LAB RECORD

SUBJECT: MACHINE LEARNING LAB

SUBJECT CODE: BCSE 0133

SESSION: 2023-24

SUBMITTED TO

Alicia Passah

SUBMITTED BY

Naveen Agrawal

COURSE: B.TECH III YEAR **SECTION: D-2**

BRANCH: CS **CLASS ROLL NO: 41**

UNI. ROLL NO: 2115000653

INDEX

S. NO	Name of Experiment	Date of Perfrom	Date of Submission	Teacher's Signature

EXPERIMENT – 1

1.1 Introduction to Pandas, Upload data and data preprocessing

Pandas is a Python library used for working with data sets. It has functions for analyzing, cleaning, exploring, and manipulating data. The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis" and was created by Wes McKinney in 2008.

```
import pandas as pd

data = {'Name': ['John', 'Jane', 'Bob'], 'Age': [28, 35, 22], 'City': ['New York', 'San Francisco', 'Los Angeles']}

df = pd.DataFrame(data)

#Uploading data: import pandas as pd

 # Read CSV file

 df = pd.read_csv('your_file.csv')

# Display the first few rows of the DataFrame

print(df.head())
```

1.2 Introduction to Numpy and Matplotlib library in Python

Numpy: NumPy is a Python library used for working with arrays.It also has functions for working in domain of linear algebra, fourier transform, and matrices.NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely.NumPy stands for Numerical Python.

CODE

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
print(arr)
print(type(arr))
```

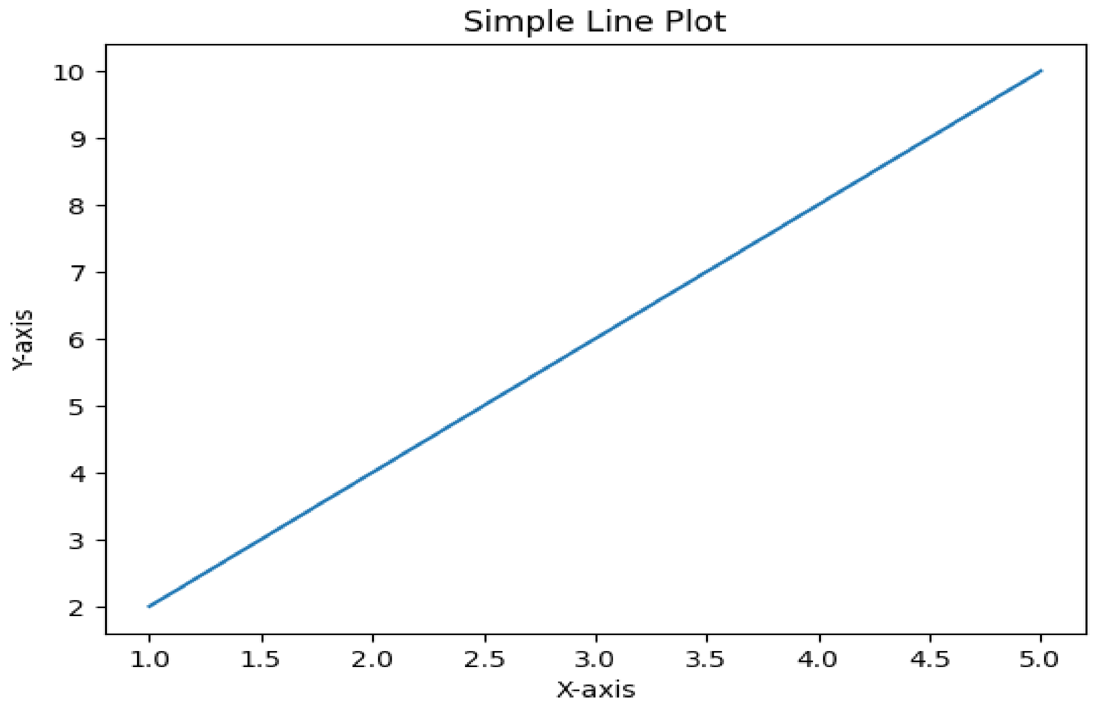
Output: [1,2,3,4,5]

<class ‘numpy.ndarray’>

Matplotlib: Matplotlib is an amazing visualization library in Python for 2D plots of arrays. Matplotlib is a multi-platform data visualization library built on NumPy arrays and designed to work with the broader SciPy stack. It was introduced by John Hunter in the year 2002. Matplotlib consists of several plots like line, bar, scatter, histogram, etc. It can be used for creating simple line plots, scatter plots, bar plots, histograms, and more.

CODE

