math function

1. **Math**:
    - The built-in `math` module provides various mathematical functions and constants.
    - It includes functions for trigonometry, logarithms, exponentiation, and more.

```
import math

# Calculate the square root

sqrt_value = math.sqrt(25)

print(f"Square root of 25: {sqrt_value:.2f}")

# Compute the factorial

factorial_value = math.factorial(5)

print(f"Factorial of 5: {factorial_value}")

# Calculate the sine of an angle (in radians)

angle_radians = math.radians(30)

sine_value = math.sin(angle_radians)

print(f"Sine of 30 degrees: {sine_value:.2f}")
```

NumPy (Numerical Python):

- NumPy is a fundamental library for numerical computations in Python.
- It provides support for multi-dimensional arrays (ndarrays) and efficient operations on these arrays.

Python

```
import numpy as np
# Create a 1D array
arr1d = np.array([1, 2, 3])
print(f"1D Array: {arr1d}")
# Create a 2D array
arr2d = np.array([[1, 2, 3], [4, 5, 6]])
print(f"2D Array: {arr2d}")
```

Example usage:

Python

```
import numpy as np#
#Create a 1D array
arr1d = np.array([1, 2, 3])
print(f"1D Array: {arr1d}")
# Create a 2D array
arr2d = np.array([[1, 2, 3], [4, 5, 6]])
print(f"2D Array: {arr2d}")
```

SciPy (Scientific Python):

- SciPy builds upon NumPy and provides additional functionality for scientific and engineering purposes.
- It includes modules for optimization, integration, linear algebra, signal processing, and more.

Python

```
import scipy.optimize as opt
#Solve an optimization problem
def objective(x):
    return x[0]**2 + x[1]**2
result = opt.minimize(objective, [1, 1])
print(f"Optimal solution: {result.x}")
```

```
# Perform numerical Integration
from scipy.integrate import quad
def integrand(x):
    return x**2
area, error = quad(integrand, 0, 2)
print(f"Integral result: {area:.2f}")
```

OUTPUT:

Calculate the square root

```
Square root of 25: 5.00
Factorial of 5: 120
Sine of 30 degrees: 0.50
```

Importing NumPy library

```
1D Array: [1 2 3]
2D Array: [[1 2 3]
           [4 5 6]]
```

Importing SciPy library

```
Optimal solution: [-1.07505143e-08 -1.07505143e-08]
Integral result: 2.67
```

RESULT: Importing libraries from python and basic calculation and implementation on sample values

3. study of python libraries for ml application such as pandas and matplotlib

AIM: procedure for importing pandas and matplotlib features from python

PROCEDURE:

Pandas is a powerful library for data manipulation and analysis. It provides data structures (such as DataFrames and Series) to handle structured data efficiently.

including Pandas and Matplotlib. These libraries play a crucial role in data manipulation, visualization, and analysis within the ML ecosystem.

Key Features:

Data cleaning: Pandas allows you to clean, transform, and preprocess data easily.

Data exploration: You can explore datasets, filter rows, and perform aggregations.

Missing data handling: Pandas provides tools to handle missing values.

Integration with other libraries: It seamlessly integrates with other ML libraries.

PROGRAM:

essential Python libraries for Machine Learning (ML) applications,

Example:

1. Pandas

```
import pandas as pd
# Create a DataFrame
data = {'Name': ['Alice', 'Bob', 'Charlie'],
'Age': [25, 30, 22]}
df = pd.DataFrame(data)
#       Display the DataFrame
print(df)
```

2. Matplotlib:

Purpose: Matplotlib is a popular plotting library for creating visualizations in Python.

Key Features: Line plots, scatter plots, bar charts, histograms, etc

Customization: You can control colors, labels, and other plot properties.

Integration with Pandas: Matplotlib works seamlessly with Pandas DataFrames.

