NETAJI SUBHAS UNIVERSITY OF TECHNOLOGY

Machine Learning

COCSC17

Submitted By:

Laksh Sachdeva

2022UCS1572

CSE-1

TABLE OF CONTENTS

1) Implementation of Linear Regression

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

df = pd.read_csv("placement (1).csv")

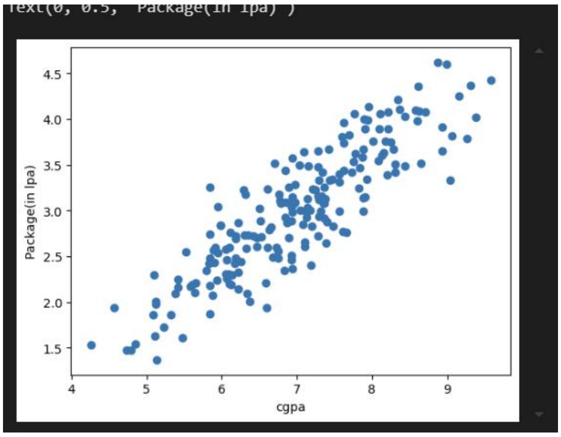
df.head()

	cgpa	package
0	6.89	3.26
1	5.12	1.98
2	7.82	3.25
3	7.42	3.67
4	6.94	3.57

plt.scatter(df['cgpa'] , df['package'])

plt.xlabel("cgpa")

plt.ylabel("Package(in lpa)")



X = df.iloc[: , 0:1]

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=0.2 , random_state=42)

lr = LinearRegression()

lr.fit(X_train , y_train)

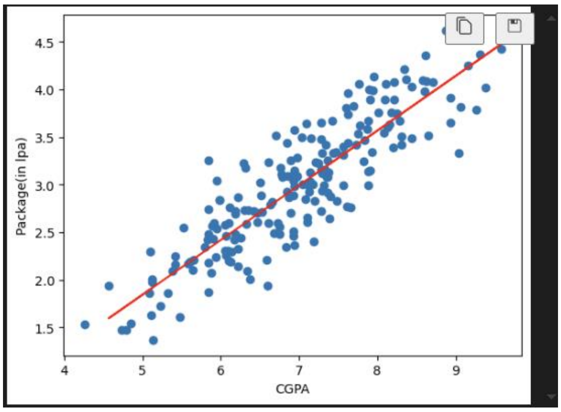
lr.predict(X_train)

plt.scatter(df['cgpa'] , df['package'])

plt.plot(X_train , lr.predict(X_train) , color = "red")

plt.xlabel("CGPA")

plt.ylabel("Package(in lpa)")



Coefficifient of Linear Regression -- SLOPE

```
m = lr.coef_
```

Intercept of Linear Regression

```
b = lr.intercept_
```

y_pred = m(x_dataset) + b

lr.predict([[8.58]])

```
array([3.9001136])
```

2) Implementation of Logistic Regression

The dataset has two columns - age (age of the person/customer) and bought_insurance (whether the customer bought insurance or not). If bought_insurance = 1, the customer bought insurance and if bought_insurance = 0, the customer did not buy the insurance.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
df = pd.read_csv("insurance_data.csv")
df.head()
```

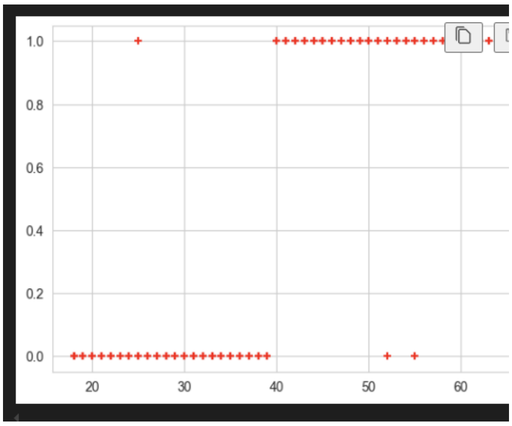
```
df = pd.read_csv("insurance_data.csv")
df.head()
```

	age	bought_insurance
0	22	0
1	25	0
2	47	1
3	52	0
4	46	1

```
print("Classification classes in the dataset
: " , np.unique(df['bought_insurance']))
```

Classification classes in the dataset : [0 1]

```
plt.scatter(df['age'],df['bought_insurance']
,marker='+',color='red')
```




```
X = df[['age']]     # input variable
y = df['bought_insurance']    # output
variable
```

Shape: (196, 1) Dimension: 2
Shape: (196,) Dimension: 1

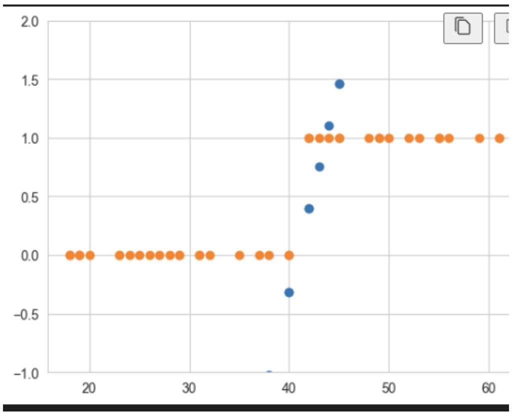
```
from sklearn.model_selection import
train_test_split
X_train, X_test, y_train, y_test =
train_test_split(X,y,test_size=0.2,
random_state = 42)
```

```
from sklearn.linear_model import
LogisticRegression
log_r = LogisticRegression()
log_r.fit(X_train , y_train)
```

```
y_pred = log_r.predict(X_test)
```

```
coeff = log_r.coef_
b = log_r.intercept_
```

```
x_in = X_test
y_val = coeff*x_in + b

plt.scatter(x_in , y_val)
plt.scatter(X_test , y_pred)
plt.ylim(-1,2)
```



print("Predict when age is 63" , log_r.predict([[63]]))

print("Predict when age is 40" , log_r.predict([[40]]))