```
import numpy as np

import matplotlib.pyplot as plt

x = np.array([1, 2, 3, 4, 5])

y =np.array([2, 4, 6, 8, 10])

plt.plot(x, y)

plt.xlabel('X-axis')

plt.ylabel('Y-axis')

plt.title('Simple Line Plot')

plt.show()
```



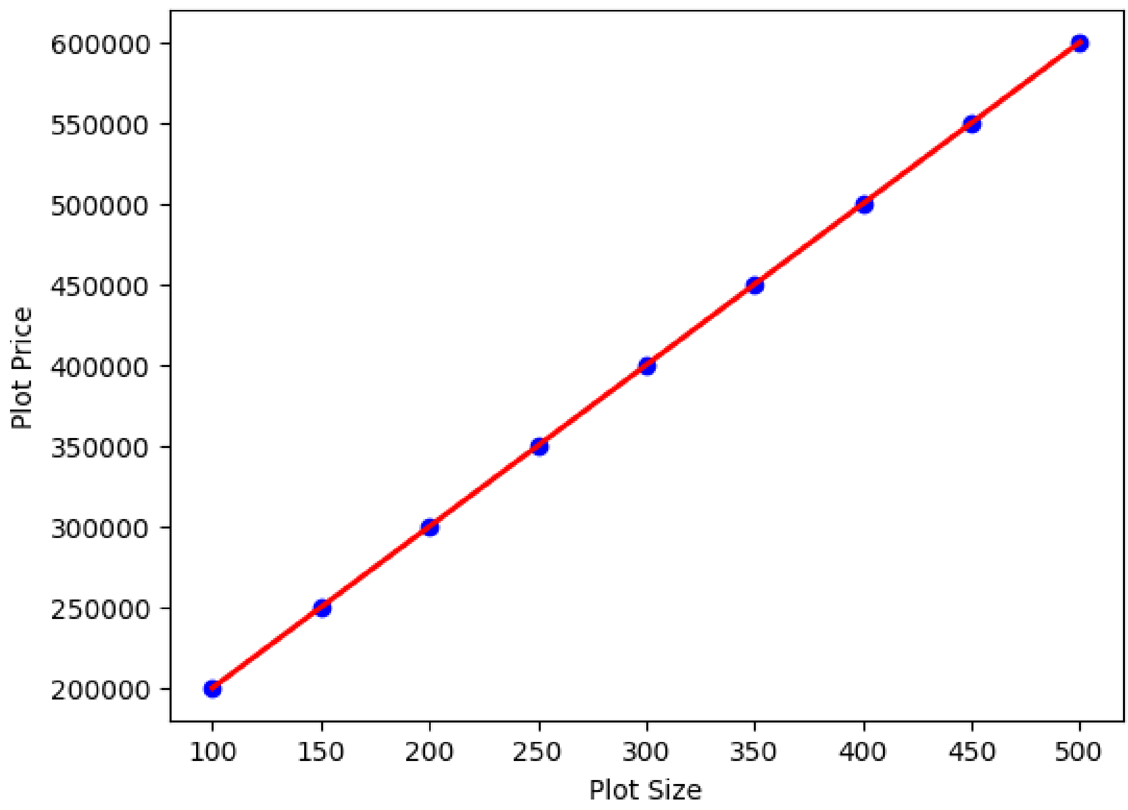
EXPERIMENT – 2

Implement Linear Regression with one variable in Python

Linear Regression: Linear regression models the relationship between a dependent variable and one or more independent variables by finding the best-fit line. It's a widely used statistical method for prediction and analysis.

CODE

```
import pandas as pd

from sklearn.linear_model import LinearRegression

import matplotlib.pyplot as plt

data={ 'plot_size' :[100,150,200,250,300,350,400,450,500],

'plot_price':[200000,250000,300000,350000,400000,450000,500000,550000,600000]}

df =pd.DataFrame(data)

x=df[['plot_size']]

y=df['plot_price']

model=LinearRegression()

model.fit(x,y)

new_sizes = [[600], [700]]

predicted_prices = model.predict(new_sizes)

print('Predicted Prices:')

