Name - Nihalahmed Munir Barudwale

Project name - Prediction of Breast cancer for a given patient

```
#

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

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

Import Required Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import os
import warnings
warnings.filterwarnings('ignore')
from scipy.stats import norm
```

Import dataset

```
dataset = pd.read_csv("/content/drive/MyDrive/TCR Internship
Project/data.csv")
```

```
dataset
```

```
           id diagnosis  ...  fractal_dimension_worst  Unnamed: 32
0      842302         M  ...                  0.11890          NaN
1      842517         M  ...                  0.08902          NaN
2    84300903         M  ...                  0.08758          NaN
3    84348301         M  ...                  0.17300          NaN
4    84358402         M  ...                  0.07678          NaN
..        ...       ...  ...                      ...          ...
564    926424         M  ...                  0.07115          NaN
565    926682         M  ...                  0.06637          NaN
566    926954         M  ...                  0.07820          NaN
567    927241         M  ...                  0.12400          NaN
568     92751         B  ...                  0.07039          NaN

[569 rows x 33 columns]
```

Shape of dataset

```
dataset.shape
```

```
(569, 33)
```

Some Operations on dataset

```
dataset.head()
```

```
         id diagnosis  ...  fractal_dimension_worst  Unnamed: 32
0    842302         M  ...                  0.11890          NaN
1    842517         M  ...                  0.08902          NaN
2  84300903         M  ...                  0.08758          NaN
3  84348301         M  ...                  0.17300          NaN
4  84358402         M  ...                  0.07678          NaN

[5 rows x 33 columns]
```

```
dataset.tail()
```

```
         id diagnosis  ...  fractal_dimension_worst  Unnamed: 32
564  926424         M  ...                  0.07115          NaN
565  926682         M  ...                  0.06637          NaN
566  926954         M  ...                  0.07820          NaN
567  927241         M  ...                  0.12400          NaN
568   92751         B  ...                  0.07039          NaN

[5 rows x 33 columns]
```

```
type(dataset)
```

```
pandas.core.frame.DataFrame
```

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   id                       569 non-null    int64
 1   diagnosis                569 non-null    object
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
```

```
 18  concavity_se             569 non-null    float64
 19  concave points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_se     569 non-null    float64
 22  radius_worst             569 non-null    float64
 23  texture_worst            569 non-null    float64
 24  perimeter_worst          569 non-null    float64
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_worst        569 non-null    float64
 28  concavity_worst          569 non-null    float64
 29  concave points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
 32  Unnamed: 32              0 non-null      float64
dtypes: float64(31), int64(1), object(1)
memory usage: 146.8+ KB
```

```
dataset.describe()
```

```
                 id  radius_mean  ...  fractal_dimension_worst
Unnamed: 32
count  5.690000e+02   569.000000  ...               569.000000
0.0
mean   3.037183e+07    14.127292  ...                 0.083946
NaN
std    1.250206e+08     3.524049  ...                 0.018061
NaN
min    8.670000e+03     6.981000  ...                 0.055040
NaN
25%    8.692180e+05    11.700000  ...                 0.071460
NaN
50%    9.060240e+05    13.370000  ...                 0.080040
NaN
75%    8.813129e+06    15.780000  ...                 0.092080
NaN
max    9.113205e+08    28.110000  ...                 0.207500
NaN

[8 rows x 32 columns]
```

```
dataset.columns
```

```
Index(['id', 'diagnosis', 'radius_mean', 'texture_mean',
'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean',
'concavity_mean',
       'concave points_mean', 'symmetry_mean',
'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se',
'smoothness_se',
```

```
       'compactness_se', 'concavity_se', 'concave points_se',
'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst', 'Unnamed: 32'],
      dtype='object')
```

```
dataset.isna().sum()
```

```
id                           0
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
```

```
dataset.isnull().sum()
```

```
id                           0
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
```

Data Preprocessing

Deleting 'Unnamed:32' column which is of no use to us and it contains only null values.

```
del dataset['Unnamed: 32']
```

```
dataset.shape
```

```
(569, 32)
```

```
dataset.isna().sum()
```

```
id                         0
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
dtype: int64
```

```
dataset.diagnosis.unique()
```

```
array(['M', 'B'], dtype=object)
```

From the results above, diagnosis is a categorical variable, because it represents a fix number of possible values (i.e, Malignant, of Benign. The machine learning algorithms wants numbers, and not strings, as their inputs so we need to convert them into numbers.

```
dataset['diagnosis'] = dataset['diagnosis'].apply(lambda x : '1' if x
== 'M' else '0')
dataset = dataset.set_index('id')
```

```
dataset.skew()
```

```
diagnosis                  0.528461
radius_mean                0.942380
texture_mean               0.650450
perimeter_mean             0.990650
area_mean                  1.645732
smoothness_mean            0.456324
compactness_mean           1.190123
concavity_mean             1.401180
concave points_mean        1.171180
symmetry_mean              0.725609
fractal_dimension_mean     1.304489
radius_se                  3.088612
```

```
texture_se                 1.646444
perimeter_se               3.443615
area_se                    5.447186
smoothness_se              2.314450
compactness_se             1.902221
concavity_se               5.110463
concave points_se          1.444678
symmetry_se                2.195133
fractal_dimension_se       3.923969
radius_worst               1.103115
texture_worst              0.498321
perimeter_worst            1.128164
area_worst                 1.859373
smoothness_worst           0.415426
compactness_worst          1.473555
concavity_worst            1.150237
concave points_worst       0.492616
symmetry_worst             1.433928
fractal_dimension_worst    1.662579
dtype: float64
```

The skew result show a positive (right) or negative (left) skew. Values closer to zero show less skew. From the graphs, we can see that radius_mean, perimeter_mean, area_mean, concavity_mean and concave_points_mean are useful in predicting cancer type due to the distinct grouping between malignant and benign cancer types in these features. We can also see that area_worst and perimeter_worst are also quite useful.

#

Exploratory Data Analysis (EDA)

Analysing the 'diagnosis' variable

```
dataset.diagnosis.describe()
```

```
count     569
unique      2
top         0
freq      357
Name: diagnosis, dtype: object
```

