GLA UNIVERSITY, MATHURA

LAB RECORD

SUBJECT: Machine Learning LAB

SUBJECT CODE: BCSE0133

SESSION: 2023-2024

SUBMITTED TO:

Dr. Arati Kushwaha

SUBMITTED BY:

Shubh Chaturvedi

B.TECH III YEAR

BRANCH: CS **SECTION: 'D' 3**

CLASS ROLL NO: 55 **UNIVERSITY ROLLNO: 2115000976**

INDEX

S. No.	Experiment	Date of Perform	Date of Submission	Signature
1	Introduction of Machine Learning with Python	01/08/23	05/12/23	
2	Linear Regression	08/08/23	05/12/23	
3	Linear Regression with CSV File	08/08/23	05/12/23	
4	Logistic Regression with CSV File	08/08/23	05/12/23	
5	Logistic Regression	23/08/23	05/12/23	
6	Logistic Regression with CSV File	05/09/23	05/12/23	
7	Principal Component Analysis (PCA)	12/09/23	05/12/23	
8	Support Vector Machine (SVM)	19/09/23	05/12/23	
9	K-Nearest Neighbour (KNN)	10/10/23	05/12/23	

10	Decision Tree	**31/10/23**	**05/12/23**	
11	Random Forest	**07/11/23**	**05/12/23**	
12	Naïve Bayes	**21/11/23**	**05/12/23**	
13	Artificial Neural Network (ANN)	**28/11/23**	**05/12/23**	

Experiment No. - 1

Introduction of Machine Learning with Python

Objective: Illustrate basic operations in Python using dictionaries, NumPy arrays, and Pandas DataFrame to demonstrate data creation, manipulation, and analysis fundamentals in machine learning.

```
1.  dict = {
      "Brand": "BMW",
      "Model": "X5",
      "Price": "60,00,000",
      "Year": "2023"
}
print(dict)
```

- ```
  {'Brand': 'BMW', 'Model': 'X5', 'Price': '60,00,000',
  'Year': '2023'}
  ```

1. import numpy as np

   arr = np.array([1,2,3,4,5,6,7,8,9,10])

   print(arr);

   print(type(arr));

   - ```
     [ 1  2  3  4  5  6  7  8  9 10]
     <class 'numpy.ndarray'>
     ```

2. import numpy as np

   arr = np.array([[1,2,3,4],[5,6,7,8]]);

   print(arr);

   - ```
     [[1 2 3 4]
     [5 6 7 8]]
     ```

3. import numpy as np

   a = np.ones(5);

   print(a);

   - ```
     [1. 1. 1. 1. 1.]
     ```

4. import numpy as np

```
b = np.zeros(10);
print(b);
```

➢ `[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]`

5. import numpy as np

```
c = np.array([2,4,6,8,10,12]);
s = c.sum();
print(s);
```

➢ `42`

6. import numpy as np

```
arr = np.array([12,25,42,56,89,76,92]);
n = np.flip(arr);
print(n);
```

➢ `[92 76 89 56 42 25 12]`

7. import pandas as pd

```
dataset = {
  "cars": ["BMW", "Audi", "Lamborghini"," Rolls Royce", "Hyundai"],
  "passing": [4,3,5,5,4]
}
new = pd.DataFrame(dataset);
print(new);
```

➢

```
          cars    passing
0          BMW          4
1         Audi          3
2  Lamborghini          5
3  Rolls Royce          5
4      Hyundai          4
```

Experiment No. - 2

Linear Regression

Objective: Perform Linear Regression analysis on a dataset of plot sizes and prices to predict plot prices for new sizes, visualize the regression line, and understand the relationship between plot size and price.

Dataset: Demonstration dataset containing plot sizes and their respective prices.

Linear Regression :

Linear Regression is a statistical technique used to establish a linear relationship between dependent and independent variables. It predicts the dependent variable's value based on independent variables, fitting a line that minimizes the difference between predicted and actual values, ideal for understanding and forecasting relationships in data.

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

#sample dataset for demostration
data = {
    'Plot_Size': [100, 150, 200, 250, 300, 350, 400, 450, 500],
    'Plot_Price': [200000, 250000, 300000, 350000, 400000, 450000,
500000, 550000, 600000]
}

#create a DatFrame from the dataset
df = pd.DataFrame(data)

#split the data into features (x) and target (y)
x = df[['Plot_Size']]
y = df['Plot_Price']

# create a linear regression model
model = LinearRegression()

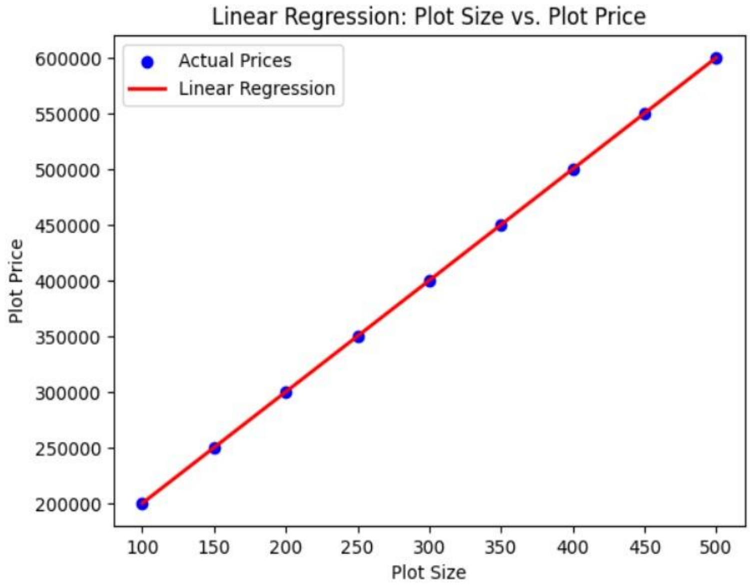
#Fit the model on the data
model.fit(x,y)

#Predict plot prices for new plot sizes (e.g., 600 and 700)
new_sizes = [[600],[700]]
predicted_prices = model.predict(new_sizes)
print("Predicted Prices for new plot sizes:")
```

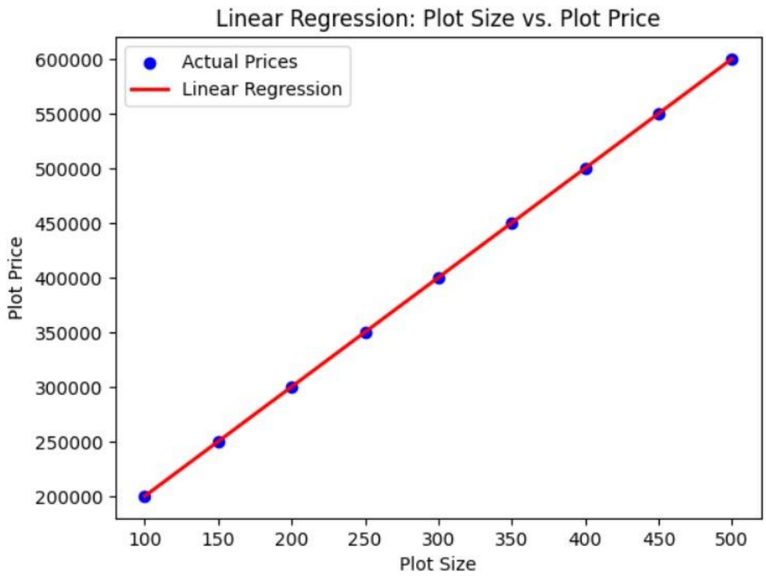
```
for size, price in zip (new_sizes, predicted_prices):
  print(f"Plot Size: {size[0]}, Predicted Price: {price:.2f}")

# visualize the data and regression line
  plt.scatter(x,y, color="blue", label='Actual Prices')
  plt.plot(x,model.predict (x), color='red', linewidth=2, label='Linear
Regression')
  plt.xlabel('Plot Size')
  plt.ylabel('Plot Price')
  plt.legend()
  plt.title('Linear Regression: Plot Size vs. Plot Price')
 plt.show()
```

```
Predicted Prices for new plot sizes:
Plot Size: 600, Predicted Price: 700000.00
```




```
Plot Size: 700, Predicted Price: 800000.00
```



Experiment No. - 3

Linear Regression with CSV File

Objective: Utilize Linear Regression on the California housing dataset to analyze the relationship between the total number of bedrooms and population, predicting population based on the number of bedrooms, and visualize the regression line to understand the correlation between these variables.

Dataset: `/content/sample_data/california_housing_test.csv`.

