Case Study - Cancer Data (Logistic Regression, D-Tree, Random Forest)

May 10, 2023

This case study is to analyse about the diagnostic data of breast cancer patients. This data is officially obtained from the University of Wisconsin Hospitals.

This model will predict which people are prone to develop cancer.

[1]:
```
# Import libraries
import numpy as np
import pandas as pd
import seaborn as sns
import sklearn
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import␣
  ↪accuracy_score,f1_score,recall_score,precision_score, confusion_matrix,␣
  ↪classification_report
```

[2]:
```
import warnings
warnings.filterwarnings('ignore')
```

[3]:
```
#import cancer dataset and look at the first five rows.
cData = pd.read_csv("Cancer_Data.csv")
cData.head()
```

[3]:
```
         id diagnosis  radius_mean  texture_mean  perimeter_mean  area_mean  \
0    842302         M        17.99         10.38          122.80     1001.0
1    842517         M        20.57         17.77          132.90     1326.0
2  84300903         M        19.69         21.25          130.00     1203.0
3  84348301         M        11.42         20.38           77.58      386.1
4  84358402         M        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

   …  texture_worst  perimeter_worst  area_worst  smoothness_worst  \
0  …          17.33           184.60      2019.0            0.1622
1  …          23.41           158.80      1956.0            0.1238
2  …          25.53           152.50      1709.0            0.1444
3  …          26.50            98.87       567.7            0.2098
4  …          16.67           152.20      1575.0            0.1374

   compactness_worst  concavity_worst  concave points_worst  symmetry_worst  \
0             0.6656           0.7119                0.2654          0.4601
1             0.1866           0.2416                0.1860          0.2750
2             0.4245           0.4504                0.2430          0.3613
3             0.8663           0.6869                0.2575          0.6638
4             0.2050           0.4000                0.1625          0.2364

   fractal_dimension_worst  Unnamed: 32
0                  0.11890          NaN
1                  0.08902          NaN
2                  0.08758          NaN
3                  0.17300          NaN
4                  0.07678          NaN

[5 rows x 33 columns]
```

Column names and meanings:

1.id: ID number

2.diagnosis: The diagnosis of breast tissues (M = malignant, B = benign)

3.radius_mean: mean of distances from center to points on the perimeter

4.texture_mean: standard deviation of gray-scale values

5.perimeter_mean: mean size of the core tumor

6.area_mean: area of the tumor

7.smoothness_mean: mean of local variation in radius lengths

8.compactness_mean: mean of perimeter^2 / area - 1.0

9.concavity_mean: mean of severity of concave portions of the contour

10.concave_points_mean: mean for number of concave portions of the contour

11.symmetry_mean

12.fractal_dimension_mean: mean for “coastline approximation” - 1

13.radius_se: standard error for the mean of distances from center to points on the perimeter

14.texture_se: standard error for standard deviation of gray-scale values

15.perimeter_se

16.area_se

17.smoothness_se: standard error for local variation in radius lengths

18.compactness_se: standard error for perimeter^2 / area - 1.0

19.concavity_se: standard error for severity of concave portions of the contour

20.concave_points_se: standard error for number of concave portions of the contour

21.symmetry_se

22.fractal_dimension_se: standard error for "coastline approximation" - 1

23.radius_worst: "worst" or largest mean value for mean of distances from center to points on the perimeter

24.texture_worst: "worst" or largest mean value for standard deviation of gray-scale values

25.perimeter_worst

26.area_worst

27.smoothness_worst: "worst" or largest mean value for local variation in radius lengths

28.compactness_worst: "worst" or largest mean value for perimeter^2 / area - 1.0

29.concavity_worst: "worst" or largest mean value for severity of concave portions of the contour

30.concave_points_worst: "worst" or largest mean value for number of concave portions of the contour

31.symmetry_worst

32.fractal_dimension_worst: "worst" or largest mean value for "coastline approximation" - 1

