Machine Learning Practical Record

Department of Computer Science & Application

Institute of Engineering & Technology

SUBMITTED TO: -

Dr. Pappu Kumar Bhagat

SUBMITTED BY: -

Abhishek Pal

201500013
Sec-k

Experiment:-1

Objective:- Linear Regration

```
import numpy as np
import pandas as pd
```

```
from google.colab import drive
drive.mount('/content/drive/')
```

```
Mounted at /content/drive/
```

```
series = pd.read_csv('/content/drive/MyDrive/annual_csv.csv',index_col = 'Date')
```

```
series
```

Date	Price
1950-12	34.720
1951-12	34.660
1952-12	34.790
1953-12	34.850
1954-12	35.040
...	...
2015-12	1068.317
2016-12	1152.165
2017-12	1265.674
2018-12	1249.887
2019-12	1480.025

70 rows × 1 columns

```
series.head()
```

Date	Price
1950-12	34.72
1951-12	34.66
1952-12	34.79
1953-12	34.85
1954-12	35.04

```
series.tail()
```

```
         Price
Date
```

```
trainingSize = int(len(series) * 0.75)
```

```
2016-12   1152.165
```

```
testingSize = int(len(series) - trainingSize)
```

```
trainingSize, testingSize
```

```
(52, 18)
```

```
train,test = series[0:trainingSize],series[trainingSize:len(series)]
```

```
len(test), len(train)
```

```
(18, 52)
```

```
from os import set_inheritable
def create_dataset(series, time_steps = 3):
  Xs, Ys = [], []
  for i in range(len(series) - time_steps):
    v = series.iloc[i:(i + time_steps)].values
    Xs.append(v)
    Ys.append(series['Price'].iloc[i+time_steps])
  return np.array(Xs),np.array(Ys)
```

```
series.shape
```

```
(70, 1)
```

```
time_steps = 3
X_train, Y_train = create_dataset(train, time_steps)
X_test, Y_test = create_dataset(test, time_steps)
```

```
print(X_train.shape, Y_train.shape)
```

```
(49, 3, 1) (49,)
```

```
x_tr = X_train.reshape(len(X_train),3)
x_t = X_test.reshape(len(X_test),3)
print(x_tr.shape)
```

```
(49, 3)
```

```
print(X_train)
```

```
[[[ 34.72 ]
  [ 34.66 ]
  [ 34.79 ]]

 [[ 34.66 ]
  [ 34.79 ]
  [ 34.85 ]]

 [[ 34.79 ]
  [ 34.85 ]
  [ 35.04 ]]

 [[ 34.85 ]
  [ 35.04 ]
  [ 34.97 ]]

 [[ 35.04 ]
  [ 34.97 ]
  [ 34.9  ]]

 [[ 34.97 ]
  [ 34.9  ]
  [ 34.99 ]]

 [[ 34.9  ]
  [ 34.99 ]
```

```
 [ 35.09 ]]

[[ 34.99 ]
 [ 35.09 ]
 [ 35.05 ]]

[[ 35.09 ]
 [ 35.05 ]
 [ 35.54 ]]

[[ 35.05 ]
 [ 35.54 ]
 [ 35.15 ]]

[[ 35.54 ]
 [ 35.15 ]
 [ 35.08 ]]

[[ 35.15 ]
 [ 35.08 ]
 [ 35.08 ]]

[[ 35.08 ]
 [ 35.08 ]
 [ 35.12 ]]

[[ 35.08 ]
 [ 35.12 ]
 [ 35.13 ]]

[[ 35.12 ]
```

```
print(Y_train)
```

```
[ 34.85   35.04   34.97   34.9    34.99   35.09   35.05   35.54   35.15
  35.08   35.08   35.12   35.13   35.18   35.19   41.113  35.189  37.434
  43.455  63.779 106.236 183.683 139.279 133.674 160.48  207.895 463.666
 596.712 410.119 444.776 388.06  319.622 321.985 391.595 487.079 419.248
 409.655 378.161 361.875 334.657 383.243 379.48  387.445 369.338 288.776
 291.357 283.743 271.892 275.992]
```

```
X_test
```

```
array([[[ 333.3  ],
        [ 407.674],
        [ 442.974]],

       [[ 407.674],
        [ 442.974],
        [ 509.423]],

       [[ 442.974],
        [ 509.423],
        [ 629.513]],

       [[ 509.423],
        [ 629.513],
        [ 803.618]],

       [[ 629.513],
        [ 803.618],
        [ 819.94 ]],

       [[ 803.618],
        [ 819.94 ],
        [1135.012]],

       [[ 819.94 ],
        [1135.012],
        [1393.512]],

       [[1135.012],
        [1393.512],
        [1652.725]],

       [[1393.512],
        [1652.725],
        [1687.342]],

       [[1652.725],
        [1687.342],
        [1221.588]],
```

```
       [[1687.342],
        [1221.588],
        [1200.44 ]],

       [[1221.588],
        [1200.44 ],
        [1068.317]],

       [[1200.44 ],
        [1068.317],
        [1152.165]],

       [[1068.317],
        [1152.165],
        [1265.674]],

       [[1152.165],
```

```
from sklearn.linear_model import LinearRegression
reg1 = LinearRegression()
reg1.fit(x_tr, Y_train)
y_pred = reg1.predict(x_t)
```

```
y_pred
```

```
array([ 425.24878102,  499.21195843,  614.7460528 ,  776.8956425 ,
        757.00982513, 1127.57060386, 1307.89266184, 1560.03761342,
       1555.61827349, 1077.90051716, 1217.71508902,  998.82312162,
       1131.7006029 , 1205.02246018, 1164.02035864])
```

```
from sklearn.metrics import mean_squared_error, r2_score
rmse_lr = np.sqrt(mean_squared_error(Y_test, y_pred))
r2_lr = r2_score(Y_test, y_pred)
print(rmse_lr, r2_lr)
```

```
216.41959383878694 0.5761964443390513
```

