In [ ]:

```
import numpy as np # linear algebra
import pandas as pd
import os
cwd = os.getcwd()
print(cwd)

import pandas as pd

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

print(df.to_string())

import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

df.head()

X = df.iloc[:, :-1].values    # Features => Years of experience => Independent Variable
y = df.iloc[:, -1].values     # Target => Salary => Dependent Variable

X


y

# divide the dataset in some amount of training and testing data
from sklearn.model_selection import train_test_split
import sklearn.metrics as sm


# random_state => seed value used by random number generator
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train, y_train)

predictions = model.predict(X_test)
predictions

y_test

import seaborn as sns
sns.distplot(predictions-y_test)

plt.scatter(X_train, y_train, color='red')
plt.plot(X_train, model.predict(X_train))

r_sq = model.score(X_train, y_train)
print('coefficient of determination:', r_sq)

# Print the Intercept:
print('intercept:', model.intercept_)

# Print the Slope:
print('slope:', model.coef_)

# Predict a Response and print it:
y_pred = model.predict(X_train)
```

```
print('Predicted response:', y_pred, sep='\n')

print('y='+str(float(model.coef_))+'X+'+str(float(model.intercept_)))
```

In [ ]:

```
## Program 2: K - Means Clustering
K-Means is an unsupervised machine learning algorithm that clusters data together
based on similarity metrics like
Euclidean Distance.
K-Means works like this :
1. First determines the number of groups/clusters (called as K),
2. Then it randomly chooses initial K centroids from data points,
3. Next it assigns data points to the cluster of nearest centroid
3. Then it updates centroids in each iteration until clusters converge.

The K in K-Means comes from the number of clusters that need to be set prior to
starting the iteration process.
We can choose the best value of K using The Elbow Method.
The best value of K is one that results in groups with minimum variance within a
single cluster.
This measure is called Within Cluster Sum of Squares, or WCSS for short.
The smaller the WCSS is, the closer our points
are, therefore we have a more well-formed cluster.

### Data Set: wine_clustering.csv
These data are the results of a chemical analysis of wines grown in the
same region in Italy but derived from three
different cultivars. The analysis determined the quantities of 13
constituents found in each of the three types of wines.

The attributes are:

- Alcohol
- Malic acid
- Ash
- Alcalinity of ash
- Magnesium
- Total phenols
- Flavanoids
- Nonflavanoid phenols
- Proanthocyanins
- Color intensity
- Hue
- OD280/OD315 of diluted wines
- Proline

# Import required libraries and read data into a dataframe
import pandas as pd
df = pd.read_csv('wine-clustering.csv')
df.head()

# Do some data exploration
df.describe().T


df.info()

# Visual data exploration to identify correlation among columns of data
import seaborn as sns
sns.pairplot(df)

#Import algorithms from sklearn
from sklearn.cluster import KMeans
```

```
#the columns we will use for clustering are only two - the
OD280 and Alcohol content of wines
selected_features = df[['OD280', 'Alcohol']]

# The random_state needs to be the same number to get reproducible results
kmeans_obj = KMeans(n_clusters=3, random_state=42)

# Fit the Kmeans algorithm on selected columns
kmeans_obj.fit(selected_features)
# Predict the cluster labels for data
y_kmeans = kmeans.fit_predict(selected_features)

#Print the predicted labels
print(y_kmeans)

# Printing the cluster centers
centers = kmeans.cluster_centers_
print(centers)

#Visualize the Groups created
sns.scatterplot(x = selected_features['OD280'],
                y = selected_features['Alcohol'], hue=kmeans_obj.labels_)
#Visualize the cluster centroids
plt.scatter(kmeans.cluster_centers_[:, 0],
            kmeans.cluster_centers_[:, 1], s=200, c='red')
```