Example (creating a simple line plot):

Matplotlib libraries import from Python for drawing sample graph

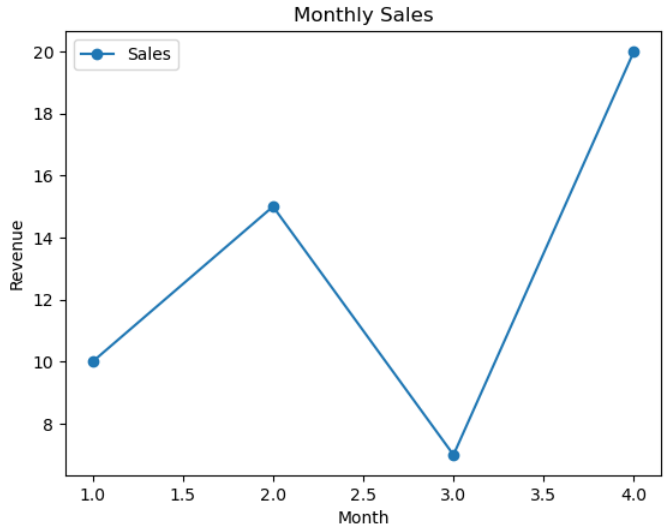
```
import matplotlib.pyplot as plt
# Sample data
x = [1, 2, 3, 4]
y = [10, 15, 7, 20]
# Create a line plot
plt.plot(x, y, marker='o', label='Sales')
plt.xlabel('Month')
plt.ylabel('Revenue')
plt.title('Monthly Sales')
plt.legend()
plt.show()
```

OUTPUT:

Importing pandas from python libraries for sample values

```
      Name   Age
0    Alice   25
1      Bob   30
2  Charlie   22
```

Importing matplot from python libraries for sample values



RESULT: Importing libraries from python and basic calculation and plotting graph on sample values

4. write a python program to implement simple linear regression

AIM: procedure for drawing a decision line in python with linear regression technique

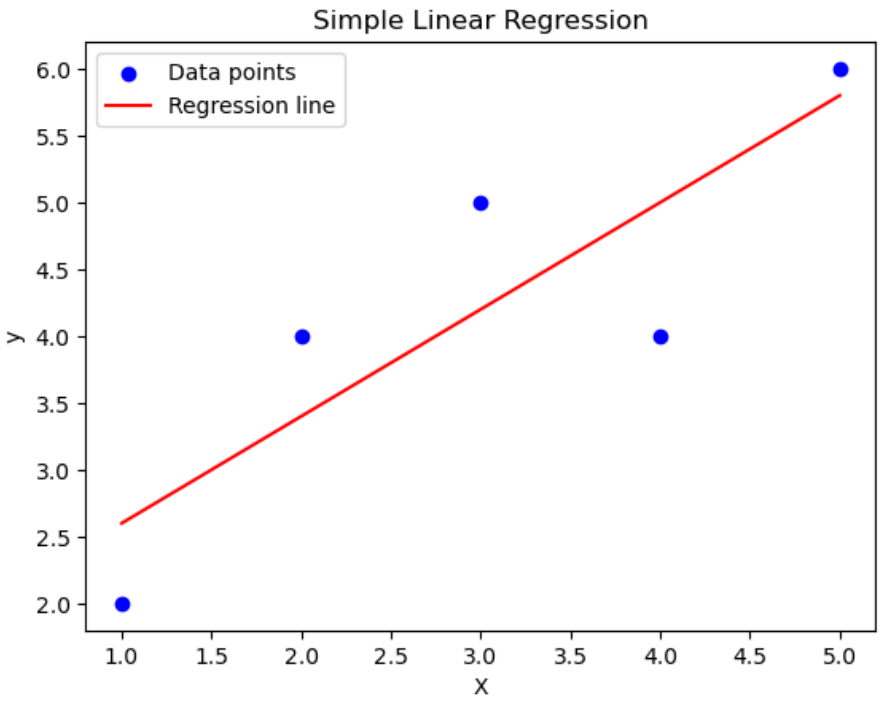
PROCEDURE: in machine learning for predicting the output which samples are trained to machine decision line is vital for take the output consideration for acquiring results how decision line is used and implementation of libraries is as follows.