Experiment:-2

Objective:- logistic regression

```
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
```

```
data_set= pd.read_csv('/content/sample_data/User_Data.csv')
```

```
x= data_set.iloc[:, [2,3]].values
y= data_set.iloc[:, 4].values
```

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_state=0)
```

```
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)
```

```
# import the regressor
from sklearn.tree import DecisionTreeRegressor

# create a regressor object
regressor = DecisionTreeRegressor(random_state = 0)

# fit the regressor with X and Y data
regressor.fit(x_train, y_train)
```

```
▾ DecisionTreeRegressor
DecisionTreeRegressor(random_state=0)
```

```
y_pred = regressor.predict(x_test)
```

```
print('Decision tree',y_pred)
```

```
Decision tree [0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 1. 1. 0. 1. 0. 0. 0. 0. 1.
 0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0.
 0. 1. 1. 0. 0. 1. 1. 1. 0. 0. 1. 0. 0. 1. 0. 1. 0. 1. 0. 0. 0. 1. 1. 0.
 0. 1. 0. 0. 0. 0. 1. 1. 1. 1. 0. 0. 1. 0. 0. 1. 1. 0. 0. 1. 0. 0. 0. 1.
 0. 1. 1. 1.]
```

```
from sklearn.metrics import confusion_matrix
cm= confusion_matrix(y_test, y_pred)
print('Decision tree',cm)
```

```
Decision tree [[62  6]
 [ 4 28]]
```

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

```
print('Decision tree',accuracy_score(y_test, y_pred))
```

```
Decision tree 0.9
```

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_state=0)
```

```
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)
```

```
from sklearn.neighbors import KNeighborsClassifier
classifier= KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=2 )
classifier.fit(x_train, y_train)
```

```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

```
y_pred= classifier.predict(x_test)
print(y_pred)
```

```
[0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 0 1 0 0 1 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0
 0 0 1 0 0 0 0 1 0 0 1 0 1 1 0 0 1 1 1 0 0 1 0 0 1 0 1 0 1 0 0 0 0 1 0 0 1
 0 0 0 0 1 1 1 1 0 0 1 0 0 1 1 0 0 1 0 0 0 0 0 1 1 1]
```

```
from sklearn.metrics import confusion_matrix
cm= confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[64  4]
 [ 3 29]]
```

```
from sklearn.metrics import accuracy_score, f1_score
print('KNN',accuracy_score(y_test, y_pred))
```

```
KNN 0.93
```

```
from sklearn import linear_model
logr = linear_model.LogisticRegression()
logr.fit(x_train,y_train);
```

```
y_pred = logr.predict(x_test)
print('logistic', y_pred)
```

```
logistic [0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0
 0 0 1 0 0 0 0 1 0 0 1 0 1 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0
 0 0 1 0 1 1 1 1 0 0 1 1 0 1 0 0 0 1 0 0 0 0 0 0 1 1]
```

```
print('Logistic',accuracy_score(y_test, y_pred));
```

```
Logistic 0.89
```

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
```

```
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(x_train, y_train)
```

```
▾ GaussianNB
GaussianNB()
```

```
y_pred = classifier.predict(x_test)
print('naive bayes', y_pred)
```

```
naive bayes [0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0
 0 0 1 0 0 0 0 1 0 0 1 0 1 1 0 0 1 1 0 0 0 1 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0
 0 0 0 0 1 1 1 1 0 0 1 0 0 1 0 0 0 1 0 0 0 0 0 1 1 1]
```

```
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[65  3]
 [ 7 25]]
```

```
print('naive bayes',accuracy_score(y_test, y_pred));
```

```
naive bayes 0.9
```

```
from sklearn.svm import SVC # "Support vector classifier"
classifier = SVC(kernel='linear', random_state=0)
classifier.fit(x_train, y_train)
```

```
▾ SVC
SVC(kernel='linear', random_state=0)
```

```
y_pred= classifier.predict(x_test)
print('SVM',y_pred)
```

```
SVM [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0
 0 0 1 0 0 0 0 1 0 0 1 0 1 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0
 0 0 1 0 1 1 1 1 0 0 1 1 0 1 0 0 0 1 0 0 0 0 0 0 1 1]
```

```
print('SVM',accuracy_score(y_test, y_pred));
```

```
SVM 0.9
```