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


data = pd.read_csv("/content/sample_data/california_housing_test.csv")


df = pd.DataFrame(data)

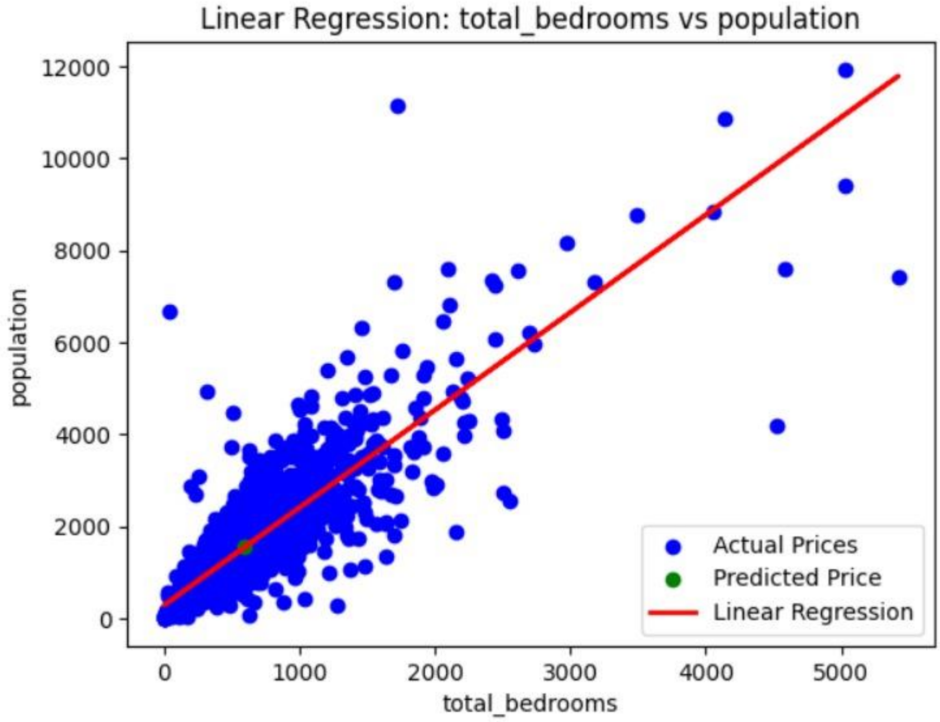

x = df[['total_bedrooms']]
y = df['population']


model = LinearRegression()

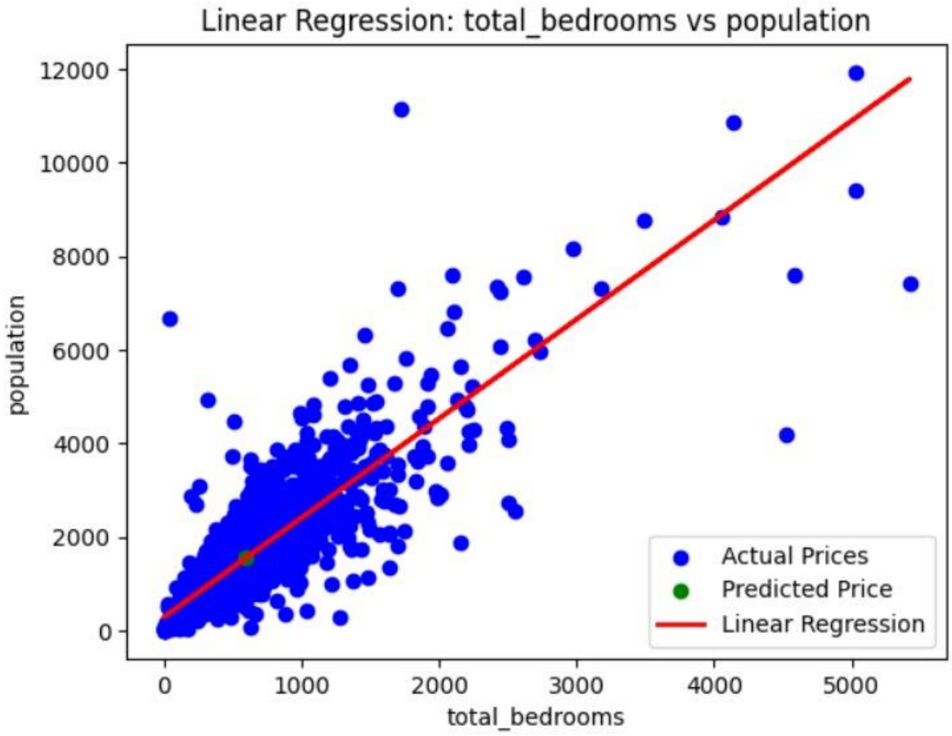

model.fit(x,y)


new_sizes = [[600],[700]]
predicted_prices = model.predict(new_sizes)
print("Predicted Prices for new plot sizes:")
for size, price in zip (new_sizes, predicted_prices):
  print(f"Plot Size: {size[0]}, Predicted Price: {price:.2f}")
  plt.scatter(x,y, color="blue", label='Actual Prices')
  plt.scatter(600,1551.53, color="green", label='Predicted Price')
  plt.plot(x,model.predict (x), color='red', linewidth=2, label='Linear
Regression')
  plt.xlabel('total_bedrooms')
  plt.ylabel('population')
  plt.legend()
  plt.title('Linear Regression: total_bedrooms vs population')
  plt.show()
```

```
Predicted Prices for new plot sizes:
Plot Size: 600, Predicted Price: 1551.53
```




```
Plot Size: 700, Predicted Price: 1763.86
```



Experiment No. - 4

Linear Regression with CSV File

Objective: Utilize Linear Regression to predict profits based on marketing spend from the '50_Startups.csv' dataset, showcasing a regression model's predictive capability in a business context.

Dataset: '50_Startups.csv' dataset containing information about startups including marketing spends and profits for regression analysis.

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

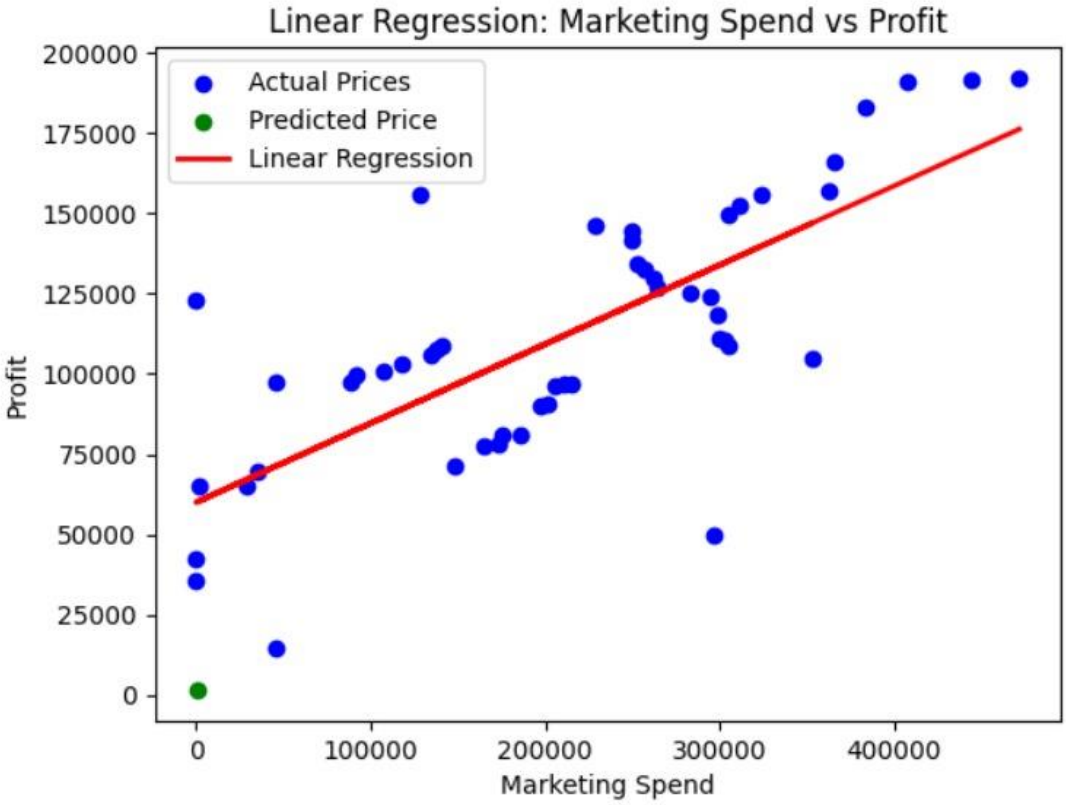
df = pd.read_csv('/content/50_Startups.csv')

x = df[['Marketing Spend']]
y = df['Profit']

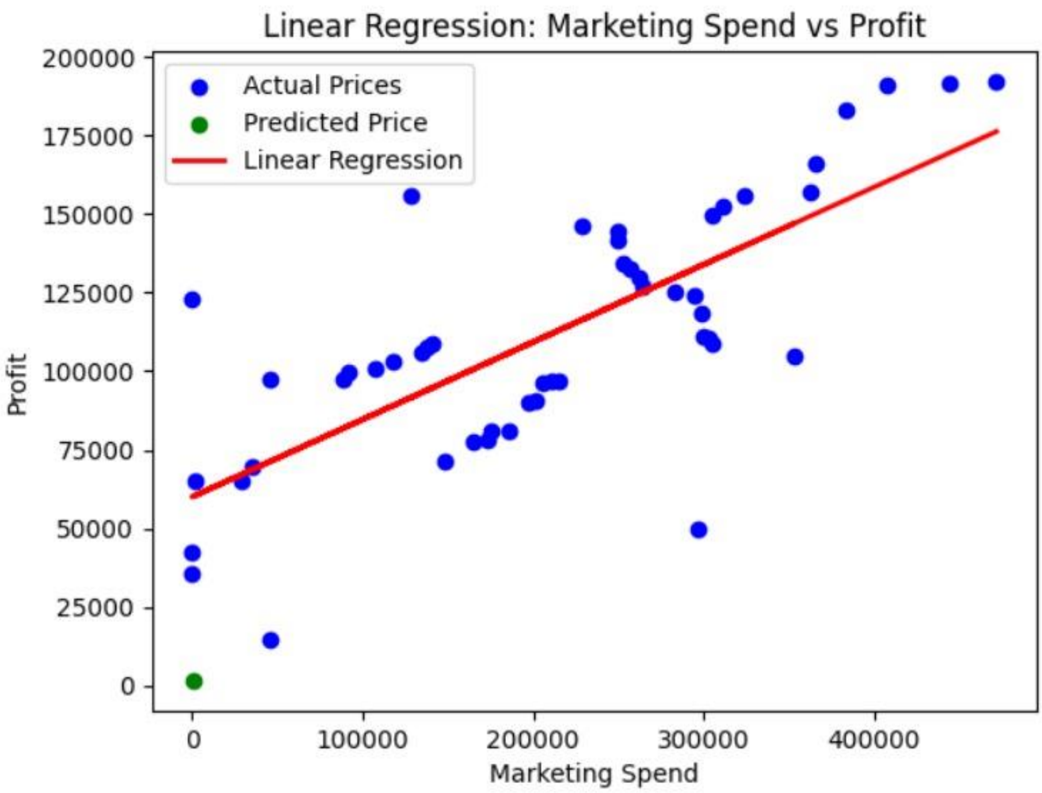
logr = linear_model.LinearRegression()
logr.fit(x,y)

new_spends = [[254597.82],[402651.34]]
predicted_profit = logr.predict(new_spends)
print("Predicted profit according the marketing spend:")
for size, profit in zip (new_spends, predicted_profit):
  print(f"profit: {size[0]}, Predicted profit: {profit:.2f}")
  plt.scatter(x,y, color="blue", label='Actual Prices')
  plt.scatter(600,1551.53, color="green", label='Predicted Price')
  plt.plot(x,logr.predict (x), color='red', linewidth=2, label='Linear
Regression')
  plt.xlabel('Marketing Spend')
  plt.ylabel('Profit')
  plt.legend()
  plt.title('Linear Regression: Marketing Spend vs Profit')
  plt.show()
```

```
Predicted profit according the marketing spend:
profit: 254597.82, Predicted profit: 122751.54
```




```
profit: 402651.34, Predicted profit: 159240.70
```