In [ ]:

```
#### Machine Learning Lab Program 3

Aim : Write a python program to classify the medical dataset using
    K Nearest Neighbor Algorithm.
    The students are expected to demonstrate how you can perform
    basic data processing operations,
    split the dataset into training and test sets, train the model,
    score the test dataset,
    and evaluate the predictions.

Description: In this program,  you will use Breast Cancer Wisconsin
    dataset
    (originally from UCI Machine Learning Repository) to train a K-nearest
    neighbor model.
    This model will be used to classify the test data into one of the two classes
    - benign or malignant.
    i.e. you will predict the diagnosis: B = benign, M = malignant

DataSet:  The Breast Cancer Wisconsin dataset from UCI machine learning repository
    is a classification dataset.
    It contains a total of 32 columns, first column is patient id, second column is
    the diagnosis - "B" for Benign ,
    "M" for Malignant. Remaining 30 columns each represent the features that are the
    measurements for breast cancer patients.
    The features are computed from a digitized image of a fine needle aspirate (FNA)
    of a breast mass.
    They describe characteristics of the cell nuclei present in the image.

Dataset Description:
https://data.world/health/breast-cancer-wisconsin/workspace/file?
    filename=DatasetDescription.txt


Dataset url:
https://data.world/health/breast-cancer-wisconsin/workspace/file?
    filename=breast-cancer-wisconsin-data%2Fdata.csv

# import required libraries
import numpy as np
import pandas as pd



#if you have downloaded the dataset to you local computer you can use this
syntax to read the file into your pandas dataframe

data = pd.read_csv("breast-cancer-wisconsin-data_data.csv")

data.head()

data.columns

data = data.drop(['id', 'Unnamed: 32'], axis = 1)

data.shape


data.describe()
```

```
data.info()

data.columns

# Extract the columns that will be the features and the target variable
(diagnosis) in X and y respectively
X = data.loc[:, ['radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst']]
y = data.loc[:, 'diagnosis']

X.head()

y.head()

#Train - Test Split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
                                                    random_state=42)


# Fit the KNN Classifier
from sklearn.neighbors import KNeighborsClassifier
knn_cfr = KNeighborsClassifier(n_neighbors=3)
knn_cfr.fit(X_train, y_train)

# Use the fitted model to make prediction for test data
y_pred = knn_cfr.predict(X_test)

# Print the accuracy score of your model
# accuracy = Number of correct predictions / total number of predictions
# accuracy_score = the number of test samples for which y_pred == y_test

from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)

# This model has an accuracy of 94.14 %
```

In [ ]:

```
Machine Learning Lab Program 4

Aim : Predict the real estate sales price of a house based upon various
    quantitative features about the house and sale.

Implementation: Demonstrate basic data processing operations, split
    dataset into training and test sets,
    train the model, score the test dataset and evaluate the predictions.

DataSet: Dataset containing house features and prices

Dataset url: https://data.world/swarnapuri-sude/house-data/workspace/file?
        filename=kc_house_data.csv

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

data = pd.read_csv("kc_house_data.csv")

data.head()

data  = data.drop(["id", "date"], axis = 1)
data.head()

data.describe()

data['bedrooms'].value_counts().plot(kind='bar')
plt.title('number of Bedroom')
plt.xlabel('Bedrooms')
plt.ylabel('Count')
sns.despine()

plt.figure(figsize=(10,10))
sns.jointplot(x=data.lat.values, y=data.long.values, height=10)
plt.ylabel('Longitude',fontsize=12)
plt.xlabel('Latitude',fontsize=12)
sns.despine()
plt.show()

#Visualizing common factors are affecting the price of the houses

plt.scatter(data.price,data.sqft_living)
plt.title("Price vs Square Feet")

plt.scatter(data.price,data.long)
plt.title("Price vs Location of the area")

plt.scatter(data.price,data.lat)
plt.xlabel("Price")
plt.ylabel('Latitude')
plt.title("Latitude vs Price")

plt.scatter(data.bedrooms,data.price)
plt.title("Bedroom and Price ")
plt.xlabel("Bedrooms")
plt.ylabel("Price")
sns.despine()
```

```
plt.show()

plt.scatter((data['sqft_living']+data['sqft_basement']),data['price'])

plt.scatter(data.waterfront,data.price)
plt.title("Waterfront vs Price ( 0= no waterfront)")

#Extracting X features and y label
y = data['price']
X = data.drop(['price'],axis=1)

from sklearn.model_selection import train_test_split
x_train , x_test , y_train , y_test = train_test_split(X , y ,
                                    test_size = 0.10,random_state =2)

from sklearn.linear_model import LinearRegression
reg = LinearRegression()
reg.fit(x_train,y_train)
reg.score(x_test,y_test)
```