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.plot(x,model.predict(x), color='red', linewidth=2,label='Linear Regression')

plt.xlabel('Plot Size')

plt.ylabel('Plot Price')

plt.show()
```



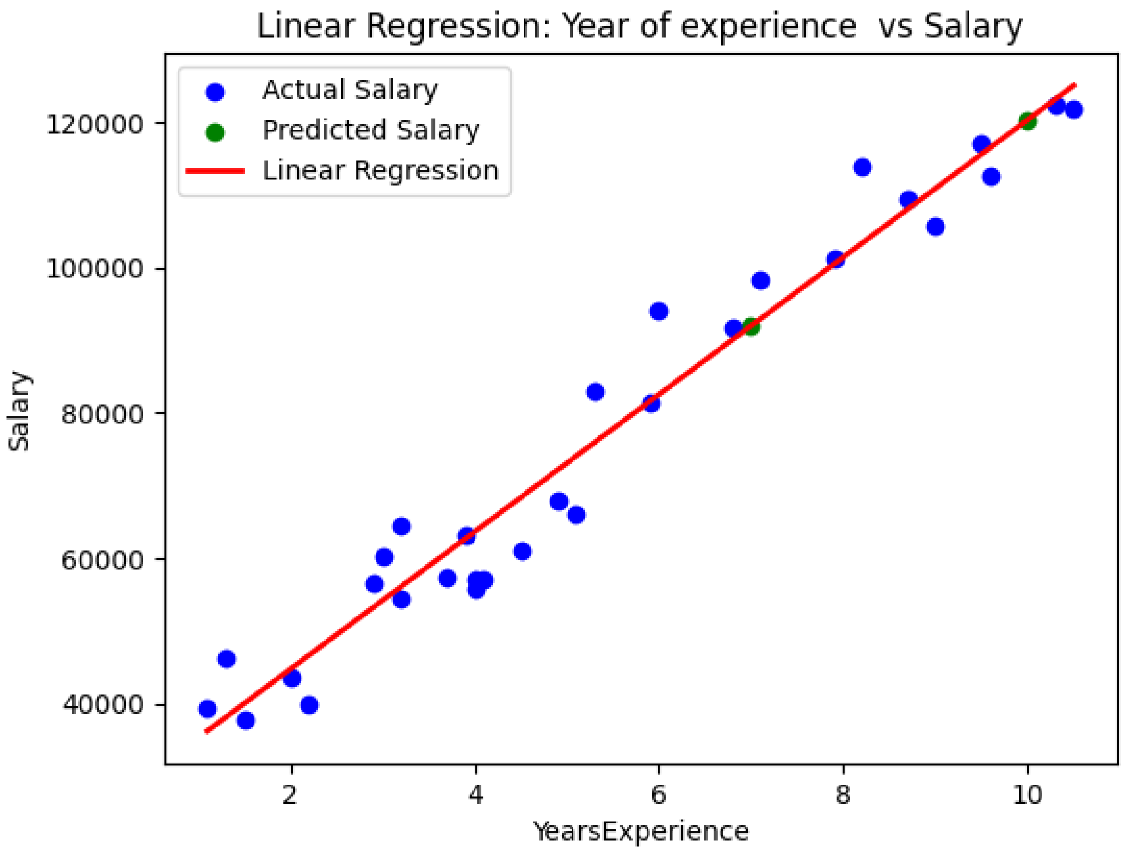
EXPERIMENT – 3

Implement Linear Regression with multiple variables in Python

CODE