[4]:
```
cData.shape
```

[4]:
```
(569, 33)
```

[5]:
```
cData.info()
```

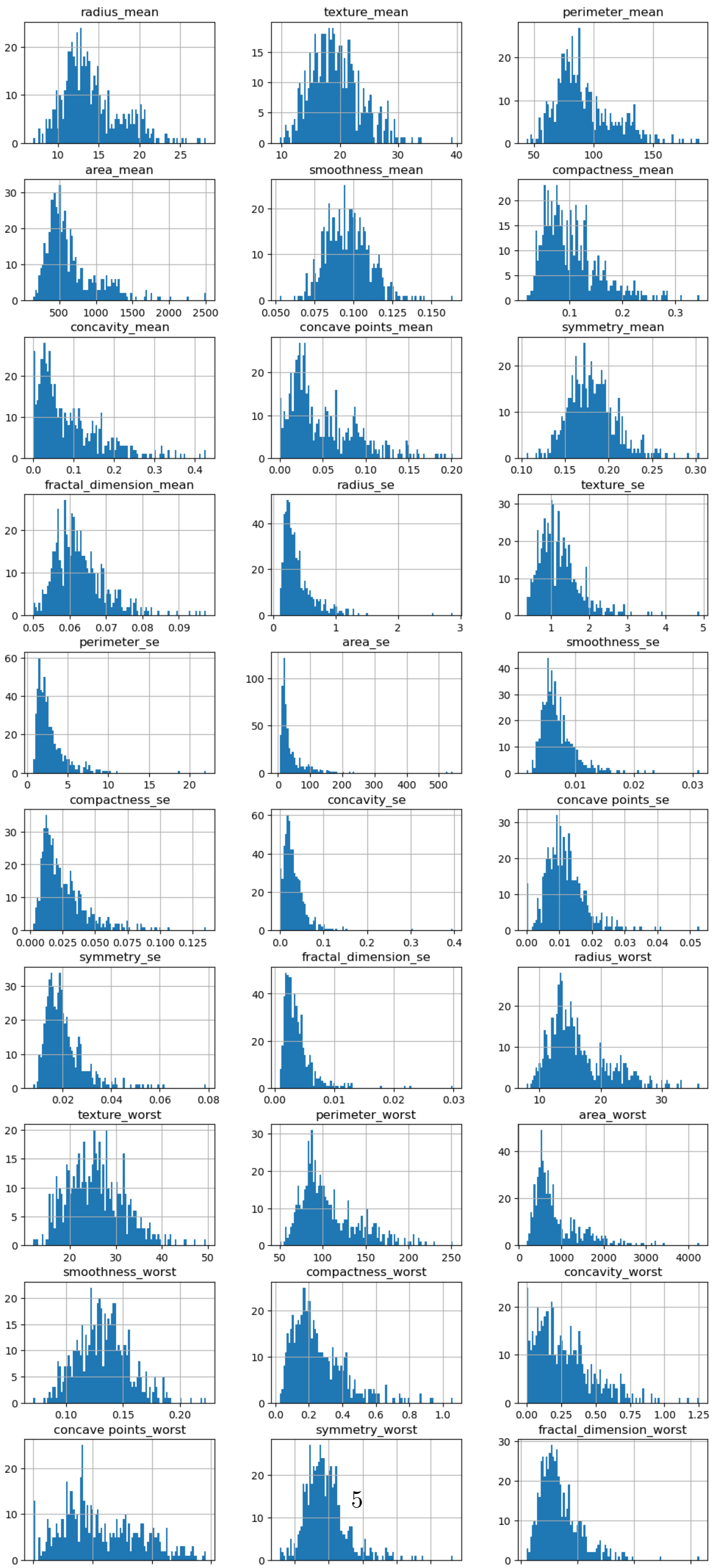
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   id                       569 non-null    int64
 1   diagnosis                569 non-null    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
```

The column “Id” is no longer required for our analysis and the column “Unnamed: 32” seems to have incorrect/ irrelevant data, so lets drop both of them.

[6]:
```
cData = cData.drop(['id','Unnamed: 32'],axis=1)
```

[7]:
```
cData.hist(stacked = False,bins = 100, figsize=(12,30),layout=(11,3))
plt.show()
```


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

Refer above graph to know the frequency of the Data distribution in cancer dataset.

[8]:
```
cData['diagnosis'] = cData['diagnosis'].replace({'M':1,'B':0})

#The above code can also be achieved with map function as well.