```
Predict when age is 63 [1]
Predict when age is 40 [0]
```

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

tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
total_samples = len(y_test)
tn_percent = (tn / total_samples) * 100
fp_percent = (fp / total_samples) * 100
fn_percent = (fn / total_samples) * 100
tp_percent = (tp / total_samples) * 100

print(f"True Negative is: {tn_percent:.2f}%")
print(f"False Positive is: {fp_percent:.2f}%")
print(f"False Negative is: {fn_percent:.2f}%")
print(f"True Positive is: {tp_percent:.2f}%")

```
True Negative is: 47.50%
False Positive is: 0.00%
False Negative is: 7.50%
True Positive is: 45.00%
```

from sklearn.metrics import accuracy_score
acc = accuracy_score(y_test, y_pred)*100
print("Accuracy of model is: " , acc ,"%")

```
Accuracy of model is:  92.5 %
```

from sklearn.metrics import precision_score,recall_score
print("Precision " , precision_score(y_test,y_pred,average=None))

```
Precision  [0.86363636 1.        ]
```

3) Implementation of SVM

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

%matplotlib inline

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

df.shape
```

```
(17898, 9)
```

```
df.head()
```

	Mean of the integrated profile	Standard deviation of the integrated profile	Excess kurtosis of the integrated profile	Skewness of the integrated profile	Mean of the DM-SNR curve	S d D
0	140.562500	55.683782	-0.234571	-0.699648	3.199833	19
1	102.507812	58.882430	0.465318	-0.515088	1.677258	14
2	103.015625	39.341649	0.323328	1.051164	3.121237	21
3	136.750000	57.178449	-0.068415	-0.636238	3.642977	20
4	88.726562	40.672225	0.600866	1.123492	1.178930	11

We can see that there are 9 variables in the dataset. 8 are continuous variables and 1 is discrete variable. The discrete variable is target_class variable. It is also the target variable.

```
df.columns = ['IP Mean', 'IP Sd', 'IP Kurtosis', 'IP Skewness',
              'DM-SNR Mean', 'DM-SNR Sd', 'DM-SNR Kurtosis', 'DM-SNR Skewness', 'target_class']

df['target_class'].value_counts()
```

```
target_class
0    16259
1     1639
Name: count, dtype: int64
```

There are 9 numerical variables in the dataset.

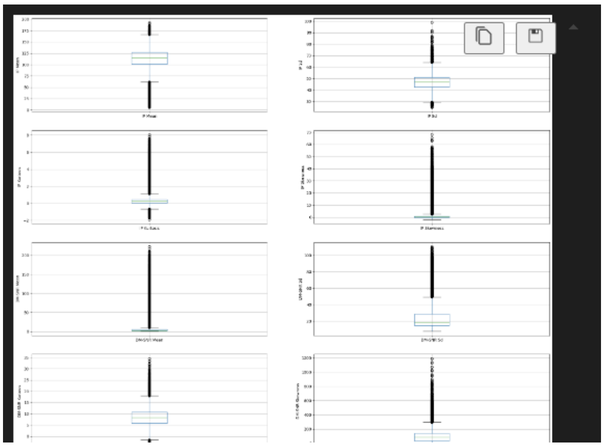
8 are continuous variables and 1 is discrete variable.

The discrete variable is target_class variable. It is also the target variable.

There are no missing values in the dataset.

```
# draw boxplots to visualize outliers

plt.figure(figsize=(24,20))

plt.subplot(4, 2, 1)
fig = df.boxplot(column='IP Mean')
fig.set_title('')
fig.set_ylabel('IP Mean')

plt.subplot(4, 2, 2)
fig = df.boxplot(column='IP Sd')
fig.set_title('')
fig.set_ylabel('IP Sd')

plt.subplot(4, 2, 3)
fig = df.boxplot(column='IP Kurtosis')
fig.set_title('')
fig.set_ylabel('IP Kurtosis')

plt.subplot(4, 2, 4)
fig = df.boxplot(column='IP Skewness')
fig.set_title('')
fig.set_ylabel('IP Skewness')

plt.subplot(4, 2, 5)
```

```
fig = df.boxplot(column='DM-SNR Mean')

fig.set_title('')

fig.set_ylabel('DM-SNR Mean')


plt.subplot(4, 2, 6)

fig = df.boxplot(column='DM-SNR Sd')

fig.set_title('')

fig.set_ylabel('DM-SNR Sd')


plt.subplot(4, 2, 7)

fig = df.boxplot(column='DM-SNR Kurtosis')

fig.set_title('')

fig.set_ylabel('DM-SNR Kurtosis')


plt.subplot(4, 2, 8)

fig = df.boxplot(column='DM-SNR Skewness')

fig.set_title('')

fig.set_ylabel('DM-SNR Skewness')
```



here are 2 variants of SVMs. They are hard-margin variant of SVM and soft-margin variant of SVM.

The hard-margin variant of SVM does not deal with outliers. In this case, we want to find the hyperplane with maximum margin such that every training point is correctly classified with margin at least 1. This technique does not handle outliers well.

Another version of SVM is called soft-margin variant of SVM. In this case, we can have a few points incorrectly classified or classified with a margin less than 1. But for every such point, we have to pay a penalty in the form of C parameter, which controls the outliers. Low C implies we are allowing more outliers and high C implies less outliers.