```
dataset.diagnosis.unique()
```

```
array(['1', '0'], dtype=object)
```

```
dataset.diagnosis.value_counts()
```

```
0    357
1    212
Name: diagnosis, dtype: int64
```

1= Malignant (Cancerous) - Present

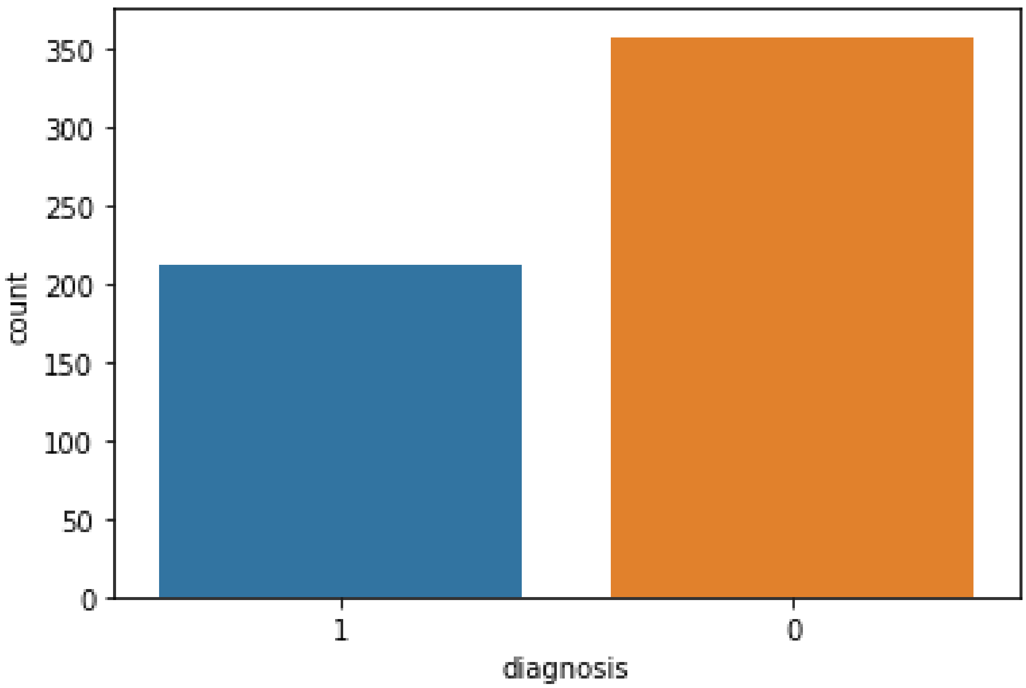
0= Benign (Not Cancerous) -Absent

```
print("Percentage of patients which have breast cancer: "+str(round(212*100/569,2)))
print("Percentage of patient which does not have breast cancer: "+str(round(357*100/569,2)))
```

```
Percentage of patients which have breast cancer: 37.26
Percentage of patient which does not have breast cancer: 62.74
```

```
y = dataset["diagnosis"]
sns.countplot(y)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f18ac644850>
```

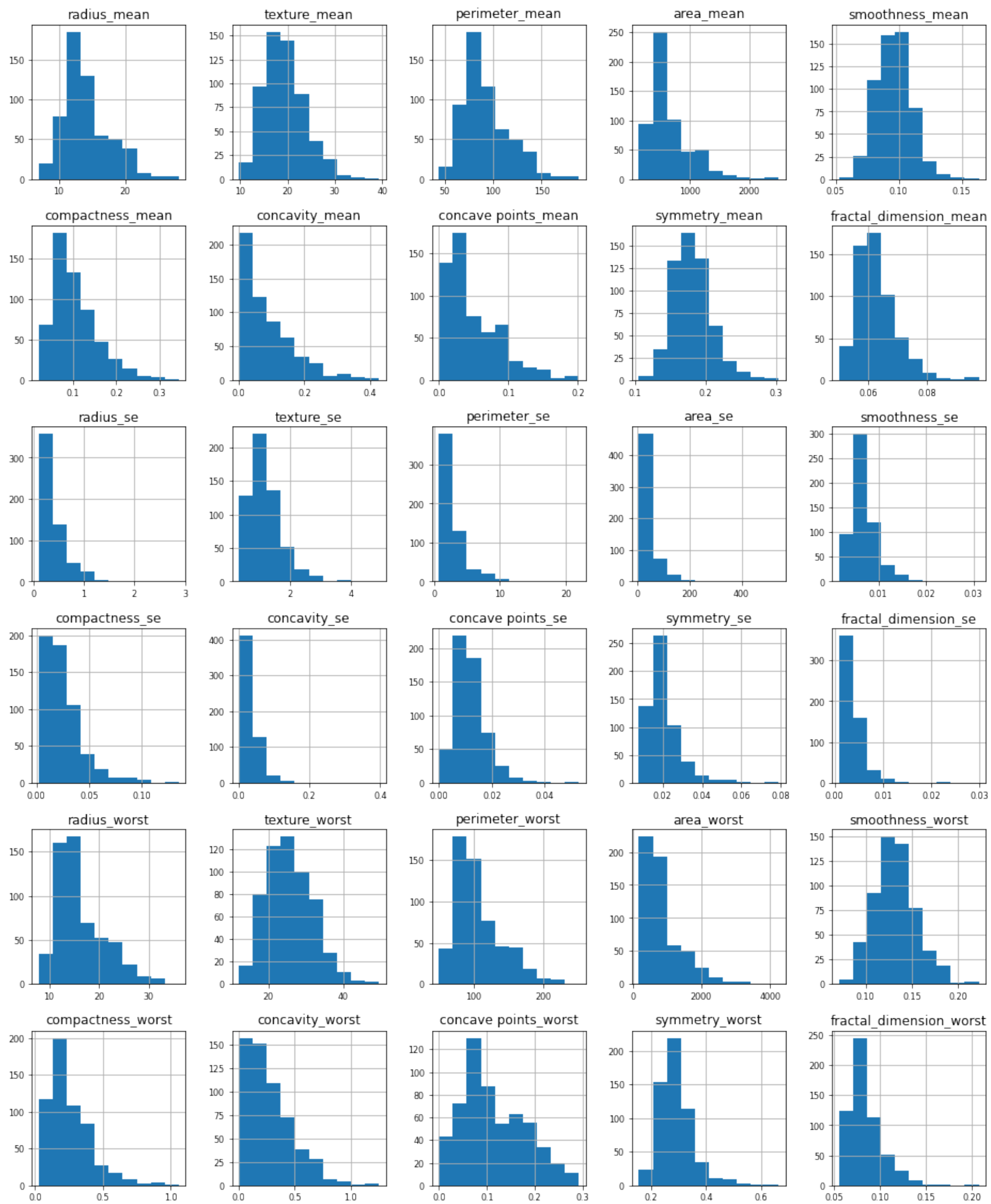


Get an overview distribution of each column