#cData['diagnosis'] = data['diagnosis'].map({'M':1,'B':0})
```

[9]:
```
cData.head()
```

[9]:
```
   diagnosis  radius_mean  texture_mean  perimeter_mean  area_mean  \
0          1        17.99         10.38          122.80     1001.0
1          1        20.57         17.77          132.90     1326.0
2          1        19.69         21.25          130.00     1203.0
3          1        11.42         20.38           77.58      386.1
4          1        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  …  radius_worst  texture_worst  perimeter_worst  \
0         0.2419  …         25.38          17.33           184.60
1         0.1812  …         24.99          23.41           158.80
2         0.2069  …         23.57          25.53           152.50
3         0.2597  …         14.91          26.50            98.87
4         0.1809  …         22.54          16.67           152.20

   area_worst  smoothness_worst  compactness_worst  concavity_worst  \
0      2019.0            0.1622             0.6656           0.7119
1      1956.0            0.1238             0.1866           0.2416
2      1709.0            0.1444             0.4245           0.4504
3       567.7            0.2098             0.8663           0.6869
4      1575.0            0.1374             0.2050           0.4000

   concave points_worst  symmetry_worst  fractal_dimension_worst
0                0.2654          0.4601                  0.11890
1                0.1860          0.2750                  0.08902
2                0.2430          0.3613                  0.08758
3                0.2575          0.6638                  0.17300
4                0.1625          0.2364                  0.07678
```

```
[5 rows x 31 columns]
```

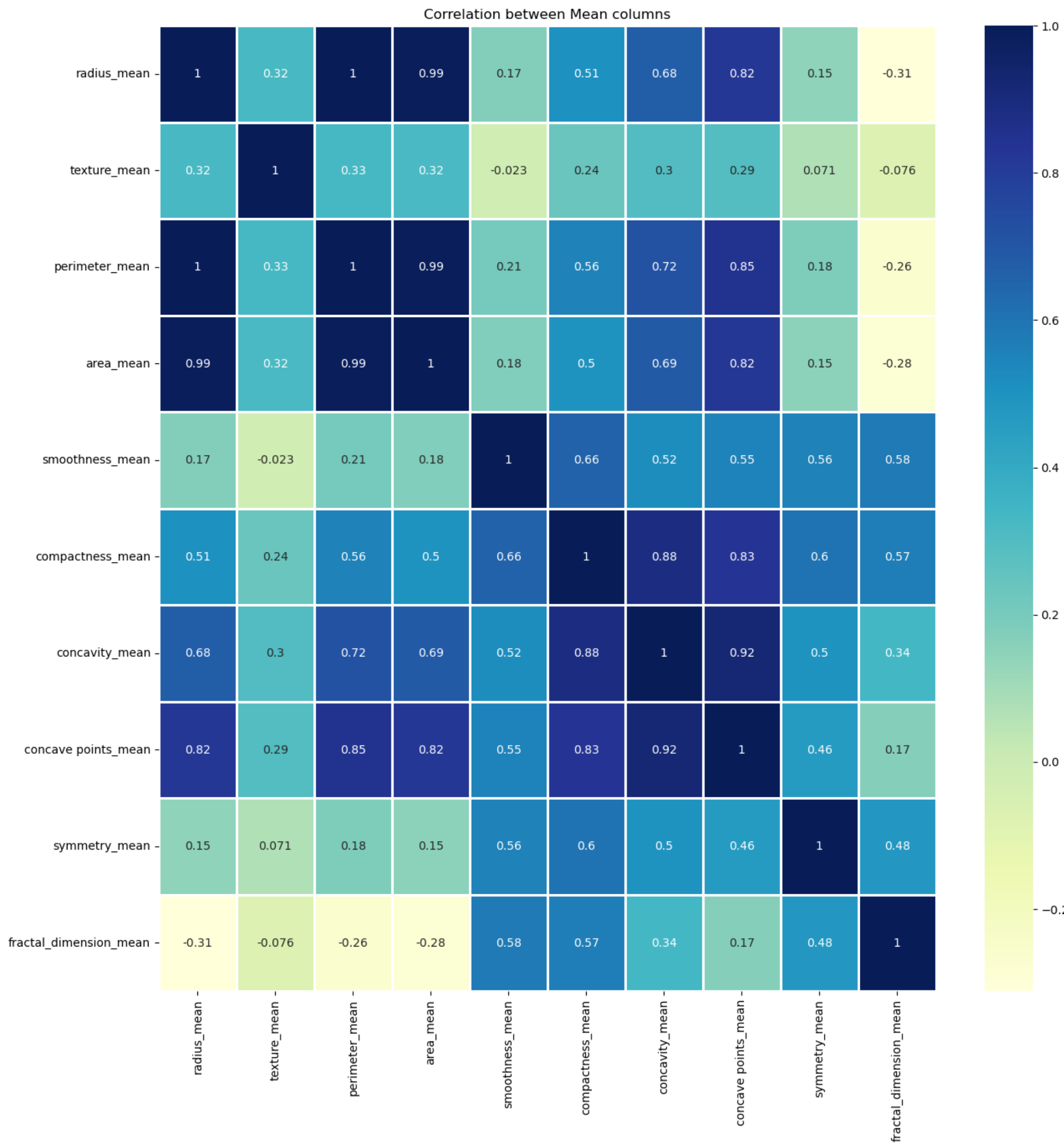
[10]:
```
cData.describe()
```

[10]:
```
       diagnosis  radius_mean  texture_mean  perimeter_mean    area_mean  \
count  569.000000   569.000000    569.000000      569.000000   569.000000
mean     0.372583    14.127292     19.289649       91.969033   654.889104
std      0.483918     3.524049      4.301036       24.298981   351.914129
min      0.000000     6.981000      9.710000       43.790000   143.500000
25%      0.000000    11.700000     16.170000       75.170000   420.300000
50%      0.000000    13.370000     18.840000       86.240000   551.100000
75%      1.000000    15.780000     21.800000      104.100000   782.700000
max      1.000000    28.110000     39.280000      188.500000  2501.000000

       smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
count       569.000000        569.000000      569.000000           569.000000
mean          0.096360          0.104341        0.088799             0.048919
std           0.014064          0.052813        0.079720             0.038803
min           0.052630          0.019380        0.000000             0.000000
25%           0.086370          0.064920        0.029560             0.020310
50%           0.095870          0.092630        0.061540             0.033500
75%           0.105300          0.130400        0.130700             0.074000
max           0.163400          0.345400        0.426800             0.201200

       symmetry_mean  …  radius_worst  texture_worst  perimeter_worst  \
count     569.000000  …    569.000000     569.000000       569.000000
mean        0.181162  …     16.269190      25.677223       107.261213
std         0.027414  …      4.833242       6.146258        33.602542
min         0.106000  …      7.930000      12.020000        50.410000
25%         0.161900  …     13.010000      21.080000        84.110000
50%         0.179200  …     14.970000      25.410000        97.660000
75%         0.195700  …     18.790000      29.720000       125.400000
max         0.304000  …     36.040000      49.540000       251.200000

        area_worst  smoothness_worst  compactness_worst  concavity_worst  \
count   569.000000        569.000000         569.000000       569.000000
mean    880.583128          0.132369           0.254265         0.272188
std     569.356993          0.022832           0.157336         0.208624
min     185.200000          0.071170           0.027290         0.000000
25%     515.300000          0.116600           0.147200         0.114500
50%     686.500000          0.131300           0.211900         0.226700
75%    1084.000000          0.146000           0.339100         0.382900
max    4254.000000          0.222600           1.058000         1.252000

       concave points_worst  symmetry_worst  fractal_dimension_worst
count            569.000000      569.000000               569.000000
mean               0.114606        0.290076                 0.083946
```

```
std            0.065732    0.061867         0.018061
min            0.000000    0.156500         0.055040
25%            0.064930    0.250400         0.071460
50%            0.099930    0.282200         0.080040
75%            0.161400    0.317900         0.092080
max            0.291000    0.663800         0.207500

