SHIV NADAR UNIVERSITY
CHENNAI

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
SCHOOL OF ENGINEERING

LABORATORY RECORD

B.TECH
(YEAR : 20 - 20)

NAME : KARTHICK.N.G

REG. NO. : 21110108

DEPT. : CSE **SEM. :** 4 **CLASS & SEC :** AI DS -'A"

www.snuchennai.edu.in

SHIV NADAR UNIVERSITY
UNIVERSITY
CHENNAI

BONAFIDE CERTIFICATE

Certified that this is the bonafide record of the practical work done in the

Machine Learning Techniques .... **Laboratory by**

Name KARTHICK.N.G ....

Register Number 21110108 ....

Semester 4 ....Class & Sec. AI DS -'A' ....

Branch AI & DS ....

SHIV NADAR UNIVERSITY Chennai

During the Academic year 2023 ....

Faculty **Head of the Department**

Submitted for the...Practical Examination held at SNU CHENNAI on.........................

Internal Examiner **External Examiner**

INDEX

Name : .... Reg. No. ....

Sem : .... Class & Sec : ....

Ex. No: 1	Regression
Date: 26-12-2022	

Aim:
To build a multiple linear regression model to predict the sale price of the house.

Algorithm:
Multiple Linear Regression:
- Import the necessary modules and the csv file.
- Fill the null values present in the data and pre-process the data.
- Use variance influence factor in the data and check for correlation and select the columns that have variance influence factor lesser than 3.
- Split the data into train and test using train-test-split from sklearn library.
- Import Linear regression from sklearn and fit the model.
- Predict the values using the test data and print the accuracy metrics.

Simple Linear Regression:
- Import the necessary modules and the csv file.
- Fix the dependent variable and independent variable calculate the slope and y-intercept using the mathematical formula calculate the variances of dependent and independent variable.
- Calculate the root mean square of the predicted values and find the line of best fit.

Program:

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

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

df.head()

df.info()

df.describe()

columns_numerical=df.select_dtypes(exclude='object')
column_numerical=columns_numerical.columns
column_numerical

columns_object=df.select_dtypes(include='object')
column_object=columns_object.columns
column_object
```

```
columns_numerical.info()

df['LotArea'].mean()
df['LotArea'].median()

columns_numerical['LotFrontage'].fillna(value=columns_numerical['LotFront
age'].mean(),inplace=True)

columns_numerical['MasVnrArea'].fillna(value=columns_numerical['MasVnrAre
a'].mean(),inplace=True)

columns_numerical['GarageYrBlt'].fillna(value=columns_numerical['GarageYr
Blt'].median(),inplace=True)

columns_numerical.info()

columns_object.info()

columns_object.drop(['Alley','FireplaceQu','PoolQC','Fence','MiscFeature'
],inplace=True,axis=1)

df1=columns_object.isna().sum()

df1

columns_object.fillna(method='ffill',inplace=True)

columns_object.columns

df_uni=columns_object.describe(include='object').T

def dummies(x):
    a=pd.get_dummies(columns_object[x],drop_first=True)
    return a

column_object=list(columns_object.columns)

len(column_object)

columns_object.describe(include='object').T

a=pd.DataFrame()
for i in column_object:
    c=dummies(i)
    a=pd.concat([a,c],axis=1)
    #c.drop_duplicates(inplace=True)


#l=a.columns[a.columns.duplicated()]
#s=list(a.columns.drop_duplicates())
```

```
a=a.loc[:,~a.columns.duplicated()]

len(l)

len(s)

df_final=a.join(columns_numerical)

df_final

def iqr(x):
    q1=df_final[x].quantile(0.25)
    q3=df_final[x].quantile(0.75)
    iqr=q3-q1
    upper=q3+1.5*iqr
    lower=q1-1.5*iqr
    print(i)
    print(upper,lower)
    return upper,lower

df_f=df_final.copy()

for i in column_numerical:
    a,b=iqr(i)
    df_f=df_f[(df_f[i]>b) | (df_f[i]<a)]

df_f

plt.boxplot(df_final['EnclosedPorch'])

for i in column_numerical:
    plt.boxplot(df_final[i])

import seaborn as sns

sns.boxplot(columns_numerical)

plt.figure(figsize=(8,250))
c=1
for i in columns_numerical:
    ax=plt.subplot(38,1,c)
    sns.boxplot(columns_numerical[i],orient='h',x=columns_numerical[i])
    c+=1
plt.show()

df_final

x=df_final.iloc[:,:-1].values
y=df_final.iloc[:,-1].values
x.shape
```

```
x=pd.DataFrame(x)
x.columns=df_final.columns[:-1]
x['intercept']=1

vif=pd.DataFrame()
vif['variables']=x.columns
from statsmodels.stats.outliers_influence import
variance_inflation_factor
vif['values']=[variance_inflation_factor(x.values,i) for i in
range(x.shape[1])]
vif

e=vif[vif['values']>3]

out=e['variables'].values[:-1]

df_final.drop(out,axis=1,inplace=True)

from sklearn.model_selection import train_test_split

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_s
tate=42)

from sklearn.linear_model import LinearRegression

reg=LinearRegression()
reg.fit(x_train,y_train)

y_pred=reg.predict(x_test)

from sklearn.metrics import
r2_score,mean_absolute_error,mean_squared_error
print("R2 score:",r2_score(y_pred,y_test))
print("Mean squared error:",mean_squared_error(y_pred,y_test))
print("Mean absolute error:",mean_absolute_error(y_pred,y_test))

x=x.values

x

df_test=pd.read_csv('test.csv')

df_test.info()

df_test.drop(['Alley','FireplaceQu','PoolQC','Fence','MiscFeature'],axis=
1,inplace=True)

df_test_num=df_test.select_dtypes(exclude='object')
df_test_obj=df_test.select_dtypes(include='object')
```

```
df_test_num.info()

df_test_num['LotFrontage'].fillna(value=columns_numerical['LotFrontage'].
mean(),inplace=True)
df_test_num['MasVnrArea'].fillna(value=columns_numerical['MasVnrArea'].me
an(),inplace=True)
df_test_num['GarageYrBlt'].fillna(value=columns_numerical['GarageYrBlt'].
median(),inplace=True)

df_test_obj.describe().T

df_test_obj.fillna(method='ffill',inplace=True)

def dummies(x):
    a=pd.get_dummies(df_test_obj[x],drop_first=True)
    return a

b=pd.DataFrame()
for i in column_object:
    c=dummies(i)
    b=pd.concat([b,c],axis=1)
    #c.drop_duplicates(inplace=True)

l=b.columns[b.columns.duplicated()]
s=list(b.columns.drop_duplicates())
b=b.loc[:,~b.columns.duplicated()]

len(b.columns)

b=b.join(df_test_num)

len(b.columns)

w=vif[vif['values']<3]

w=w['variables'].values

for i in w:
    if i not in b.columns:
        b[i]=0

test_x=b[w]

test_x.columns=x_train.columns

test_x['intercept']=1

test_x
```

```
x_train

house_price_predict=reg.predict(test_x)

test_x['SalePrice']=house_price_predict

test_x.head()

df_test['SalePrice']=house_price_predict
df_test.to_csv('Predicted House Price.csv')

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import math
df=pd.read_csv('data1.csv')
df.head()
x=df['x'].values
y=df['y'].values
mean_x=np.mean(x)
mean_y=np.mean(y)
r=f=d=0

f=((x-mean_x)*(y-mean_y)).sum()
d=(np.sqrt(((x-mean_x)**2)*((y-mean_y)**2))).sum()
r=f/d
S_y=((np.sqrt((y-mean_y)**2))).sum()/(len(y)-1)
S_x=((np.sqrt((x-mean_x)**2))).sum()/(len(x)-1)
b1=r*(S_y/S_x)
b1
b0=mean_y-b1*mean_x
b0
r
rmse=math.sqrt((1/len(y))*((y-mean_y)**2).sum())
rmse
mae=(abs(y-mean_y).sum())/len(y)
mae
print("Root mean squared error:",rmse)
print("Mean absolute error:",mae)
```

Output Screenshot:

```
R2 score: 0.16748732785203146
Mean squared error: 3018442661.65523
Mean absolute error: 38863.85721170338
```

43]:

;	Low	NoSeWa	CulDSac	FR2	FR3	...	OpenPorchSF	EnclosedPorch	3SsnPorch	ScreenPorch	PoolArea	MiscVal	MoSold	YrSold	intercept	SalePrice
)	0	0	0	0	0	...	88	0	0	0	0	0	7	2006	1	93304.995968
)	0	0	1	0	0	...	70	0	0	0	0	0	6	2009	1	159191.864368
)	0	0	0	0	0	...	42	0	0	0	0	0	11	2006	1	38853.260651
)	0	0	0	0	0	...	23	0	0	0	0	0	5	2006	1	117322.810292
)	0	0	0	0	0	...	0	0	0	0	0	0	10	2008	1	29592.051038