```
dataset.hist(figsize=(16, 20), xlabelsize=8, ylabelsize=8)
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f18ab0fbe90>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f18ab0b4450>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f18ab069a50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f18ab097b50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f18aafde6d0>],
       [<matplotlib.axes._subplots.AxesSubplot object at
```

```
0x7f18ab013d10>,
         <matplotlib.axes._subplots.AxesSubplot object at
0x7f18aafd63d0>,
         <matplotlib.axes._subplots.AxesSubplot object at
0x7f18aaf8c8d0>,
         <matplotlib.axes._subplots.AxesSubplot object at
0x7f18aaf8c910>,
         <matplotlib.axes._subplots.AxesSubplot object at
0x7f18aaf50090>],
        [<matplotlib.axes._subplots.AxesSubplot object at
0x7f18aaebbb90>,
         <matplotlib.axes._subplots.AxesSubplot object at
0x7f18aae7d1d0>,
         <matplotlib.axes._subplots.AxesSubplot object at
0x7f18aae337d0>,
         <matplotlib.axes._subplots.AxesSubplot object at
0x7f18aadeadd0>,
         <matplotlib.axes._subplots.AxesSubplot object at
0x7f18aadaf410>],
        [<matplotlib.axes._subplots.AxesSubplot object at
0x7f18aad66a10>,
         <matplotlib.axes._subplots.AxesSubplot object at
0x7f18aad1cfd0>,
         <matplotlib.axes._subplots.AxesSubplot object at
0x7f18aacdf610>,
         <matplotlib.axes._subplots.AxesSubplot object at
0x7f18aad13c50>,
         <matplotlib.axes._subplots.AxesSubplot object at
0x7f18aacd7290>],
        [<matplotlib.axes._subplots.AxesSubplot object at
0x7f18aac8c890>,
         <matplotlib.axes._subplots.AxesSubplot object at
0x7f18aac43e90>,
         <matplotlib.axes._subplots.AxesSubplot object at
0x7f18aac074d0>,
         <matplotlib.axes._subplots.AxesSubplot object at
0x7f18aabbcad0>,
         <matplotlib.axes._subplots.AxesSubplot object at
0x7f18aab80110>],
        [<matplotlib.axes._subplots.AxesSubplot object at
0x7f18aab35710>,
         <matplotlib.axes._subplots.AxesSubplot object at
0x7f18aaaecd10>,
         <matplotlib.axes._subplots.AxesSubplot object at
0x7f18aaab0350>,
         <matplotlib.axes._subplots.AxesSubplot object at
0x7f18aaa65950>,
         <matplotlib.axes._subplots.AxesSubplot object at
0x7f18aaa1df50>]],
       dtype=object)
```



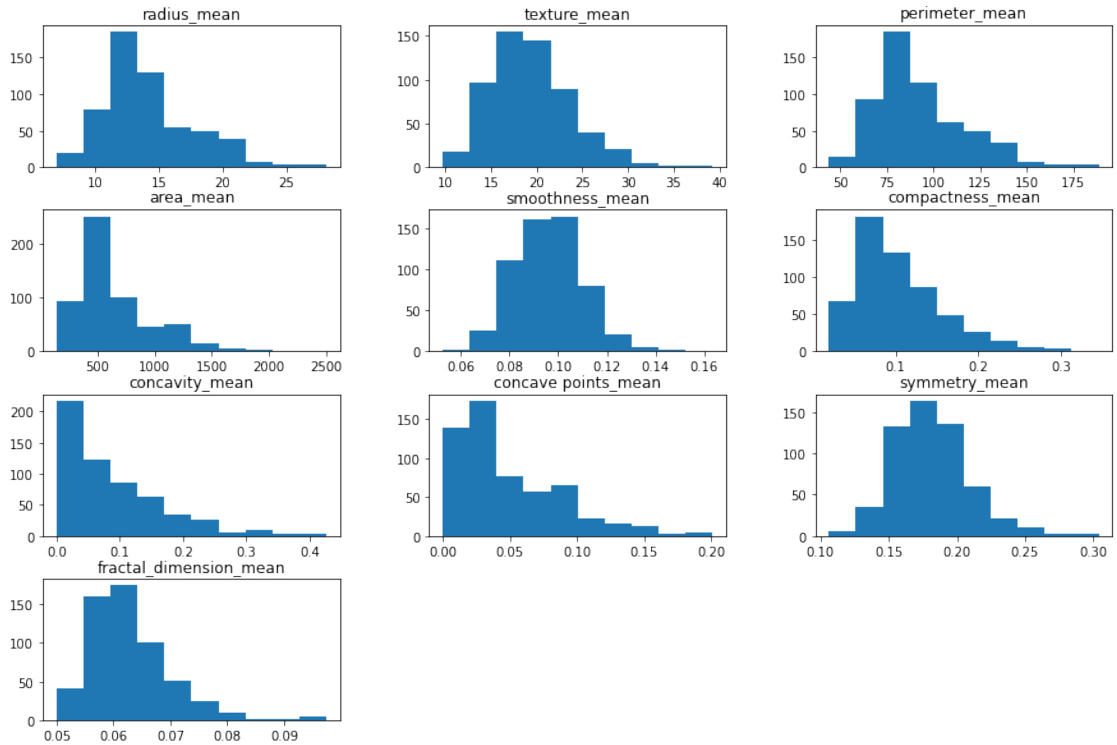
Separate columns into smaller dataframes to perform visualization

```
data_id_diag=dataset.reindex(columns=['id', 'diagnosis'])
data_diag=dataset.reindex(columns=['diagnosis'])