[8 rows x 31 columns]
```

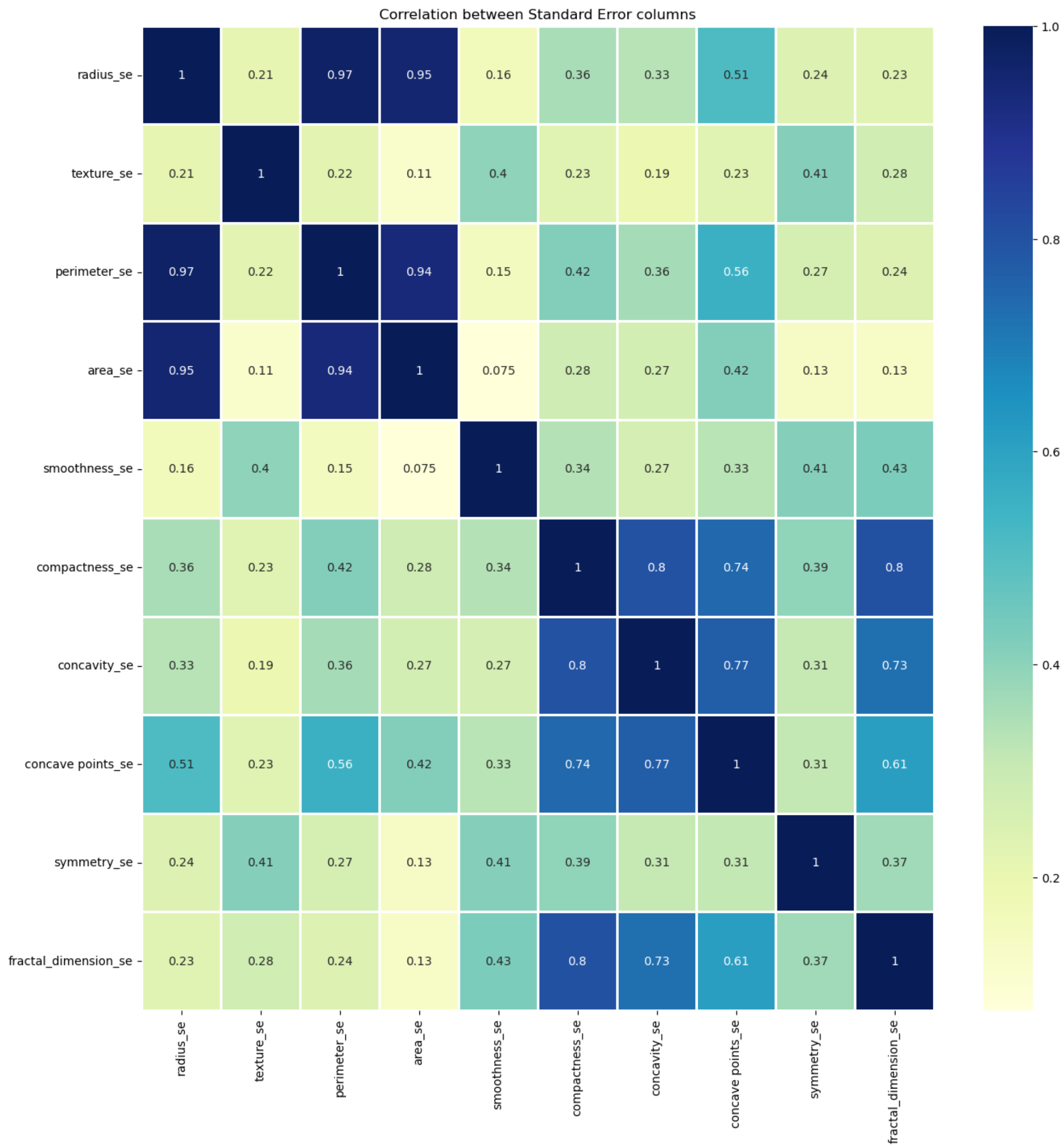
[11]:
```
# Heatmap for mean columns
mean_columns = ['radius_mean','texture_mean','perimeter_mean','area_mean',
                'smoothness_mean','compactness_mean','concavity_mean','concave␣
 ↪points_mean',
                'symmetry_mean','fractal_dimension_mean']