In [ ]:

```
## Prog 5: Write a python program to predict income levels of adult individuals
#using Decision  Tree  Model.  The process   includes   training,   testing
#and   evaluating   the model  on  the  Adult  dataset.

In  this experiment you  need  to  train  a  classifier  on  the  "adult"
dataset,  and predict whether an
individual’s income is greater or less than $50,000. Perform  basic
data  processing  operations, split
the  dataset  into  training  and  test  sets,  train  the model,
score    the    test  dataset,    and
evaluate    the predictions.


Dataset: The Adult dataset is from the Census Bureau and the task is
    to predict whether a given adult
    earns more than $50,000 a year or not based attributes such as
    education, hours of work per week, etc..
URL: https://www.kaggle.com/datasets/wenruliu/adult-income-dataset/
        download?datasetVersionNumber=2

It has a total of 15 columns,
Target Column is "Income", The income is divide into two classes:
    <=50K and >50K
Number of attributes: 14, These are the demographics and other
    features to describe a person

14 attributes are:

- Age.
- Workclass.
- Final Weight.
- Education.
- Education Number of Years.
- Marital-status.
- Occupation.
- Relationship.
- Race.
- Gender.
- Capital-gain.
- Capital-loss.
- Hours-per-week.
- Native-country.
The dataset contains missing values that are marked with a question
mark character (?).
There are a total of 48,842 rows of data, and 3,620 with missing values,
leaving 45,222 complete rows.
There are two class values ‘>50K‘ and ‘<=50K‘, i.e.,
it is a binary classification task.

#Required imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

#Read dataset
df = pd.read_csv("adult.csv")

df.head()
```

```
df.columns

df.shape

# See the columns that contain a "?" and how many "?"
#are there in those columns
df.isin(['?']).sum()

#Replace ? with NaN
df['workclass'] = df['workclass'].replace('?', np.nan)
df['occupation'] = df['occupation'].replace('?', np.nan)
df['native-country'] = df['native-country'].replace('?', np.nan)

#Now the ? has been replaced by NaN, so count of ? is 0
df.isin(['?']).sum()

#Check missing values - NaN values
df.isnull().sum()

#Drop all rows that contain a missing value
df.dropna(how='any', inplace=True)

#Check duplicate values in dataframe now
print(f"There are {df.duplicated().sum()} duplicate values")

df = df.drop_duplicates()

df.shape

df.columns

#Drop non-relevant columns
df.drop(['fnlwgt','educational-num','marital-status','relationship',
         'race',], axis = 1, inplace = True)

df.columns

#Extract X and y from the dataframe , income column is the target column,
rest columns are features
X = df.loc[:,['age', 'workclass', 'education', 'occupation',
              'gender', 'capital-gain',
       'capital-loss', 'hours-per-week', 'native-country']]
y = df.loc[:,'income']

X.head()

y.head()

# Since y is a binary categorical column we will use label encoder to
#convert it into numerical columns with values 0 and 1
from sklearn.preprocessing import LabelEncoder
y = LabelEncoder().fit_transform(y)
y = pd.DataFrame(y)
y.head()

#First identify caterogical features and numeric features
numeric_features = X.select_dtypes('number')
categorical_features = X.select_dtypes('object')
categorical_features

numeric_features
```

```
#Convert categorical features into numeric
converted_categorical_features = pd.get_dummies(categorical_features)
converted_categorical_features.shape

#combine the converted categorical features and the numeric features
together into a new dataframe called "newX"
all_features = [converted_categorical_features, numeric_features]
newX = pd.concat(all_features,axis=1, join='inner')
newX.shape

newX.columns

#Do a train test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(newX, y,
                                        test_size=0.33, random_state=42)

# Load Decision Tree Classifier, max_depth = 5 and
#fit it with X-train and y-train
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier(max_depth=5)
clf.fit(X_train, y_train)

# Make predictions
y_pred = clf.predict(X_test)

y_test.shape

y_pred.shape

predictions_df = pd.DataFrame()
predictions_df['precdicted_salary_class'] = y_pred
predictions_df['actual_salary_class'] = y_test[0].values
predictions_df

#Evaluate the performance of fitting
from sklearn.metrics import accuracy_score
print(accuracy_score(y_pred,y_test))

#Plot your decision tree
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt
plt.figure(figsize=(14,14))
plot_tree(clf, fontsize=10, filled=True)
plt.title("Decision tree trained on the selected features")
plt.show()
```