#For a merge + slice:
data_mean=dataset.iloc[:,1:11]
data_se=dataset.iloc[:,11:22]
data_worst=dataset.iloc[:,23:]
```

Visualise distribution of data via histograms

1.Histogram the "_mean" suffix designition

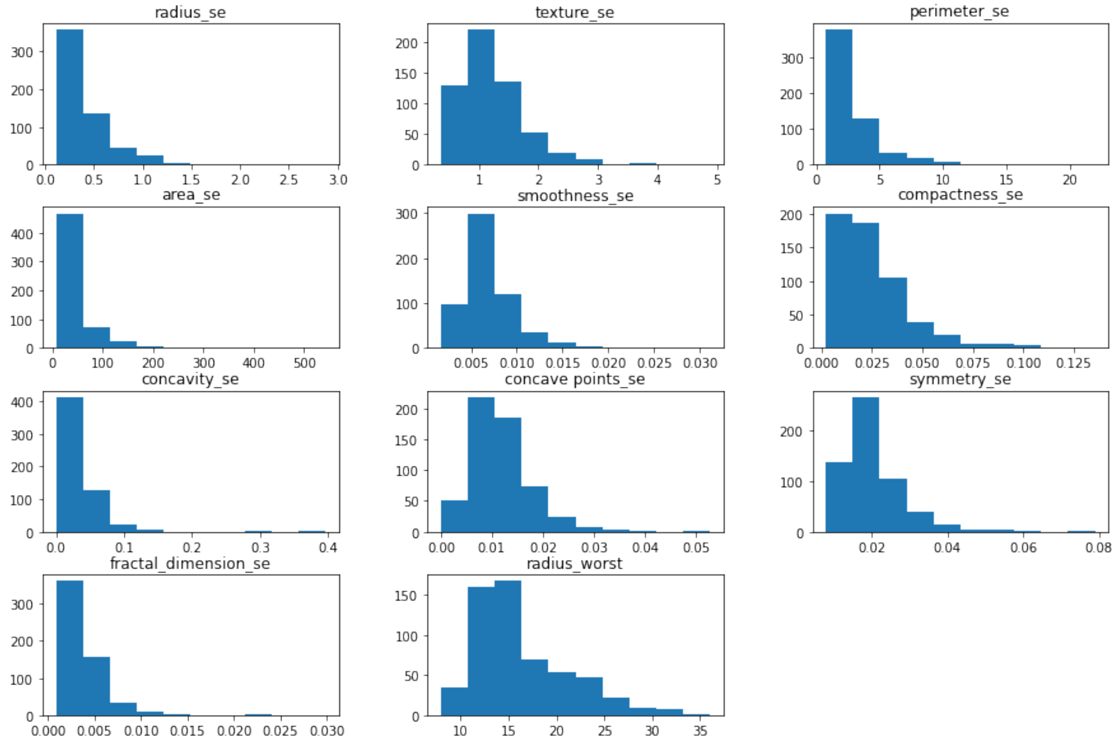
```
hist_mean=data_mean.hist(bins=10, figsize=(15, 10),grid=False,)
```



#

2.Histogram for the "_se" suffix designition

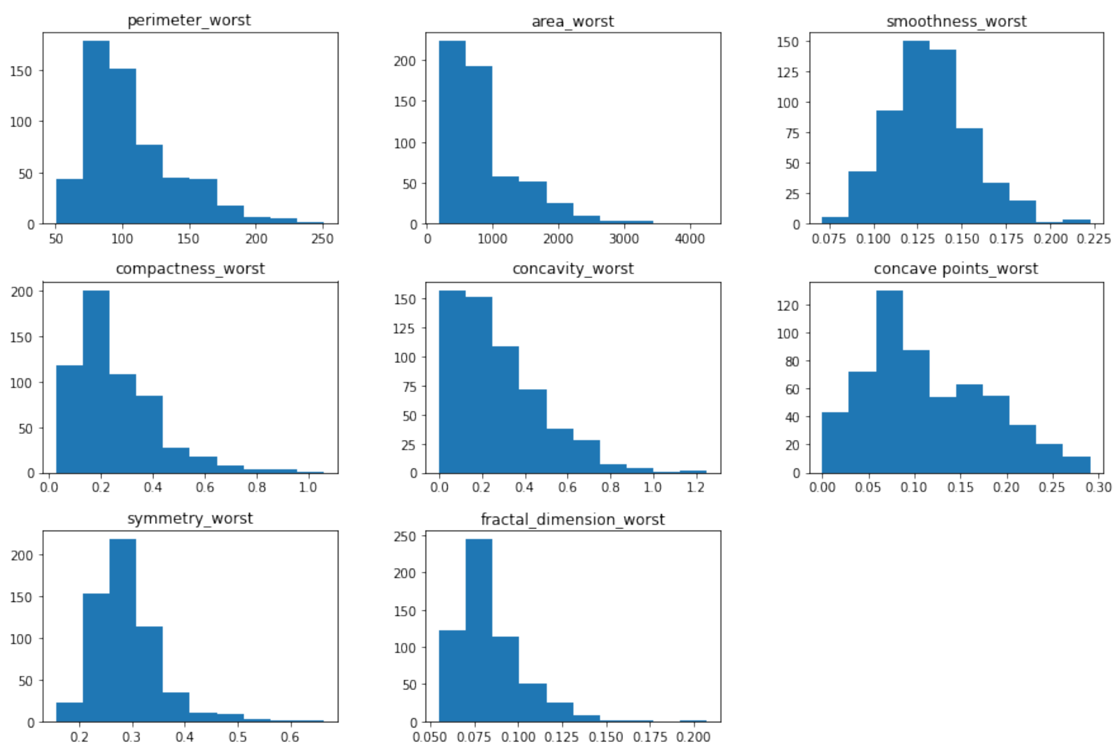
```
hist_se=data_se.hist(bins=10, figsize=(15, 10),grid=False,)
```



#

3.Histogram "_worst" suffix designition