plt.figure(figsize=(15,15))
ax = plt.axes()
sns.heatmap(cData[mean_columns].corr(), annot=True,␣
 ↪cmap='YlGnBu',linewidths=1,ax=ax)
ax.set_title('Correlation between Mean columns')
```

[11]:
```
Text(0.5, 1.0, 'Correlation between Mean columns')
```



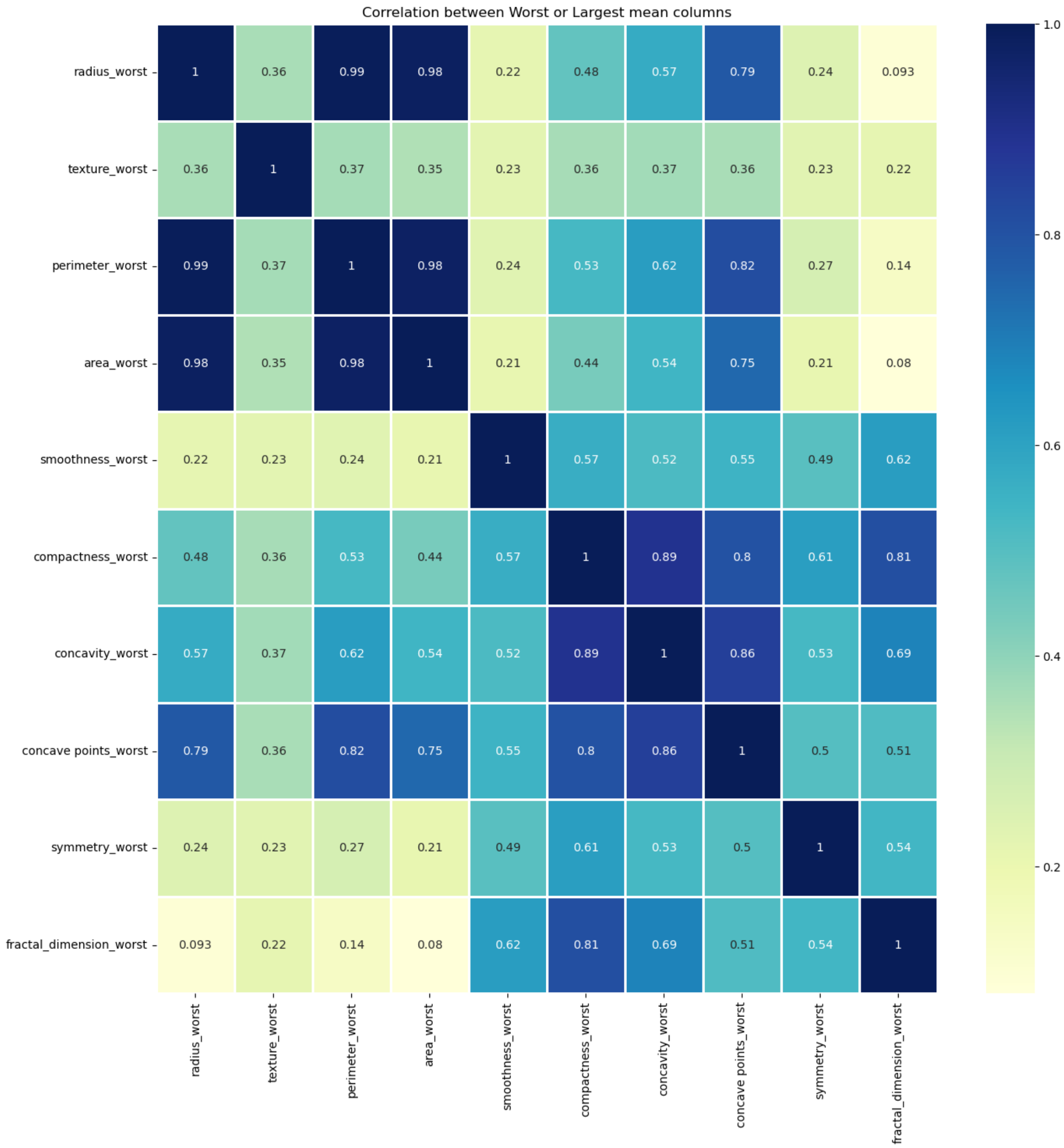