The message is that since the dataset contains outliers, so the value of C should be high while training the model.

```
X = df.drop(['target_class'], axis=1)


y = df['target_class']

from sklearn.model_selection import train_test_split


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)


# check the shape of X_train and X_test


from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)


from sklearn.svm import SVC

from sklearn.metrics import accuracy_score

svc = SVC()

svc.fit(X_train ,y_train)

y_pred = svc.predict(X_test)

print('Model accuracy score with default hyperparameters: {0:0.4f}'.
format(accuracy_score(y_test, y_pred)))
```

```
Model accuracy score: 0.9796
```

from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred)

print('Confusion matrix\n\n', cm)

print('\nTrue Positives(TP) = ', cm[0,0])

print('\nTrue Negatives(TN) = ', cm[1,1])

print('\nFalse Positives(FP) = ', cm[0,1])

print('\nFalse Negatives(FN) = ', cm[1,0])

```
Confusion matrix

 [[3242   17]
 [  56  265]]

True Positives(TP) =  3242

True Negatives(TN) =  265

False Positives(FP) =  17

False Negatives(FN) =  56
```

from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred))

	precision	recall	f1-score	support
0	0.98	0.99	0.99	3259
1	0.94	0.83	0.88	321
accuracy			0.98	3580
macro avg	0.96	0.91	0.93	3580
weighted avg	0.98	0.98	0.98	3580

4) Implementation of Random Forest

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

%matplotlib inline

df = pd.read_csv('car_evaluation (1).csv')


df.shape
```

```
(1727, 7)
```

```
df.head()
```

	vhigh	vhigh.1	2	2.1	small	low	unacc
0	vhigh	vhigh	2	2	small	med	unacc
1	vhigh	vhigh	2	2	small	high	unacc
2	vhigh	vhigh	2	2	med	low	unacc
3	vhigh	vhigh	2	2	med	med	unacc
4	vhigh	vhigh	2	2	med	high	unacc

```
col_names = ['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety', 'class']


df.columns = col_names

df.head()
```

	buying	maint	doors	persons	lug_boot
0	vhigh	vhigh	2	2	small
1	vhigh	vhigh	2	2	small
2	vhigh	vhigh	2	2	med
3	vhigh	vhigh	2	2	med
4	vhigh	vhigh	2	2	med

```
col_names = ['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety', 'class']

for col in col_names:


    print(df[col].value_counts())
```

Summary of Model

There are 7 variables in the dataset. All the variables are of categorical data type.

These are given by buying, maint, doors, persons, lug_boot, safety and class.

class is the target variable.

```
X = df.drop(['class'], axis=1)


y = df['class']

# split data 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 = 0.33, random_state = 42)

X_train.shape, X_test.shape
```

```
((1157, 6), (570, 6))
```

```
import category_encoders as ce

encoder = ce.OrdinalEncoder(cols=['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety'])
```

X_train = encoder.fit_transform(X_train)

X_test = encoder.transform(X_test)

X_train.head()

	buying	maint	doors	persons	lug_boot	safety
83	1	1	1	1	1	1
48	1	1	2	2	1	2
468	2	1	2	3	2	2
155	1	2	2	2	1	1
1043	3	2	3	2	2	1

import Random Forest classifier

from sklearn.ensemble import RandomForestClassifier

instantiate the classifier

rfc = RandomForestClassifier(random_state=0)

fit the model

rfc.fit(X_train, y_train)

Predict the Test set results

y_pred = rfc.predict(X_test)

Check accuracy score

from sklearn.metrics import accuracy_score

print('Model accuracy score with 10 decision-trees : {0:0.4f}'.

```
Model accuracy score with 10 decision-trees : 0.9649
```

```
Random Forest Classifier model with paramet
n_estimators=100
```

rfc_100 = RandomForestClassifier(n_estimators=100, random_state=0)

fit the model to the training set

rfc_100.fit(X_train, y_train)

Predict on the test set results

y_pred_100 = rfc_100.predict(X_test)

Check accuracy score

print('Model accuracy score with 100 decision-trees : {0:0.4f}'. format(accuracy_score(y_test, y_pred_100)))

```
Model accuracy score with 100 decision-
+ Code    + Markdown
```

clf = RandomForestClassifier(n_estimators=100, random_state=0)

fit the model to the training set

clf.fit(X_train, y_train)

feature_scores = pd.Series(clf.feature_importances_, index=X_train.columns).sort_values(ascending=False)

feature_scores

```
safety      0.291657
persons     0.235380
buying      0.160692
maint       0.134143
lug_boot    0.111595
doors       0.066533
dtype: float64
```

We can see that the most important featu
important feature is doors.

sns.barplot(x=feature_scores, y=feature_scores.index)

Add labels to the graph

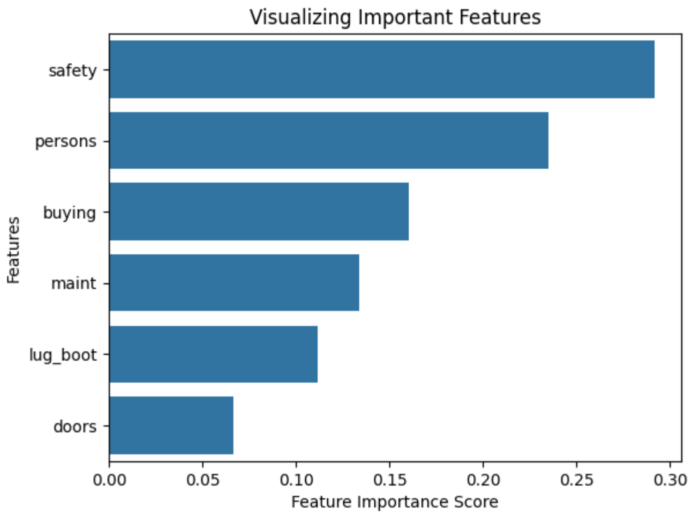
plt.xlabel('Feature Importance Score')

plt.ylabel('Features')

Add title to the graph

plt.title("Visualizing Important Features")

plt.show()



declare feature vector and target variable

X = df.drop(['class', 'doors'], axis=1)

y = df['class']

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)

encoder = ce.OrdinalEncoder(cols=['buying', 'maint', 'persons', 'lug_boot', 'safety'])

X_train = encoder.fit_transform(X_train)

X_test = encoder.transform(X_test)

clf = RandomForestClassifier(random_state=0)

fit the model to the training set

clf.fit(X_train, y_train)

Predict on the test set results

y_pred = clf.predict(X_test)

Check accuracy score

print('Model accuracy score with doors variable removed : {0:0.4f}'. format(accuracy_score(y_test, y_pred)))