```
hist_worst=data_worst.hist(bins=10, figsize=(15, 10),grid=False,)
```

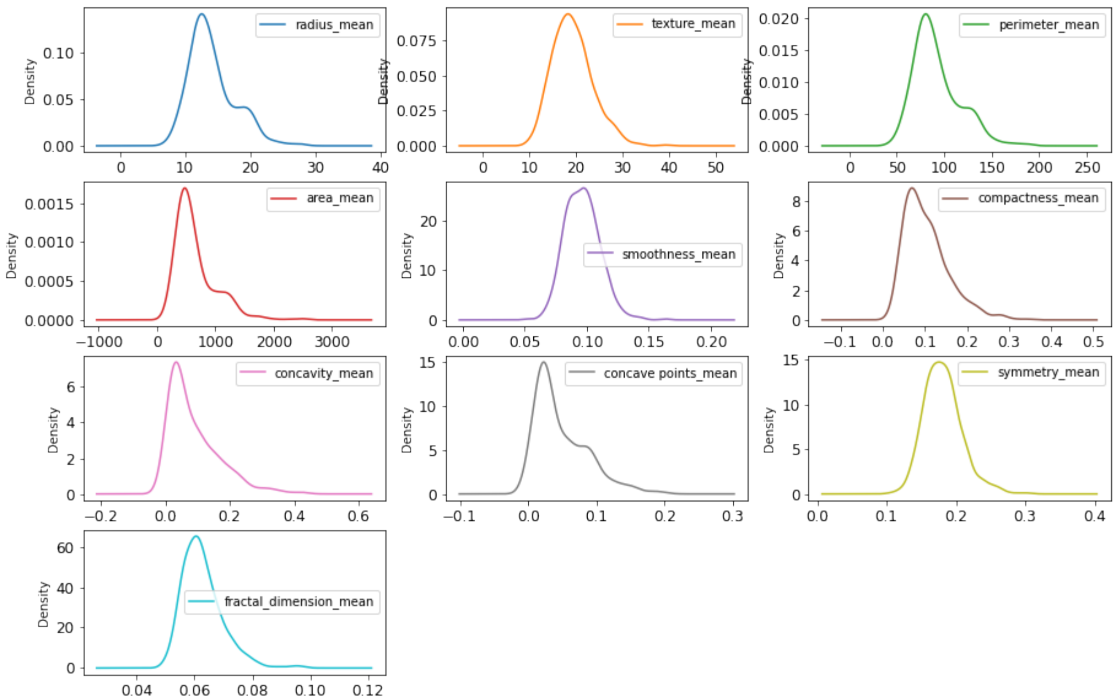


#

Visualise distribution of data via Density plots

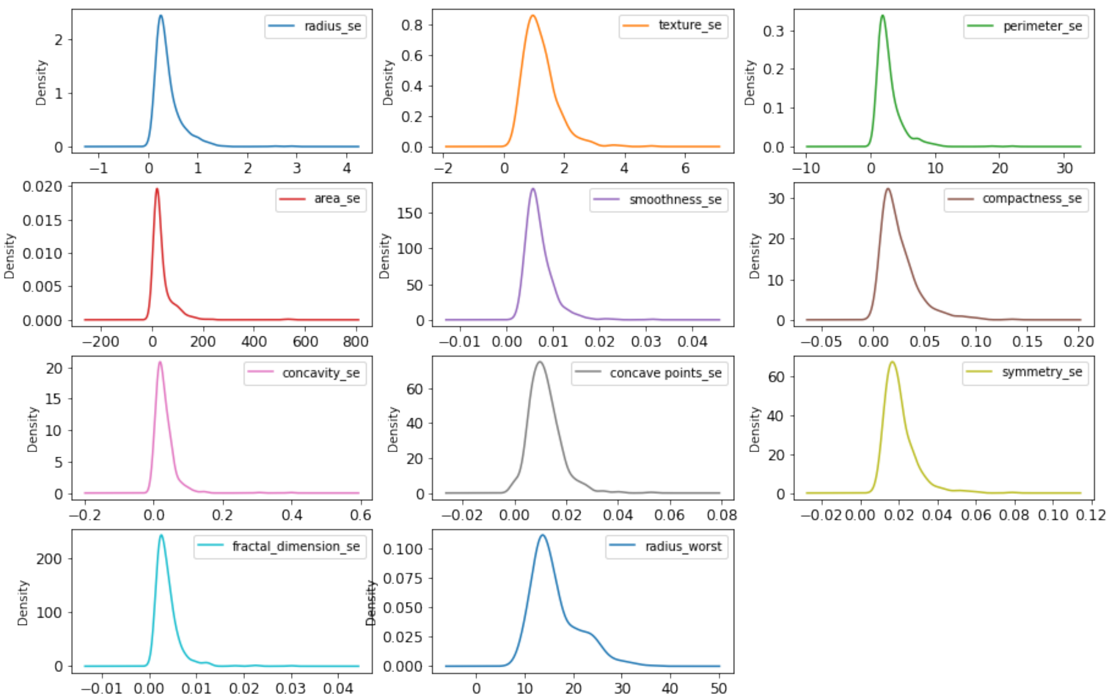
1.Density plots "_mean" suffix designition

```
plt = data_mean.plot(kind= 'density', subplots=True, layout=(4,3),
sharex=False,
                     sharey=False,fontsize=12, figsize=(15,10))
```



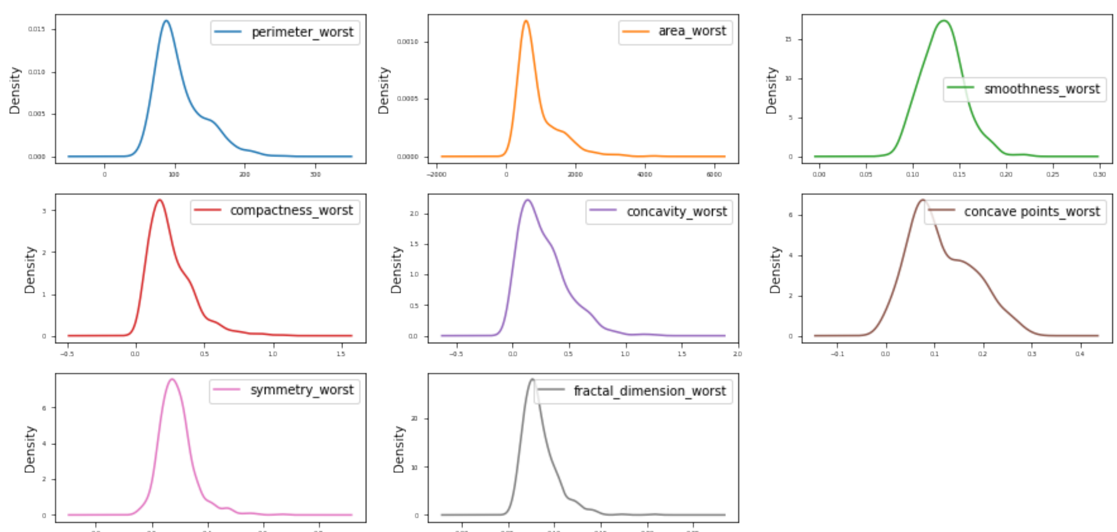
2.Density plots "_se" suffix designition

```
plt = data_se.plot(kind= 'density', subplots=True, layout=(4,3),
sharex=False,
                   sharey=False,fontsize=12, figsize=(15,10))
```



3.Density plot "_worst" suffix designition