PROGRAM:

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

# Generate some sample data
X = np.array([[1], [2], [3], [4], [5]]) # Independent variable (feature)
y = np.array([2, 4, 5, 4, 6]) # Dependent variable (response)

# Create a linear regression model
reg = LinearRegression().fit(X, y)

# Get the coefficients and intercept
slope = reg.coef_[0]
intercept = reg.intercept_
print(f"Slope (coefficient): {slope:.2f}")
print(f"Intercept: {intercept:.2f}")

# Predict a new value
new_X = np.array([[6]])
predicted_y = reg.predict(new_X)
```

```
print(f"Predicted value for X = 6: {predicted_y[0]:.2f}")


# Plot the data and regression line
plt.scatter(X, y, color='blue', label='Data points')
plt.plot(X, reg.predict(X), color='red', label='Regression line')
plt.xlabel('X')
plt.ylabel('y')
plt.title('Simple Linear Regression')
plt.legend()
plt.show()
```

OUTPUT:



RESULT: Importing libraries from python and basic calculation and plotting graph on sample values a python program to implement simple linear regression

5.implementation of multiple linear regression for house price prediction using sklearn.

AIM: procedure for drawing a multiple linear regression for house price prediction using sklearn.

PROCEDURE: in machine learning for predicting the output which samples are trained to machine decision line is vital for take the output consideration for multiple linear regression for house price prediction using sklearn.

PROGRAM:

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

# importing data
df = pd.read_csv('Real-estate1.csv')
df.drop('No', inplace=True, axis=1)
  print(df.head())
 print(df.columns)

# plotting a scatterplot
sns.scatterplot(x='X4 number of convenience stores',
                y='Y house price of unit area', data=df)
# creating feature variables
X = df.drop('Y house price of unit area', axis=1)
```

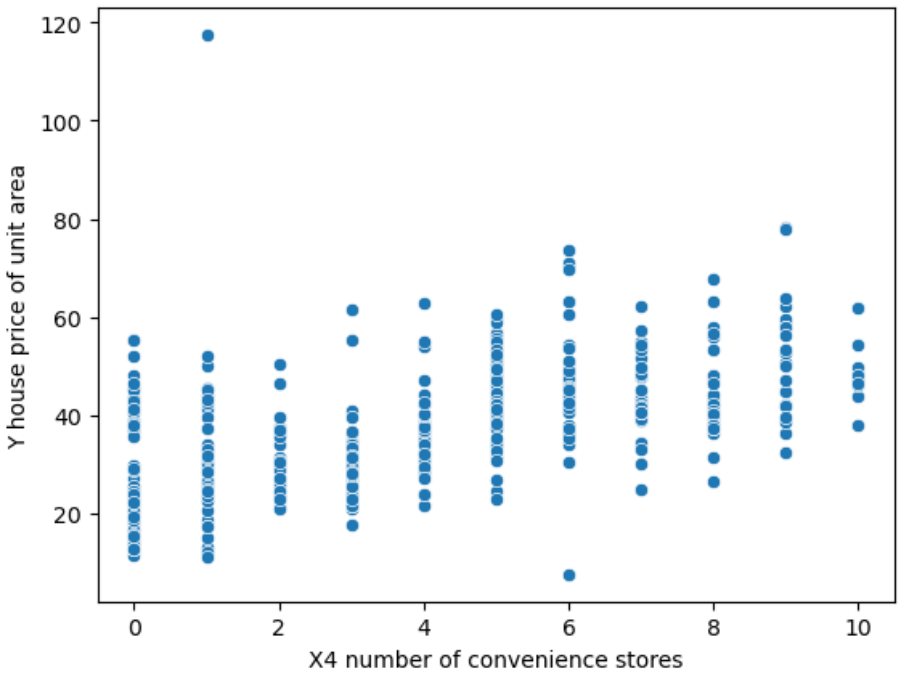
```
y = df['Y house price of unit area']

print(X)
print(y)

# creating train and test sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=101)

# creating a regression model
model = LinearRegression()

# fitting the model
model.fit(X_train, y_train)

# making predictions
predictions = model.predict(X_test)


# model evaluation
print('mean_squared_error : ', mean_squared_error(y_test,
predictions))
print('mean_absolute_error : ', mean_absolute_error(y_test,
predictions))
```