In [ ]:

```
### Program 6: Write a python program to predict income levels
#of adult individuals using Support Vector Machine Model.
The process includes training, testing and evaluating the
model on the Adult dataset. In this experiment
you need to train a classifier on the Adult dataset,
to predict whether an individual’s income is
greater or less than $50,000.



Dataset: We have used a smaller version of adult income dataset.
    This dataset has 3574 rows and 7 columns.
It has a total of 15 columns, Target Column is "Income",
The income is divide into two classes: <=50K and >50K
    Number of attributes: 6, These are the demographics
        and other features to describe a person

6 attributes are:

- Age.
- Workclass.
- Education Number of Years.
- Occupation.
- gender.
- Hours-per-week.

The dataset contains missing values that are marked
with a question mark character (?).
There are two class values ‘>50K‘ and ‘<=50K‘
in target column i.e., it is a binary classification task.

#Required imports
import numpy as np
import pandas as pd


#Read dataset
df = pd.read_csv("smaller_adult.csv")
df.head()

df.columns

df.shape

df.info()


df.describe()

# See the columns that contain a "?" and how many "?"
#are there in those columns
df.isin(['?']).sum()

df.columns

#Replace ? with NaN
df['workclass'] = df['workclass'].replace('?', np.nan)
df['occupation'] = df['occupation'].replace('?', np.nan)
```

```
#Now the ? has been replaced by NaN, so count of ? is 0
df.isin(['?']).sum()

#Check missing values - NaN values
df.isnull().sum()

#Drop all rows that contain a missing value
df.dropna(how='any', inplace=True)

#Check duplicate values in dataframe now
print(f"There are {df.duplicated().sum()} duplicate values")

df = df.drop_duplicates()
df.shape

df.columns

#Extract X and y from the dataframe , income column is the
#target column, rest columns are features
X = df.loc[:,['age', 'workclass', 'educational-num',
              'occupation', 'gender', 'hours-per-week']]
y = df.loc[:,'income']

# Since y is a binary categorical column we will use label
#encoder to convert it into numerical columns with values 0 and 1
from sklearn.preprocessing import LabelEncoder
y = LabelEncoder().fit_transform(y)
y = pd.DataFrame(y)
y.head()

#First identify caterogical features and numeric features
numeric_features = X.select_dtypes('number')
categorical_features = X.select_dtypes('object')
categorical_features

numeric_features

#Convert categorical features into numeric
converted_categorical_features = pd.get_dummies(categorical_features)
converted_categorical_features.shape

#combine the converted categorical features and the numeric features
#together into a new dataframe called "newX"
all_features = [converted_categorical_features, numeric_features]
newX = pd.concat(all_features,axis=1, join='inner')
newX.shape

newX.columns

#Do a train test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test =
train_test_split(newX, y, test_size=0.33, random_state=42)

# Load Support Vector Machine Classifier
from sklearn.svm import SVC
clf = SVC(kernel="linear", gamma = 'auto')
clf.fit(X_train, y_train.values.ravel())
```

```
# Make predictions
y_pred = clf.predict(X_test)

predictions_df = pd.DataFrame()
predictions_df['precdicted_salary_class'] = y_pred
predictions_df['actual_salary_class'] = y_test[0].values
predictions_df

#Evaluate the performance of fitting
from sklearn.metrics import accuracy_score
print(accuracy_score(y_pred,y_test))
```

In [ ]:

```
### Program7: Write a python program to classify the medical
dataset using Multilayer Perceptron Classifier.
The students are expected to demonstrate how you can perform
basic data processing operations, split the dataset
into training and test sets, train the model, score the test
dataset, and evaluate the predictions.

Dataset:
Description: In this program, you will use Breast Cancer
    Wisconsin dataset (originally from UCI Machine Learning
                       Repository)
    to train a K-nearest neighbor model. This model will be
    used to classify the test data into one of the two classes -
    benign or malignant. i.e. you will predict the diagnosis:
        B = benign, M = malignant

DataSet: The Breast Cancer Wisconsin dataset from UCI machine
    learning repository is a classification dataset.
    It contains a total of 32 columns, first column is patient
    id, second column is the diagnosis - "B" for Benign ,
    "M" for Malignant. Remaining 30 columns each represent the
    features that are the measurements for breast cancer patients.
    The features are computed from a digitized image of a fine
    needle aspirate (FNA) of a breast mass.
    They describe characteristics of the cell nuclei present in the image.

Dataset Description: https://data.world/health/breast
        -cancer-wisconsin/workspace/file?filename=DatasetDescription.txt

Dataset url: https://data.world/health/breast-cancer-wisconsin
        /workspace/file?filename=breast-cancer-wisconsin-data%2Fdata.csv

import numpy as np
import pandas as pd

df = pd.read_csv("breast-cancer-wisconsin-data_data.csv")
df.head()

df.shape

df = df.drop(['id', 'Unnamed: 32'], axis = 1)

df.columns

df.describe()

df.info()

# Extract the columns that will be the features and the target variable
#(diagnosis) in X and y respectively
X = df.loc[:, ['radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst']]
```

```
y = df.loc[:, 'diagnosis']

#Train - Test Split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.33, random_state=42)

from sklearn.neural_network import MLPClassifier
clf = MLPClassifier(random_state=1, max_iter=300).fit(X_train, y_train)

y_pred = clf.predict(X_test)

y_pred

# Print the accuracy score of your model
# accuracy = Number of correct predictions / total number of predictions
# accuracy_score = the number of test samples for which y_pred == y_test

from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)
```