```
Root mean squared error: 10.982258419833325
Mean absolute error: 8.9
```

Result:
Hence, the multiple regression model is built and the sale price of the house is predicted and a simple linear regression model is built.

Ex. No: 2	Logistic Regression
Date: 02-01-2023	

Aim:
To build a logistic regression model to classify customer status.

Algorithm:

- Import the necessary modules and the csv file.
- Fill the null values present in the data and pre-process the data.
- Split the data into train and test using train-test-split from sklearn library.
- Using the standard scaler normalize the train data and the test data.
- Import Logistic regression from sklearn and fit the model.
- Predict the values using the test data and print the confusion matrix and classification report.

Program:

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

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

df.head()

df.info()

df.drop(['Churn Category','Churn Reason'],axis=1,inplace=True)

obj_col=df.select_dtypes(include='object')

for i in obj_col:
    print(i,":",obj_col[i].unique())

obj_col.drop(['Customer ID','City'],axis=1,inplace=True)

obj_col['Offer'].value_counts().index[0]

for i in obj_col:
    obj_col[i].fillna(method='ffill',inplace=True)

for i in obj_col:
    print(i,":",obj_col[i].unique())
```

```
df_obj=pd.DataFrame()
from sklearn.preprocessing import LabelEncoder

le=LabelEncoder()

for i in obj_col:
    df_obj[i]=le.fit_transform(obj_col[i])

df_obj.head()

df_obj.info()

num_col=df.select_dtypes(exclude='object')

num_col.head()

num_col.info()

num_col.describe()

plt.figure(figsize=(8,250))
c=1
for i in num_col:
    ax=plt.subplot(38,1,c)
    sns.boxplot(num_col[i],orient='h',x=num_col[i])
    c+=1
plt.show()

num_col.drop(['Zip Code','Latitude','Longitude'],axis=1,inplace=True)

sns.kdeplot(num_col['Avg Monthly Long Distance Charges'])

sns.kdeplot(num_col['Avg Monthly GB Download'])

num_col['Avg Monthly Long Distance Charges'].fillna(num_col['Avg Monthly
   Long Distance Charges'].mean(),inplace=True)

num_col['Avg Monthly GB Download'].fillna(num_col['Avg Monthly GB
   Download'].mean(),inplace=True)

num_col.info()

df_final=df_obj.join(num_col)

df_final.head()

x=df_final.iloc[:,:-1]
y=df_final.iloc[:,-1].values

from sklearn.model_selection import train_test_split
```

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

l=x_train.select_dtypes(include=['int64','float64']).columns

from sklearn.preprocessing import StandardScaler

sc=StandardScaler()

x_train[l]=sc.fit_transform(x_train[l])
x_test[l]=sc.transform(x_test[l])

x_train=x_train.values
x_test=x_test.values

x_train

x_test

from sklearn.linear_model import LogisticRegression

reg=LogisticRegression()

reg.fit(x_train,y_train)
y_pred=reg.predict(x_test)

from sklearn.metrics import
    accuracy_score,confusion_matrix,classification_report

print(confusion_matrix(y_pred,y_test))
print(classification_report(y_pred,y_test))
print("Accuracy of model is:",accuracy_score(y_pred,y_test))
```

Output Screenshot:

```
[[ 407  152]
 [ 163 1255]]
              precision    recall  f1-score   support

           0       0.71      0.73      0.72       559
           1       0.89      0.89      0.89      1418

    accuracy                           0.84      1977
   macro avg       0.80      0.81      0.80      1977
weighted avg       0.84      0.84      0.84      1977

Accuracy of model is: 0.8406676783004552
```

Result:
Hence, the logistic regression model is built, and the customer status is classified.

Ex. No: 3	**Decision Tree from scratch**
Date: 09-01-2023	

Aim:
To build a decision tree classifier from scratch.

Algorithm:

- Import the necessary modules and the csv file.
- Fill the null values present in the data and pre-process the data.
- Create a class and define the functions for building the tree, calculating information gain and gini index, print the tree, fit and predict the values.
- Split the data into train and test data using train test split.
- Create a object for Decision Tree Classifier and fit the model.
- Predict the values using the test data and print the accuracy score.

Program:

```
import numpy as np
import pandas as pd

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

df.head()

class Node():
    def
   __init__(self,feature_index=None,threshold=None,left=None,right=None,i
   nfo_gain=None,value=None):

        self.feature_index=feature_index
        self.threshold=threshold
        self.left=left
        self.right=right
        self.info_gain=info_gain

        self.value=value

class DecisionTreeClassifier():
    def__init__(self, min_samples_split=2, max_depth=2):
        self.root = None
        self.min_samples_split = min_samples_split
        self.max_depth = max_depth

    def build_tree(self, dataset, curr_depth=0):
        X, Y = dataset[:,:-1], dataset[:,-1]
```

```
     num_samples, num_features = np.shape(X)
     if num_samples>=self.min_samples_split and
curr_depth<=self.max_depth:
         best_split = self.get_best_split(dataset, num_samples,
num_features)
         if best_split["info_gain"]>0:
             left_subtree =
self.build_tree(best_split["dataset_left"], curr_depth+1)
             right_subtree =
self.build_tree(best_split["dataset_right"], curr_depth+1)
             return Node(best_split["feature_index"],
best_split["threshold"],left_subtree, right_subtree,
best_split["info_gain"])
     leaf_value = self.calculate_leaf_value(Y)
     return Node(value=leaf_value)

 def get_best_split(self, dataset, num_samples, num_features):
     best_split = {}
     max_info_gain = -float("inf")
     for feature_index in range(num_features):
         feature_values = dataset[:, feature_index]
         possible_thresholds = np.unique(feature_values)
         for threshold in possible_thresholds:
             dataset_left, dataset_right = self.split(dataset,
feature_index, threshold)
             if len(dataset_left)>0 and len(dataset_right)>0:
                 y, left_y, right_y = dataset[:, -1], dataset_left[:,
-1], dataset_right[:, -1]
                 curr_info_gain = self.information_gain(y, left_y,
right_y, "gini")
                 if curr_info_gain>max_info_gain:
                     best_split["feature_index"] = feature_index
                     best_split["threshold"] = threshold
                     best_split["dataset_left"] = dataset_left
                     best_split["dataset_right"] = dataset_right
                     best_split["info_gain"] = curr_info_gain
                     max_info_gain = curr_info_gain
     return best_split

 def split(self, dataset, feature_index, threshold):
     dataset_left = np.array([row for row in dataset if
row[feature_index]<=threshold])
     dataset_right = np.array([row for row in dataset if
row[feature_index]>threshold])
     return dataset_left, dataset_right

 def information_gain(self, parent, l_child, r_child,mode='gini'):
     weight_l = len(l_child) / len(parent)
     weight_r = len(r_child) / len(parent)
     if mode=="gini":
```

```
            gain = self.gini_index(parent) -
    (weight_l*self.gini_index(l_child) +
    weight_r*self.gini_index(r_child))
        else:
            gain = self.entropy(parent) - (weight_l*self.entropy(l_child)
    + weight_r*self.entropy(r_child))
        return gain

    def gini_index(self, y):
        class_labels = np.unique(y)
        gini = 0
        for cls in class_labels:
            p_cls = len(y[y == cls]) / len(y)
            gini += p_cls**2
        return 1 - gini

    def calculate_leaf_value(self, Y):
        Y = list(Y)
        return max(Y, key=Y.count)

    def print_tree(self, tree=None, indent=" "):
        if not tree:
            tree = self.root
        if tree.value is not None:
            print(tree.value)
        else:
            print("X_"+str(tree.feature_index), "<=", tree.threshold,
    "?", tree.info_gain)
            print("%sleft:" % (indent), end="")
            self.print_tree(tree.left, indent + indent)
            print("%sright:" % (indent), end="")
            self.print_tree(tree.right, indent + indent)

    def fit(self, X, Y):
        dataset = np.concatenate((X, Y), axis=1)
        self.root = self.build_tree(dataset)

    def predict(self, X):
        preditions = [self.make_prediction(x, self.root) for x in X]
        return preditions

    def make_prediction(self, x, tree):
        if tree.value!=None: return tree.value
        feature_val =  x[tree.feature_index]
        if feature_val<=tree.threshold:
            return self.make_prediction(x, tree.left)
        else:
            return self.make_prediction(x, tree.right)

x=df.iloc[:,:-1].values
```

```
y=df.iloc[:,-1].values.reshape(-1,1)
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=42,test_s
    ize=0.3)

tree=DecisionTreeClassifier(min_samples_split=3)

tree.fit(x_train,y_train)

tree.print_tree()

y_pred=tree.predict(x_test)
from sklearn.metrics import accuracy_score
accuracy_score(y_pred,y_test)
```

Output Screenshot:

```
X_0 <= 42 ? 0.15750816115920058
 left:X_1 <= 90000 ? 0.1841174682367247
  left:X_0 <= 36 ? 0.006683318626474102
    left:0
    right:0
  right:X_1 <= 117000 ? 0.035416666666666624
    left:1
    right:1
 right:X_1 <= 38000 ? 0.027824495449149056
  left:1
  right:X_0 <= 52 ? 0.0357568027210885
    left:1
    right:1
```

