Import Library

```
import pandas as pd
import numpy as np
from matplotlib import pyplot
```

Choose Dataset from Local Directory

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

Choose Files data.csv

- **data.csv**(text/csv) - 125141 bytes, last modified: 3/28/2023 - 100% done

```
Saving data.csv to data.csv
```

Load Dataset

```
dataset = pd.read_csv('data.csv')
dataset
```

	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_me
0	842302	17.99	10.38	122.80	1001.0	0.118
1	842517	20.57	17.77	132.90	1326.0	0.084
2	84300903	19.69	21.25	130.00	1203.0	0.109
3	84348301	11.42	20.38	77.58	386.1	0.142
4	84358402	20.29	14.34	135.10	1297.0	0.100
...	...	...	...	...	...	
564	926424	21.56	22.39	142.00	1479.0	0.111
565	926682	20.13	28.25	131.20	1261.0	0.097
566	926954	16.60	28.08	108.30	858.1	0.084
567	927241	20.60	29.33	140.10	1265.0	0.117
568	92751	7.76	24.54	47.92	181.0	0.052

569 rows × 32 columns

Summarize Dataset

```
print(dataset.shape)
print(dataset.head(5))
```

```
(569, 32)
         id  radius_mean  texture_mean  perimeter_mean  area_mean  \
0    842302        17.99         10.38          122.80     1001.0
1    842517        20.57         17.77          132.90     1326.0
2  84300903        19.69         21.25          130.00     1203.0
3  84348301        11.42         20.38           77.58      386.1
4  84358402        20.29         14.34          135.10     1297.0

   smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
0          0.11840           0.27760          0.3001              0.14710
1          0.08474           0.07864          0.0869              0.07017
2          0.10960           0.15990          0.1974              0.12790
3          0.14250           0.28390          0.2414              0.10520
4          0.10030           0.13280          0.1980              0.10430

   symmetry_mean  ...  texture_worst  perimeter_worst  area_worst  \
0         0.2419  ...          17.33           184.60      2019.0
1         0.1812  ...          23.41           158.80      1956.0
2         0.2069  ...          25.53           152.50      1709.0
3         0.2597  ...          26.50            98.87       567.7
4         0.1809  ...          16.67           152.20      1575.0

   smoothness_worst  compactness_worst  concavity_worst  concave points_worst  \
0            0.1622             0.6656           0.7119                0.2654
1            0.1238             0.1866           0.2416                0.1860
2            0.1444             0.4245           0.4504                0.2430
3            0.2098             0.8663           0.6869                0.2575
4            0.1374             0.2050           0.4000                0.1625

   symmetry_worst  fractal_dimension_worst  diagnosis
0          0.4601                  0.11890          M
1          0.2750                  0.08902          M
2          0.3613                  0.08758          M
3          0.6638                  0.17300          M
```

```
4         0.2364                  0.07678         M

[5 rows x 32 columns]
```

Mapping Class String Values to Numbers

```
dataset['diagnosis'] = dataset['diagnosis'].map({'B': 0, 'M': 1}).astype(int)
print(dataset.head)
```

```
565    926682        20.13         28.25          131.20     1261.0
566    926954        16.60         28.08          108.30      858.1
567    927241        20.60         29.33          140.10     1265.0
568     92751         7.76         24.54           47.92      181.0

     smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
0            0.11840           0.27760         0.30010              0.14710
1            0.08474           0.07864         0.08690              0.07017
2            0.10960           0.15990         0.19740              0.12790
3            0.14250           0.28390         0.24140              0.10520
4            0.10030           0.13280         0.19800              0.10430
..               ...               ...             ...                  ...
564          0.11100           0.11590         0.24390              0.13890
565          0.09780           0.10340         0.14400              0.09791
566          0.08455           0.10230         0.09251              0.05302
567          0.11780           0.27700         0.35140              0.15200
568          0.05263           0.04362         0.00000              0.00000

     symmetry_mean  ...  texture_worst  perimeter_worst  area_worst  \
0           0.2419  ...          17.33           184.60      2019.0
1           0.1812  ...          23.41           158.80      1956.0
2           0.2069  ...          25.53           152.50      1709.0
3           0.2597  ...          26.50            98.87       567.7
4           0.1809  ...          16.67           152.20      1575.0
..             ...  ...            ...              ...         ...
564         0.1726  ...          26.40           166.10      2027.0
565         0.1752  ...          38.25           155.00      1731.0
566         0.1590  ...          34.12           126.70      1124.0
567         0.2397  ...          39.42           184.60      1821.0
568         0.1587  ...          30.37            59.16       268.6

     smoothness_worst  compactness_worst  concavity_worst  \
0             0.16220            0.66560           0.7119
1             0.12380            0.18660           0.2416
2             0.14440            0.42450           0.4504
3             0.20980            0.86630           0.6869
4             0.13740            0.20500           0.4000
..                ...                ...              ...
564           0.14100            0.21130           0.4107
565           0.11660            0.19220           0.3215
566           0.11390            0.30940           0.3403
567           0.16500            0.86810           0.9387
568           0.08996            0.06444           0.0000

     concave points_worst  symmetry_worst  fractional_dimension_worst  diagnosis
0                  0.2654          0.4601                  0.11890          1
1                  0.1860          0.2750                  0.08902          1
2                  0.2430          0.3613                  0.08758          1
3                  0.2575          0.6638                  0.17300          1
4                  0.1625          0.2364                  0.07678          1
..                    ...             ...                      ...        ...
564                0.2216          0.2060                  0.07115          1
565                0.1628          0.2572                  0.06637          1
566                0.1418          0.2218                  0.07820          1
567                0.2650          0.4087                  0.12400          1
568                0.0000          0.2871                  0.07039          0

[569 rows x 32 columns]>
```