In [ ]:

```
8th
from sklearn.datasets import load_iris
from sklearn.datasets import load_diabetes
from sklearn import metrics
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB

iris = load_diabetes()

# store the feature matrix (X) and response vector (y)
X = iris.data
y = iris.target

# splitting X and y into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.4, random_state=1)

# training the model on training set

gnb = GaussianNB()
gnb.fit(X_train, y_train)

# making predictions on the testing set
y_pred = gnb.predict(X_test)

# comparing actual response values (y_test) with predicted response values (y_pred)
from sklearn import metrics
print("Gaussian Naive Bayes model accuracy(in %):",
      metrics.accuracy_score(y_test, y_pred)*100)
```

In [ ]:

```
## Program 9: Comparison of performance of classifiers

In this program we will compare the classification
performance of
5 classifiers - K Nearest Neighbors, Support Vector
Machines, Decision Tree, Multilayer perceptron and
Gaussian Naive Bayes for the classification of Iris Data set.

Iris dataset contains 4 features
(SepalLength,   SepalWidth, PetalLength,    PetalWidth) and the
target "Species" the species names of different kinds of flowers.

import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.naive_bayes import GaussianNB



names = [
    "K-Nearest Neighbors",
    "Linear SVM",
    "Decision Tree",
    "Multilayer Perceptron",
    "Gaussian Naive Bayes",
    ]

classifiers = [
    KNeighborsClassifier(3),
    SVC(kernel="linear", C=0.025),
    DecisionTreeClassifier(max_depth=5),
    MLPClassifier(alpha=1, max_iter=1000),
    GaussianNB(),
]

df = pd.read_csv("Iris.csv")
df.head()

df = df.drop("Id", axis = 1)
df.head()

#Extract X and y as features and target
X= df.iloc[:, :-1]
X.head()

y = df.iloc[:, -1]
y.head()

#Since target column is categorical , we will
#convert it to numerical usign LabelEncoder
from sklearn.preprocessing import LabelEncoder
y = LabelEncoder().fit_transform(y)
y = pd.DataFrame(y)
y.head()
```

```
#Train-test split
X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.33, random_state=42)

# Using a for loop fit all the classifiers to
#X_train and y_train and print the accuracy score of each classifier
for name, clf in zip(names, classifiers):
    clf.fit(X_train, y_train.values.ravel())
    score = clf.score(X_test, y_test)
    print("Classifier Name: ", name, "Score: ",  score)
```

In [ ]:

```
### Program 10: Write a Python program to classify authentic
#and fake currency notes using Learning
(Deep Neural Network) Classifier

Dateset: This dataset was extracted from images that were
    taken from genuine and forged banknote-like specimens.
    The images were digitized. The final images have 400x
    400 pixels and a resolution of 660 dpi.
    There are four features in data variance, skewness,
    surtosis and entropy. Class is the target variable,
    it has two values 0 and 1 for the authentic and fake
    notes. Wavelet Transform tool were used to extract features
    from images.

# To create a deep learning model you will need to install
#the libraries Tensorflow and Keras on your computers
# To install these libraries uncomment the below two lines
#of code in this cell. This installs using pip command of Python

#!pip install tensorflow
#!pip install keras

#Required imports
import pandas as pd
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense

#Read data into a dataframe
df = pd.read_csv('BankNote_Authentication.csv')
df.head()

#Extract Features X and target y
X = df.values[:, :-1]
y = df.values[:, -1]

#Train-test split
X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.33, random_state=42)

#Initialize the deep neural network model with 7 layers,
#Sigmoid activation function at the Output last layer
model = Sequential()
model.add(Dense(7, input_shape=(X.shape[1],)))
model.add(Dense(1, activation = 'sigmoid'))

#Fit the model on X_train and y_train for 10 epochs
model.compile(optimizer='adam',loss='binary_crossentropy')
model.fit(X_train, y_train, epochs=30, batch_size=32)

#Make predictions using the model, this prediction is
#a floating point probability value
y_pred = model.predict(X_test)


#y_pred contains the predicted probability for each row in X-test
y_pred
```

```
#We are apply a condition that if a value in y_pred >0.5
#then it is made 1 else it is made 0.
# The resulting array is flattened to one dimensions
#and converted to integer datatype
y_pred = (y_pred>0.5).flatten().astype(int)

#Now y_pred contains integers 0 and 1 to represent
#two classes of noted - authentic and fake
y_pred

#Printing the accuracy of the model
print(accuracy_score(y_test, y_pred))
```