```
0.9333333333333333
```

Result:

Hence, the decision tree classifier is built from scratch and the purchased status is classified.

Ex. No: 4	Hyper parameter Tuning
Date: 23-01-2023	

Aim:
To build an hyperparameter tuning model using grid search and random search.

Algorithm:

- Import the necessary modules and the csv file.
- Fill the null values present in the data and pre-process the data.
- Split the data into train and test data using train test split.
- Import GridSearchCV and RandomSearchCV and declare the parameters for each model.
- Train the model with the predicted values and predict the values and print the accuracy score.

Program:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')

df=pd.read_csv('Telco-Customer-Churn.csv')

df.head()

df.info()

df.describe()

df.drop('customerID',inplace=True,axis=1)

df['TotalCharges']=df['TotalCharges'].apply(pd.to_numeric,errors='coerce'
    )

df.isnull().sum()

import seaborn as sns
sns.kdeplot(df['TotalCharges'])
print(df['TotalCharges'].mean(),df['TotalCharges'].median())

df['TotalCharges'].fillna(df['TotalCharges'].mean(),inplace=True)

a=df.select_dtypes(include='object')
```

```
for i in a:
    print(i,':',df[i].unique())

df['MultipleLines']=df['MultipleLines'].replace('No phone service','No')
df['OnlineBackup']=df['OnlineBackup'].replace('No internet service','No')
df['OnlineSecurity']=df['OnlineSecurity'].replace('No internet
    service','No')
df['DeviceProtection']=df['DeviceProtection'].replace('No internet
    service','No')
df['StreamingTV']=df['StreamingTV'].replace('No internet service','No')
df['TechSupport']=df['TechSupport'].replace('No internet service','No')
df['StreamingMovies']=df['StreamingMovies'].replace('No internet
    service','No')

from sklearn.preprocessing import LabelEncoder,StandardScaler
le=LabelEncoder()
a=a.columns
df[a]=df[a].apply(lambda x : le.fit_transform(x))

df.info()

x=df.iloc[:,:-1].values
y=df.iloc[:,-1].values

from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_s
    tate=42)

s=StandardScaler()
x_train=s.fit_transform(x_train)
x_test=s.transform(x_test)

## KNN

from sklearn.model_selection import GridSearchCV,RandomizedSearchCV
from sklearn.neighbors import KNeighborsClassifier

knn=KNeighborsClassifier()
knn.fit(x_train,y_train)

knn_p={'n_neighbors':np.arange(1,50)}

print("Grid Search:")
grid_knn=GridSearchCV(knn,knn_p,scoring='accuracy',cv=9,n_jobs=-1)
grid_knn.fit(x_train,y_train)
print(grid_knn.best_params_)
print(grid_knn.best_score_)

print("\nRandomized Search:")
rand_knn=RandomizedSearchCV(knn,knn_p,scoring='accuracy',cv=9,n_jobs=-1)
```

```
rand_knn.fit(x_train,y_train)
print(rand_knn.best_params_)
print(rand_knn.best_score_)


from sklearn.metrics import accuracy_score
y_pred=knn.predict(x_test)
y_pred_grid=grid_knn.predict(x_test)
y_pred_rand=rand_knn.predict(x_test)

print("Normal KNN:",accuracy_score(y_pred,y_test))
print("Grid Search KNN:",accuracy_score(y_pred_grid,y_test))
print("Random Search KNN:",accuracy_score(y_pred_rand,y_test))

## Logistic Regression

from sklearn.linear_model import LogisticRegression

lr=LogisticRegression()
lr.fit(x_train,y_train)

lr_params={'solver':['newton-cg','lbfgs','sag'],'C':np.logspace(-
    3,2.45,7),'penalty':['l1','l2']}

print("Grid Search:")
grid_lr=GridSearchCV(lr,lr_params,scoring='accuracy',cv=9,n_jobs=-1)
grid_lr.fit(x_train,y_train)
print(grid_lr.best_params_)
print(grid_lr.best_score_)

print("\nRandomized Search:")
rand_lr=RandomizedSearchCV(lr,lr_params,scoring='accuracy',cv=9,n_jobs=-
    1)
rand_lr.fit(x_train,y_train)
print(rand_lr.best_params_)
print(rand_lr.best_score_)

y_pred=lr.predict(x_test)
y_pred_grid=grid_lr.predict(x_test)
y_pred_rand=rand_lr.predict(x_test)

print("Normal Logistic Regression:",accuracy_score(y_pred,y_test))
print("Grid Search Logistic
    Regression:",accuracy_score(y_pred_grid,y_test))
print("Random Search Logistic
    Regression:",accuracy_score(y_pred_rand,y_test))

## Naive Bayes Classification

from sklearn.naive_bayes import GaussianNB
```

```
gnb=GaussianNB()
gnb.fit(x_train,y_train)

gnb_params={'var_smoothing':np.logspace(5,-7,79)}

print("Grid Search:")
grid_gnb=GridSearchCV(gnb,gnb_params,scoring='accuracy',cv=9,n_jobs=-1)
grid_gnb.fit(x_train,y_train)
print(grid_gnb.best_params_)
print(grid_gnb.best_score_)

print("\nRandomized Search:")
rand_gnb=RandomizedSearchCV(gnb,gnb_params,scoring='accuracy',cv=9,n_jobs
    =-1)
rand_gnb.fit(x_train,y_train)
print(rand_gnb.best_params_)
print(rand_gnb.best_score_)



y_pred=gnb.predict(x_test)
y_pred_grid=grid_gnb.predict(x_test)
y_pred_rand=rand_gnb.predict(x_test)

print("Normal Naive Bayes:",accuracy_score(y_pred,y_test))
print("Grid Search Naive Bayes:",accuracy_score(y_pred_grid,y_test))
print("Random Search Naive Bayes:",accuracy_score(y_pred_rand,y_test))

## Decision Trees

from sklearn.tree import DecisionTreeClassifier
dtc=DecisionTreeClassifier()
dtc.fit(x_train,y_train)

dtc_params={'criterion':['gini','entropy'],'max_depth':np.arange(1,20),'m
    in_samples_split':np.arange(1,10)
           ,'max_features':np.arange(1,7)}

print("Grid Search:")
grid_dtc=GridSearchCV(dtc,dtc_params,scoring='accuracy',cv=9,n_jobs=-1)
grid_dtc.fit(x_train,y_train)
print(grid_dtc.best_params_)
print(grid_dtc.best_score_)

print("Randomized Search:")
rand_dtc=RandomizedSearchCV(dtc,dtc_params,scoring='accuracy',cv=9,n_jobs
    =-1)
rand_dtc.fit(x_train,y_train)
print(grid_dtc.best_params_)
```

```
print(grid_dtc.best_score_)



y_pred=dtc.predict(x_test)
y_pred_grid=grid_dtc.predict(x_test)
y_pred_rand=rand_dtc.predict(x_test)

print("Normal Decision  Tree:",accuracy_score(y_pred,y_test))
print("Grid Search Decision Tree:",accuracy_score(y_pred_grid,y_test))
print("Random Search Decision Tree:",accuracy_score(y_pred_rand,y_test))

## Support Vector Machines

from sklearn.svm import SVC

svc=SVC()
svc.fit(x_train,y_train)

svc_params={'C':[0.1,1,10,100],'gamma':[0.1,0.001,0.01,0.001],'kernel':['
    poly','sigmoid','rbf']}

print("Grid Search:")
grid_svc=GridSearchCV(svc,svc_params,scoring='accuracy',cv=9,n_jobs=-1)
grid_svc.fit(x_train,y_train)
print(grid_svc.best_params_)
print(grid_svc.best_score_)

print("\nRandomized Search:")
rand_svc=RandomizedSearchCV(svc,svc_params,scoring='accuracy',cv=9,n_jobs
    =-1)
rand_svc.fit(x_train,y_train)
print(rand_svc.best_params_)
print(rand_svc.best_score_)




y_pred=svc.predict(x_test)
y_pred_grid=grid_svc.predict(x_test)
y_pred_rand=rand_svc.predict(x_test)

print("Normal Support Vector Machines:",accuracy_score(y_pred,y_test))
print("Grid Search Decision Tree:",accuracy_score(y_pred_grid,y_test))
print("Random Search Decision Tree:",accuracy_score(y_pred_rand,y_test))
```

Output Screenshot:

```
Grid Search:
{'criterion': 'entropy', 'max_depth': 7, 'max_features': 5, 'min_samples_split': 8}
0.7910726650268144
Randomized Search:
{'criterion': 'entropy', 'max_depth': 7, 'max_features': 5, 'min_samples_split': 8}
0.7910726650268144
```

```
Grid Search:
{'C': 10, 'gamma': 0.001, 'kernel': 'sigmoid'}
0.7989868796991925

Randomized Search:
{'kernel': 'sigmoid', 'gamma': 0.01, 'C': 1}
0.7987837515253147
```

```
Grid Search:
{'var_smoothing': 2.424462017082326}
0.7870152909551621

Randomized Search:
{'var_smoothing': 0.004124626382901348}
0.7519267522770372
```

```
Grid Search:
{'n_neighbors': 49}
0.7793079111751632

Randomized Search:
{'n_neighbors': 18}
0.7746485660188805
```

```
Grid Search:
{'C': 0.06556418494179789, 'penalty': 'l2', 'solver': 'newton-cg'}
0.8000006672093303

Randomized Search:
{'solver': 'lbfgs', 'penalty': 'l2', 'C': 0.06556418494179789}
0.8000006672093303
```

Result:

Hence, parameters for each classification model are selected using Grid Search and Random Search.

Ex. No: 5	Ensemble methods
Date: 30-01-2023	

Aim:
To build an ensemble of classification models.