```
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
data=pd.read_csv('/content/Salary_Data.csv')
print(data)
df =pd.DataFrame(data)
# Extract the features (Plot_Size) and target variable (Plot_Price)
x = df[['YearsExperience']]
y = df['Salary']
# Create and train a linear regression model
model = LinearRegression()
model.fit(x, y)
new_sizes = [[10], [7]]
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 the regression line
plt.scatter(x, y, color='blue', label='Actual Salary')
plt.scatter(new_sizes,predicted_prices ,color='green' ,label='Predicted Salary')
plt.plot(x, model.predict(x), color='red', linewidth=2, label='Linear Regression')
plt.xlabel('YearsExperience')
plt.ylabel('Salary')
plt.legend()
plt.title('Linear Regression: Year of experience  vs Salary')
plt.show()
```



EXPERIMENT – 4

Implement binary classification using Logistic Regression in Python

Logistic Regression is a statistical method used for binary and multiclass classification tasks in machine learning. Despite its name, it's a classification algorithm, not a regression one. The algorithm models the probability of an instance belonging to a particular class using a logistic (or sigmoid) function.Logistic Regression is widely used due to its simplicity, efficiency, and interpretability.

CODE

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
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
data=pd.get_dummies(data,columns=['country','gender',])
data.head()
data.shape
d=data.isnull()
data=data.dropna()
data.shape
x=data.drop('churn',axis=1)
y=data['churn']
x.head()
x.shape
y.shape
y.head()
#split data
x_train , x_test , y_train , y_test = train_test_split(x,y , test_size=0.3 ,
random_state =42)
#model training
model = LogisticRegression(random_state=42)
```

```
model.fit(x_train , y_train)
#model evolution
y_pred = model.predict(x_test)
#display performance
accuracy = accuracy_score(y_test,y_pred)
print(f'Accuracy {accuracy : .2f}')
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'Precison {precision : .2f}')
print(f'Recall {recall : .2f}')
print(f'f1_score {f1 : .2f}')
print('Confusion matrix')
print(conf_matrix)
data['churn'].value_counts()
```

OUTPUT

```
Precison  0.00
Recall  0.00
f1_score  0.00
Confusion matrix
[[2416    0]
 [ 584    0]]
```

EXPERIMENT – 5

Implement Principle Component Analysis (PCA) in Python

PCA: PCA (Principal Component Analysis) in machine learning is a dimensionality reduction technique that transforms high-dimensional data into a lower-dimensional representation while preserving the most important information. It achieves this by identifying and retaining the principal components, which are the directions of maximum variance in the data. PCA is commonly used for feature extraction and simplifying complex datasets.