Experiment:-3

```
import pandas as pd
```

```
from google.colab import files
uploades = files.upload()
```

Choose Files diabetes.csv

- **diabetes.csv**(text/csv) - 23873 bytes, last modified: 3/20/2023 - 100% done

```
Saving diabetes.csv to diabetes.csv
```

```
data = pd.read_csv("diabetes.csv")
```

```
data.head(10)
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigre
0	6	148	72	35	0	33.6	
1	1	85	66	29	0	26.6	
2	8	183	64	0	0	23.3	
3	1	89	66	23	94	28.1	
4	0	137	40	35	168	43.1	
5	5	116	74	0	0	25.6	
6	3	78	50	32	88	31.0	
7	10	115	0	0	0	35.3	
8	2	197	70	45	543	30.5	
9	8	125	96	0	0	0.0	

Automatic saving failed. This fle was updated remotely or in another tab. Show diff

```
data.dtypes
```

```
Pregnancies                   int64
Glucose                       int64
BloodPressure                 int64
SkinThickness                 int64
Insulin                       int64
BMI                         float64
DiabetesPedigreeFunction    float64
Age                           int64
Outcome                       int64
dtype: object
```

```
data.describe()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFu
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.

```
from sklearn.model_selection import train_test_split

X = data.drop("Outcome", axis = 1)
y = data[["Outcome"]]
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.30, random_state=1)
```

```
from sklearn.naive_bayes import GaussianNB
```

```
model = GaussianNB()
model.fit(X_train, y_train)
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A
  colum y = column_or_1d(y, warn=True)
▾ GaussianNB

GaussianNB()
```

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

```
from sklearn import metrics
```

```
print ("Accuracy:", metrics.accuracy_score (y_test, y_pred))
```

```
Accuracy: 0.7835497835497836
```

```
test_pred = model.predict(X_test)
```

```
print(metrics.classification_report(y_test, test_pred))
print(metrics.confusion_matrix(y_test, test_pred))
```

```
              precision    recall  f1-score   support

           0       0.80      0.88      0.84       146
           1       0.75      0.62      0.68        85

    accuracy                           0.78       231
   macro avg       0.77      0.75      0.76       231
weighted avg       0.78      0.78      0.78       231

[[128  18]
 [ 32  53]]
```

Automatic saving failed. This fle was updated remotely or in another tab. Show diff

Colab paid products - Cancel contracts here

✓ 1s completed at 10:03 AM

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.

Experiment:-4

Objective:- KNN

```
import sklearn
import pandas as pd
from sklearn.datasets import load_iris
```

```
iris=load_iris()
iris.keys()
df=pd.DataFrame(iris['data'])
df.head()
print(df)
print(iris['target_names'])
iris['feature_names']
```

```
       0    1    2    3
0    5.1  3.5  1.4  0.2
1    4.9  3.0  1.4  0.2
2    4.7  3.2  1.3  0.2
3    4.6  3.1  1.5  0.2
4    5.0  3.6  1.4  0.2
..   ...  ...  ...  ...
145  6.7  3.0  5.2  2.3
146  6.3  2.5  5.0  1.9
147  6.5  3.0  5.2  2.0
148  6.2  3.4  5.4  2.3
149  5.9  3.0  5.1  1.8

[150 rows x 4 columns]
['setosa' 'versicolor' 'virginica']
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

```
X=df
y=iris['target']
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
```