Algorithm:

- Import the necessary modules and the csv file.
- Fill the null values present in the data and pre-process the data.
- Split the data into train and test data using train test split.
- Create 5 bags consisting of Logistic regression, K-Nearest Neighbours, Naïve bayes, Decision Tree and Support Vector Machines and predict the classes.
- Create a separate data frame consisting of the predicted values for each model and use max voting to predict the appropriate final class.

Program:

```
import numpy as np
import pandas as pd
import warnings
warnings.filterwarnings('ignore')

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

df.head()

df.info()

df.drop(['Churn Category','Churn Reason'],axis=1,inplace=True)

df.drop(['Customer ID','City'],axis=1,inplace=True)

df['Offer'].value_counts().index[0]

obj_col=df.select_dtypes(include='object')
for i in obj_col:
    df[i].fillna(method='ffill',inplace=True)

for i in obj_col:
    print(i,":",df[i].unique())

df_obj=pd.DataFrame()
from sklearn.preprocessing import LabelEncoder

le=LabelEncoder()
```

```
for i in obj_col:
    df[i]=le.fit_transform(obj_col[i])

df.drop(['Latitude','Longitude','Zip Code'],inplace=True,axis=1)
df['Avg Monthly GB Download']=df['Avg Monthly GB
   Download'].fillna(value=df['Avg Monthly GB Download'].mean())
df['Avg Monthly Long Distance Charges']=df['Avg Monthly Long Distance
   Charges'].fillna(value=df['Avg Monthly Long Distance Charges'].mean())

x=df.iloc[:,:-1]
y=df.iloc[:,-1]
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)

x_train.reset_index(inplace=True,drop=True)
y_train.reset_index(inplace=True,drop=True)

x_train.columns

x_train.info()

b1_x_train=x_train.sample(n=992)

b1_y_train=y_train[b1_x_train.index]

b2_x_train=x_train.sample(n=992)
b2_y_train=y_train[b2_x_train.index]

b3_x_train=x_train.sample(n=992)
b3_y_train=y_train[b3_x_train.index]

b4_x_train=x_train.sample(n=992)
b4_y_train=y_train[b4_x_train.index]

b5_x_train=x_train.sample(n=644)
b5_y_train=y_train[b5_x_train.index]


## Logistic Regression -Bag 1

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV

lr=LogisticRegression()
lr.fit(b1_x_train,b1_y_train)

lr_params={'solver':['newton-cg','lbfgs','sag'],'C':np.logspace(-
   10,2.45,10),'penalty':['l1','l2']}
```

```
print("Grid Search:")
grid_lr=GridSearchCV(lr,lr_params,scoring='accuracy',cv=9,n_jobs=-1)
grid_lr.fit(b1_x_train,b1_y_train)
print(grid_lr.best_score_)


## KNN-Bag 2

from sklearn.neighbors import KNeighborsClassifier

knn=KNeighborsClassifier()
knn.fit(b2_x_train,b2_y_train)

knn_p={'n_neighbors':np.arange(1,50)}


grid_knn=GridSearchCV(knn,knn_p,scoring='accuracy',cv=9,n_jobs=-1)
grid_knn.fit(b2_x_train,b2_y_train)

print(grid_knn.best_score_)


## Naive Bayes-Bag 3

from sklearn.naive_bayes import GaussianNB

gnb=GaussianNB()
gnb.fit(b3_x_train,b3_y_train)

gnb_params={'var_smoothing':np.logspace(5,-7,79)}


grid_gnb=GridSearchCV(gnb,gnb_params,scoring='accuracy',cv=9,n_jobs=-1)
grid_gnb.fit(b3_x_train,b3_y_train)

print(grid_gnb.best_score_)


## Decision Tree-Bag 4

from sklearn.tree import DecisionTreeClassifier
dtc=DecisionTreeClassifier()
dtc.fit(b4_x_train,b4_y_train)

dtc_params={'criterion':['gini','entropy'],'max_depth':np.arange(1,20),'m
    in_samples_split':np.arange(1,10)
            ,'max_features':np.arange(1,7)}


grid_dtc=GridSearchCV(dtc,dtc_params,scoring='accuracy',cv=9,n_jobs=-1)
```

```
grid_dtc.fit(b4_x_train,b4_y_train)

print(grid_dtc.best_score_)


## Support Vector Machine-Bag 5

from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
svc=SVC()
svc.fit(b5_x_train,b5_y_train)

print(accuracy_score(svc.predict(x_test),y_test))

print()

predicted_data=pd.DataFrame()
predicted_data['Logistic Regression']=grid_lr.predict(x_test)
predicted_data['KNN']=grid_knn.predict(x_test)
predicted_data['Naive Bayes']=grid_gnb.predict(x_test)
predicted_data['Decision Tree']=grid_dtc.predict(x_test)
predicted_data['Support Vector Machines']=svc.predict(x_test)

predicted_data

for i in predicted_data:
    print(i)

df1=predicted_data.iloc[0]

l=[]
for i in range(0,len(predicted_data)):
    c=d=0
    pred=0
    for i in predicted_data.iloc[i]:
        if i==0:
            c+=1
        elif i==1:
            d+=1
        if c>d:
            pred=0
        elif d>c:
            pred=1
    l.append(pred)

predicted_data
predicted_data['Final']=l
from sklearn.metrics import accuracy_score
print(accuracy_score(predicted_data['Final'],y_test))
```

Output Screenshot:

	Logistic Regression	KNN	Naive Bayes	Decision Tree	Support Vector Machines
0	1	1	1	1	1
1	1	1	0	1	1
2	0	0	0	0	0
3	0	0	0	0	0
4	1	1	1	1	1
...	...	...	...	...	...
1972	1	1	1	0	1
1973	1	1	0	1	0
1974	1	1	0	1	1
1975	1	1	1	1	1
1976	0	0	0	0	0

```
0.8300455235204856
```

Result:
Hence, using ensemble methods of 5 classification models, churn is classified.

Ex. No: 6	K Means Clustering
Date: 06-02-2023	

Aim:
To build a clustering model using K-Means algorithm.

Algorithm:

- Import the necessary modules and the csv file.
- Fill the null values present in the data and pre-process the data.
- Find the optimal number of clusters using WCSS graph.
- Retrain the model with the optimal number of clusters and predict the labels of each data point.
- Visualize the clusters in 3-dimensional using mpl_toolkits and print the silhouette score.

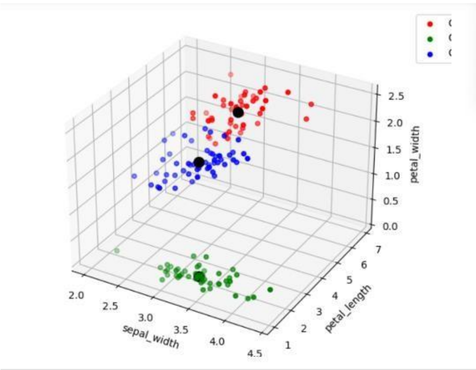
Program:

```
import numpy as np
import pandas as pd
import seaborn as sns
import warnings
import matplotlib.pyplot as plt
warnings.filterwarnings('ignore')
### IRIS dataset
df=sns.load_dataset('iris')
df.head()
sns.pairplot(df)
dataset=df.iloc[:,[1,2,3]].values
from sklearn.cluster import KMeans
wcss=[]
for i in range(1,15):
    km=KMeans(n_clusters=i,random_state=42)
    km.fit(dataset)
    wcss.append(km.inertia_)
plt.plot(range(1,15),wcss)
km=KMeans(n_clusters=3,random_state=42)
y=km.fit_predict(dataset)
y
%matplotlib widget
from mpl_toolkits.mplot3d import Axes3D

fig=plt.figure()
#ax=fig.add_subplot(111,projection='3d')
ax=Axes3D(fig)
ax.scatter(dataset[y==0,0],dataset[y==0,1],dataset[y==0,2],c='red',label=
    'Cluster-1')
```

```
ax.scatter(dataset[y==1,0],dataset[y==1,1],dataset[y==1,2],c='green',labe
    l='Cluster-2')
ax.scatter(dataset[y==2,0],dataset[y==2,1],dataset[y==2,2],c='blue',label
    ='Cluster-3')

ax.scatter(km.cluster_centers_[:,0],km.cluster_centers_[:,1],km.cluster_c
    enters_[:,2],c='black',s=100,depthshade=False)

ax.set_xlabel('sepal_width')
ax.set_ylabel('petal_length')
ax.set_zlabel('petal_width')
plt.legend(bbox_to_anchor=(1.05, 1), loc=2)

from sklearn.metrics import silhouette_score
score=silhouette_score(dataset,y)
print(score)
### data.csv
df1=pd.read_csv('data.csv')
df1.head()
dataset1=df1.iloc[:,[2,3,4]].values
from sklearn.cluster import KMeans
wcss=[]
for i in range(1,15):
    km=KMeans(n_clusters=i,random_state=42)
    km.fit(dataset1)
    wcss.append(km.inertia_)
%matplotlib inline
plt.plot(range(1,15),wcss)
km1=KMeans(n_clusters=6,random_state=42)
y1=km1.fit_predict(dataset1)
y1
%matplotlib widget
from mpl_toolkits.mplot3d import Axes3D

fig1=plt.figure()
#ax=fig.add_subplot(111,projection='3d')
ax=Axes3D(fig1)
ax.scatter(dataset1[y1==0,0],dataset1[y1==0,1],dataset1[y1==0,2],c='red',
    label='Cluster-1')
ax.scatter(dataset1[y1==1,0],dataset1[y1==1,1],dataset1[y1==1,2],c='green
    ',label='Cluster-2')
ax.scatter(dataset1[y1==2,0],dataset1[y1==2,1],dataset1[y1==2,2],c='blue'
    ,label='Cluster-3')
ax.scatter(dataset1[y1==3,0],dataset1[y1==3,1],dataset1[y1==3,2],c='yello
    w',label='Cluster-4')
ax.scatter(dataset1[y1==4,0],dataset1[y1==4,1],dataset1[y1==4,2],c='magen
    ta',label='Cluster-5')
ax.scatter(dataset1[y1==5,0],dataset1[y1==5,1],dataset1[y1==5,2],c='orang
    e',label='Cluster-6')
```

```
ax.scatter(km1.cluster_centers_[:,0],km1.cluster_centers_[:,1],km1.cluste
    r_centers_[:,2],c='black',s=100,depthshade=False)

ax.set_xlabel('Age')
ax.set_ylabel('Annual Income')
ax.set_zlabel('Spending Score')
plt.legend(bbox_to_anchor=(1.05, 1), loc=2)

from sklearn.metrics import silhouette_score
score=silhouette_score(dataset1,y1)
print(score)
```

Output Screenshot:




```
0.5932903117719593
```

Result:
Hence, cluster classification model using K-Means is built and the labels of the data points are predicted.