Experiment No. - 5

Logistic Regression

Objective : Implementation of Logistic Regression

Logistic Regression: Logistic Regression is a statistical method used for binary classification, estimating the probability of a binary outcome based on one or more predictor variables. It models the relationship between the dependent binary variable and independent variables, providing insights into the likelihood of a particular event occurrence

```
import numpy
from sklearn import linear_model

#x represents the size of a tumor in centimeters.
x = numpy.array([3.78, 2.44, 2.09, 0.14, 1.72, 1.65, 4.92, 4.37, 4.96,
4.52, 3.69, 5.88]).reshape(-1,1)
#x.reshape(-1,1) #Note: x has to be reshaped into a column from now a
row for the LogisticRegression() function to work.

#Note: x has to be reshaped into a column from a row for the
LogisticRegression() function to work.
#y represents whether or not the tumor is cancerous (0 for "No", 1 for
"Yes").
y = numpy.array([0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1])

logr = linear_model.LogisticRegression()
logr.fit(x,y)

#predict if tumor is cancerous where the size is 3.46mm:
predicted = logr.predict(numpy.array([3.5]).reshape(-1,1))
print(predicted)
```

- [1]

Experiment No. - 6

Logistic Regression with CSV File

Objective : Implementation of Logistic Regression with CSV file

Datasheet : /content/Bank Customer Churn Prediction.csv

Logistic Regression: Logistic Regression is a statistical method used for binary classification, estimating the probability of a binary outcome based on one or more predictor variables. It models the relationship between the dependent binary variable and independent variables, providing insights into the likelihood of a particular event occurrence

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics  import accuracy_score

data = pd.read_csv('/content/Bank Customer Churn Prediction.csv')

data.head()
data.shape
```

- (10000, 12)

data.head()

d = data.isnull()

d.head()

d.shape

- (10000, 15)

x.head()

x.shape

- (10000, 14)

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=42)

model = LogisticRegression(random_state=42)
model.fit(x_train, y_train)

y_pred = model.predict(x_test)

accuracy = accuracy_score(y_test,y_pred)

print(f'Accuracy:{accuracy:.2f}')
```

- Accuracy:0.81

```
df = pd.DataFrame(d)

df.to_csv('Check_MissingValues.csv',index=False)

data=data.dropna()

from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, confusion_matrix

precision = precision_score(y_test,y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

print(f'Precision:{precision:.2f}')
print(f'Recall:{recall:.2f}')
print(f'F1:{f1:.2f}')
print('Confusion Matrix:')
print(conf_matrix)
```

- ```
  Precision:0.00

  Recall:0.00
  F1:0.00
  Confusion Matrix:
  [[2416    0]
   [ 584    0]]
  ```
  ```

Experiment No. - 7

Principal Component Analysis (PCA)

Objective: Implement Principal Component Analysis (PCA) on a 2D dataset to reduce dimensions and project the data into a lower-dimensional space, visually illustrating the transformation from a higher-dimensional space to a one-dimensional space.

Dataset: Demonstration dataset containing 2D points used to showcase PCA transformation and dimensionality reduction.

Principal Component Analysis (PCA) : Principal Component Analysis (PCA) is a dimensionality reduction technique that identifies new, uncorrelated variables (principal components) by projecting high-dimensional data onto a lower-dimensional space. It simplifies data while preserving important information, aiding in visualization, noise reduction, and computational efficiency in machine learning tasks.

```
import numpy as np
import matplotlib.pyplot as plt


# original data
data = np.array([[2,3],
                 [3,4],
                 [4,5],
                 [5,6],
                 [6,7],])


# step 1 normalize data
mean = np.mean(data, axis=0)
std_dev = np.std(data, axis=0)
data_std = (data - mean) / std_dev



# step 2 compute covariance matrix
cov_mat = np.cov(data_std.T)
print(cov_mat)


# step 3 compute eigen values and eigen vectors
eig_val, eig_vec = np.linalg.eig(cov_mat)
print(eig_val)
```

```
# step 4 select principal components
principal_components = eig_vec[:, np.argmax(eig_val)]

# step 5 project data
projected_data = principal_components.reshape(-1, 1)

# step 6 transform data
transformed_data = data_std.dot(projected_data)

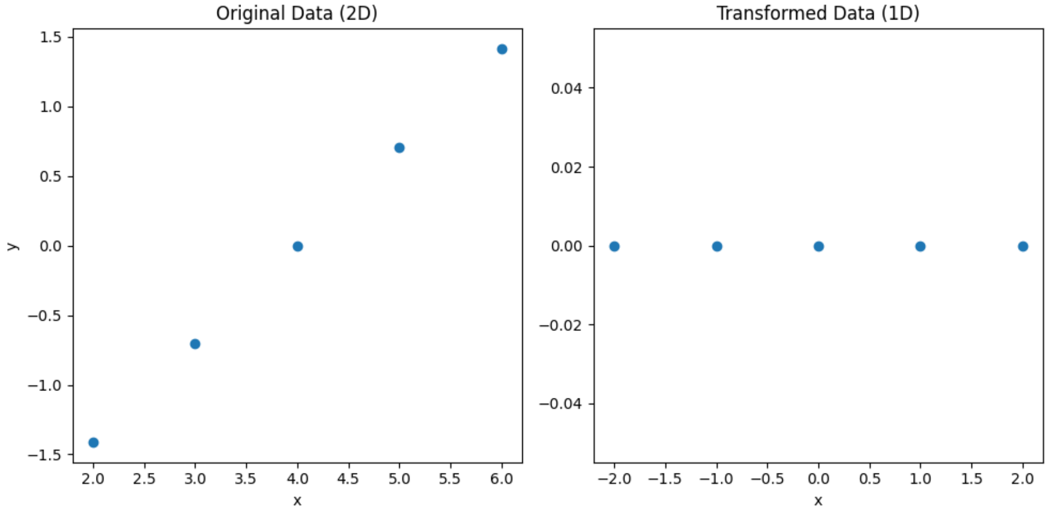

plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.title('Original Data (2D)')
plt.xlabel('x')
plt.ylabel('y')
plt.scatter(data[:, 0], data_std[:, 1])



# plot transformed data
plt.subplot(1, 2, 2)
plt.title('Transformed Data (1D)')
plt.xlabel('x')
plt.scatter(transformed_data, np.zeros_like(transformed_data))
plt.ylabel('')
plt.tight_layout()
plt.show()
```

```
[[1.25 1.25]
 [1.25 1.25]]
[ 2.50000000e+00 -2.22044605e-16]
```



Performing Principle Component Analysis (PCA) on a sample dataset with 5 features and 10 data points, reducing dimensions to retain three components, showcasing transformation and reconstruction of the data.