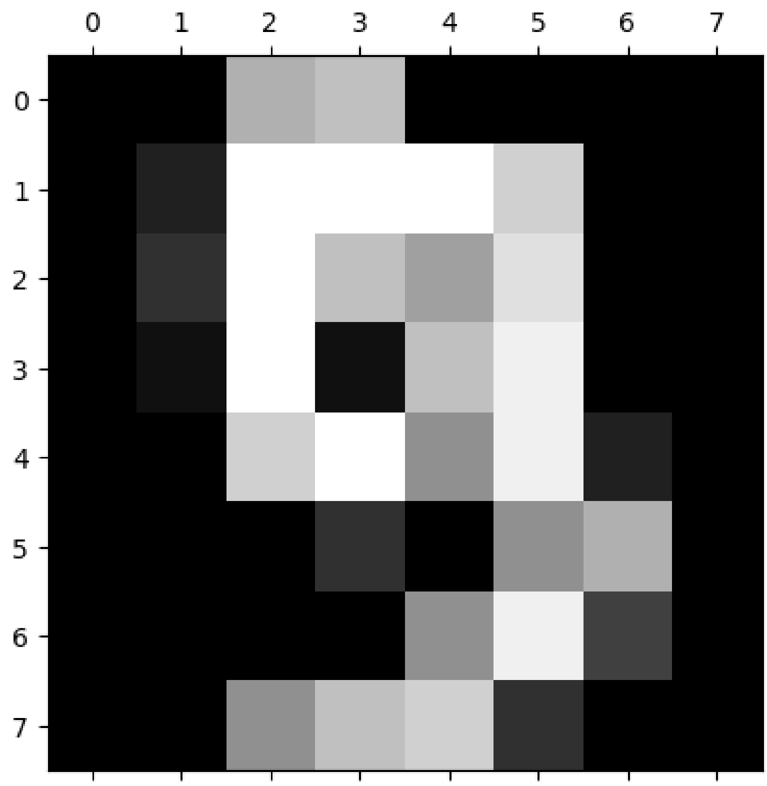
CODE

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_digits
import pandas as pd
dataset=load_digits()
dataset.keys()
dataset.target_names
dataset.data.shape
dataset.data[1796]
a=dataset.data[9].reshape(8,8)
from matplotlib import pyplot as plt
#matplotlib innline
plt.gray()
plt.matshow(a)
df = pd.DataFrame(dataset.data,columns=dataset.feature_names)
df.head()
x=df
y=dataset.target
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
x_scaled=scaler.fit_transform(x)
x_scaled
#scaled the value from -1 to 1
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(x_scaled,y,test_size=0.2,
random_state=30)
from sklearn.linear_model import LogisticRegression
model =LogisticRegression()
model.fit(X_train, y_train)
#model.score(x_test,y_test)
y_pred=model.predict(X_test)
from sklearn.metrics import accuracy_score
accuracy=accuracy_score(y_test,y_pred)
print(f'Accuracy:{accuracy: .5f}')
#use PCA to reduce dimension from 64 to a lower number and check if
results improve or degraded
from sklearn.decomposition import PCA
```

```
pca= PCA(n_components=40) #this means to reduce and use those number
of components such that 95%
X_pca=pca.fit_transform(x_scaled)
X_pca.shape
scaler=StandardScaler()
x_scaled=scaler.fit_transform(x)
x_scaled
#scaled the value from -1 to 1
X_train_pca,X_test_pca,y_train,y_test=train_test_split(X_pca,y,test_size=0
.2,random_state=30)
model=LogisticRegression(max_iter=1000)
model.fit(X_train_pca,y_train)
y_pred_pca=model.predict(X_test_pca)
accuracy=accuracy_score(y_test,y_pred_pca)
print(f'Accuracy:{accuracy: .5f}')
```

OUTPUT

```
Accuracy: 0.97222
Accuracy: 0.96111
<Figure size 640x480 with 0 Axes>
```



EXPERIMENT – 6

Implement Support Vector Machine (SVM) classifier in Python

SVM : Support Vector Machine (SVM) is a powerful algorithm for classification and regression tasks. It finds a hyperplane that maximizes the margin between classes, using support vectors (closest data points). SVM can handle high-dimensional data and employs the kernel trick for non-linear relationships. It is effective but can be sensitive to noise.

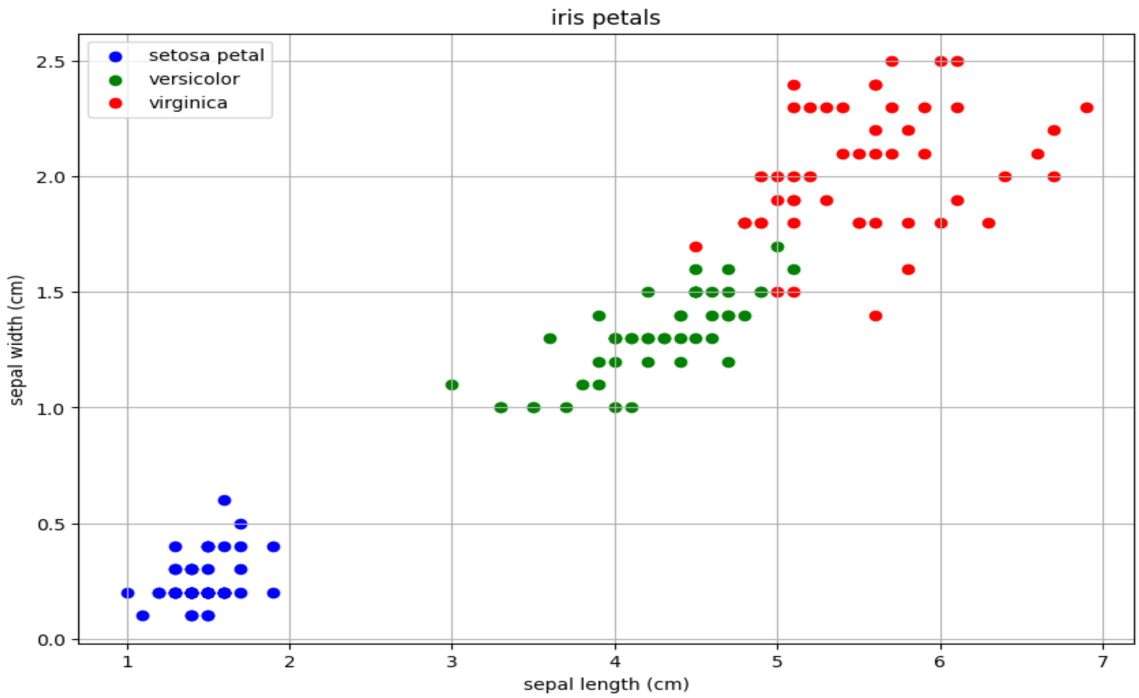
CODE