Ex. No: 7	**Principal Component Analysis**
Date: 20-02-2023	

Aim:
To build a principal component analysis from scratch.

Algorithm:

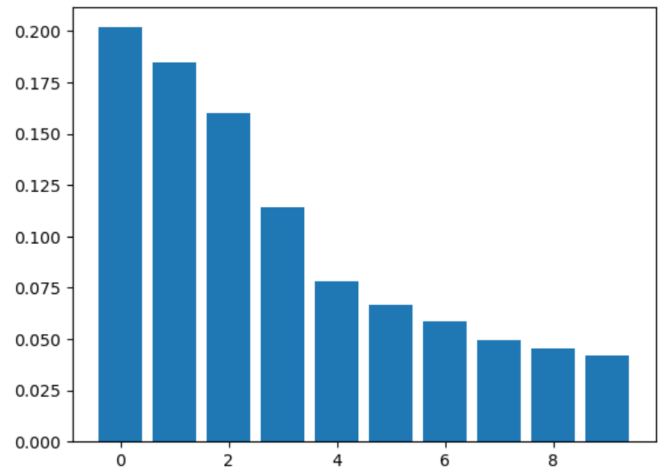
- Import the necessary modules and the necessary datasets.
- Fill the null values present in the data and pre-process the data.
- Normalize the data and compute the covariance matrix.
- Calculate the eigen values and eigen vectors of the covariance matrix.
- Reproject the data by scalar dot operation between the highest eigen vector and the normalized data i.e., mean is subtracted from all the data points.
- Calculate the variance and plot it using scree plot and select the number of features.

Program:

```
import numpy as np

from sklearn.datasets import load_digits

digit=load_digits()

df=pd.DataFrame(digit.data)
df.columns=digit.feature_names
label=digit.target
x=digit.data

df

x=df

mean_l=np.mean(x.T,axis=1)

mean_l

x_center=x-mean_l
x=x_center


q=[]
for i in df:
    w=np.array(df[i])
    q.append(w)
```

```
cov=np.cov(q,bias=False)

cov

cov.shape

eigen=np.linalg.eig(cov)

eigen_values=eigen[0]
eigen_vectors=eigen[1]

print("Eigen values:",eigen_values)
print("Eigen vectors:",eigen_vectors)

import matplotlib.pyplot as plt

plt.bar(np.arange(64),eigen_values/np.sum(eigen_values))
plt.show()

selected_eigen_values=eigen_values[:10]
selected_eigen_vectors=eigen_vectors[:,:10]

plt.bar(np.arange(10),selected_eigen_values/np.sum(selected_eigen_values)
    )
plt.show()

main_x=x @ selected_eigen_vectors

main_x.shape

from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(main_x,label,stratify=labe
    l,random_state=42)

y_train.shape

from sklearn.neural_network import MLPClassifier

mlp=MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(20,
    10), random_state=1)
mlp.fit(x_train,y_train)

mlp.score(x_test,y_test)
```

Output Screenshot:




```
[67]: mlp.score(x_test,y_test)
[67]: 0.92
```

Result:

Hence, principal component analysis is built from scratch and the feature are selected.

Ex. No: 8	**Recommendation System**
Date:06-03-2023	

Aim:
To build a content based and collaborative filter-based recommendation system.

Algorithm:
- Import the necessary modules and the csv file.
- Fill the null values present in the data and pre-process the data.

Content based recommendation system:
- Create a tfidf sparse matrix using TfidfVectorizer from sklearn library.
- Create a cosine similarity matrix using linear kernel function and store the keywords in a data frame.
- Create a user defined recommendation function to check if the keyword is present and sort the cosine similarity values and return the top recommended the top n items.

Collaborative filter-based recommendation system:
- Create a csr matrix using the scipy module and use the nearest neighbours algorithm to fit the csr matrix.
- Find similar users using the nearest neighbour model trained with the csr matrix.
- Predict the rating a user would give to the item and recommend the top n-items.

Program:

```
import numpy as np
import pandas as pd
## Content based recommendation system
df=pd.read_csv('result_final.csv')
df.head()
df.info()
df.drop(columns=['Unnamed: 0.1','link','date'],axis=1,inplace=True)
def string(x):
    x=x[1:-1]
    s=''
    for i in x.split(','):
       s+=i+','
    return s
df.fillna(method='ffill',inplace=True)
df['keywords']=df['keywords'].agg(string)
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf=TfidfVectorizer(stop_words='english')

tfidf_mat=tfidf.fit_transform(df['keywords'])
tfidf_mat
```

```
from sklearn.metrics.pairwise import linear_kernel

cosine_sim=linear_kernel(tfidf_mat,tfidf_mat)
cosine_sim
df1=pd.DataFrame(pd.Series(df['keywords'],index=df.index))
df1
import re
def recomendation(x):

    ind=df1[df1['keywords'].str.contains(x,flags=re.IGNORECASE,regex=True)
    ].index[0]
    sim_score=list(enumerate(cosine_sim[ind]))
    sim_score=sorted(sim_score,key=lambda a:a[1],reverse=True)
    sim_score=sim_score[1:8]
    final_ind=[i[0] for i in sim_score]
    return final_ind

r=input('Enter keyword for news title:')
ind=recomendation(r)

print('Recommended news are:')
for i in ind:
    print('>>',df['title'][i],'\n \t>>>',df['title_summary'][i])
```

<u>Collaborative filtering recommendation system</u>

```
df2=pd.read_csv('Netflix_Dataset_Movie.csv')
df3=pd.read_csv('Netflix_Dataset_Rating.csv')
df3.head()
users=df3['User_ID'].unique().shape[0]
movie=df3['Movie_ID'].unique().shape[0]
df3.sort_values('User_ID',inplace=True)
user_uni
df3
from sklearn.preprocessing import LabelEncoder

le=LabelEncoder()
df3['User_ID']=le.fit_transform(df3['User_ID'])
df3.sort_values('Movie_ID',inplace=True)
df3['Movie_id']=le.fit_transform(df3['Movie_ID'])
a=np.zeros((users,movie))
for i in df3.itertuples():
    a[i[1]-1,i[4]-1]=i[2]
a
from scipy.sparse import csr_matrix
sparse_matrix=csr_matrix(a)
from sklearn.neighbors import NearestNeighbors

nn=NearestNeighbors(n_neighbors=5,algorithm='brute',metric='cosine',n_job
   s=-1)
```

```
nn.fit(sparse_matrix)
df3
data=df3.sort_values('User_ID',ascending=True)
colab_filter=data[data['User_ID']==69].Movie_ID
colab_filter=colab_filter.tolist()
len(colab_filter)
item_id=[]
for i in colab_filter:
    dist,ind=nn.kneighbors(sparse_matrix[i],n_neighbors=5)
    ind=ind.flatten()
    ind=ind[:1]
    item_id.extend(ind)
item_id=item_id[:10]
item_id
df2
print("Movies recommended by User 69:")
for i in item_id:
    print(">>>>",df2['Name'][i-1])
```

Output Screenshot:

```
Recommended news are:
>> Motor racing: No 'Andretti Curse' in August, says Marco
        >>> Motor racing: No 'Andretti Curse' in August, says Marco (Reuters) - After more than five decades of Brickyard frustration the "Andretti Curse" has

Aug 17, 2020; Indianapolis, Indiana, USA; Indy Car Series driver Marco Andretti poses for a family photo shoot with wife Marta and his sister and his father M

American motor racing royalty, the Andrettis have ruled over open wheel racing in the United States with a string of victories that have connected generations

But for all their success at circuits around the world, Mario's Indy 500 victory in 1969 stands alone.

Over the years what was first shrugged off as bad luck, became cruel misfortune then later a full blown jinx that now ranks right up there with the Boston Red

For those looking for signs that whatever voodoo planted on the Andrettis will be lifted this year, there were hints on Sunday that something was different wh

The Andrettis have happily played along when it comes to the curse, neither embracing or dismissing it.

Marco routinely turns the tables saying his family is not cursed but blessed while at the same time conceding the results suggest darker forces at work.

A theory this year has the curse thrown out of whack by the COVID-19 pandemic which forced the Indy 500 to move from its traditional May 24th date to Aug. 23.

"I don't think it (the curse) exists in August so we are good," Andretti told Reuters with a laugh. "As a family we would talk that we are blessed not cursed

"Results wise, yeah, maybe there is something to it because man the three of us have been so dominate here.
```

```
Movies recommended by User 69:
>>>> Taking Lives
>>>> Scary Movie 2
>>>> Speed
>>>> Shrek 2
>>>> Jurassic Park III
>>>> Rudy
>>>> 2 Fast 2 Furious
>>>> Holiday in the Sun
>>>> Encino Man
>>>> Madeline
```

Result:

Hence, for the given keyword input the items are content based recommendations are recommended to the user and according to the given user collaborative filter-based recommendations are recommended.