```
import numpy as np
from sklearn.decomposition import PCA


# Create a sample dataset with 5 features (columns) and 10 data points
(rows)
data = np.array([[1.0, 2.0, 3.0, 4.0, 5.0],
                 [2.0, 3.0, 4.0, 5.0, 6.0],
                 [3.0, 4.0, 5.0, 6.0, 7.0],
                 [4.0, 5.0, 6.0, 7.0, 8.0],
                 [5.0, 6.0, 7.0, 8.0, 9.0],
                 [6.0, 7.0, 8.0, 9.0, 10.0],
                 [7.0, 8.0, 9.0, 10.0, 11.0],
                 [8.0, 9.0, 10.0, 11.0, 12.0],
                 [9.0, 10.0, 11.0, 12.0, 13.0],
                 [10.0, 11.0, 12.0, 13.0, 14.0]])


# specify the number of principle components
n_components_to_retain = 3

# create a PCA object
pca = PCA(n_components=n_components_to_retain)
data_transformed = pca.fit_transform(data)


# recover the original data using the inverse transform method
data_recovered = pca.inverse_transform(data_transformed)

# display the original data and

# Display the original data and the reconstructed data
print("Original Data:")
print(data)
print("\Reduced Feature (Data):")
print(data_transformed)

print("\nData After PCA (Reconstructed):")
print(data_recovered)
```

```
Original Data:
[[ 1.  2.  3.  4.  5.]
 [ 2.  3.  4.  5.  6.]
 [ 3.  4.  5.  6.  7.]
 [ 4.  5.  6.  7.  8.]
 [ 5.  6.  7.  8.  9.]
 [ 6.  7.  8.  9. 10.]
 [ 7.  8.  9. 10. 11.]
 [ 8.  9. 10. 11. 12.]
 [ 9. 10. 11. 12. 13.]
 [10. 11. 12. 13. 14.]]
\Reduced Feature (Data):
[[ 1.00623059e+01  9.94877574e-16 -2.74935550e-32]
 [ 7.82623792e+00 -1.40849427e-16  2.48933714e-31]
 [ 5.59016994e+00 -2.14094009e-16 -3.86364558e-32]
 [ 3.35410197e+00 -8.87358591e-17 -1.98242585e-32]
 [ 1.11803399e+00 -6.26790751e-17 -9.40609865e-33]
 [-1.11803399e+00  6.26790751e-17  9.40609865e-33]
 [-3.35410197e+00  8.87358591e-17  1.98242585e-32]
 [-5.59016994e+00  2.14094009e-16  3.86364558e-32]
 [-7.82623792e+00  1.40849427e-16  4.06605781e-32]
 [-1.00623059e+01  4.64810310e-16  7.62608504e-32]]

Data After PCA (Reconstructed):
[[ 1.  2.  3.  4.  5.]
 [ 2.  3.  4.  5.  6.]
 [ 3.  4.  5.  6.  7.]
 [ 4.  5.  6.  7.  8.]
 [ 5.  6.  7.  8.  9.]
 [ 6.  7.  8.  9. 10.]
 [ 7.  8.  9. 10. 11.]
 [ 8.  9. 10. 11. 12.]
 [ 9. 10. 11. 12. 13.]
 [10. 11. 12. 13. 14.]]
```

Perform Logistic Regression on a red wine dataset after data preprocessing **(scaling and PCA)**, predicting wine quality, and evaluating model performance using a confusion matrix and classification report.

```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd



# Importing the dataset
dataset = pd.read_csv('winequality-red.csv')

X = dataset.drop('quality', axis=1)
y = dataset['quality']


# Splitting the dataset into the Training set and Test set
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,
random_state=0)


from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```python
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)

explained_variance = pca.explained_variance_ratio_

from sklearn.linear_model import LogisticRegression

classifier = LogisticRegression(random_state=0)
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import confusion_matrix, classification_report

cm = confusion_matrix(y_test, y_pred)
print("confusion matrix: \n", cm)


print("classification report: \n")
print(classification_report(y_test, y_pred))
```

```
confusion matrix:
 [[ 0  0  0  2  0  0]
 [ 0  0  4  7  0  0]
 [ 0  0 89 45  1  0]
 [ 0  0 55 81  6  0]
 [ 0  0  4 21  2  0]
 [ 0  0  0  2  1  0]]
classification report:

              precision    recall  f1-score   support

           3       0.00      0.00      0.00         2
           4       0.00      0.00      0.00        11
           5       0.59      0.66      0.62       135
           6       0.51      0.57      0.54       142
           7       0.20      0.07      0.11        27
           8       0.00      0.00      0.00         3

    accuracy                           0.54       320
   macro avg       0.22      0.22      0.21       320
weighted avg       0.49      0.54      0.51       320
```

Experiment No. - 8

Support Vector Machine (SVM)

Objective: Apply a Support Vector Machine (SVM) using a linear kernel on the red wine dataset for quality prediction, assess its accuracy, generate a confusion matrix, and visualize the classification.

Dataset: `winequality-red.csv`

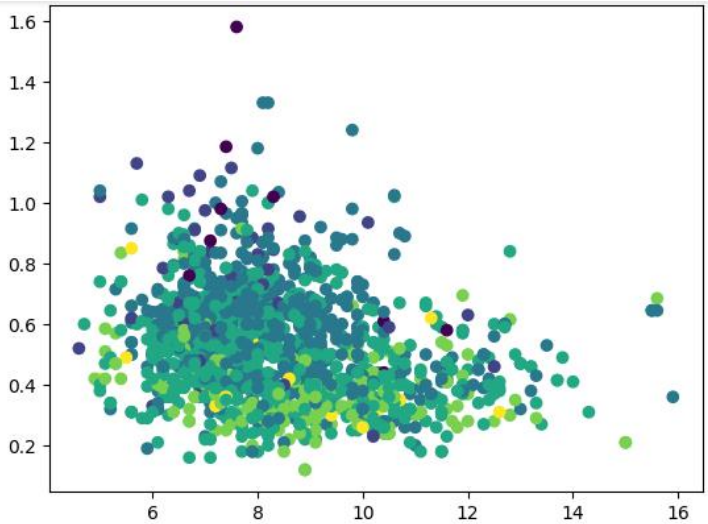
Support Vector Machine (SVM) :

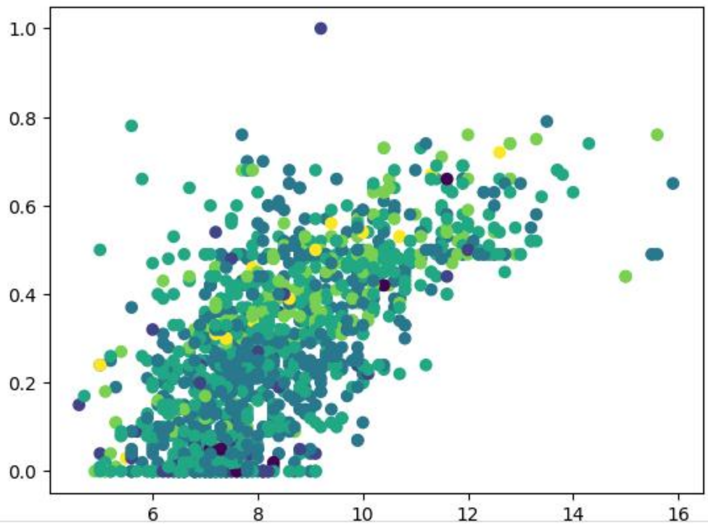
Support Vector Machine (SVM) is a supervised machine learning algorithm used for classification and regression tasks. It finds an optimal hyperplane that best separates data points into different classes in high-dimensional space, aiming to maximize the margin between classes for accurate predictions. SVMs handle complex datasets efficiently, making them suitable for various applications.