```
Model accuracy score with doors variable removed
```

from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred)

print('Confusion matrix\n\n', cm)

```
Confusion matrix

 [[108   5  12   2]
 [  1  10   2   5]
 [ 10   0 389   0]
 [  4   1   0  21]]
```

from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred))

```
              precision    recall  f1-score   support

         acc       0.88      0.85      0.86       127
        good       0.62      0.56      0.59        18
       unacc       0.97      0.97      0.97       399
       vgood       0.75      0.81      0.78        26

    accuracy                           0.93       570
   macro avg       0.80      0.80      0.80       570
weighted avg       0.93      0.93      0.93       570
```

5) Implementation of Decision Trees

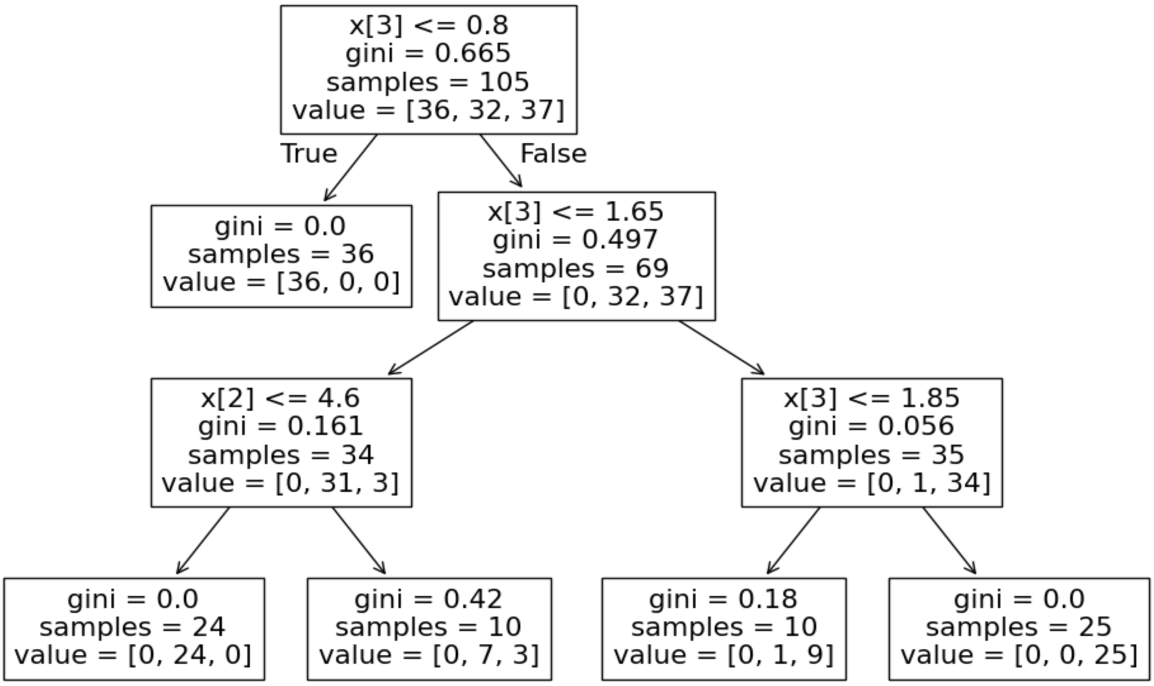
```
#Loading Libraries

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

%matplotlib inline


import sklearn.datasets as datasets

from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy_score, roc_auc_score, roc_curve

from sklearn.tree import plot_tree


from sklearn.tree import DecisionTreeClassifier

from sklearn.model_selection import GridSearchCV, RandomizedSearchCV


from sklearn.metrics import confusion_matrix

from sklearn.metrics import classification_report

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

df.head()
```

	sepal.length	sepal.width	petal.length	petal.width	variety
0	5.1	3.5	1.4	0.2	Setosa
1	4.9	3.0	1.4	0.2	Setosa
2	4.7	3.2	1.3	0.2	Setosa
3	4.6	3.1	1.5	0.2	Setosa
4	5.0	3.6	1.4	0.2	Setosa

```
from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()

df['variety'] = encoder.fit_transform(df['variety'])


df.head()
```

	sepal.length	sepal.width	petal.length	petal.width	variety
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

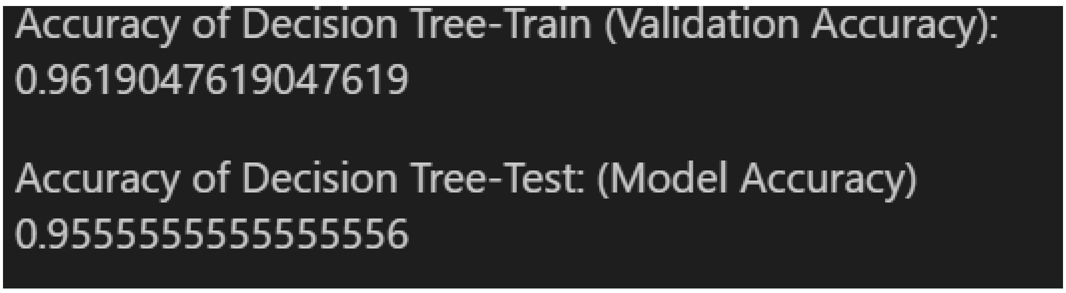
```
# Splitting the data into train and test sets

X = df.drop("variety",axis=1)

y = df["variety"]

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3, random_state= 1)