Ex. No: 9	Neural Network
Date: 13-03-2023	

Aim:
To build a neural network using tensorflow and pytorch modules.

Algorithm:
- Import the necessary modules and the csv file.
- Fill the null values present in the data and pre-process the data.
- Standardize the data using Standard Scalar from skleran.preprocessing module.

Tensorflow neural network:
- Create the neural network by adding the layers and the activation functions of each layer.
- Compile the neural network and declare early stopping to avoid overtraining of the model.
- Fit the neural network and predict the values using validation data and print the classification report and confusion matrix.

Pytorch neural network:
- Change the datatype of the train, test data and to torch datatypes and concert too tensor dataset.
- Create the neural network by adding the hidden layers and declare the optimizer and loss functions and train the network.
- Predict the values and print the accuracy of the network.

Program:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
df=pd.read_csv('Bank_Personal_Loan_Modelling.csv')
df.head()
df.info()
df.isnull().sum()
df.duplicated().sum()
sns.countplot(x='Personal Loan',data=df)
df.describe()
df['Experience']=abs(df['Experience'])
df['Annual_CCAvg']=df['CCAvg']*12
df.drop(['ID','ZIP Code','CCAvg'],axis=1,inplace=True)
x=df.drop('Personal Loan',axis=1).values
y=df['Personal Loan'].values.reshape(-1,1)
from sklearn.model_selection import train_test_split
```

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_s
tate=42)
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x_train=sc.fit_transform(x_train)
x_test=sc.fit_transform(x_test)
from sklearn.decomposition import PCA

pca=PCA()
x_train=pca.fit_transform(x_train)
x_test=pca.fit_transform(x_test)
np.cumsum(pca.explained_variance_ratio_)
from sklearn.decomposition import PCA

pca=PCA(n_components=10)
x_train=pca.fit_transform(x_train)
x_test=pca.fit_transform(x_test)
x_train.shape
## Using keras classification
import tensorflow.keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
model=Sequential()

model.add(Dense(9,activation='relu'))
model.add(Dense(5,activation='relu'))
model.add(Dense(1,activation='sigmoid'))

model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['
accuracy'])
model.fit(x_train,y_train,epochs=10,validation_data=(x_test,y_test),verbo
se=1)
model_loss=pd.DataFrame(model.history.history)
model_loss.plot()
model=Sequential()

model.add(Dense(9,activation='relu'))
model.add(Dense(5,activation='relu'))
model.add(Dense(1,activation='sigmoid'))

model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['
accuracy'])
from tensorflow.keras.callbacks import EarlyStopping
early_stop=EarlyStopping(monitor='val_loss',mode='min',verbose=1,patience
=25)
model.fit(x_train,y_train,epochs=100,validation_data=(x_test,y_test),verb
ose=1,callbacks=[early_stop])
model_loss=pd.DataFrame(model.history.history)
model_loss.plot()
pred=model.predict(x_test)
```

```
threshold=0.5

pred=np.where(pred>threshold,1,0)
from sklearn.metrics import classification_report,confusion_matrix

print(confusion_matrix(y_test,pred))
print('\n')
print(classification_report(y_test,pred))
## Using PyTorch classification
import torch
import torch.nn as nn
from torch.optim import SGD
from torch.utils.data import TensorDataset,DataLoader
x=df.drop('Personal Loan',axis=1).values
y=df['Personal Loan'].values.reshape(-1,1)
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_s
tate=42)
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x_train=sc.fit_transform(x_train)
x_test=sc.fit_transform(x_test)
x_train=torch.tensor(x_train).to(torch.float32)
x_test=torch.tensor(x_test).to(torch.float32)
y_train=torch.tensor(y_train).to(torch.float32)
y_test=torch.tensor(y_test).to(torch.float32)
dataset=TensorDataset(x_train,y_train)
data=DataLoader(dataset,batch_size=24,shuffle=True)
model=nn.Sequential(nn.Linear(11,8),nn.ReLU(),nn.Linear(8,4),nn.ReLU(),nn
.Linear(4,1),nn.Sigmoid())
model
fn_loss=nn.BCELoss()
optimr=SGD(model.parameters(),lr=0.001)
def train(model,epoch,data):
    train_loss=[0]*epoch
    train_acc=[0]*epoch

    for i in range(epoch):
        for x_batch,y_batch in data:

            pred=model(x_batch)
            loss=fn_loss(pred,y_batch)

            loss.backward()

            optimizer.step()
            optimizer.zero_grad()

            train_loss[i]+=loss.item()*x_batch.size(0)
```

```
            crt=(torch.where(model(x_batch)>=0.5,1,0)==y_batch).sum()
            train_acc[i]+=crt

        train_loss[i] = train_loss[i]/len(data.dataset)
        train_acc[i]=train_acc[i]/len(data.dataset)


print(f'Epoch:{i+1}|Loss:{train_loss[i]}|Accuracy:{train_acc[i]}')
    return train_loss,train_acc
train_loss,train_acc=train(model,20,data)
testdata=TensorDataset(x_test,y_test)
tdata=DataLoader(testdata,batch_size=1500)
with torch.no_grad():
    for x_whole,y_whole in tdata:
        pred=model(x_whole)
        crt_cnt=(torch.where(pred>=0.5,1,0)==y_whole).sum()
    print(f'Accuracy:{crt_cnt/len(tdata.dataset)}')
```

Output Screenshot:

```
Epoch 1/100
110/110 [==============================] - 0s 2ms/step - loss: 0.0000e+00 - accuracy: 0.8706 - val_loss: 0.0000e+00 - val_accuracy: 0.8953
Epoch 2/100
110/110 [==============================] - 0s 1ms/step - loss: 0.0000e+00 - accuracy: 0.9077 - val_loss: 0.0000e+00 - val_accuracy: 0.8953
Epoch 3/100
110/110 [==============================] - 0s 1ms/step - loss: 0.0000e+00 - accuracy: 0.9077 - val_loss: 0.0000e+00 - val_accuracy: 0.8953
Epoch 4/100
110/110 [==============================] - 0s 1ms/step - loss: 0.0000e+00 - accuracy: 0.9077 - val_loss: 0.0000e+00 - val_accuracy: 0.8953
Epoch 5/100
110/110 [==============================] - 0s 1ms/step - loss: 0.0000e+00 - accuracy: 0.9077 - val_loss: 0.0000e+00 - val_accuracy: 0.8953
Epoch 6/100
110/110 [==============================] - 0s 1ms/step - loss: 0.0000e+00 - accuracy: 0.9077 - val_loss: 0.0000e+00 - val_accuracy: 0.8953
Epoch 7/100
110/110 [==============================] - 0s 1ms/step - loss: 0.0000e+00 - accuracy: 0.9077 - val_loss: 0.0000e+00 - val_accuracy: 0.8953
Epoch 8/100
110/110 [==============================] - 0s 1ms/step - loss: 0.0000e+00 - accuracy: 0.9077 - val_loss: 0.0000e+00 - val_accuracy: 0.8953
Epoch 9/100
110/110 [==============================] - 0s 1ms/step - loss: 0.0000e+00 - accuracy: 0.9077 - val_loss: 0.0000e+00 - val_accuracy: 0.8953
Epoch 10/100
110/110 [==============================] - 0s 1ms/step - loss: 0.0000e+00 - accuracy: 0.9077 - val_loss: 0.0000e+00 - val_accuracy: 0.8953
Epoch 11/100
110/110 [==============================] - 0s 1ms/step - loss: 0.0000e+00 - accuracy: 0.9077 - val_loss: 0.0000e+00 - val_accuracy: 0.8953
Epoch 12/100
110/110 [==============================] - 0s 1ms/step - loss: 0.0000e+00 - accuracy: 0.9077 - val_loss: 0.0000e+00 - val_accuracy: 0.8953
Epoch 13/100
...
110/110 [==============================] - 0s 1ms/step - loss: 0.0000e+00 - accuracy: 0.9077 - val_loss: 0.0000e+00 - val_accuracy: 0.8953
Epoch 26/100
```

```
Epoch:1|Loss:0.6308542529514858|Accuracy:0.907714307308197
Epoch:2|Loss:0.6308542481831142|Accuracy:0.907714307308197
Epoch:3|Loss:0.6308542514528547|Accuracy:0.907714307308197
Epoch:4|Loss:0.6308542485918317|Accuracy:0.907714307308197
Epoch:5|Loss:0.6308542495455061|Accuracy:0.907714307308197
Epoch:6|Loss:0.6308542491367886|Accuracy:0.907714307308197
Epoch:7|Loss:0.6308542485918317|Accuracy:0.907714307308197
Epoch:8|Loss:0.6308542485918317|Accuracy:0.907714307308197
Epoch:9|Loss:0.630854254586356|Accuracy:0.907714307308197
Epoch:10|Loss:0.6308542522021703|Accuracy:0.907714307308197
Epoch:11|Loss:0.6308542476381575|Accuracy:0.907714307308197
Epoch:12|Loss:0.6308542504991804|Accuracy:0.907714307308197
Epoch:13|Loss:0.6308542451858521|Accuracy:0.907714307308197
Epoch:14|Loss:0.6308542516572135|Accuracy:0.907714307308197
Epoch:15|Loss:0.630854248387473|Accuracy:0.907714307308197
Epoch:16|Loss:0.6308542541776384|Accuracy:0.907714307308197
Epoch:17|Loss:0.6308542503629412|Accuracy:0.907714307308197
Epoch:18|Loss:0.6308542497498648|Accuracy:0.907714307308197
Epoch:19|Loss:0.6308542477743966|Accuracy:0.907714307308197
Epoch:20|Loss:0.6308542463438851|Accuracy:0.907714307308197
```