```python
# implement svm model


import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import svm
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report



# read data
data = pd.read_csv('winequality-red.csv')
#print(data.head())
#print(data.shape)
#print(data.describe())


# split data
X = data.iloc[:, :-1]
y = data.iloc[:, -1]
# print(X.shape)
# print(y.shape)
# print(X.head())
```

```
# print(y.head())


# split data into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2)
# print(X_train.shape)
# print(X_test.shape)
# print(y_train.shape)
# print(y_test.shape)


# train model
model = svm.SVC(kernel='linear')
model.fit(X_train, y_train)



# predict
print("Predict")
y_pred = model.predict(X_test)
print(y_pred)
# print(y_test)


# evaluate and confusion matrix
print(accuracy_score(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))


# plot
plt.scatter(X.iloc[:, 0], X.iloc[:, 1], c=y)
plt.show()

# plot
plt.scatter(X.iloc[:, 0], X.iloc[:, 2], c=y)
plt.show()
```

```
Predict
[5 6 5 6 6 5 6 6 5 6 6 5 5 6 6 6 6 6 6 5 5 5 5 5 6 5 6 6 6 5 5 5 5 5 6 5 5
 5 5 6 6 6 6 6 5 5 5 5 6 5 6 6 5 6 5 5 5 6 5 6 5 5 5 5 5 5 5 6 6 6 6 6 5 6
 6 5 6 6 5 5 5 5 5 5 5 5 6 5 5 6 6 6 6 5 5 5 6 6 6 6 6 6 6 6 5 5 6 6 5 5 6
 5 6 6 5 5 5 6 6 5 6 5 6 5 6 6 6 5 5 5 5 6 5 6 5 6 6 6 5 5 5 6 5 6 5 6 6 5
 5 6 5 5 6 5 6 6 5 5 6 5 5 5 6 5 6 6 6 6 5 5 5 6 6 5 5 6 6 6 5 5 5 6 6 5 5
 6 5 5 6 5 5 5 5 5 5 6 6 5 5 5 5 6 5 6 6 6 6 5 5 6 5 5 5 5 5 5 5 6 6 5 6 5
 5 6 6 5 5 6 5 6 6 5 5 6 5 6 5 6 6 5 5 6 6 6 6 5 5 6 6 5 6 6 6 5 5 6 5 5 5
 5 6 6 6 6 6 5 5 5 5 6 5 6 5 6 6 6 5 6 5 5 6 5 6 6 6 5 6 5 6 6 5 6 5 6 5 6
 5 6 6 5 5 5 6 5 5 5 5 6 6 5 5 5 6 6 6 5 6 5 6 5]
0.625
[[  0   0   2   0   0   0]
 [  0   0  11   2   0   0]
 [  0   0 114  25   0   0]
 [  0   0  40  86   0   0]
 [  0   0   1  34   0   0]
 [  0   0   0   5   0   0]]
              precision    recall  f1-score   support

           3       0.00      0.00      0.00         2
           4       0.00      0.00      0.00        13
           5       0.68      0.82      0.74       139
           6       0.57      0.68      0.62       126
           7       0.00      0.00      0.00        35
           8       0.00      0.00      0.00         5

    accuracy                           0.62       320
   macro avg       0.21      0.25      0.23       320
weighted avg       0.52      0.62      0.57       320
```





Experiment No. - 9

K-Nearest Neighbour (KNN)

Objective: Employ the K-Nearest Neighbors (KNN) classifier on the red wine dataset to predict wine quality, evaluate accuracy, generate a confusion matrix, and save the trained model.

Dataset: `winequality-red.csv`

K-Nearest Neighbors (KNN):

K-Nearest Neighbors (KNN) is a simple, non-parametric, and versatile classification algorithm. It predicts the classification of a new data point by considering its neighbors' majority class, making decisions based on the majority class among its k nearest data points in the feature space. This algorithm is effective for various applications due to its simplicity and adaptability in handling both classification and regression tasks.

```
# implement knn model

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import pickle
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix

# load data
df = pd.read_csv('winequality-red.csv')
df.head()

# check for null values
# df.isnull().sum()

df = pd.get_dummies(df, columns=['quality'])
df.head()

# split data


X = df.iloc[:, :-1]
y = df.iloc[:, -1]
```

```python
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
print('X_train shape: ', X_train.shape)
print('X_test shape: ', X_test.shape)
print('y_train shape: ', y_train.shape)
print('y_test shape: ', y_test.shape)


# train model
y_train = y_train.astype('int')
y_test = y_test.astype('int')
knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
print('Accuracy: ', accuracy_score(y_test, y_pred))


# confusion matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)

# save model
pickle.dump(knn, open('model.pkl', 'wb'))
```

```
X_train shape:  (1279, 16)
X_test shape:  (320, 16)
y_train shape:  (1279,)
y_test shape:  (320,)
Accuracy:  0.984375
[[315   0]
 [  5   0]]
```

Experiment No. - 10

Decision Tree

Objective: Apply a Decision Tree Classifier to the Red Wine Quality dataset, assess model accuracy, generate a confusion matrix, and classification report, and visualize the decision tree.

Dataset: https://raw.githubusercontent.com/aniruddhachoudhury/Red-Wine-Quality/master/winequality-red.csv

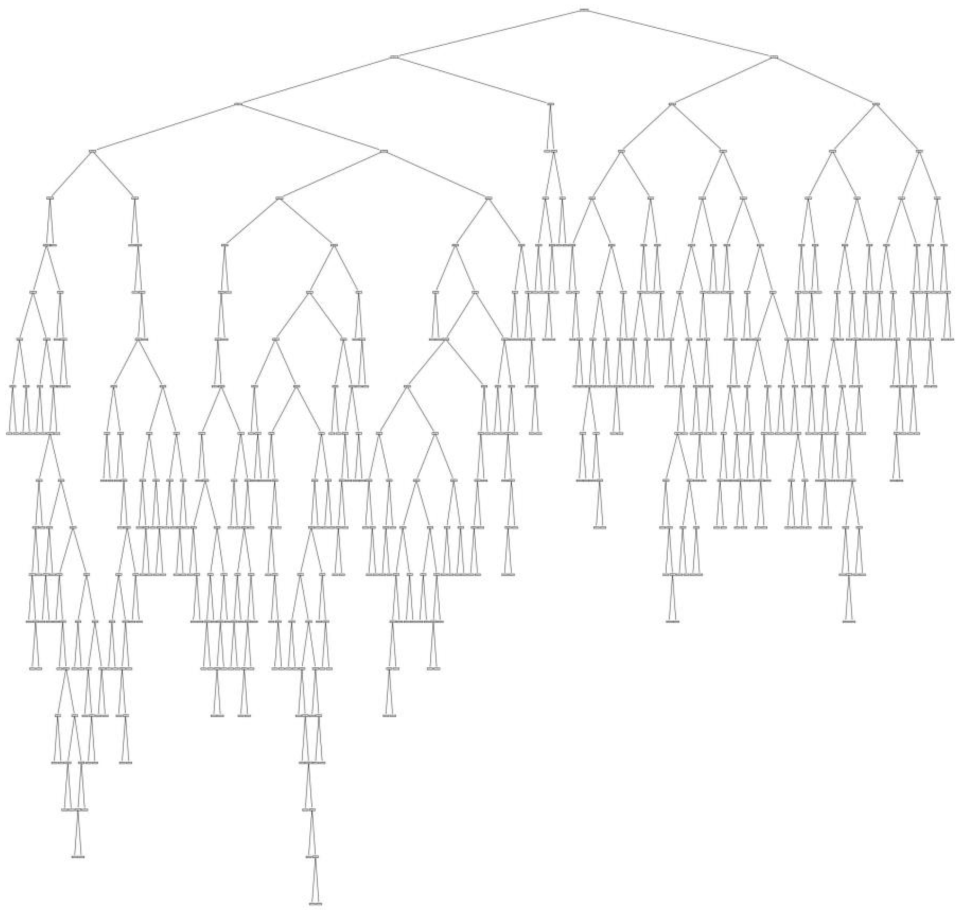
Random Forest:

Random Forest, an ensemble learning method, constructs multiple decision trees and amalgamates their predictions. By utilizing diverse trees, it improves accuracy, mitigates overfitting, and accommodates large datasets efficiently, making it a robust technique for classification and regression tasks in machine learning.