# Defining an object for DTC and fitting for whole dataset

dt = DecisionTreeClassifier(criterion="gini" , max_depth=3, min_samples_leaf=10, random_state=1 )

dt.fit(X, y)

plt.figure(figsize=(12,8))


from sklearn import tree


tree.plot_tree(dt.fit(X_train, y_train))
```




```
from sklearn.metrics import accuracy_score ,

classification_report , confusion_matrix , recall_score,precision_score,roc_curve , roc_auc_score

y_pred_train = dt.predict(X_train)

y_pred = dt.predict(X_test)

print('Accuracy of Decision Tree-Train  (Validation Accuracy): ', accuracy_score(y_pred_train, y_train))

print('Accuracy of Decision Tree-Test: (Model Accuracy) ', accuracy_score(y_pred, y_test))
```

Accuracy of Decision Tree-Train (Validation Accuracy): 0.9619047619047619

Accuracy of Decision Tree-Test: (Model Accuracy) 0.9555555555555556


Accuracy of Decision Tree-Train (Validation Accuracy):
0.9619047619047619

Accuracy of Decision Tree-Test: (Model Accuracy)
0.9555555555555556

```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        14
           1       0.94      0.94      0.94        18
           2       0.92      0.92      0.92        13

    accuracy                           0.96        45
   macro avg       0.96      0.96      0.96        45
weighted avg       0.96      0.96      0.96        45
```

print('Confusion Matrix - Train:','\n',confusion_matrix(y_train,y_pred_train))

print('\n','Confusion Matrix - Test:','\n',confusion_matrix(y_test,y_pred))

```
Confusion Matrix - Train:
 [[36  0  0]
 [ 0 31  1]
 [ 0  3 34]]

 Confusion Matrix - Test:
 [[14  0  0]
 [ 0 17  1]
 [ 0  1 12]]
```

6) Implementation of Adaboost

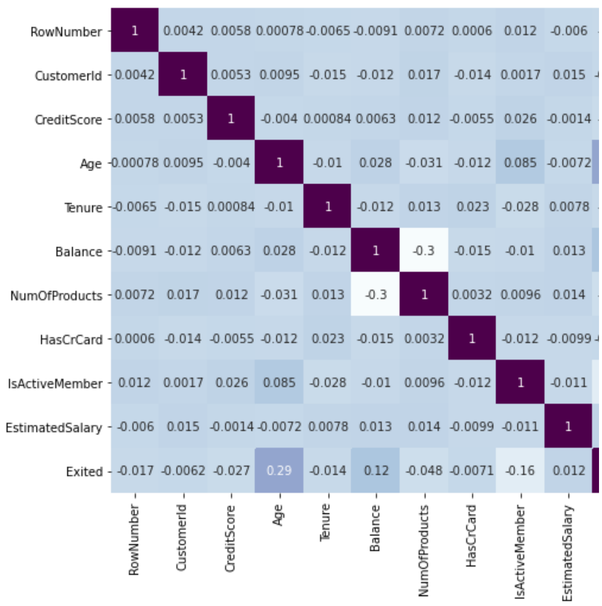
```
import pandas as pd
import numpy as np
from mlxtend.plotting import
plot_decision_regions


df = pd.DataFrame()


df['X1'] = [1,2,3,4,5,6,6,7,9,9]
df['X2'] = [5,3,6,8,1,9,5,8,9,2]
df['label'] = [1,1,0,1,0,1,0,1,0,0]


import seaborn as sns
sns.scatterplot(x=df['X1'],y=df['X2'],hue=d
f['label'])

df['weights'] = 1/df.shape[0]


df


from sklearn.tree import
DecisionTreeClassifier


dt1 =
DecisionTreeClassifier(max_depth=1)


X = df.iloc[:,0:2].values
y = df.iloc[:,2].values


# Step 2 - Train 1st model
dt1.fit(X,y)

DecisionTreeClassifier(ccp_alpha=0.0,
class_weight=None, criterion='gini',
            max_depth=1,
max_features=None,
max_leaf_nodes=None,
            min_impurity_decrease=0.0,
min_impurity_split=None,
            min_samples_leaf=1,
min_samples_split=2,
            min_weight_fraction_leaf=0.
0, presort='deprecated',
            random_state=None,
splitter='best')

from sklearn.tree import plot_tree
plot_tree(dt1)


plot_decision_regions(X, y, clf=dt1,
legend=2)


df['y_pred'] = dt1.predict(X)


def calculate_model_weight(error):

  return 0.5*np.log((1-error)/(error))


# Step 3 - calculate model weight
alpha1 = calculate_model_weight(0.3)
alpha1


# Step 4 - Update weights
def
update_row_weights(row,alpha=0.423):
  if row['label'] == row['y_pred']:
    return row['weights'] * np.exp(-alpha)
  else:
    return row['weights'] * np.exp(alpha)


df['updated_weights'] =
df.apply(update_row_weights,axis=1)


df['updated_weights'].sum()
```

```
df['nomalized_weights'] = df['updated_weights']/df['updated_weights'].sum()

df

df['nomalized_weights'].sum()

df['cumsum_upper'] = np.cumsum(df['nomalized_weights'])

df['cumsum_lower'] = df['cumsum_upper'] - df['nomalized_weights']

df[['X1','X2','label','weights','y_pred','updated_weights','cumsum_lower','cumsum_upper']]

def create_new_dataset(df):

  indices = []

  for i in range(df.shape[0]):
    a = np.random.random()
    for index,row in df.iterrows():
      if row['cumsum_upper'] > a and a > row['cumsum_lower']:
        indices.append(index)
  return indices

index_values = create_new_dataset(df)

second_df = df.iloc[index_values,[0,1,2,3]]

dt2 = DecisionTreeClassifier(max_depth=1)

X = second_df.iloc[:,0:2].values
y = second_df.iloc[:,2].values

dt2.fit(X,y)

DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=1, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')

plot_tree(dt2)

plot_decision_regions(X, y, clf=dt2, legend=2)

second_df['y_pred'] = dt2.predict(X)

alpha2 = calculate_model_weight(0.1)

# Step 4 - Update weights
def update_row_weights(row,alpha=1.09):
  if row['label'] == row['y_pred']:
    return row['weights'] * np.exp(-alpha)
  else:
    return row['weights'] * np.exp(alpha)

second_df['updated_weights'] = second_df.apply(update_row_weights,axis=1)

second_df['nomalized_weights'] = second_df['updated_weights']/second_df['updated_weights'].sum()
```

```
second_df['nomalized_weights'].sum()

second_df['cumsum_upper'] = np.cumsum(second_df['nomalized_weights'])

second_df['cumsum_lower'] = second_df['cumsum_upper'] - second_df['nomalized_weights']

second_df[['X1','X2','label','weights','y_pred','nomalized_weights','cumsum_lower','cumsum_upper']]

index_values = create_new_dataset(second_df)

third_df = second_df.iloc[index_values,[0,1,2,3]]

dt3 = DecisionTreeClassifier(max_depth=1)

X = second_df.iloc[:,0:2].values
y = second_df.iloc[:,2].values

dt3.fit(X,y)

DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                max_depth=1, max_features=None, max_leaf_nodes=None,
                min_impurity_decrease=0.0, min_impurity_split=None,
                min_samples_leaf=1, min_samples_split=2,
                min_weight_fraction_leaf=0.0, presort='deprecated',
                random_state=None, splitter='best')

plot_decision_regions(X, y, clf=dt3, legend=2)

third_df['y_pred'] = dt3.predict(X)

third_df

alpha3 = calculate_model_weight(0.7)
alpha3

print(alpha1,alpha2,alpha3)

query = np.array([1,5]).reshape(1,2)
dt1.predict(query)

dt2.predict(query)

dt3.predict(query)

alpha1*1 + alpha2*(1) + alpha3*(1)

np.sign(1.09)

query = np.array([9,9]).reshape(1,2)
dt1.predict(query)

dt2.predict(query)

dt3.predict(query)

alpha1*(1) + alpha2*(-1) + alpha3*(-1)

np.sign(-0.25)
```

7) Implementation of ANN

```
#import libraries

import numpy as np

import pandas as pd

import tensorflow as tf

#importing dataset

dataset = pd.read_csv('Churn_Modelling.csv')

#partial view of dataset from top

dataset.head()

#partial view of dataset from bottom

dataset.tail()

#dimention of the dataset

dataset.shape

#finding correlation between the features

corr_var=dataset.corr()

print(corr_var)

plt.figure(figsize=(10,7.5))

sns.heatmap(corr_var, annot=True, cmap='BuPu')
```




```
#as there is no importance in cust id, row no and sur name for modelling we are not included here in independent feature

X = dataset.iloc[:, 3:-1].values

#target value

y = dataset.iloc[:, -1].values


#as we have two columns as categorical terms we go for encoding we need to convert to numericals

#Categorical encoding


#gender will have some correlation with other feature so we go for label encoding

from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

#gender column in index 2

X[:, 2] = le.fit_transform(X[:, 2])


#country name wont be that much correlation added it has more than 2 names so go for one hot encoding

from sklearn.compose import ColumnTransformer

from sklearn.preprocessing import OneHotEncoder

#country name is present in 1st index value

ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [1])], remainder='passthrough')

X = np.array(ct.fit_transform(X))


#training and testing split

from sklearn.model_selection import train_test_split
```

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 0)

#feature scaling is an important and mandatory for ann process before modelling

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X_train = sc.fit_transform(X_train)

X_test = sc.transform(X_test)

#ANN - initializing

ann = tf.keras.models.Sequential()

#input layer

6 features

ann.add(tf.keras.layers.Dense(units=6, activation='relu'))

#hidden layer

ann.add(tf.keras.layers.Dense(units=6, activation='relu'))

#output layer

#as target value is binary - AF

ann.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))

#compiling

#loss - target is binary

ann.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])

#training set

ann.fit(X_train, y_train, batch_size = 32, epochs = 50)

#test result - prediction

y_pred = ann.predict(X_test)

#insted of values we ll get 0 or 1

y_pred = (y_pred > 0.5)

#actual vs prediicted outputs

print(np.concatenate((y_pred.reshape(len(y_pred), 1), y_test.reshape(len(y_test),1)),1))

#accuracy and confusion matrix

from sklearn.metrics import confusion_matrix, accuracy_score

cm = confusion_matrix(y_test, y_pred)

print(cm)

accuracy_score(y_test, y_pred)

Use our ANN model to predict if the customer with the following informations will leave the bank:

Geography: France

Credit Score: 750

Gender: Female

Age: 48 years old

Tenure: 5 years

Balance: $ 62500

Number of Products: 3

Does this customer have a credit card ? Yes

Is this customer an Active Member: Yes

Estimated Salary: $ 80000

So, should we say goodbye to that customer ?

#for predicting single sample

print(ann.predict(sc.transform([[1, 0, 0, 750, 0, 48,53, 62500, 3, 1, 1, 80000]])) > 0.5)