OUTPUT:

```
X1 transaction date  X2 house age  X3 distance to the nearest MRT stati
on  \
0             2012.917          32.0                                  84.
87882
1             2012.917          19.5                                 306.
59470
2             2013.583          13.3                                 561.
98450
3             2013.500          13.3                                 561.
98450
4             2012.833           5.0                                 390.
56840

   X4 number of convenience stores  X5 latitude  X6 longitude  \
```

```
0                                10     24.98298     121.54024
1                                 9     24.98034     121.53951
2                                 5     24.98746     121.54391
3                                 5     24.98746     121.54391
4                                 5     24.97937     121.54245

   Y house price of unit area
0                        37.9
1                        42.2
2                        47.3
3                        54.8
4                        43.1
Index(['X1 transaction date', 'X2 house age',
       'X3 distance to the nearest MRT station',
       'X4 number of convenience stores', 'X5 latitude', 'X6 longitude'
,
       'Y house price of unit area'],
      dtype='object')
     X1 transaction date  X2 house age  \
0               2012.917          32.0
1               2012.917          19.5
2               2013.583          13.3
3               2013.500          13.3
4               2012.833           5.0
..                   ...           ...
409             2013.000          13.7
410             2012.667           5.6
411             2013.250          18.8
412             2013.000           8.1
413             2013.500           6.5

     X3 distance to the nearest MRT station  X4 number of convenience s
tores  \
0                                  84.87882
10
1                                 306.59470
9
2                                 561.98450
5
3                                 561.98450
5
4                                 390.56840
5
..                                      ...
...
409                              4082.01500
0
410                                90.45606
9
411                               390.96960
7
412                               104.81010
5
413                                90.45606
9

     X5 latitude  X6 longitude
0       24.98298     121.54024
```

```
1       24.98034     121.53951
2       24.98746     121.54391
3       24.98746     121.54391
4       24.97937     121.54245
..           ...           ...
409     24.94155     121.50381
410     24.97433     121.54310
411     24.97923     121.53986
412     24.96674     121.54067
413     24.97433     121.54310

[414 rows x 6 columns]
0      37.9
1      42.2
2      47.3
3      54.8
4      43.1
       ...
409    15.4
410    50.0
411    40.6
412    52.5
413    63.9
Name: Y house price of unit area, Length: 414, dtype: float64
mean_squared_error :  46.211797834938665
mean_absolute_error :  5.39229368475692
```



RESULT: after importing libraries sample data is trained and imported from csv file for predicted expected target data by extravagating math formulae.

6. implementation of decision tree using sklearn and its parameters tuning.