```
# decision tree in ml

import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')

data=pd.read_csv("https://raw.githubusercontent.com/aniruddhachoudhury/
Red-Wine-Quality/master/winequality-red.csv")

data.quality.unique()

data.quality.value_counts()
print(data.quality.unique())

X = data.drop(columns=('quality'))
y = data['quality']

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

from sklearn.tree import DecisionTreeClassifier
```

```python
dt = DecisionTreeClassifier()
dt.fit(X_train, y_train)
print(dt.score(X_test, y_test))


y_pred = dt.predict(X_test)

from sklearn.metrics import confusion_matrix, classification_report

print("confusion matrix: ", confusion_matrix(y_test, y_pred))

print("classification report: ", classification_report(y_test, y_pred))

from sklearn.tree import plot_tree

plt.figure(figsize=(40,40))
plot_tree(dt)
plt.show()
```

```
[5 6 7 4 8 3]
0.5604166666666667
confusion matrix:  [[  0   0   1   0   0   0]
 [  0   1   8   7   1   0]
 [  2  11 123  53   5   1]
 [  0   4  50 114  28   4]
 [  0   1   5  25  29   1]
 [  0   0   0   2   2   2]]
classification report:                precision    recall  f1-score   support

           3       0.00      0.00      0.00         1
           4       0.06      0.06      0.06        17
           5       0.66      0.63      0.64       195
           6       0.57      0.57      0.57       200
           7       0.45      0.48      0.46        61
           8       0.25      0.33      0.29         6

    accuracy                           0.56       480
   macro avg       0.33      0.34      0.34       480
weighted avg       0.57      0.56      0.56       480
```



Experiment No. - 11

Random Forest

Objective: Implement a Random Forest Classifier on the Red Wine Quality dataset to predict wine quality, evaluate model accuracy, and visualize decision trees for insights.

Datasheet:
https://raw.githubusercontent.com/aniruddhachoudhury/Red-Wine-Quality/master/winequality-red.csv

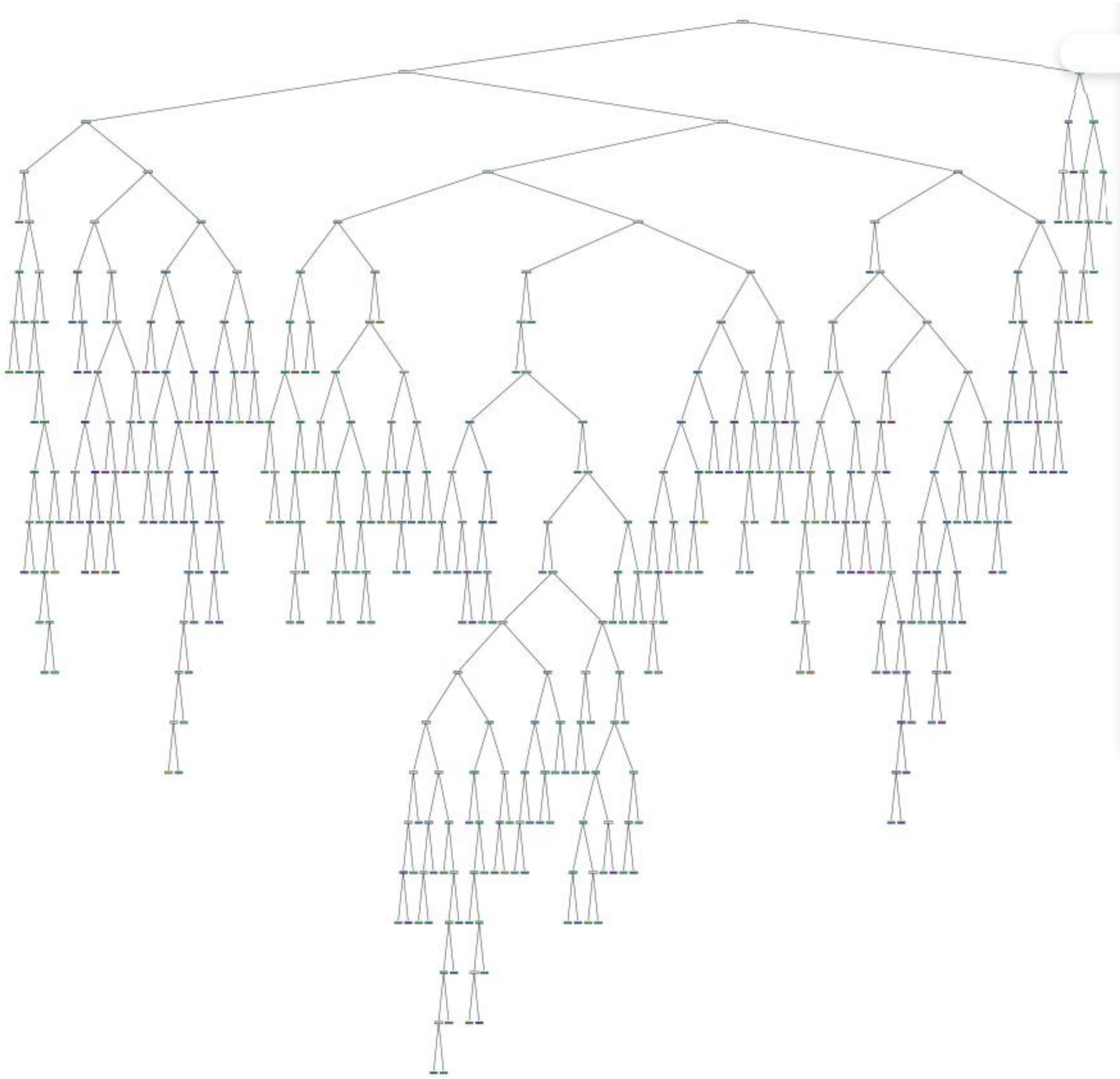
Random Forest :

Random Forest is an ensemble learning method employing multiple decision trees during training and outputting the mode of the classes (classification) or the average prediction (regression). It combines diverse decision trees to enhance accuracy, reduces overfitting, and handles large datasets efficiently, making it versatile for various predictive tasks.

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')

data=pd.read_csv("https://raw.githubusercontent.com/aniruddhachoudhury/
Red-Wine-Quality/master/winequality-red.csv")

data.quality.unique()

data.quality.value_counts()
print(data.quality.unique())

X = data.drop(columns=('quality'))
y = data['quality']

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

from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier()
rf.fit(X_train, y_train)
print(rf.score(X_test, y_test))
```

```
y_pred = rf.predict(X_test)

from sklearn.metrics import confusion_matrix, classification_report

print("confusion matrix: ", confusion_matrix(y_test, y_pred))

print("classification report: ", classification_report(y_test, y_pred))

from sklearn.tree import plot_tree

plt.figure(figsize=(40,40))

estimator = rf.estimators_[5]

plot_tree(estimator, feature_names = X.columns, filled = True)
plt.show()
```

```
[5 6 7 4 8 3]
0.6729166666666667
confusion matrix:  [[  0   0   1   0   0   0]
 [  0   0  11   6   0   0]
 [  0   0 152  40   3   0]
 [  0   0  46 142  12   0]
 [  0   0   0  32  28   1]
 [  0   0   0   1   4   1]]
classification report:                precision    recall  f1-score
support

           3       0.00      0.00      0.00         1
           4       0.00      0.00      0.00        17
           5       0.72      0.78      0.75       195
           6       0.64      0.71      0.67       200
           7       0.60      0.46      0.52        61
           8       0.50      0.17      0.25         6

    accuracy                           0.67       480
   macro avg       0.41      0.35      0.37       480
weighted avg       0.64      0.67      0.66       480
```



Experiment No. - 12

Naïve Bayes

Objective: Implement a Naive Bayes classifier on the Red Wine Quality dataset for prediction accuracy assessment using confusion matrix and classification report.

Datasheet :
https://raw.githubusercontent.com/aniruddhachoudhury/Red-Wine-Quality/master/winequality-red.csv

Naïve Bayes :

Naive Bayes is a probabilistic classification algorithm based on Bayes' theorem. It assumes independence between features and predicts the probability of an event's occurrence given prior knowledge. Despite its simplicity and fast computation, Naive Bayes performs well in text classification, spam filtering, and recommendation systems due to its efficiency in handling large datasets.

```
import pandas as pd
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings('ignore')

data=pd.read_csv("https://raw.githubusercontent.com/aniruddhachoudhury/
Red-Wine-Quality/master/winequality-red.csv")

data.quality.unique()

data.quality.value_counts()
print(data.quality.unique())

X = data.drop(columns=('quality'))
y = data['quality']

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

from sklearn.naive_bayes import GaussianNB

nb = GaussianNB()
nb.fit(X_train, y_train)
print(nb.score(X_test, y_test))



y_pred = nb.predict(X_test)
```

```python
from sklearn.metrics import confusion_matrix, classification_report

print("confusion matrix: ", confusion_matrix(y_test, y_pred))

print("classification report: ", classification_report(y_test, y_pred))
```

```
[5 6 7 4 8 3]
0.5416666666666666
confusion matrix:  [[  0   0   1   0   0   0]
 [  1   2   8   6   0   0]
 [  0   6 121  63   5   0]
 [  0   9  46 107  35   3]
 [  0   0   3  28  30   0]
 [  0   0   0   1   5   0]]
classification report:                precision    recall  f1-score
support

           3       0.00      0.00      0.00         1
           4       0.12      0.12      0.12        17
           5       0.68      0.62      0.65       195
           6       0.52      0.54      0.53       200
           7       0.40      0.49      0.44        61
           8       0.00      0.00      0.00         6

    accuracy                           0.54       480
   macro avg       0.29      0.29      0.29       480
weighted avg       0.55      0.54      0.54       480
```