```
plt = data_worst.plot(kind= 'kde', subplots=True, layout=(4,3),
sharex=False, sharey=False,fontsize=5,
                     figsize=(15,10))
```



Correlation heatmap

```
dataset.corr()
```

```
                         radius_mean  ...  fractal_dimension_worst
radius_mean                 1.000000  ...                 0.007066
texture_mean                0.323782  ...                 0.119205
```

```
perimeter_mean              0.997855  ...                 0.051019
area_mean                   0.987357  ...                 0.003738
smoothness_mean             0.170581  ...                 0.499316
compactness_mean            0.506124  ...                 0.687382
concavity_mean              0.676764  ...                 0.514930
concave points_mean         0.822529  ...                 0.368661
symmetry_mean               0.147741  ...                 0.438413
fractal_dimension_mean     -0.311631  ...                 0.767297
radius_se                   0.679090  ...                 0.049559
texture_se                 -0.097317  ...                -0.045655
perimeter_se                0.674172  ...                 0.085433
area_se                     0.735864  ...                 0.017539
smoothness_se              -0.222600  ...                 0.101480
compactness_se              0.206000  ...                 0.590973
concavity_se                0.194204  ...                 0.439329
concave points_se           0.376169  ...                 0.310655
symmetry_se                -0.104321  ...                 0.078079
fractal_dimension_se       -0.042641  ...                 0.591328
radius_worst                0.969539  ...                 0.093492
texture_worst               0.297008  ...                 0.219122
perimeter_worst             0.965137  ...                 0.138957
area_worst                  0.941082  ...                 0.079647
smoothness_worst            0.119616  ...                 0.617624
compactness_worst           0.413463  ...                 0.810455
concavity_worst             0.526911  ...                 0.686511
concave points_worst        0.744214  ...                 0.511114
symmetry_worst              0.163953  ...                 0.537848
fractal_dimension_worst     0.007066  ...                 1.000000

[30 rows x 30 columns]
```

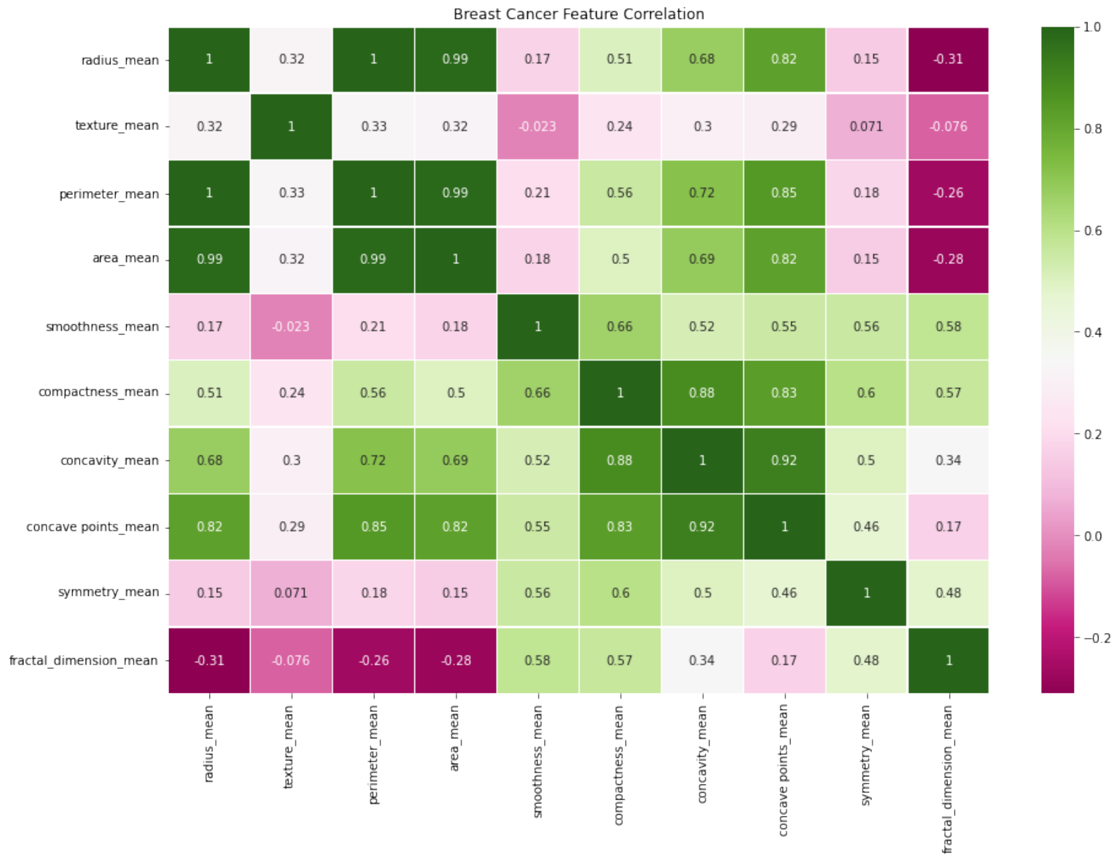
```
data_mean.corr()
```

```
                        radius_mean  ...  fractal_dimension_mean
radius_mean                1.000000  ...               -0.311631
texture_mean               0.323782  ...               -0.076437
perimeter_mean             0.997855  ...               -0.261477
area_mean                  0.987357  ...               -0.283110
smoothness_mean            0.170581  ...                0.584792
compactness_mean           0.506124  ...                0.565369
concavity_mean             0.676764  ...                0.336783
concave points_mean        0.822529  ...                0.166917
symmetry_mean              0.147741  ...                0.479921
fractal_dimension_mean    -0.311631  ...                1.000000

[10 rows x 10 columns]
```

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

```
f, ax = plt.subplots(figsize=(15, 10))
plt.title('Breast Cancer Feature Correlation')
sns.heatmap(data_mean.corr(),annot=True,cmap='PiYG',linewidths=.5)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f18a7a35ad0>
```



Splitting the data - Train Test split

Label encoding

```
#Here, I assign the 30 features to a NumPy array x, and transform the class labels from their original string representation (M and B) into integers.
array = dataset.values
x = array[:,1:31]
y = array[:,0]
```

```
from sklearn.model_selection import train_test_split
array = dataset.values
x = array[:,1:31]
y = array[:,0]
X_train,X_test,Y_train,Y_test = train_test_split(x,y,test_size=0.20,random_state=0)
```