	precision	recall	f1-score	support
0	0.90	1.00	0.94	1343
1	0.00	0.00	0.00	157
accuracy			0.90	1500
macro avg	0.45	0.50	0.47	1500
eighted avg	0.80	0.90	0.85	1500

```
Accuracy:0.8953333497047424
```

Result:
Hence, neural networks are built using tensorflow and pytorch and the data is classsified.

Ex. No: 10	Convolutional Neural Network
Date: 27-03-2023	

Aim:
To build a convolutional neural network using tensorflow for image classification.

Algorithm:

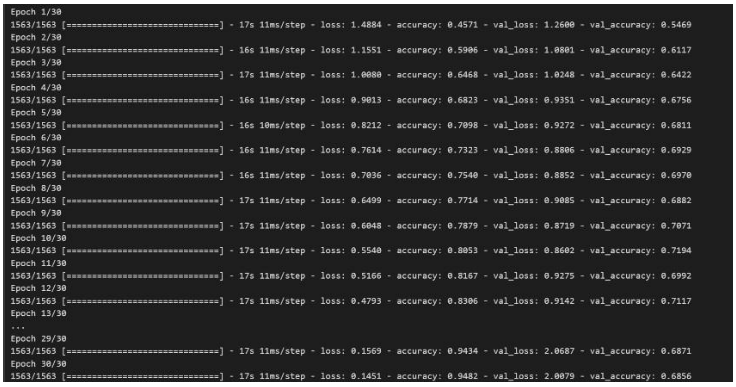
- Import the necessary modules and the database.
- Label the images of each class and spilt the data into train and test data.
- Create the convolutional neural network by adding the layers of Conv2D and MaxPooling2D and the activation functions of each layer.
- Create an early stop to avoid overtraining of the model and compile the model and fit the model with training data.
- Predict the accuracy of the model using the test data and print the accuracy.

Program:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras import datasets,layers,models
(train_images,train_labels),(test_images,test_labels)=datasets.cifar10.lo
ad_data()

train_images,test_images=train_images/255.0,test_images/255.0
class_names=['airplane','automobile','bird','cat','deer','dog','frog','ho
rse','ship','truck']

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

for i in range(10):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i])
    plt.xlabel(class_names[train_labels[i][0]])
plt.show()
from keras.layers.core import Dense,Activation,Dropout,Flatten
from tensorflow.keras.models import Sequential
from keras.layers.convolutional import Conv2D,MaxPooling2D
model=Sequential()

model.add(Conv2D(32,(3,3),activation='relu',input_shape=(32,32,3)))
```

```
model.add(MaxPooling2D(2,2))
model.add(Conv2D(64,(3,3),activation='relu'))
model.add(MaxPooling2D(2,2))
model.add(Conv2D(64,(3,3),activation='relu'))
model.add(layers.Flatten())
model.add(Dense(128,activation='relu'))
model.add(Dense(10))
model.summary()
from keras.callbacks import EarlyStopping
early_stop=EarlyStopping(monitor='val_loss',mode='min',verbose=1,patience
=25)
model.compile(optimizer='adam',loss=tf.keras.losses.SparseCategoricalCros
sentropy(from_logits=True),metrics=['accuracy'])
history=model.fit(train_images,train_labels,epochs=30,validation_data=(te
st_images,test_labels),callbacks=[early_stop])
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.xlabel(['Epoch'])
plt.ylabel(['Accuracy'])
plt.ylim([.5,1])
plt.legend(loc='lower right')
```

Output Screenshot:



```
Epoch 1/30
1563/1563 [==============================] - 17s 11ms/step - loss: 1.4884 - accuracy: 0.4571 - val_loss: 1.2600 - val_accuracy: 0.5469
Epoch 2/30
1563/1563 [==============================] - 16s 11ms/step - loss: 1.1551 - accuracy: 0.5906 - val_loss: 1.0801 - val_accuracy: 0.6117
Epoch 3/30
1563/1563 [==============================] - 17s 11ms/step - loss: 1.0080 - accuracy: 0.6468 - val_loss: 1.0248 - val_accuracy: 0.6422
Epoch 4/30
1563/1563 [==============================] - 16s 11ms/step - loss: 0.9013 - accuracy: 0.6823 - val_loss: 0.9351 - val_accuracy: 0.6756
Epoch 5/30
1563/1563 [==============================] - 16s 10ms/step - loss: 0.8212 - accuracy: 0.7098 - val_loss: 0.9272 - val_accuracy: 0.6811
Epoch 6/30
1563/1563 [==============================] - 16s 11ms/step - loss: 0.7614 - accuracy: 0.7323 - val_loss: 0.8806 - val_accuracy: 0.6929
Epoch 7/30
1563/1563 [==============================] - 16s 11ms/step - loss: 0.7036 - accuracy: 0.7540 - val_loss: 0.8852 - val_accuracy: 0.6970
Epoch 8/30
1563/1563 [==============================] - 17s 11ms/step - loss: 0.6499 - accuracy: 0.7714 - val_loss: 0.9085 - val_accuracy: 0.6882
Epoch 9/30
1563/1563 [==============================] - 17s 11ms/step - loss: 0.6048 - accuracy: 0.7879 - val_loss: 0.8719 - val_accuracy: 0.7071
Epoch 10/30
1563/1563 [==============================] - 17s 11ms/step - loss: 0.5540 - accuracy: 0.8053 - val_loss: 0.8602 - val_accuracy: 0.7194
Epoch 11/30
1563/1563 [==============================] - 17s 11ms/step - loss: 0.5166 - accuracy: 0.8167 - val_loss: 0.9275 - val_accuracy: 0.6992
Epoch 12/30
1563/1563 [==============================] - 17s 11ms/step - loss: 0.4793 - accuracy: 0.8306 - val_loss: 0.9142 - val_accuracy: 0.7117
Epoch 13/30
...
Epoch 29/30
1563/1563 [==============================] - 17s 11ms/step - loss: 0.1569 - accuracy: 0.9434 - val_loss: 2.0687 - val_accuracy: 0.6871
Epoch 30/30
1563/1563 [==============================] - 17s 11ms/step - loss: 0.1451 - accuracy: 0.9482 - val_loss: 2.0079 - val_accuracy: 0.6856
```

Result:

Hence, the convolutional neural network is built, and the images are classified.

Ex. No: 11	**MLOps-Using Flask**
Date: 23-03-2023	

Aim:
To build a flask model that implements a ML model (IPL MATCH PREDICTION BASED ON TOSS).

Algorithm:

- Import the necessary modules and the necessary datasets.
- Fill the null values present in the data and pre-process the data.
- Create a prediction model that receives user and input and predicts winner, then rename the file as model.py.
- Connect mysql using pymysql library to verify the login credentials.
- Create html page to display the output, receive input and a login page and render all the html pages using appropriate functions.
- Pass the input values to python from html to check the login credentials using sql.
- Pass the input for prediction and print the predictions in the html page.

Program:

```
import pickle as pkl
from flask import Flask, render_template, request, url_for, redirect
import numpy as np
import mysql.connector as sql

app = Flask(__name__)

@app.route('/')
def home():
    return render_template("login.html")

@app.route("/login",methods=["GET","POST"])
def login():
    con=sql.connect(user='root',
password='Karthic@2206*',host='localhost',database='login')
    cur=con.cursor()

    username = request.form['username']
    password = request.form['password']
    cur.execute('select * from details')
    t=cur.fetchone()
    # Check if the username exists and the password matches
    if username ==t[0] and password==t[1]:
        return render_template("index.html")

    else:
        return render_template("login.html")