```
from sklearn.datasets import load_digits
import pandas as pd
dataset=load_digits()
dataset.keys()
dataset.target_names
dataset.data.shape
dataset.data[1796]
a=dataset.data[9].reshape(8,8)
from matplotlib import pyplot as plt
#matplotlib innline
plt.gray()
plt.matshow(a)
df = pd.DataFrame(dataset.data,columns=dataset.feature_names)
df.head()
x=df
y=dataset.target
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
x_scaled=scaler.fit_transform(x)
x_scaled
#scaled the value from -1 to 1
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(x_scaled,y,test_size=0.2,
random_state=30)
from sklearn.linear_model import LogisticRegression
model =LogisticRegression()
model.fit(X_train, y_train)
#model.score(x_test,y_test)

y_pred=model.predict(X_test)
```

```
from sklearn.metrics import accuracy_score
accuracy=accuracy_score(y_test,y_pred)
print(f'Accuracy:{accuracy: .5f}')
#use PCA to reduce dimension from 64 to a lower number and check if results improve or degraded
from sklearn.decomposition import PCA
pca= PCA(n_components=40) #this means to reduce and use those number of components such that 95%
X_pca=pca.fit_transform(x_scaled)
X_pca.shape
scaler=StandardScaler()
x_scaled=scaler.fit_transform(x)
x_scaled
#scaled the value from -1 to 1
X_train_pca,X_test_pca,y_train,y_test=train_test_split(X_pca,y,test_size=0.2,random_state=30)
model=LogisticRegression(max_iter=1000)
model.fit(X_train_pca,y_train)
y_pred_pca=model.predict(X_test_pca)
accuracy=accuracy_score(y_test,y_pred_pca)
print(f'Accuracy:{accuracy: .5f}')
```

OUTPUT

Accuracy using linear: 0.9333333333333333
Accuracy using rbf: 0.9666666666666667
Accuracy using poly: 0.9333333333333333



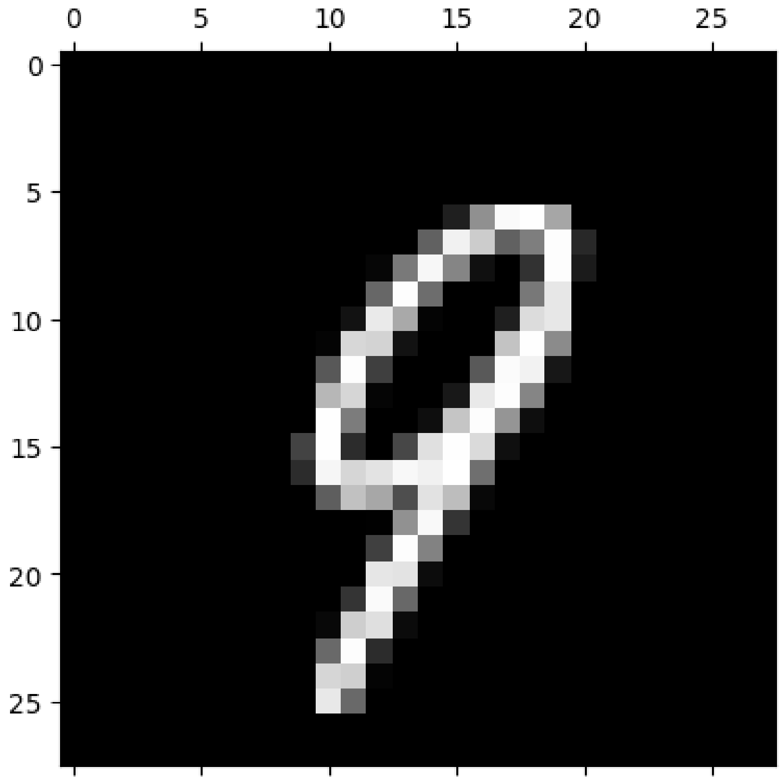
EXPERIMENT – 7

Implement multi-classification using Artificial Neural Network (ANN) in Python

ANN : An Artificial Neural Network (ANN) is an information processing paradigm that is inspired by the brain. ANNs, like people, learn by examples. An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process. Learning largely involves adjustments to the synaptic connections that exist between the neurons.

CODE

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
#%matplotlib inline
(x_train, y_train),(x_test, y_test)=keras.datasets.mnist.load_data()
len(x_train)
len(x_test)
x_train.shape
x_test.shape
plt.matshow(x_train[22])
x_train[0]
x_train=x_train/255
x_test=x_test/255
x_train[44000]
# flatting the dataset in order to compute for model building
x_train_flatten =x_train.reshape(len(x_train),28*28)
x_test_flatten =x_test.reshape(len(x_test),28*28)
x_train_flatten[0]
model = keras. Sequential
([keras.layers.Dense(10,input_shape=(784,),activation='sigmoid')])
model.compile(optimizer='adam',loss='sparse_categorical_crossentropy',m
etrics=['accuracy'])
model.fit(x_train_flatten, y_train,epochs=5)
model.evaluate(x_test_flatten, y_test)
```