AIM: procedure for decision tree parameters and its tuning by sklearn libraries

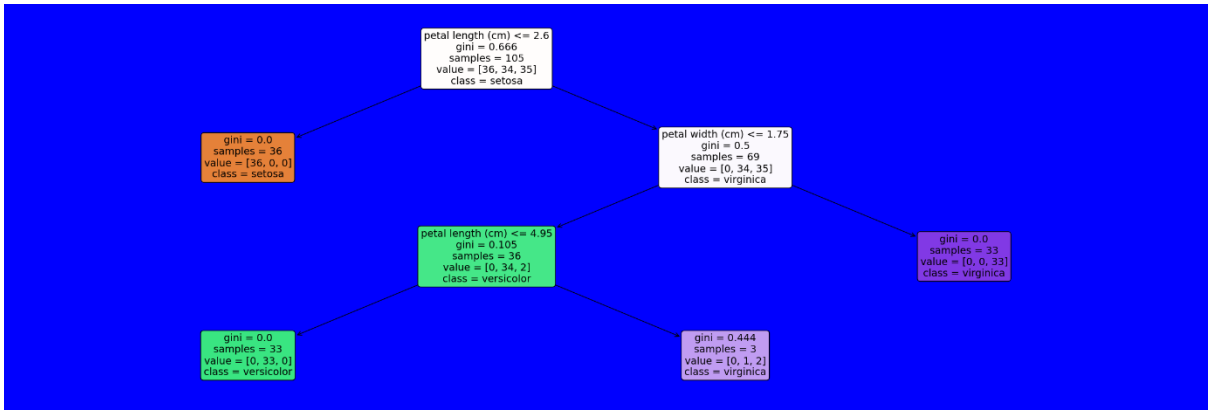
PROCEDURE: in machine learning for predicting the output which samples are trained to machine decision line is vital for take the output consideration for Decision tree for house price prediction using sklearn.

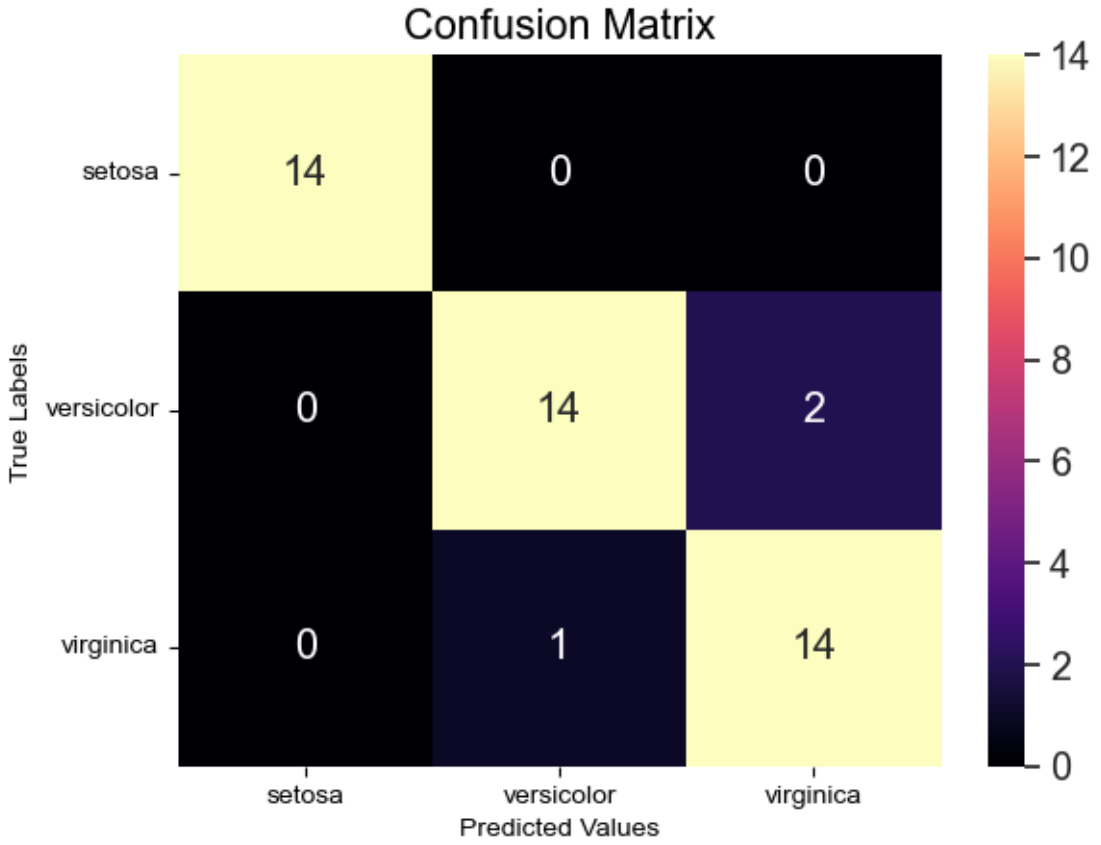
PROGRAM:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import metrics
import seaborn as sns
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn import tree