@app.route("/predict", methods=['get','post'])

def predict():
```

```
    team1 = str(request.args.get('team1'))
    team2 = str(request.args.get('team2'))

    toss_win = int(request.args.get('toss_winner'))
    choose = int(request.args.get('toss_decision'))

    with open('inv_vocab.pkl', 'rb') as f:
        inv_vocab = pkl.load(f)

    with open('model.pkl', 'rb') as f:
        model = pkl.load(f)

    cteam1 = inv_vocab[team1]
    cteam2 = inv_vocab[team2]

    if cteam1 == cteam2:
        return redirect(url_for('index'))

    lst = np.array([cteam1, cteam2, choose,toss_win],dtype='int32').reshape(1,-1)

    prediction = model.predict(lst)

    if prediction == 0:
        return render_template('success.html', data=team1)


    else:
        return render_template('success.html', data=team2)

if __name__=='__main__':
    app.run(host='localhost',port=5000,debug=True)


if __name__ == "__main__":
      app.run(debug=True)
```

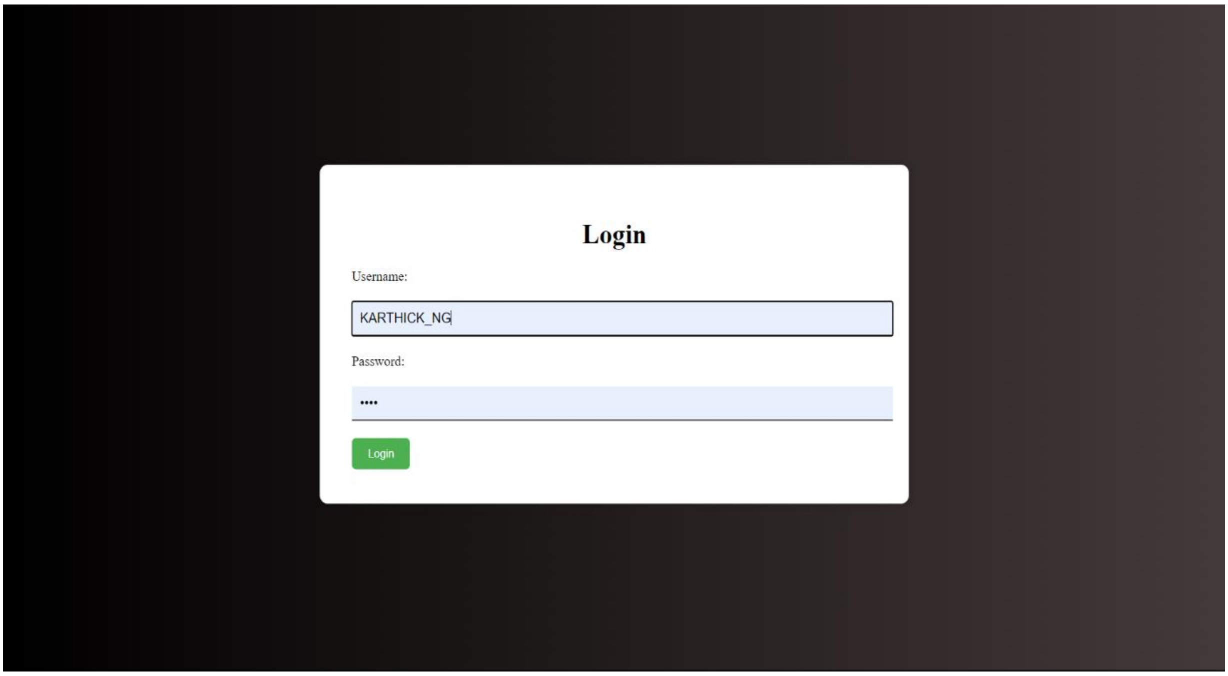
PREDICTION MODEL:

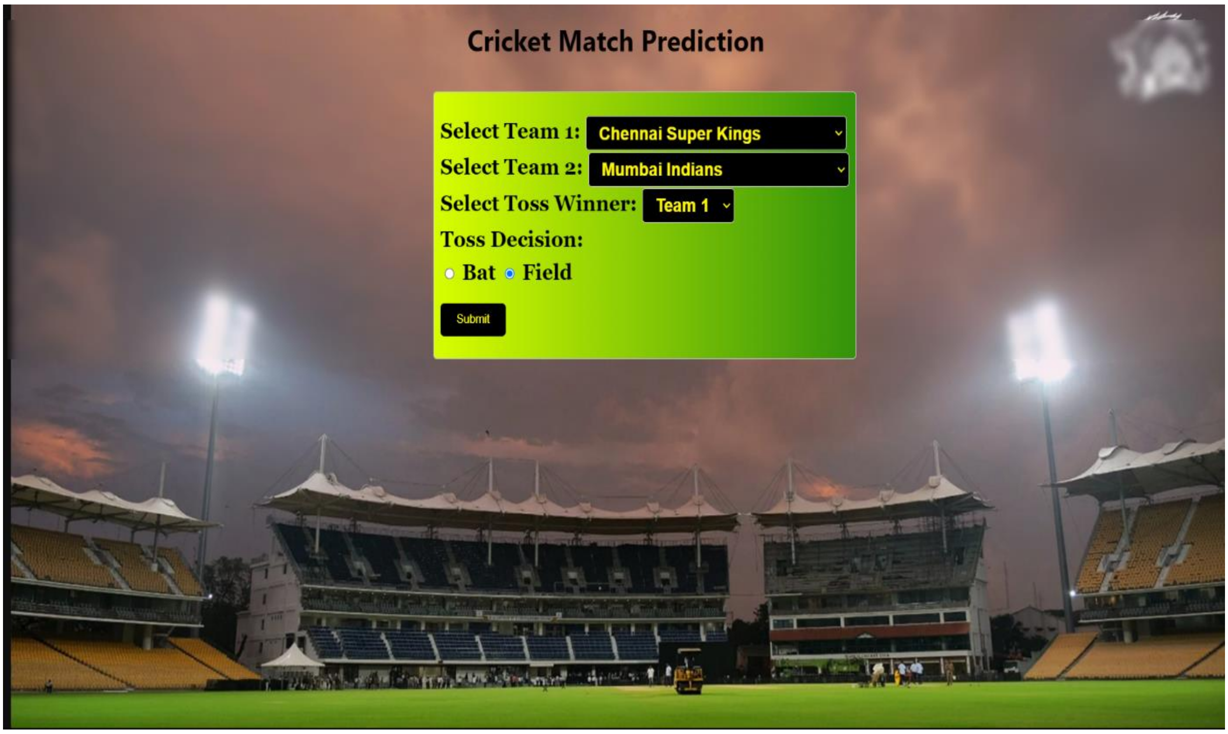
```
import pandas as pd
import numpy as np
import pickle as pkl
df = pd.read_csv('matches.csv')
new_df = df[['team1', 'team2', 'winner', 'toss_decision', 'toss_winner']]



new_df.dropna(inplace=True)

X = new_df[['team1', 'team2', 'toss_decision', 'toss_winner']]
y = new_df[['winner']]

#AFTER PREPROCESSING

all_teams = {}
cnt = 0
for i in range(len(df)):
    if df.loc[i]['team1'] not in all_teams:
        all_teams[df.loc[i]['team1']] = cnt
        cnt += 1

    if df.loc[i]['team2'] not in all_teams:
        all_teams[df.loc[i]['team2']] = cnt
        cnt += 1



from sklearn.preprocessing import LabelEncoder
teams = LabelEncoder()
teams.fit(list(all_teams.keys()))

encoded_teams = teams.transform(list(all_teams.keys()))

with open('vocab.pkl', 'wb') as f:
    pkl.dump(encoded_teams, f)
with open('inv_vocab.pkl', 'wb') as f:
    pkl.dump(all_teams, f)


X = np.array(X)
y = np.array(y)
y = np.squeeze(y)

X[:, 0] = teams.transform(X[:, 0])
X[:, 1] = teams.transform(X[:, 1])
X[:, 3] = teams.transform(X[:, 3])

y[:] = teams.transform(y[:])

fb = {'field' : 0, 'bat' : 1}
for i in range(len(X)):
    X[i][2] = fb[X[i][2]]

y = y.astype('int')
```

```python
X = np.array(X, dtype='int32')
y = np.array(y, dtype='int32')
y_backup = y.copy()

y = y_backup.copy()

ones, zeros = 0,0
for i in range(len(X)):
    if y[i] == X[i][0] :
        if zeros <= 375:
            y[i] = 0
            zeros += 1
        else:
            y[i] = 1
            ones += 1
            t = X[i][0]
            X[i][0] = X[i][1]
            X[i][1] = t



    if y[i] == X[i][1] :
        if ones <= 375:
            y[i] = 1
            ones += 1
        else:
            y[i] = 0
            zeros += 1
            t = X[i][0]
            X[i][0] = X[i][1]
            X[i][1] = t


from sklearn.model_selection import train_test_split
X_train, X_test, y_train,y_test = train_test_split(X, y, test_size=0.05)


from sklearn.svm import SVC
model1 = SVC().fit(X_train, y_train)
model1.score(X_test, y_test)

from sklearn.tree import DecisionTreeClassifier
model2 = DecisionTreeClassifier().fit(X_train, y_train)
model2.score(X_test, y_test)

from sklearn.ensemble import RandomForestClassifier
model3 = RandomForestClassifier(n_estimators=250).fit(X, y)
model3.score(X_test, y_test)

test = np.array([2,4, 1, 4]).reshape(1,-1)
model1.predict(test)
model2.predict(test)
model3.predict(test)





import pickle as pkl

with open('model.pkl', 'wb') as f:
```

```
    pkl.dump(model3, f)

with open('model.pkl', 'rb') as f:
    model = pkl.load(f)

model.predict(test)
```

Output Screenshot:





Result:
Hence, the prediction model is built and implemented using flask.