Segregate Dataset into X & Y

```
X = dataset.iloc[:, 1:31].values
X
```

```
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]])
```

```
Y = dataset.iloc[:, -1].values
Y
```

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1,
```

```
       0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1,
       0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1,
       1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0,
       0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1,
       1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1,
       1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1,
       1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0,
       0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0,
       0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 1, 1, 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, 0])
```

Splitting Dataset into Train and Test

```
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)
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
print(X_train)
print(X_test)
```

```
[[-0.65079907 -0.43057322 -0.68024847 ... -0.36433881  0.32349851
  -0.7578486 ]
 [-0.82835341  0.15226547 -0.82773762 ... -1.45036679  0.62563098
  -1.03071387]
 [ 1.68277234  2.18977235  1.60009756 ...  0.72504581 -0.51329768
  -0.96601386]
 ...
 [-1.33114223 -0.22172269 -1.3242844  ... -0.98806491 -0.69995543
  -0.12266325]
 [-1.25110186 -0.24600763 -1.28700242 ... -1.75887319 -1.56206114
  -1.00989735]
 [-0.74662205  1.14066273 -0.72203706 ... -0.2860679  -1.24094654
   0.2126516 ]]
[[-0.21395901  0.3125461  -0.14355187 ...  1.37043754  1.08911166
   1.53928319]
 [-0.26750714  1.461224   -0.32955207 ... -0.84266106 -0.71577388
  -0.88105993]
 [-0.03922298 -0.86770223 -0.10463112 ... -0.505318   -1.20298225
  -0.92494342]
 ...
 [-0.51270124 -1.69096186 -0.54095317 ... -0.12632201  0.33773512
  -0.42872244]
 [-0.17732081 -2.01395163 -0.17345939 ... -0.62875108 -0.29500302
  -0.65432858]
 [ 1.5305829  -0.26300709  1.57961296 ...  1.6694843   1.18085869
   0.48889253]]
```

Validating some ML algorithm by its accuracy - Model Score

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC

from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
```

```
models = []
models.append(('LR', LogisticRegression(solver='liblinear', multi_class='ovr')))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC(gamma='auto')))
print(models)
```

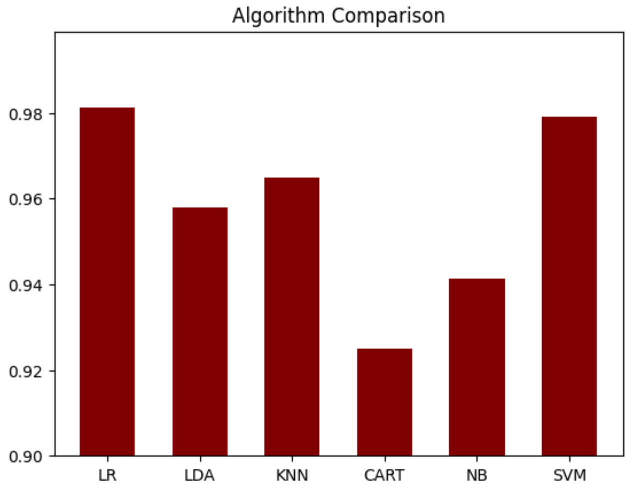
```
[('LR', LogisticRegression(multi_class='ovr', solver='liblinear')), ('LDA', LinearDiscriminantAnalysis()), ('KNN', KNeighborsClassifier()), ('CART', DecisionTr
```

```
results = []
names = []
res = []
for name, model in models:
  kfold = StratifiedKFold(n_splits=10, random_state=None)
  cv_results = cross_val_score(model, X_train, y_train, cv=kfold, scoring='accuracy')
  results.append(cv_results)
  names.append(name)
  res.append(cv_results.mean())
  print('%s: %f' % (name, cv_results.mean()))

pyplot.ylim(.900, .999)
pyplot.bar(names, res, color='maroon', width=0.6)

pyplot.title('Algorithm Comparison')
pyplot.show()
```

```
LR: 0.981285
LDA: 0.957863
KNN: 0.964839
CART: 0.924917
NB: 0.941417
SVM: 0.979014
```



Training & Prediction using the algorithm with high accuracy

```
model = LogisticRegression(solver='liblinear', multi_class='ovr')
model.fit(X_train, y_train)
value = [[17.99, 10.38, 122.8, 1001, 0.1184, 0.2776, 0.3801, 0.1471, 0.2419, 0.07871, 1.095, 0.999,
          8.604, 100.4, 0.0023, 17.99, 10.38, 122.8, 1001, 0.1184, 0.2776, 0.3801, 0.1471, 0.2419, 0.07871, 1.095, 0.999, 8.604, 100.4, 0.0023]]
y_pred = model.predict(value)
print(y_pred)
```

```
[1]
```