EXPERIMENT – 8

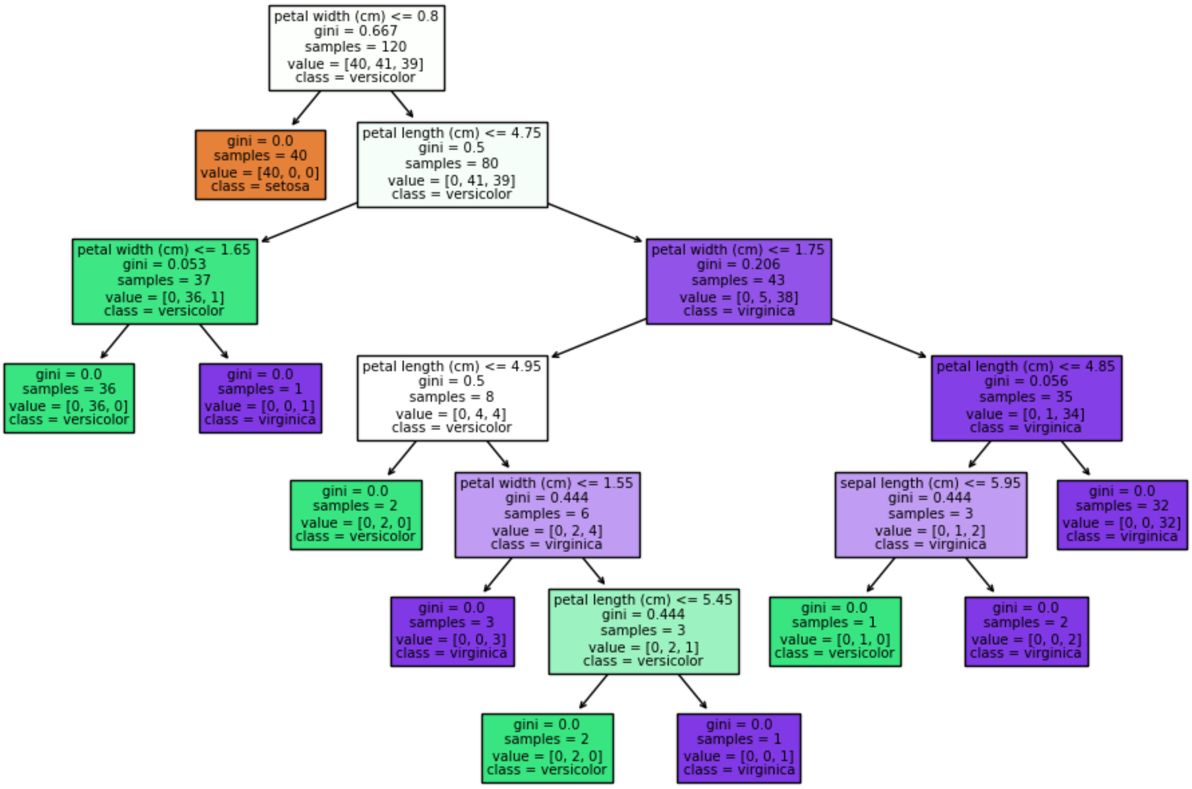
Implement Decision Tree (DT) classification in Python

Decision Tree: A decision tree is a non-parametric supervised learning algorithm, which is utilized for both classification and regression tasks. It has a hierarchical, tree structure, which consists of a root node, branches, internal nodes and leaf nodes.