```
[[False]]
```

```
Epoch 1/50
219/219 [==============================] - 1s 1ms/step - loss: 0.8414 - accuracy: 0.2877
Epoch 2/50
219/219 [==============================] - 0s 1ms/step - loss: 0.6430 - accuracy: 0.7643
Epoch 3/50
219/219 [==============================] - 0s 1ms/step - loss: 0.5725 - accuracy: 0.8061
Epoch 4/50
219/219 [==============================] - 0s 954us/step - loss: 0.4945 - accuracy: 0.7956
Epoch 5/50
219/219 [==============================] - 0s 881us/step - loss: 0.4599 - accuracy: 0.7990
Epoch 6/50
219/219 [==============================] - 0s 867us/step - loss: 0.4518 - accuracy: 0.7947
Epoch 7/50
219/219 [==============================] - 0s 872us/step - loss: 0.4339 - accuracy: 0.8013
Epoch 8/50
219/219 [==============================] - 0s 890us/step - loss: 0.4292 - accuracy: 0.8081
Epoch 9/50
219/219 [==============================] - 0s 922us/step - loss: 0.4118 - accuracy: 0.8195
Epoch 10/50
219/219 [==============================] - 0s 922us/step - loss: 0.4127 - accuracy: 0.8170
Epoch 11/50
219/219 [==============================] - 0s 895us/step - loss: 0.3895 - accuracy: 0.8421
Epoch 12/50
219/219 [==============================] - 0s 876us/step - loss: 0.3917 - accuracy: 0.8392
Epoch 13/50
...
Epoch 49/50
219/219 [==============================] - 0s 895us/step - loss: 0.3259 - accuracy: 0.8675
Epoch 50/50
219/219 [==============================] - 0s 908us/step - loss: 0.3431 - accuracy: 0.8563
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

8) Implementation of CNN

```
import mnist

import numpy as np

from conv import Conv3x3

from maxpool import MaxPool2

from softmax import Softmax


# We only use the first 1k examples of each set in the interest of time.

# Feel free to change this if you want.

train_images = mnist.train_images()[:1000]

train_labels = mnist.train_labels()[:1000]

test_images = mnist.test_images()[:1000]

test_labels = mnist.test_labels()[:1000]


conv = Conv3x3(8)              # 28x28x1 -> 26x26x8

pool = MaxPool2()              # 26x26x8 -> 13x13x8

softmax = Softmax(13 * 13 * 8, 10) # 13x13x8 -> 10


def forward(image, label):

  '''

  Completes a forward pass of the CNN and calculates the accuracy and

  cross-entropy loss.

  - image is a 2d numpy array

  - label is a digit

  '''

  # We transform the image from [0, 255] to [-0.5, 0.5] to make it easier

  # to work with. This is standard practice.

  out = conv.forward((image / 255) - 0.5)

  out = pool.forward(out)

  out = softmax.forward(out)


  # Calculate cross-entropy loss and accuracy. np.log() is the natural log.

  loss = -np.log(out[label])

  acc = 1 if np.argmax(out) == label else 0


  return out, loss, acc


def train(im, label, lr=.005):

  '''

  Completes a full training step on the given image and label.

  Returns the cross-entropy loss and accuracy.

  - image is a 2d numpy array

  - label is a digit

  - lr is the learning rate

  '''

  # Forward

  out, loss, acc = forward(im, label)


  # Calculate initial gradient

  gradient = np.zeros(10)

  gradient[label] = -1 / out[label]


  # Backprop

  gradient = softmax.backprop(gradient, lr)

  gradient = pool.backprop(gradient)
```

```
  gradient = conv.backprop(gradient, lr)

  return loss, acc

print('MNIST CNN initialized!')

# Train the CNN for 3 epochs
for epoch in range(3):
  print('--- Epoch %d ---' % (epoch + 1))

  # Shuffle the training data
  permutation =
np.random.permutation(len(train_images))
  train_images = train_images[permutation]
  train_labels = train_labels[permutation]

  # Train!
  loss = 0
  num_correct = 0
  for i, (im, label) in enumerate(zip(train_images,
train_labels)):
    if i % 100 == 99:
      print(
        '[Step %d] Past 100 steps: Average Loss %.3f |
Accuracy: %d%%' %
        (i + 1, loss / 100, num_correct)
      )
      loss = 0
      num_correct = 0

    l, acc = train(im, label)
    loss += l
    num_correct += acc

# Test the CNN
print('\n--- Testing the CNN ---')
loss = 0
num_correct = 0
for im, label in zip(test_images, test_labels):
  _, l, acc = forward(im, label)
  loss += l
  num_correct += acc

num_tests = len(test_images)
print('Test Loss:', loss / num_tests)
print('Test Accuracy:', num_correct / num_tests)
```

9) Implementation of KNN

```
import numpy as np

import pandas as pd

df = pd.read_csv('data (1).csv')

df.drop(columns=['id','Unnamed: 32'],inplace=True)

df.head()
```

	diagnosis	radius_mean	texture_mean	perir
0	M	17.99	10.38	
1	M	20.57	17.77	
2	M	19.69	21.25	
3	M	11.42	20.38	
4	M	20.29	14.34	