# Loading the dataset
iris = load_iris()

#converting the data to a pandas dataframe
data = pd.DataFrame(data = iris.data, columns = iris.feature_names)

#creating a separate column for the target variable of iris dataset
data['Species'] = iris.target

#replacing the categories of target variable with the actual names of the species
target = np.unique(iris.target)
target_n = np.unique(iris.target_names)
target_dict = dict(zip(target, target_n))
data['Species'] = data['Species'].replace(target_dict)

# Separating the independent dependent variables of the dataset
x = data.drop(columns = "Species")
y = data["Species"]
names_features = x.columns
target_labels = y.unique()

# Splitting the dataset into training and testing datasets
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.3,
random_state = 93)

# Importing the Decision Tree classifier class from sklearn
from sklearn.tree import DecisionTreeClassifier

# Creating an instance of the classifier class
dtc = DecisionTreeClassifier(max_depth = 3, random_state = 93)

# Fitting the training dataset to the model
dtc.fit(x_train, y_train)

# Plotting the Decision Tree
plt.figure(figsize = (30, 10), facecolor = 'b')
Tree = tree.plot_tree(dtc, feature_names = names_features, class_names =
target_labels, rounded = True, filled = True, fontsize = 14)
plt.show()
y_pred = dtc.predict(x_test)

# Finding the confusion matrix
confusion_matrix = metrics.confusion_matrix(y_test, y_pred)
matrix = pd.DataFrame(confusion_matrix)
axis = plt.axes()
sns.set(font_scale = 1.3)
plt.figure(figsize = (10,7))

# Plotting heatmap
sns.heatmap(matrix, annot = True, fmt = "g", ax = axis, cmap = "magma")
axis.set_title('Confusion Matrix')
axis.set_xlabel("Predicted Values", fontsize = 10)
axis.set_xticklabels([''] + target_labels)
axis.set_ylabel( "True Labels", fontsize = 10)
axis.set_yticklabels(list(target_labels), rotation = 0)
plt.show()
```

Expected Output:





RESULT: after importing libraries sample data is trained from samples for predicted expected parameter data by extravagating its tuning.

7. implementation of knn using sklearn

AIM: procedure for K- Nearest Neighbour values detecting by sklearn libraries

PROCEDURE: in machine learning for predicting the output which samples are trained to machine decision line is vital for take the output consideration for K-NN for data csv format file using sklearn.

PROGRAM:

implementation of knn using sklearn

```
# Step 1: Importing the required Libraries
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt
import seaborn as sns

# Step 2: Reading the Dataset

df = pd.read_csv('data.csv') # Replace 'data.csv' with your dataset e path
y = df['diagnosis']
X = df.drop(['diagnosis', 'Unnamed: 32', 'id'], axis=1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)

# Step 3: Training the model
K = []
training = []
test = []
scores = {}
for k in range(2, 21):
        clf = KNeighborsClassifier(n_neighbors=k)
        clf.fit(X_train, y_train)
        training_score = clf.score(X_train, y_train)
        test_score = clf.score(X_test, y_test)
        K.append(k)
        training.append(training_score)
        test.append(test_score)
        scores[k] = [training_score, test_score]

# Step 4: Evaluating the model

for k, values in scores.items():
        print(f"{k}: Training Score = {values[0]:.2f}, Test Score = {values[1]:.2f}")
```