```
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier(n_neighbors=5)
```

```
knn.fit(X_train, y_train)
```

```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

```
from sklearn import metrics
y_pred = knn.predict(X_test)
i = 0
print ("\n-------------------------------------------------------------------------")
print ('%-25s %-25s %-25s' % ('Original Label', 'Predicted Label', 'Correct/Wrong'))
print ("-------------------------------------------------------------------------")
for label in y_test:
    print ('%-25s %-25s' % (label, y_pred[i]), end="")
    if (label == y_pred[i]):
        print (' %-25s' % ('Correct'))
    else:
        print (' %-25s' % ('Wrong'))
    i = i + 1
print ("-------------------------------------------------------------------------")
print("\nConfusion Matrix:\n",metrics.confusion_matrix(y_test, y_pred))
print ("-------------------------------------------------------------------------")
print("\nClassification Report:\n",metrics.classification_report(y_test, y_pred))
```

```
print ("------------------------------------------------------------------------")
print('Accuracy of the classifer is %0.2f' % metrics.accuracy_score(y_test,y_pred))
print ("------------------------------------------------------------------------")
```

```
------------------------------------------------------------------------
Original Label            Predicted Label          Correct/Wrong
------------------------------------------------------------------------
1                         1                        Correct
0                         0                        Correct
2                         2                        Correct
1                         1                        Correct
1                         1                        Correct
0                         0                        Correct
1                         1                        Correct
2                         2                        Correct
1                         1                        Correct
1                         1                        Correct
2                         2                        Correct
0                         0                        Correct
0                         0                        Correct
0                         0                        Correct
0                         0                        Correct
1                         1                        Correct
2                         2                        Correct
1                         1                        Correct
1                         1                        Correct
2                         2                        Correct
0                         0                        Correct
2                         2                        Correct
0                         0                        Correct
2                         2                        Correct
2                         2                        Correct
2                         2                        Correct
2                         2                        Correct
2                         2                        Correct
0                         0                        Correct
0                         0                        Correct
0                         0                        Correct
0                         0                        Correct
1                         1                        Correct
0                         0                        Correct
0                         0                        Correct
2                         2                        Correct
1                         1                        Correct
0                         0                        Correct
0                         0                        Correct
0                         0                        Correct
2                         2                        Correct
1                         1                        Correct
1                         1                        Correct
0                         0                        Correct
0                         0                        Correct
1                         1                        Correct
                                                   Wrong
2                         2                        Correct
1                         1                        Correct
2                         2                        Correct
------------------------------------------------------------------------

Confusion Matrix:
 [[19  0  0]
```

```
seconds_in_a_day = 24 * 60 * 60
seconds_in_a_day
```

```
86400
```

```
seconds_in_a_week = 7 * seconds_in_a_day
seconds_in_a_week
```

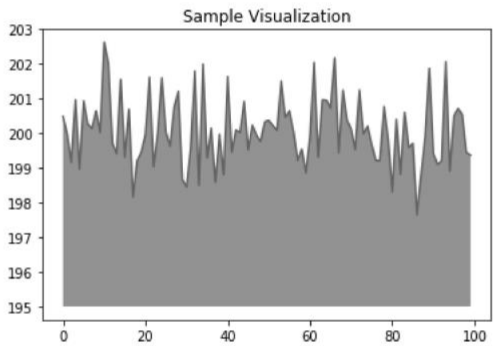
```
604800
```

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

ys = 200 + np.random.randn(100)
x = [x for x in range(len(ys))]

plt.plot(x, ys, '-')
plt.fill_between(x, ys, 195, where=(ys > 195), facecolor='g', alpha=0.6)
```

```
plt.title("Sample Visualization")
plt.show()
```



Experiment:-5

Objective:- SVM

```
import numpy as np


import pandas as pd
import matplotlib.pyplot as plt



from sklearn.datasets import load_iris
iris = load_iris()


dir(iris)
```

```
['DESCR',
 'data',
 'data_module',
 'feature_names',
 'filename',
 'frame',
 'target',
 'target_names']
```

```
iris.DESCR
```

```
'.. _iris_dataset:\n\nIris plants dataset\n--------------------\n\n**Data Set Cha
racteristics:**\n\n    :Number of Instances: 150 (50 in each of three classes)\n
:Number of Attributes: 4 numeric, predictive attributes and the class\n    :Attri
bute Information:\n        - sepal length in cm\n        - sepal width in cm\n
- petal length in cm\n        - petal width in cm\n        - class:\n
- Iris-Setosa\n                - Iris-Versicolour\n                - Iris-Virgini
ca\n                \n    :Summary Statistics:\n\n    ============== ==== ==== ==
===== ===== ====================\n                    Min  Max   Mean    SD   Cla
```

```
iris.data
```

```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
       [4.8, 3. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1],
       [5.8, 4. , 1.2, 0.2],
       [5.7, 4.4, 1.5, 0.4],
       [5.4, 3.9, 1.3, 0.4],
       [5.1, 3.5, 1.4, 0.3],
       [5.7, 3.8, 1.7, 0.3],
       [5.1, 3.8, 1.5, 0.3],
       [5.4, 3.4, 1.7, 0.2],
       [5.1, 3.7, 1.5, 0.4],
       [4.6, 3.6, 1. , 0.2],
       [5.1, 3.3, 1.7, 0.5],
       [4.8, 3.4, 1.9, 0.2],
       [5. , 3. , 1.6, 0.2],
       [5. , 3.4, 1.6, 0.4],
       [5.2, 3.5, 1.5, 0.2],
       [5.2, 3.4, 1.4, 0.2],
       [4.7, 3.2, 1.6, 0.2],
       [4.8, 3.1, 1.6, 0.2],
       [5.4, 3.4, 1.5, 0.4],
       [5.2, 4.1, 1.5, 0.1],
```