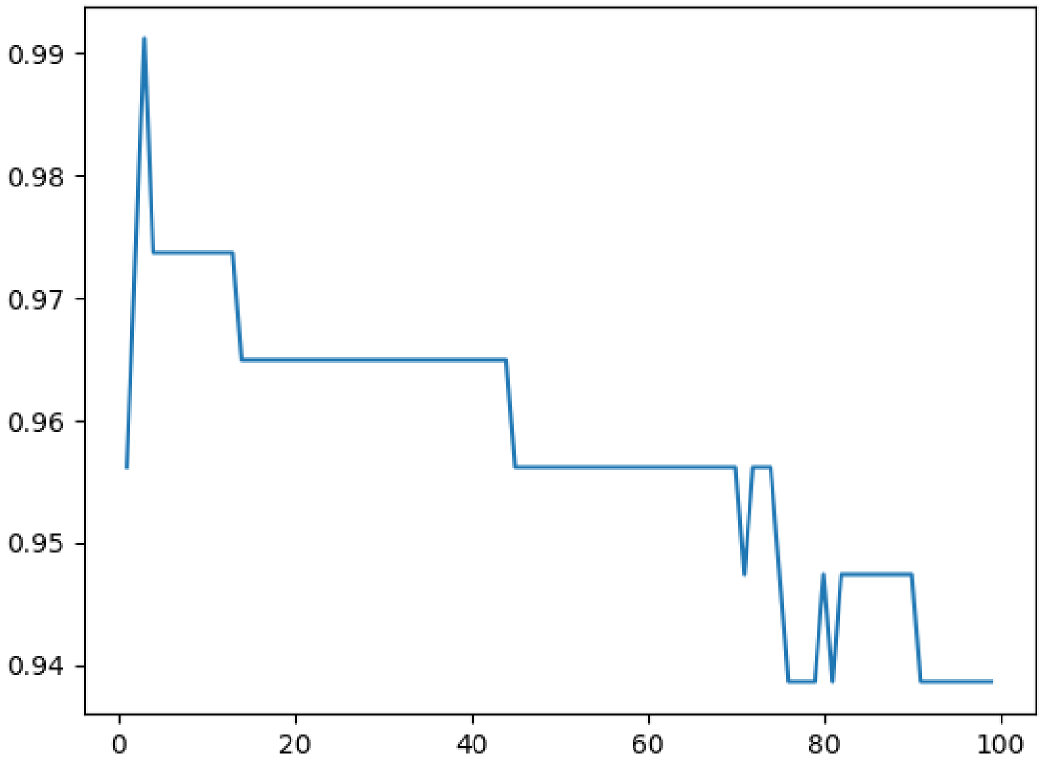
```
df.shape
```

```
(569, 31)
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(df.iloc[:,1:], df.iloc[:,0],test_size=0.2, random_state=2)

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)

from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=5)

knn.fit(X_train,y_train)

from sklearn.metrics import accuracy_score


y_pred = knn.predict(X_test)


accuracy_score(y_test, y_pred)
```

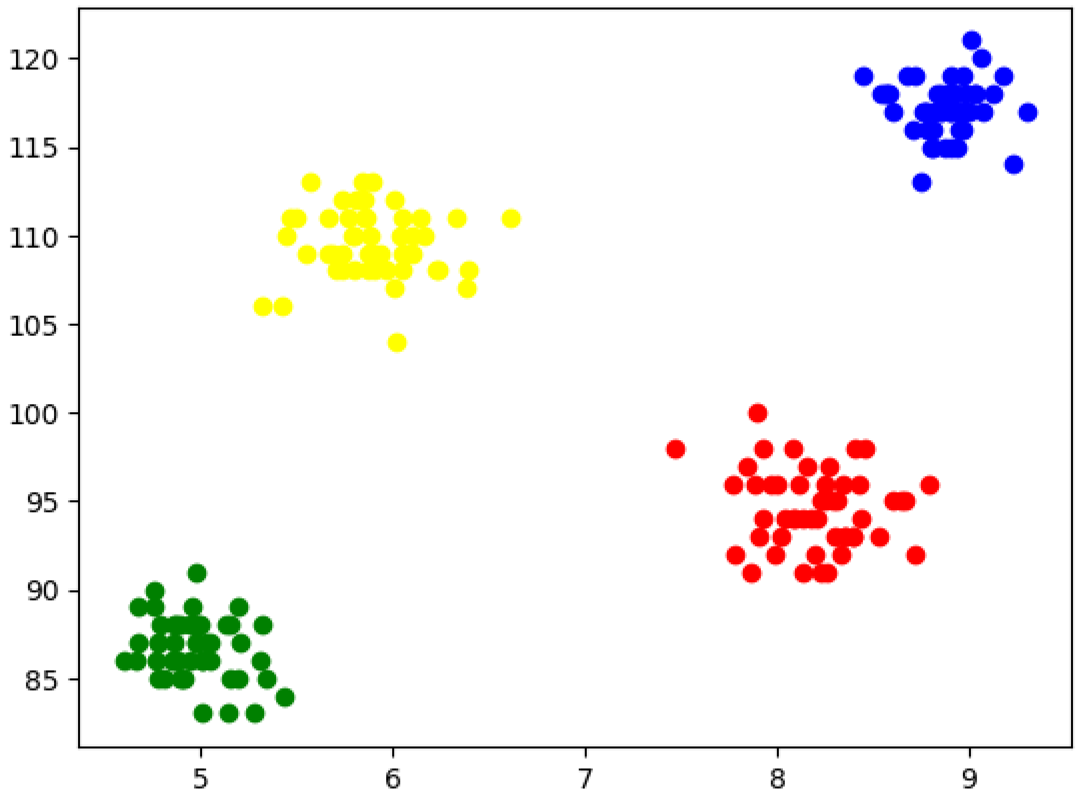
```
0.9736842105263158
```

```
scores = []


for i in range(1,100):

    knn = KNeighborsClassifier(n_neighbors=i)

    knn.fit(X_train,y_train)

    y_pred = knn.predict(X_test)

    scores.append(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt


plt.plot(range(1,100),scores)
```


0.99
0.98
0.97
0.96
0.95
0.94
0
20
40
60
80
100

10) Implementation of K-Means clustering

```
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

import pandas as pd

#centroids = [(-5,-5),(5,5),(-2.5,2.5),(2.5,-2.5)]
#cluster_std = [1,1,1,1]

#X,y =
make_blobs(n_samples=100,cluster_std=cluster_std,centers=centroids,n_features=2,random_state=2
)

#plt.scatter(X[:,0],X[:,1])

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

X = df.iloc[:,:].values

km = KMeans(n_clusters=4,max_iter=500)
y_means = km.fit_predict(X)

plt.scatter(X[y_means == 0,0],X[y_means == 0,1],color='red')
plt.scatter(X[y_means == 1,0],X[y_means == 1,1],color='blue')
plt.scatter(X[y_means == 2,0],X[y_means == 2,1],color='green')
plt.scatter(X[y_means == 3,0],X[y_means == 3,1],color='yellow')
plt.show()
```


120
115
110
105
100
95
90
85
5
6
7
8
9