```
# Step 5: Plotting the training and test scores
ax = sns.stripplot(K, training)
ax.set(xlabel='Values of k', ylabel='Training Score')
plt.show()

ax = sns.stripplot(K, test)
ax.set(xlabel='Values of k', ylabel='Test Score')
plt.show()

plt.scatter(K, training, color='k', label='Training Score')
plt.scatter(K, test, color='g', label='Test Score')
plt.legend()
plt.show()
```

output:

2: Training Score = 0.97, Test Score = 0.92
3: Training Score = 0.96, Test Score = 0.94
4: Training Score = 0.95, Test Score = 0.94
5: Training Score = 0.95, Test Score = 0.94
6: Training Score = 0.94, Test Score = 0.94
7: Training Score = 0.94, Test Score = 0.94
8: Training Score = 0.94, Test Score = 0.94
9: Training Score = 0.94, Test Score = 0.94
10: Training Score = 0.94, Test Score = 0.94
11: Training Score = 0.94, Test Score = 0.94
12: Training Score = 0.94, Test Score = 0.94
13: Training Score = 0.94, Test Score = 0.94
14: Training Score = 0.94, Test Score = 0.94
15: Training Score = 0.94, Test Score = 0.94
16: Training Score = 0.94, Test Score = 0.94
17: Training Score = 0.94, Test Score = 0.94
18: Training Score = 0.94, Test Score = 0.94
19: Training Score = 0.94, Test Score = 0.94
20: Training Score = 0.94, Test Score = 0.94

RESULT: after importing libraries sample data is trained from samples for predicted expected parameter data by extravagating its tuning.

8. implementation of logic regression using sklearn

AIM: procedure for logic regrerssion by sklearn libraries

PROCEDURE: in machine learning for predicting the output which samples are trained to machine decision line is vital for take the output consideration for logic regression for data csv format file using sklearn.

PROGRAM:

implement logic regression using sklearn

```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report

# Load your dataset (replace 'data.csv' with your own data)
data = pd.read_csv('data.csv')
X = data.drop(columns=['target_column'])
y = data['target_column']

# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Create a Logistic Regression model
model = LogisticRegression()
# Fit the model with training data
model.fit(X_train, y_train)

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

# Evaluate model performance
print(f"Accuracy: {accuracy_score(y_test, y_pred):.2f}")
print(classification_report(y_test, y_pred))
```

output:

```
Accuracy: 0.95
              precision    recall  f1-score   support

     Class A       0.94      0.97      0.96        30
     Class B       0.97      0.94      0.95        35

    accuracy                           0.95        65
   macro avg       0.95      0.95      0.95        65
weighted avg       0.95      0.95      0.95        65
```

RESULT: after importing libraries sample data is trained from samples for predicted expected values towards sample csv file.

9. implementation of k means clustering

AIM: procedure for k means clustering

PROCEDURE: in machine learning for predicting the output which samples are trained to machine decision line is vital for take the output consideration for k means for samples under clustering method.

PROGRAM:

implementation of k means clustering

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

# Load the dataset (replace with your own data)
data = pd.read_csv('your_dataset.csv') # Replace 'your_dataset.csv' with your file path
X = data[['feature1', 'feature2']] # Select relevant features

# Create and fit the K-Means model
kmeans = KMeans(n_clusters=3)

 # Choose the number of clusters
kmeans.fit(X)

# Get cluster centers and labels
cluster_centers = kmeans.cluster_centers_
labels = kmeans.labels_

# Visualize the clusters
plt.scatter(X['feature1'], X['feature2'], c=labels, cmap='rainbow')
plt.scatter(cluster_centers[:, 0], cluster_centers[:, 1], color='black', marker='x', s=100)
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')

plt.title('K-Means Clustering')

plt.show()
```

Output:

RESULT: after importing libraries sample data is trained from samples for predicted expected values towards sample csv file.