CODE

```
import numpy as np
import pandas as pd
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree
# load the iris dayaset
iris=datasets.load_iris()

x=iris.data
y=iris.target
```

```
#spliting the dataset into training and testing sets
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_stat
e=42)
#create a decision tree model
dt_model=DecisionTreeClassifier()
#train the model on the training set
dt_model.fit(x_train,y_train)
#make predictions on the testing set
y_pred=dt_model.predict(x_test)
#evaluate the model
accuracy=metrics.accuracy_score(y_test,y_pred)
print(f"accuracy: {accuracy}" )
#plot the decision tree
plt.figure(figsize=(12,8))
plot_tree(dt_model, feature_names=iris.feature_names,
class_names=iris.target_names, filled=True)
plt.show()
```



EXPERIMENT – 9

Implement K-Nearest Neighbor (KNN) in Python

KNN : KNN, or k-Nearest Neighbors, is a versatile machine learning algorithm used for classification and regression. It assigns a data point's outcome based on the majority class or average value of its k closest neighbors in the feature space. The algorithm is straightforward and relies on the notion that similar data points often share similar outcomes, making it especially useful for pattern recognition and recommendation systems.

CODE

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
#load the datsets
data=pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data',names=['sepal_length','sepal_width','petal_length','petal_width','species'])
#seperate features and target lables
x=data[['sepal_length','sepal_width','petal_length','petal_width']]
y=data['species']
#splitting the data
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.25,random_state=42)
#create and train the KNN classifier on the dataset
knn=KNeighborsClassifier(n_neighbors=5)
knn.fit(x_train,y_train)
#make predictions with test data
y_pred=knn.predict(x_test)
#evaluate performance
accuracy=accuracy_score(y_test,y_pred)
print("k_nearest neighbors classfier accuracy:",accuracy)
```

OUTPUT

```
k_nearest neighbors classfier accuracy: 1.0
```

EXPERIMENT – 10

Implement Random Forest in Python

Random Forest : Random Forest is an ensemble learning technique that builds multiple decision trees during training and merges their predictions. It enhances accuracy and reduces overfitting by aggregating the results of individual trees. Each tree is trained on a random subset of data and features, contributing to a diverse set of models. This method is robust, handles high-dimensional data well, and is widely used for classification and regression tasks in machine learning.

CODE

```
import numpy as np
import pandas as pd
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree
# load the iris dayaset
iris=datasets.load_iris()
x=iris.data
y=iris.target
#spliting the dataset into training and testing sets
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_stat
e=42)
#create a decision tree model
dt_model=RandomForestClassifier()
#train the model on the training set
dt_model.fit(x_train,y_train)
#make predictions on the testing set
y_pred=dt_model.predict(x_test)
#evaluate the model
accuracy=metrics.accuracy_score(y_test,y_pred)
print(f"accuracy: {accuracy}" )
```

OUTPUT

```
accuracy: 1.0
```

EXPERIMENT – 11

Implement Naïve Bayes Claasifier (NB) in Python

Naïve Bayes : Naive Bayes is a probabilistic machine learning algorithm based on Bayes' theorem. It is particularly suited for classification tasks. The "naive" assumption in Naive Bayes is that features are conditionally independent given the class label, simplifying the computation of probabilities. Despite its simplicity, Naive Bayes often performs well in various real-world applications, such as text classification and spam filtering.

CODE

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score,classification_report
iris=load_iris()
x=iris.data
y=iris.target
iris.data.shape
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.25,ra
ndom_state=42)
nb=MultinomialNB()
nb.fit(x_train,y_train)
y_pred=nb.predict(x_test)
accuracy=accuracy_score(y_test,y_pred)
print(f"Accuracy: {accuracy}")
report=classification_report(y_test,y_pred,target_names=iris.targ
et_names)
print("classfication report:\n",report)
```

OUTPUT

Accuracy: 0.9736842105263158
classfication report:

	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	15
versicolor	0.92	1.00	0.96	11
virginica	1.00	0.92	0.96	12
accuracy			0.97	38
macro avg	0.97	0.97	0.97	38
weighted avg	0.98	0.97	0.97	38