Experiment No. - 13

Artificial Neural Network(ANN)

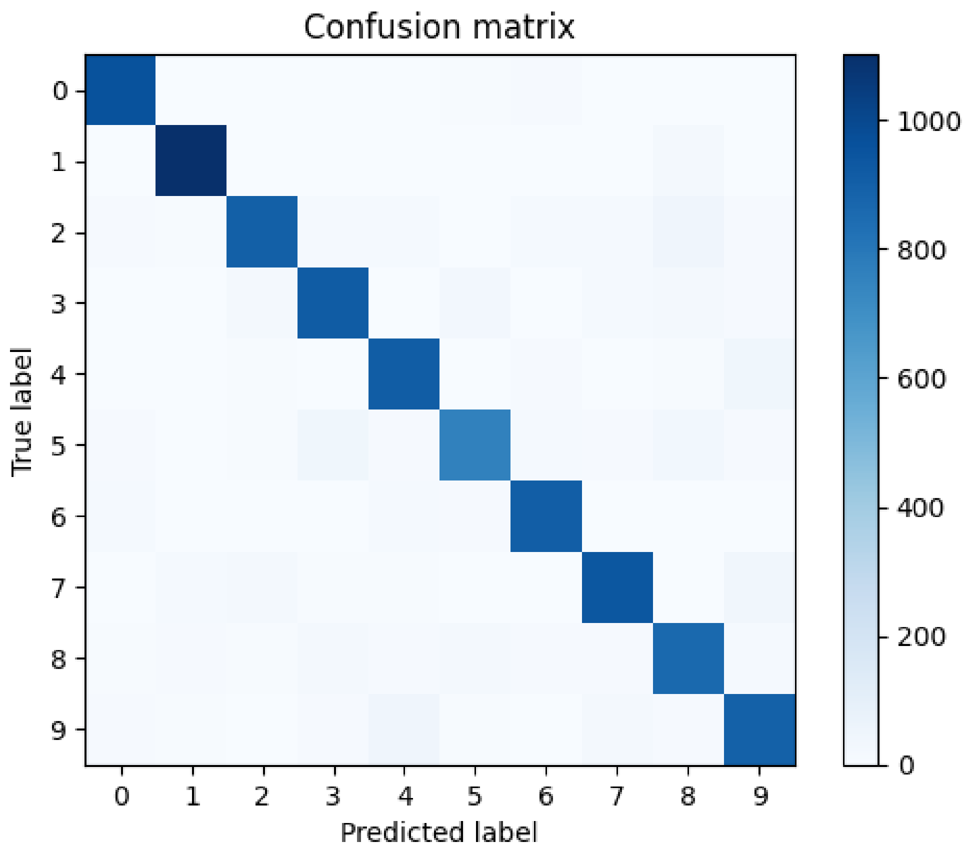
Objective: Implement a single-layer perceptron using an Artificial Neural Network (ANN)

Datasheet : mnist.load() data

Artificial Neural Network (ANN):
Artificial Neural Networks (ANNs) imitate the brain's structure, using interconnected nodes to learn complex patterns in data through weighted transformations and activation functions. Their adaptability and learning during training enable accurate predictions, proving valuable in various tasks like predictive modeling and pattern recognition.

```
# ANN implementation

import tensorflow as tf
import numpy as np
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt

# Load the data

(train_images, train_labels), (test_images, test_lables) =
mnist.load_data()

# Flatten the images into 1D array

train_images = train_images.reshape((60000,
28*28)).astype('float32')/255

test_images = test_images.reshape((10000, 28*28)).astype('float32')/255

#  one hot encode the labels

train_labels = to_categorical(train_labels)
test_lables = to_categorical(test_lables)

# Build the single layer preceptron model

model = models.Sequential()
model.add(layers.Dense(10, activation='softmax', input_shape=(28*28,)))
```

```
# Compile the model
model.compile(optimizer='sgd', loss='categorical_crossentropy',
metrics=['accuracy'])

# Train the model
history = model.fit(train_images, train_labels, epochs=8,
validation_data=(test_images, test_lables))

# Evaluate the model

test_loss, test_acc = model.evaluate(test_images, test_lables)
print(f'Test accuracy: {test_acc * 100:.2f}%')


from sklearn.metrics import confusion_matrix, classification_report

predictions = model.predict(test_images)
predictions_lables = np.argmax(predictions, axis=1)
true_lables = np.argmax(test_lables, axis=1)

# Confusion matrix
conf_matrix = confusion_matrix(true_lables, predictions_lables)

# plot confusion matrix

plt.imshow(conf_matrix, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion matrix')
plt.colorbar()

classes = [str(i) for i in range(10)]
tick_marks = np.arange(len(classes))

plt.xticks(tick_marks, classes)
plt.yticks(tick_marks, classes)
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.show()

# Classification report
report = classification_report(true_lables, predictions_lables,
target_names=classes)
print("Classification report: \n", report)
```

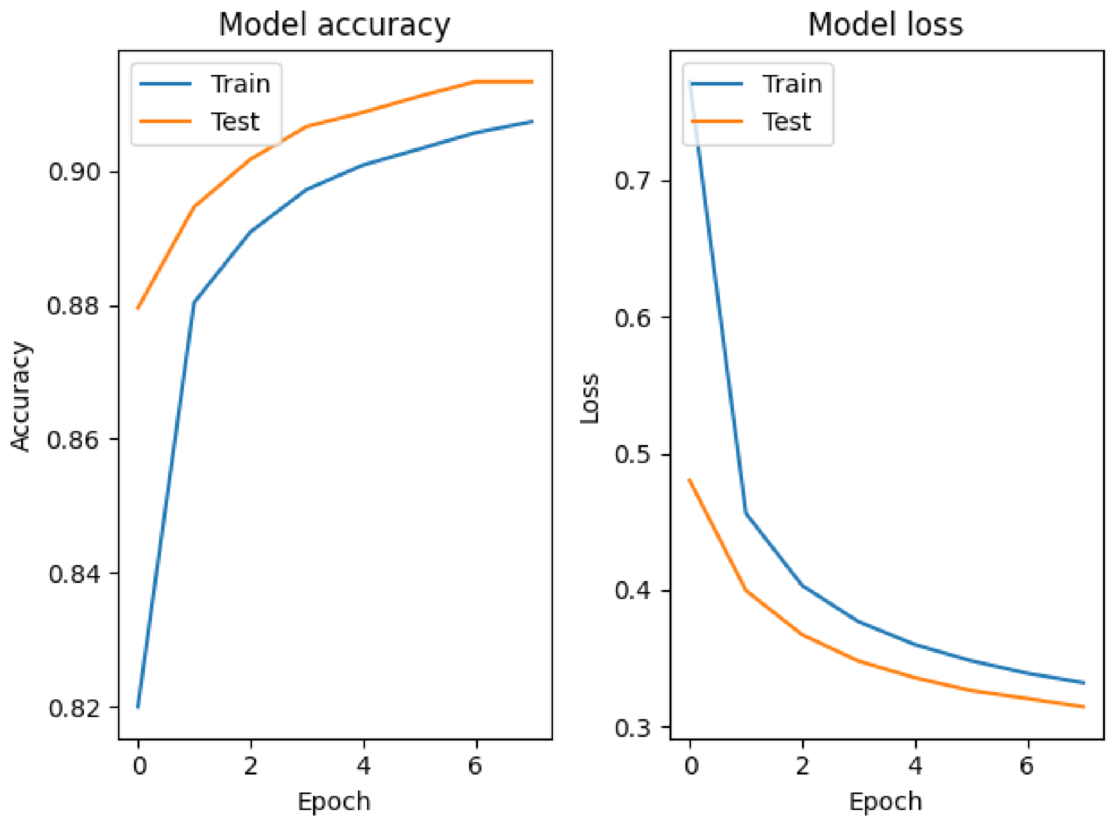


Classification report:

	precision	recall	f1-score	support
0	0.94	0.98	0.96	980
1	0.96	0.97	0.97	1135
2	0.93	0.87	0.90	1032
3	0.90	0.90	0.90	1010
4	0.90	0.93	0.91	982
5	0.91	0.85	0.88	892
6	0.93	0.95	0.94	958
7	0.92	0.91	0.92	1028
8	0.86	0.88	0.87	974
9	0.88	0.89	0.88	1009
accuracy			0.91	10000
macro avg	0.91	0.91	0.91	10000
weighted avg	0.91	0.91	0.91	10000

Objective: Implement a single-layer perceptron using an Artificial Neural Network (ANN)