```
X_train.shape
```

```
(455, 30)
```

```
X_test.shape
```

```
(114, 30)
```

```
Y_train.shape
```

```
(455,)
```

```
Y_test.shape
```

```
(114,)
```

```
from sklearn.metrics import accuracy_score
```

Logistic Regression

```
from sklearn.linear_model import LogisticRegression
model_logistic_reg = LogisticRegression()
model_logistic_reg.fit(X_train,Y_train)
Y_pred_logistic_reg = model_logistic_reg.predict(X_test)
```

```
Y_pred_logistic_reg.shape
```

```
(114,)
```

```
print("Predicted Values : ",Y_pred_logistic_reg)
```

```
Predicted Values :  ['1' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0'
'0' '1' '0' '1' '0' '1'
 '1' '1' '1' '1' '0' '0' '1' '0' '0' '1' '0' '1' '0' '1' '0' '1' '0'
'1'
 '0' '1' '0' '1' '1' '0' '1' '0' '1' '1' '0' '0' '0' '1' '1' '0' '1'
'0'
 '0' '0' '0' '0' '0' '1' '1' '1' '0' '0' '1' '0' '1' '1' '1' '0' '1'
'1'
 '0' '0' '1' '0' '0' '0' '0' '0' '1' '1' '1' '0' '1' '0' '0' '0' '1'
'1'
 '0' '1' '1' '1' '0' '0' '1' '0' '0' '0' '0' '0' '0' '0' '1' '0' '1'
'0'
 '1' '1' '0' '1' '1' '0']
```

```
Y_test[0:10] #You can check accuracy by observing predicted results
and test data.
```

```
array(['1', '0', '0', '0', '0', '0', '0', '0', '0', '0'],
dtype=object)
```

```
accuracy_score_logistic_reg =
round(accuracy_score(Y_pred_logistic_reg,Y_test)*100,2)
print("The accuracy score achieved using Logistic Regression is:
"+str(accuracy_score_logistic_reg)+" %")
```

```
The accuracy score achieved using Logistic Regression is: 94.74 %
```

SVM

```
from sklearn import svm
model_svm = svm.SVC(kernel='linear')
model_svm.fit(X_train, Y_train)
Y_pred_svm = model_svm.predict(X_test)
```

```
Y_pred_svm.shape
```

```
(114,)
```

```
print("Predicted Values : ",Y_pred_svm)
```

```
Predicted Values :  ['1' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0'
'0' '1' '0' '1' '0' '1'
 '1' '1' '1' '1' '0' '0' '1' '0' '0' '1' '0' '1' '0' '1' '0' '1' '0'
'1'
 '0' '1' '0' '1' '1' '0' '1' '0' '1' '1' '0' '0' '0' '1' '1' '1' '1'
'0'
 '0' '0' '0' '0' '0' '1' '1' '1' '0' '0' '1' '0' '1' '1' '1' '0' '1'
'1'
 '0' '0' '1' '0' '0' '0' '0' '0' '1' '1' '1' '0' '1' '0' '0' '0' '1'
'1'
 '0' '1' '1' '1' '0' '0' '1' '0' '0' '0' '0' '0' '0' '0' '1' '0' '1'
'0'
 '1' '1' '0' '1' '1' '0']
```

```
Y_test[0:10] #You can check accuracy by observing predicted results 
and test data.
```

```
array(['1', '0', '0', '0', '0', '0', '0', '0', '0', '0'], 
dtype=object)
```

```
accuracy_score_svm = round(accuracy_score(Y_pred_svm,Y_test)*100,2)
print("The accuracy score achieved using Linear SVM is: 
"+str(accuracy_score_svm)+" %")
```

```
The accuracy score achieved using Linear SVM is: 95.61 %
```

K Nearest Neighbors

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=7)
knn.fit(X_train,Y_train)
Y_pred_knn=knn.predict(X_test)
```

```
Y_pred_knn.shape
```

```
(114,)
```

```
print("Predicted Values : ",Y_pred_knn)
```

```
Predicted Values :  ['1' '0' '0' '1' '0' '0' '0' '0' '0' '0' '0' '0'
'0' '1' '0' '1' '0' '1'
 '1' '1' '1' '1' '0' '0' '1' '0' '0' '0' '0' '1' '0' '1' '0' '1' '0'
'1'
 '0' '1' '0' '1' '1' '0' '1' '0' '1' '1' '0' '0' '0' '1' '1' '0' '1'
'0'
 '0' '0' '0' '0' '0' '1' '1' '1' '0' '0' '1' '0' '1' '1' '1' '0' '0'
'1'
 '0' '0' '1' '0' '0' '0' '0' '0' '1' '1' '1' '0' '1' '0' '0' '0' '1'
'1'
 '0' '1' '0' '1' '0' '0' '1' '0' '0' '0' '0' '0' '0' '0' '1' '0' '1'
'0'
 '1' '1' '0' '1' '1' '0']
```

```
Y_test[0:10] #You can check accuracy by observing predicted results
and test data.
```

```
array(['1', '0', '0', '0', '0', '0', '0', '0', '0', '0'],
dtype=object)
```

```
accuracy_score_knn = round(accuracy_score(Y_pred_knn,Y_test)*100,2)
print("The accuracy score achieved using KNN is:
"+str(accuracy_score_knn)+" %")
```

```
The accuracy score achieved using KNN is: 94.74 %
```

Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
max_accuracy = 0
for x in range(200):
    dt = DecisionTreeClassifier(random_state=x)
    dt.fit(X_train,Y_train)
    Y_pred_dt = dt.predict(X_test)
    current_accuracy = round(accuracy_score(Y_pred_dt,Y_test)*100,2)
    if(current_accuracy>max_accuracy):
        max_accuracy = current_accuracy
        best_x = x

dt = DecisionTreeClassifier(random_state=best_x)
dt.fit(X_train,Y_train)
Y_pred_dt = dt.predict(X_test)
```

```
print(Y_pred_dt.shape)
```

```
(114,)
```

```
print("Predicted Values : ",Y_pred_dt)
```

```
Predicted Values :  ['1' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0'
'0' '1' '1' '1' '0' '1'
 '1' '1' '1' '1' '0' '0' '1' '0' '0' '1' '0' '1' '0' '1' '0' '1' '0'
```