```
           [5.5, 4.2, 1.4, 0.2],
           [4.9, 3.1, 1.5, 0.2],
           [5. , 3.2, 1.2, 0.2],
           [5.5, 3.5, 1.3, 0.2],
           [4.9, 3.6, 1.4, 0.1],
           [4.4, 3. , 1.3, 0.2],
           [5.1, 3.4, 1.5, 0.2],
           [5. , 3.5, 1.3, 0.3],
           [4.5, 2.3, 1.3, 0.3],
           [4.4, 3.2, 1.3, 0.2],
           [5. , 3.5, 1.6, 0.6],
           [5.1, 3.8, 1.9, 0.4],
           [4.8, 3. , 1.4, 0.3],
           [5.1, 3.8, 1.6, 0.2],
           [4.6, 3.2, 1.4, 0.2],
           [5.3, 3.7, 1.5, 0.2],
           [5. , 3.3, 1.4, 0.2],
           [7. , 3.2, 4.7, 1.4],
           [6.4, 3.2, 4.5, 1.5],
           [6.9, 3.1, 4.9, 1.5],
           [5.5, 2.3, 4. , 1.3],
           [6.5, 2.8, 4.6, 1.5],
           [5.7, 2.8, 4.5, 1.3],
           [6.3, 3.3, 4.7, 1.6],
           [4.9, 2.4, 3.3, 1. ],
```

```
iris.feature_names
```

```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

```
iris.filename
```

```
'iris.csv'
```

```
iris.frame
```

```
iris.target
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

```
iris.target_names
```

```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

```
df=pd.DataFrame(iris.data,columns=iris.feature_names)
```

```
df.head()
```

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

```
df['target']=iris.target
```

```
df.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
		3.2	1.3	0.2	0

```
iris.target_names
```

```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

```
df[df.target==1]
```

```
df.info
```

```
<bound method DataFrame.info of      sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                  5.1               3.5                1.4               0.2
1                  4.9               3.0                1.4               0.2
2                  4.7               3.2                1.3               0.2
3                  4.6               3.1                1.5               0.2
4                  5.0               3.6                1.4               0.2
..                 ...               ...                ...               ...
```

```
145          6.7          3.0          5.2          2.3
146          6.3          2.5          5.0          1.9
147          6.5          3.0          5.2          2.0
148          6.2          3.4          5.4          2.3
149          5.9          3.0          5.1          1.8

     target
0         0
1         0
2         0
3         0
4         0
..      ...
145       2
146       2
147       2
148       2
149       2

[150 rows x 5 columns]>
```

```
x = df.iloc[:, 0:-1].values
y = df.iloc[:, 4].values
print(x)
```

```
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]
 [5.4 3.7 1.5 0.2]
 [4.8 3.4 1.6 0.2]
 [4.8 3.  1.4 0.1]
 [4.3 3.  1.1 0.1]
 [5.8 4.  1.2 0.2]
 [5.7 4.4 1.5 0.4]
 [5.4 3.9 1.3 0.4]
 [5.1 3.5 1.4 0.3]
 [5.7 3.8 1.7 0.3]
 [5.1 3.8 1.5 0.3]
 [5.4 3.4 1.7 0.2]
 [5.1 3.7 1.5 0.4]
 [4.6 3.6 1.  0.2]
 [5.1 3.3 1.7 0.5]
 [4.8 3.4 1.9 0.2]
 [5.  3.  1.6 0.2]
 [5.  3.4 1.6 0.4]
 [5.2 3.5 1.5 0.2]
 [5.2 3.4 1.4 0.2]
 [4.7 3.2 1.6 0.2]
 [4.8 3.1 1.6 0.2]
 [5.4 3.4 1.5 0.4]
 [5.2 4.1 1.5 0.1]
 [5.5 4.2 1.4 0.2]
 [4.9 3.1 1.5 0.2]
 [5.  3.2 1.2 0.2]
 [5.5 3.5 1.3 0.2]
 [4.9 3.6 1.4 0.1]
 [4.4 3.  1.3 0.2]
 [5.1 3.4 1.5 0.2]
 [5.  3.5 1.3 0.3]
 [4.5 2.3 1.3 0.3]
 [4.4 3.2 1.3 0.2]
 [5.  3.5 1.6 0.6]
 [5.1 3.8 1.9 0.4]
 [4.8 3.  1.4 0.3]
 [5.1 3.8 1.6 0.2]
 [4.6 3.2 1.4 0.2]
 [5.3 3.7 1.5 0.2]
 [5.  3.3 1.4 0.2]
 [7.  3.2 4.7 1.4]
 [6.4 3.2 4.5 1.5]
 [6.9 3.1 4.9 1.5]
 [5.5 2.3 4.  1.3]
 [6.5 2.8 4.6 1.5]
 [5.7 2.8 4.5 1.3]
 [6.3 3.3 4.7 1.6]
 [4.9 2.4 3.3 1. ]
```

```
y = df.iloc[:, 4].values
print(y)
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