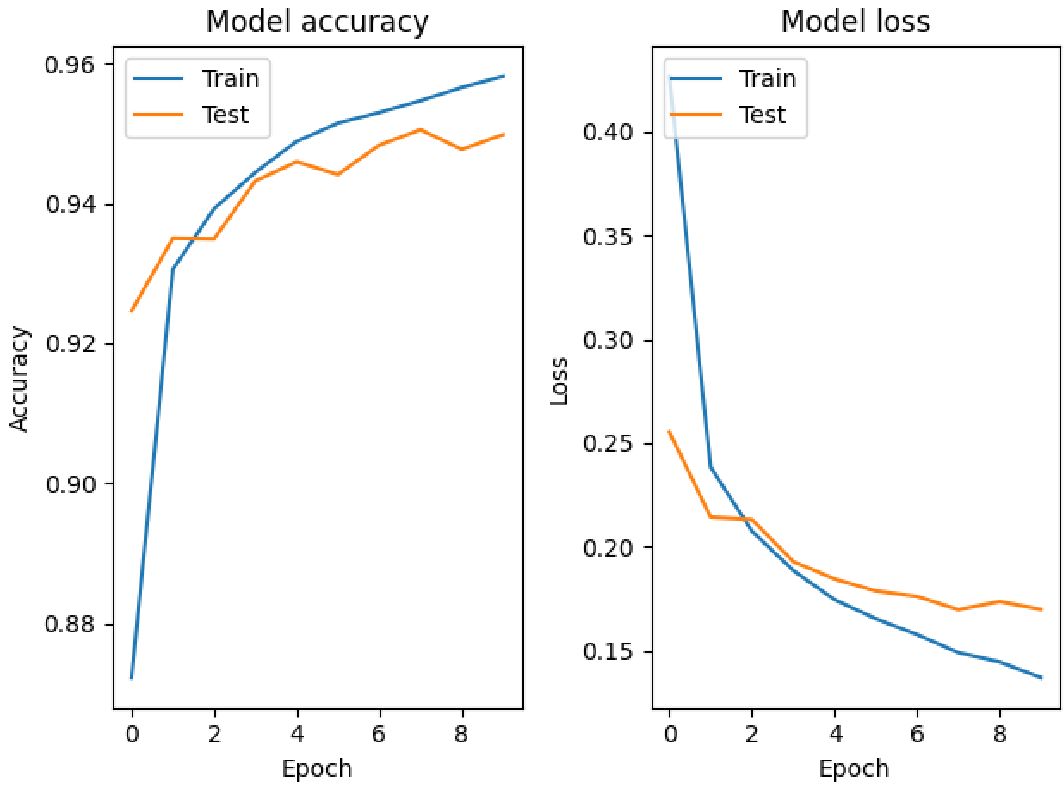
```
# plot training and validation accuracy values

plt.subplot(1, 2, 1)

plt.plot(history.history['accuracy'])

plt.plot(history.history['val_accuracy'])

plt.title('Model accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.legend(['Train', 'Test'], loc='upper left')

# plot training and validation loss values

plt.subplot(1, 2, 2)

plt.plot(history.history['loss'])

plt.plot(history.history['val_loss'])

plt.title('Model loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.legend(['Train', 'Test'], loc='upper left')

plt.tight_layout()

plt.show()
```




```
import tensorflow as tf
import numpy as np
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt



(train_images, train_labels), (test_images, test_lables) =
mnist.load_data()



train_images = train_images.reshape((60000,
28*28)).astype('float32')/255

test_images = test_images.reshape((10000, 28*28)).astype('float32')/255



train_labels = to_categorical(train_labels)
test_lables = to_categorical(test_lables)
```

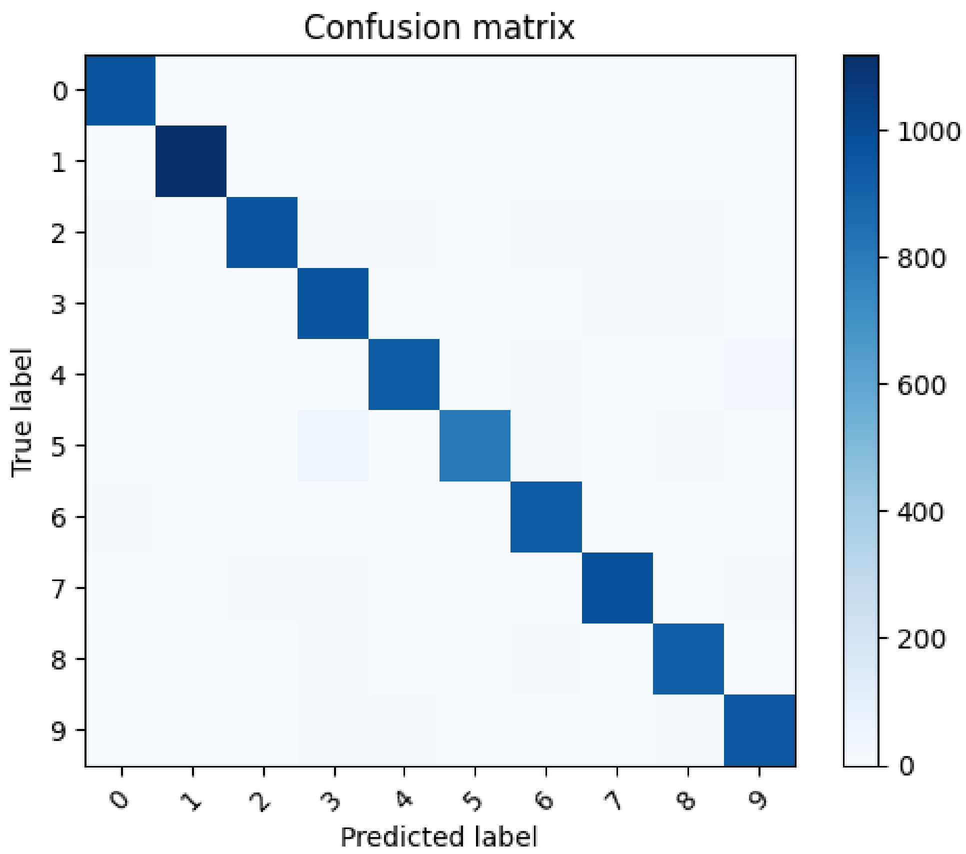
```
model = models.Sequential()
model.add(layers.Dense(10, activation='relu', input_shape=(28*28)))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10,activation='softmax'))

model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])


history = model.fit(train_images, train_labels, epochs=10,
validation_data=(test_images, test_lables))



test_loss, test_acc = model.evaluate(test_images, test_lables)
print(f'Test accuracy: {test_acc * 100:.2f}%')


plt.subplot(1, 2, 1)

plt.plot(history.history['accuracy'])

plt.plot(history.history['val_accuracy'])

plt.title('Model accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.legend(['Train', 'Test'], loc='upper left')



plt.subplot(1, 2, 2)

plt.plot(history.history['loss'])

plt.plot(history.history['val_loss'])

plt.title('Model loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.legend(['Train', 'Test'], loc='upper left')

plt.tight_layout()
```

```
plt.show()
```




```
from sklearn.metrics import confusion_matrix,classification_report

# predicition

predicition = model.predict(test_images)

predicition_labels = np.argmax(predicition, axis=1)

true_labels = np.argmax(test_lables, axis=1)

# confusion matrix

cm = confusion_matrix(true_labels, predicition_labels)

# plot confusion matrix

plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion matrix')

plt.colorbar()
classes = [str(i) for i in range(10)]

tick_marks = np.arange(len(classes))
```

```
plt.xticks(tick_marks, classes, rotation=45)

plt.yticks(tick_marks, classes)

plt.xlabel('Predicted label')

plt.ylabel('True label')


plt.show()

# classification report

print(classification_report(true_labels, predicition_labels))
```




```
              precision    recall  f1-score   support

           0       0.96      0.98      0.97       980
           1       0.97      0.98      0.98      1135
           2       0.96      0.93      0.94      1032
           3       0.90      0.95      0.93      1010
           4       0.96      0.95      0.96       982
           5       0.96      0.90      0.93       892
           6       0.94      0.97      0.96       958
           7       0.95      0.95      0.95      1028
           8       0.93      0.94      0.93       974
           9       0.95      0.94      0.95      1009
```

```
    accuracy                           0.95     10000
   macro avg       0.95      0.95      0.95     10000
weighted avg       0.95      0.95      0.95     10000
```

visualize the sample image dataset

```python
# visualize the sample image dataset

import matplotlib.pyplot as plt
import numpy as np
from tensorflow.keras.datasets import mnist


# load the dataset

(train_images, train_labels), (_, _) = mnist.load_data()

# visualize the dataset
num_sample = 5
rand_idx = np.random.randint(0, train_images.shape[0], num_sample)

for i in range(num_sample):
    index = rand_idx[i]
    image = train_images[index]
    label = train_labels[index]

    plt.subplot(1, num_sample, i+1)
    plt.imshow(image, cmap='gray')
    plt.title(f'Label: {label}')
    plt.axis('off')

plt.show()
```