```
'1'
 '0' '1' '0' '1' '0' '0' '1' '0' '1' '1' '0' '0' '0' '1' '1' '1' '1'
'0'
 '0' '0' '1' '0' '1' '1' '1' '1' '0' '0' '1' '1' '1' '1' '1' '0' '0'
'1'
 '0' '0' '1' '0' '0' '0' '0' '0' '1' '1' '1' '0' '1' '0' '0' '0' '1'
'1'
 '0' '1' '0' '1' '0' '0' '1' '0' '0' '0' '0' '0' '0' '1' '1' '0' '1'
'0'
 '1' '0' '0' '1' '1' '0']
```

```
Y_test[0:10] #You can check accuracy by observing predicted results
and test data.
```

```
array(['1', '0', '0', '0', '0', '0', '0', '0', '0', '0'],
dtype=object)
```

```
accuracy_score_dt = round(accuracy_score(Y_pred_dt,Y_test)*100,2)
print("The accuracy score achieved using Decision Tree is:
"+str(accuracy_score_dt)+" %")
```

```
The accuracy score achieved using Decision Tree is: 91.23 %
```

Random Forest

```
from sklearn.ensemble import RandomForestClassifier
max_accuracy = 0
for x in range(2000):
    rf = RandomForestClassifier(random_state=x)
    rf.fit(X_train,Y_train)
    Y_pred_rf = rf.predict(X_test)
    current_accuracy = round(accuracy_score(Y_pred_rf,Y_test)*100,2)
    if(current_accuracy>max_accuracy):
        max_accuracy = current_accuracy
        best_x = x

rf = RandomForestClassifier(random_state=best_x)
rf.fit(X_train,Y_train)
Y_pred_rf = rf.predict(X_test)
```

```
Y_pred_rf.shape
```

```
(114,)
```

```
print("Predicted Values : ",Y_pred_rf)
```

```
Predicted Values :  ['1' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0'
'0' '1' '0' '1' '0' '1'
 '1' '1' '1' '1' '0' '0' '1' '0' '0' '1' '0' '1' '0' '1' '0' '1' '0'
'1'
 '0' '1' '0' '1' '1' '0' '1' '0' '0' '1' '0' '0' '0' '1' '1' '1' '1'
'0'
```

```
 '0' '0' '0' '0' '0' '1' '1' '1' '0' '0' '1' '0' '1' '1' '1' '0' '0'
'1'
 '0' '0' '1' '0' '0' '0' '0' '0' '1' '1' '1' '0' '1' '0' '0' '0' '1'
'1'
 '0' '1' '0' '1' '0' '0' '1' '0' '0' '0' '0' '0' '0' '0' '1' '0' '1'
'0'
 '1' '1' '0' '1' '1' '0']
```

```
Y_test[0:10] #You can check accuracy by observing predicted results
and test data.
```

```
array(['1', '0', '0', '0', '0', '0', '0', '0', '0', '0'],
dtype=object)
```

```
accuracy_score_rf = round(accuracy_score(Y_pred_rf,Y_test)*100,2)
print("The accuracy score achieved using Random Forest is:
"+str(accuracy_score_rf)+" %")
```

```
The accuracy score achieved using Random Forest is: 98.25 %
```

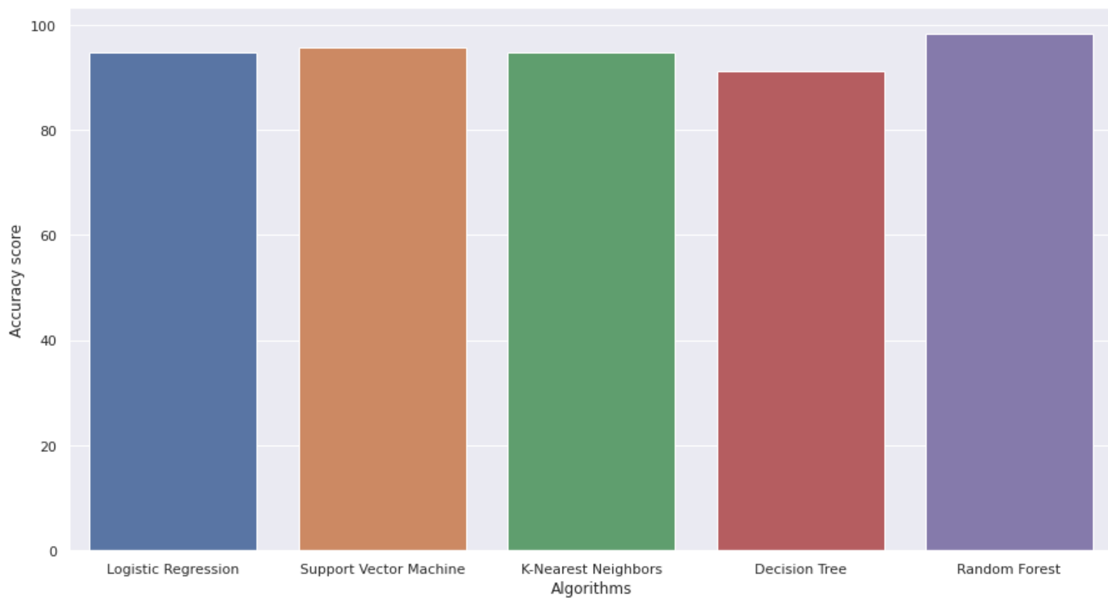
Summary of accuracy scores

```
all_accuracy_scores =
[accuracy_score_logistic_reg,accuracy_score_svm,accuracy_score_knn,acc
uracy_score_dt,accuracy_score_rf]
algorithms_used = ["Logistic Regression","Support Vector Machine","K-
Nearest Neighbors","Decision Tree","Random Forest"]

for i in range(len(algorithms_used)):
    print("\nThe accuracy score achieved using "+algorithms_used[i]+"
is: "+str(all_accuracy_scores[i])+" %")
```

```

The accuracy score achieved using Logistic Regression is: 94.74 %

The accuracy score achieved using Support Vector Machine is: 95.61 %

The accuracy score achieved using K-Nearest Neighbors is: 94.74 %

The accuracy score achieved using Decision Tree is: 91.23 %

The accuracy score achieved using Random Forest is: 98.25 %
```

```
sns.set(rc={'figure.figsize':(15,8)})
plt.xlabel("Algorithms")
plt.ylabel("Accuracy score")

sns.barplot(algorithms_used,all_accuracy_scores)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1899723350>
```



Here we can see that Random Forest is better than other algorithms.

Name - Nihalahmed Munir Barudwale

Project name - Prediction of Breast cancer for a given patient