```
#train test split
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.1)
```

```
print(x_train)
```

```
[[6.3 3.3 6.  2.5]
 [4.4 2.9 1.4 0.2]
 [7.2 3.6 6.1 2.5]
 [6.1 2.9 4.7 1.4]
 [5.8 2.7 4.1 1. ]
 [6.3 3.3 4.7 1.6]
 [5.4 3.4 1.5 0.4]
 [6.8 3.  5.5 2.1]
 [6.5 3.2 5.1 2. ]
 [6.7 3.  5.  1.7]
 [4.8 3.  1.4 0.1]
 [6.6 2.9 4.6 1.3]
 [6.  3.4 4.5 1.6]
 [4.7 3.2 1.3 0.2]
 [6.7 2.5 5.8 1.8]
 [5.1 3.3 1.7 0.5]
 [7.7 3.8 6.7 2.2]
 [5.8 2.7 5.1 1.9]
 [5.  3.5 1.6 0.6]
 [5.  3.6 1.4 0.2]
 [5.6 2.7 4.2 1.3]
 [6.7 3.1 4.7 1.5]
 [6.  2.9 4.5 1.5]
 [5.9 3.2 4.8 1.8]
 [7.2 3.2 6.  1.8]
 [5.2 4.1 1.5 0.1]
 [6.5 2.8 4.6 1.5]
 [6.2 2.2 4.5 1.5]
 [6.1 2.8 4.  1.3]
 [6.7 3.  5.2 2.3]
 [6.9 3.1 4.9 1.5]
 [4.5 2.3 1.3 0.3]
 [7.7 2.6 6.9 2.3]
 [6.3 3.4 5.6 2.4]
 [6.7 3.1 5.6 2.4]
 [5.7 2.6 3.5 1. ]
 [5.9 3.  4.2 1.5]
 [5.5 2.5 4.  1.3]
 [5.2 3.4 1.4 0.2]
 [5.  3.3 1.4 0.2]
 [5.8 2.7 3.9 1.2]
 [5.4 3.7 1.5 0.2]
 [4.4 3.2 1.3 0.2]
 [5.9 3.  5.1 1.8]
 [4.9 3.  1.4 0.2]
 [6.3 2.5 4.9 1.5]
 [4.8 3.1 1.6 0.2]
 [5.7 2.8 4.1 1.3]
 [5.5 2.4 3.8 1.1]
 [7.2 3.  5.8 1.6]
 [5.3 3.7 1.5 0.2]
 [5.5 2.3 4.  1.3]
 [6.3 2.5 5.  1.9]
 [6.9 3.2 5.7 2.3]
 [5.1 2.5 3.  1.1]
 [4.6 3.6 1.  0.2]
 [4.8 3.  1.4 0.3]
 [4.9 2.4 3.3 1. ]
```

```
from sklearn.linear_model import LinearRegression
model=LinearRegression()
```

```
from sklearn.svm import SVC
model=SVC(kernel="linear")
```

```
model.fit(x_train, y_train)
y_predict = model.predict(x_test)
```

```
print(y_predict)
```

```
[1 2 1 0 1 0 0 0 2 0 2 1 1 1 2]
```

```
print(y_test)
```

```
[1 2 1 0 1 0 0 0 2 0 2 1 2 2 2]
```

Experiment:-6

Objective:- Data Preprocessing on Titanic Data

```
import pandas as pd
```

```
data = pd.read_csv('titanic-data.csv')
```

```
data.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500
				Cumings, Mrs. John Bradley (Florence	female	38.0	1	0	PC 17599	71.2833

```
data.dtypes
```

```
PassengerId      int64
Survived         int64
Pclass           int64
Name            object
Sex             object
Age            float64
SibSp            int64
Parch            int64
Ticket          object
Fare           float64
Cabin           object
Embarked        object
dtype: object
```

```
data.columns
```

```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

Explore the columns

Unsupported Cell Type. Double-Click to inspect/edit the content.

Passangers who survived vs not survived

```
data['Survived'].value_counts()
```

```
0    549
1    342
Name: Survived, dtype: int64
```

```
data['Survived']==0
```

```
0       True
1      False
2      False
3      False
4       True
       ...
886     True
887    False
888     True
889    False
890     True
Name: Survived, Length: 891, dtype: bool
```

```
print('Total number of passangers in the training data...', len(data))
print('Number of passangers  who survived...', len(data[data['Survived'] == 1]))
print("Number of passangers  who didn't survived...", len(data[data['Survived'] == 0]))
```

```
Total number of passangers in the training data... 891
Number of passangers  who survived... 342
Number of passangers  who didn't survived... 549
```

```
data['Sex'].value_counts()
```

```
male      577
female    314
Name: Sex, dtype: int64
```

What is the % of men and women who survived?

