```
import numpy as np
import pandas as pd

pr1=pd.read_csv('/content/KNNAlgorithmDataset.csv')
pr1
```

1 to 10 of 569 entries Filter

index	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothr
0	842302	M	17.99	10.38	122.8	1001.0	
1	842517	M	20.57	17.77	132.9	1326.0	
2	84300903	M	19.69	21.25	130.0	1203.0	
3	84348301	M	11.42	20.38	77.58	386.1	
4	84358402	M	20.29	14.34	135.1	1297.0	
5	843786	M	12.45	15.7	82.57	477.1	
6	844359	M	18.25	19.98	119.6	1040.0	
7	84458202	M	13.71	20.83	90.2	577.9	
8	844981	M	13.0	21.82	87.5	519.8	
9	84501001	M	12.46	24.04	83.97	475.9	

Show 10 per page 1 2 10 50 57

Like what you see? Visit the data table notebook to learn more about interactive tables.

```
Warning: Total number of columns (33) exceeds max_columns (20) limiting to
```

```
pr1.describe()
```

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_m
count	569.000000	569.000000	569.000000	569.000000	569.000
mean	14.127292	19.289649	91.969033	654.889104	0.096
std	3.524049	4.301036	24.298981	351.914129	0.014
min	6.981000	9.710000	43.790000	143.500000	0.052
25%	11.700000	16.170000	75.170000	420.300000	0.086
50%	13.370000	18.840000	86.240000	551.100000	0.095
75%	15.780000	21.800000	104.100000	782.700000	0.105
max	28.110000	39.280000	188.500000	2501.000000	0.163

8 rows × 30 columns

```
pr1.isna().sum()
```

✓ 0s completed at 7:47 PM

```
diagnosis                  0
radius_mean                0
texture_mean               0
perimeter_mean             0
area_mean                  0
smoothness_mean            0
compactness_mean           0
concavity_mean             0
concave points_mean        0
symmetry_mean              0
fractal_dimension_mean     0
radius_se                  0
texture_se                 0
perimeter_se               0
area_se                    0
smoothness_se              0
compactness_se             0
concavity_se               0
concave points_se          0
symmetry_se                0
fractal_dimension_se       0
radius_worst               0
texture_worst              0
perimeter_worst            0
area_worst                 0
smoothness_worst           0
compactness_worst          0
concavity_worst            0
concave points_worst       0
symmetry_worst             0
fractal_dimension_worst    0
Unnamed: 32              569
dtype: int64
```

```
pr1.dtypes
```

```
id                           int64
diagnosis                   object
radius_mean                float64
texture_mean               float64
perimeter_mean             float64
area_mean                  float64
smoothness_mean            float64
compactness_mean           float64
concavity_mean             float64
concave points_mean        float64
symmetry_mean              float64
fractal_dimension_mean     float64
radius_se                  float64
texture_se                 float64
perimeter_se               float64
area_se                    float64
smoothness_se              float64
compactness_se             float64
concavity_se               float64
concave points_se          float64
symmetry_se                float64
fractal_dimension_se       float64
```

```
fractal_dimension_se       float64
radius_worst               float64
texture_worst              float64
perimeter_worst            float64
area_worst                 float64
smoothness_worst           float64
compactness_worst          float64
concavity_worst            float64
concave points_worst       float64
symmetry_worst             float64
fractal_dimension_worst    float64
Unnamed: 32                float64
dtype: object
```

```
# checking balanced or not
pr1['diagnosis'].value_counts()
```

```
B    357
M    212
Name: diagnosis, dtype: int64
```

```
pr1.drop_duplicates(inplace=True)
```

```
pr1.drop(['id','Unnamed: 32'],axis=1,inplace=True)
```

```
# seperate input and output
inp=pr1.iloc[:,1:].values
oup=pr1.iloc[:,0].values
```

```
# train and test splitting
from sklearn.model_selection import train_test_split

x_train,x_test,y_train,y_test=train_test_split(inp,oup,test_size=0.30)
```

```
# normalizaation
from sklearn.preprocessing import StandardScaler

std=StandardScaler()
std.fit(x_train)
x_train=std.transform(x_train)
x_test=std.transform(x_test)
```

```
# model creation

from sklearn.neighbors import KNeighborsClassifier

knn=KNeighborsClassifier(n_neighbors=7)
knn.fit(x_train,y_train)
y_pred=knn.predict(x_test)
```

```
#
```

```
# evaluation

from sklearn.metrics import confusion_matrix,accuracy_score,classification_repor

result=confusion_matrix(y_test,y_pred)
result
```

```
array([[101,   0],
       [  8,  62]])
```

```
score=accuracy_score(y_test,y_pred)
score
```

```
0.9532163742690059
```

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

```
                  precision    recall  f1-score   support

               B       0.93      1.00      0.96       101
               M       1.00      0.89      0.94        70

        accuracy                           0.95       171
       macro avg       0.96      0.94      0.95       171
    weighted avg       0.96      0.95      0.95       171
```