```
print('% of male who survived', 100*np.mean(data['Survived'][data['Sex']=='male']))
print('% of female who survived', 100*np.mean(data['Survived'][data['Sex']=='female']))
```

```
% of male who survived 18.890814558058924
% of female who survived 74.20382165605095
```

```
np.mean(data['Survived'][data['Sex']=='male'])
```

```
0.18890814558058924
```

what is the % of men and women who survived, and then by the same token with class and age?

```
data['Pclass'].value_counts()
```

```
3    491
1    216
2    184
Name: Pclass, dtype: int64
```

```
print('% of passengers who survived in first class', 100*np.mean(data['Survived'][data['Pclass'] == 1]))
print('% of passengers who survived in second class', 100*np.mean(data['Survived'][data['Pclass'] == 2]))
print('% of passengers who survived in third class', 100*np.mean(data['Survived'][data['Pclass'] == 3]))
```

```
% of passengers who survived in first class 62.96296296296296
% of passengers who survived in second class 47.28260869565217
% of passengers who survived in third class 24.236252545824847
```

```
#data[["Pclass","Survived"]].groupby(["Pclass"], as_index = False).mean()
data[["Pclass", "Survived"]].groupby(['Pclass']).mean()
```

	Survived
Pclass	
1	0.629630
2	0.472826
3	0.242363

Summary

```
data.shape
```

```
(891, 12)
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

```
data['Age'].value_counts()
```

```
24.00    30
22.00    27
18.00    26
19.00    25
30.00    25
         ..
55.50     1
70.50     1
66.00     1
23.50     1
0.42      1
Name: Age, Length: 88, dtype: int64
```

```
data['Cabin']
# You can see NA values here. We have to deal with them before trainig our model
```

```
0       NaN
1       C85
2       NaN
3      C123
4       NaN
       ...
886     NaN
887     B42
888     NaN
889    C148
890     NaN
Name: Cabin, Length: 891, dtype: object
```

Unsupported Cell Type. Double-Click to inspect/edit the content.

```
data['Sex']
```

```
0        male
1      female
2      female
3      female
4        male
        ...
886      male
887    female
888    female
```

```
889      male
890      male
Name: Sex, Length: 891, dtype: object
```

```
df2 = data.copy()
df2['Sex'] = data['Sex'].apply(lambda x: 1 if x == 'male' else 0)
df2['Sex']
```

```
0      1
1      0
2      0
3      0
4      1
      ..
886    1
887    0
888    0
889    1
890    1
Name: Sex, Length: 891, dtype: int64
```

def fun(x):

if x == 'male': return 1

else: return 0

Dealing with Missing Values

```
df2 = data.copy() #dataframe copy
```

```
df2.isnull().sum()
```

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

```
int(data['Age'].mean())
```

```
29
```

```
df2['Age'] = df2['Age'].fillna(np.mean(df2['Age']))
df2.isnull().sum()
```

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age              0
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

```
df2.Embarked.value_counts()
```

```
S    644
C    168
```

```
Q     77
Name: Embarked, dtype: int64
```

```
emabark = df2['Embarked'].dropna()
```

```
df2[df2['Embarked'].isnull()]
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
61	62	1	1	Icard, Miss. Amelie	0	38.0	0	0	113572	80.0

```
# while there can be many ways to deal NA values for this column
# we could have drop these NA values by dropping rows as data is less
# on the otther hand we can replace it with mode value
```

```
df2['Embarked'].mode()
```

```
0    S
dtype: object
```

```
df2['Embarked'].fillna(df2['Embarked'].mode()[0], inplace=True)
```

```
df2.isnull().sum()
```

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age              0
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         0
dtype: int64
```

```
df2['Cabin'].value_counts()
```

```
G6             4
B96 B98        4
C23 C25 C27    4
D              3
E101           3
              ..
B4             1
B102           1
A6             1
E10            1
A14            1
Name: Cabin, Length: 147, dtype: int64
```

```
df2['Cabin'].mode()
```

```
0        B96 B98
1    C23 C25 C27
2             G6
dtype: object
```

```
df2['Cabin'].fillna(df2['Cabin'].mode()[0], inplace=True)
```

```
df2.isnull().sum()
```

```
PassengerId    0
Survived       0
Pclass         0
Name           0
Sex            0
Age            0
SibSp          0
Parch          0
Ticket         0
Fare           0
Cabin          0
Embarked       0
dtype: int64
```

```
df2.corr()
```

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Pa
PassengerId	1.000000	-0.005007	-0.035144	0.042939	0.033207	-0.057527	-0.001
Survived	-0.005007	1.000000	-0.338481	-0.543351	-0.069809	-0.035322	0.081
Pclass	-0.035144	-0.338481	1.000000	0.131900	-0.331339	0.083081	0.018
Sex	0.042939	-0.543351	0.131900	1.000000	0.084153	-0.114631	-0.245
Age	0.033207	-0.069809	-0.331339	0.084153	1.000000	-0.232625	-0.179
SibSp	-0.057527	-0.035322	0.083081	-0.114631	-0.232625	1.000000	0.414
Parch	-0.001652	0.081629	0.018443	-0.245489	-0.179191	0.414838	1.000
Fare	0.012658	0.257307	-0.549500	-0.182333	0.091566	0.159651	0.216

Experiment:-7

Objective:- Decision Tree

```
from sklearn.datasets import load_breast_cancer
dataset = load_breast_cancer()
dataset
```


```
{'data': array([[1.799e+01, 1.038e+01, 1.228e+02, ..., 2.654e-01, 4.601e-01,
         1.189e-01],
        [2.057e+01, 1.777e+01, 1.329e+02, ..., 1.860e-01, 2.750e-01,
         8.902e-02],
        [1.969e+01, 2.125e+01, 1.300e+02, ..., 2.430e-01, 3.613e-01,
         8.758e-02],
        ...,
        [1.660e+01, 2.808e+01, 1.083e+02, ..., 1.418e-01, 2.218e-01,
         7.820e-02],
        [2.060e+01, 2.933e+01, 1.401e+02, ..., 2.650e-01, 4.087e-01,
         1.240e-01],
        [7.760e+00, 2.454e+01, 4.792e+01, ..., 0.000e+00, 2.871e-01,
         7.039e-02]]),
 'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
        0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0,
        1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,
        1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
        1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0,
        0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
        1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0,
        0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0,
        1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1,
        1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0,
        0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0,
        0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0,
        1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1,
        1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1,
        1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
        1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
        1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1,
        1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1]),
 'frame': None,
 'target_names': array(['malignant', 'benign'], dtype='<U9'),
 'DESCR': '.. _breast_cancer_dataset:\n\nBreast cancer wisconsin (diagnostic) dataset\n--------------------------------------------
\n\n**Data Set Characteristics:**\n\n    :Number of Instances: 569\n\n    :Number of Attributes: 30 numeric, predictive attributes
and the class\n\n    :Attribute Information:\n        - radius (mean of distances from center to points on the perimeter)\n        -
texture (standard deviation of gray-scale values)\n        - perimeter\n        - area\n        - smoothness (local variation in
radius lengths)\n        - compactness (perimeter^2 / area - 1.0)\n        - concavity (severity of concave portions of the
contour)\n        - concave points (number of concave portions of the contour)\n        - symmetry\n        - fractal dimension
("coastline approximation" - 1)\n\n        The mean, standard error, and "worst" or largest (mean of the three\n        worst/largest
values) of these features were computed for each image,\n        resulting in 30 features.  For instance, field 0 is Mean Radius,
field\n        10 is Radius SE, field 20 is Worst Radius.\n\n        - class:\n                - WDBC-Malignant\n                -
WDBC-Benign\n\n    :Summary Statistics:\n\n    ===================================== ====== ======\n
Min    Max\n    ===================================== ====== ======\n    radius (mean):                        6.981  28.11\n
texture (mean):                       9.71   39.28\n    perimeter (mean):                     43.79  188.5\n    area (mean):
143.5  2501.0\n    smoothness (mean):                    0.053  0.163\n    compactness (mean):                   0.019  0.345\n
concavity (mean):                     0.0    0.427\n    concave points (mean):                0.0    0.201\n    symmetry (mean):
0.106  0.304\n    fractal dimension (mean):             0.05   0.097\n    radius (standard error):              0.112  2.873\n
texture (standard error):             0.36   4.885\n    perimeter (standard error):           0.757  21.98\n    area (standard
error):                  6.802  542.2\n    smoothness (standard error):          0.002  0.031\n    compactness (standard error):
```


```
import pandas as pd
import numpy as np
data1 = pd.DataFrame(data= np.c_[dataset['data']],columns= dataset['feature_names'])
```

```
data1
```

	radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mea concav point
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.1471
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.0701
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.1279
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.1052
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.1043
...	...	...	...	...	...	...	...	
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.1389
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.0979
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.0530
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.1520
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.0000

569 rows × 30 columns

```
y = dataset.target
y
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0,
       1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,
       1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0,
       0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
       1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0,
       0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0,
       0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0,
       0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1,
       1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1,
       1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1])
```

```
x = data1
```

```
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)
print(x_train.shape,y_train.shape,y_test.shape,x_test.shape)
```

```
(398, 30) (398,) (171,) (171, 30)
```

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

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

```
from sklearn import tree
from matplotlib import pyplot as plt
feature_names = x.columns
```

```
fig = plt.figure(figsize = (15, 10))
_ = tree.plot_tree(clf, feature_names = feature_names, class_names={0:'maligant', 1:'benign'}, filled = True, fontsize = 12)
```