```
[12]: plt.figure(figsize=(15,15))
      se_columns = ['radius_se','texture_se','perimeter_se','area_se',
                    'smoothness_se', 'compactness_se', 'concavity_se', 'concave␣
       ↪points_se', 'symmetry_se', 'fractal_dimension_se']
      ax= plt.axes()
      sns.heatmap(cData[se_columns].corr(), annot=True, cmap='YlGnBu',linewidths=1)
      ax.set_title('Correlation between Standard Error columns')
```

```
[12]: Text(0.5, 1.0, 'Correlation between Standard Error columns')
```



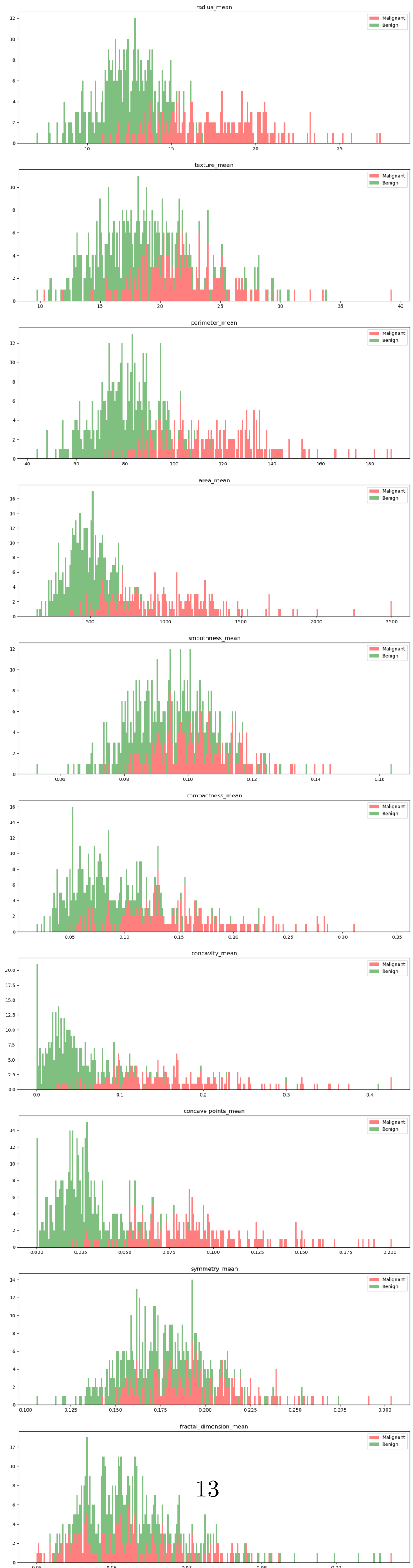
[13]:
```
plt.figure(figsize=(15,15))
worst_columns =['radius_worst','texture_worst','perimeter_worst',
                'area_worst','smoothness_worst','compactness_worst',
                'concavity_worst','concave␣
 ↪points_worst','symmetry_worst','fractal_dimension_worst']

ax= plt.axes()
sns.heatmap(cData[worst_columns].corr(), annot=True, cmap='YlGnBu',linewidths=1)
ax.set_title('Correlation between Worst or Largest mean columns')
```

[13]: Text(0.5, 1.0, 'Correlation between Worst or Largest mean columns')



[14]:
```
#In order to have better visualisation,let us split the data into malignant and
↪benign

dataMalignant=cData[cData['diagnosis'] ==1]
dataBenign=cData[cData['diagnosis'] ==0]

#Plotting these features as a histogram
fig, axes = plt.subplots(nrows=10, ncols=1, figsize=(15,60))
```

```
for idx,ax in enumerate(axes):
    ax.figure
    binwidth= (max(cData[mean_columns[idx]]) - min(cData[mean_columns[idx]]))/
 ↪250
    ax.hist([dataMalignant[mean_columns[idx]],dataBenign[mean_columns[idx]]],
            bins=np.arange(min(cData[mean_columns[idx]]),␣
 ↪max(cData[mean_columns[idx]]) + binwidth, binwidth),
            alpha=0.5,
            stacked=True,
            label=['Malignant','Benign'],
            color=['r','g'])
    ax.legend(loc='upper right')
    ax.set_title(mean_columns[idx])
plt.show()
```


radius_mean
texture_mean
perimeter_mean
area_mean
smoothness_mean
compactness_mean
concavity_mean
concave points_mean
symmetry_mean
fractal_dimension_mean
Malignant
Benign

From the above visuals, we could see that all the features are at the highest data point in case of benign tunors and not for malignant tumors. So with respect to malignancy, higher values doesnt have any accountability.

[15]:
```
#Before we start with creating a data model, let us first make sure that
↪is are no null values.
cData.isna().sum()
```

[15]:
```
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
```

Before splitting the train and test data, we will see the ratio of the cancer and non cancer patients.

This step will help us to identify whether the dataset is balanced or not.

[16]:
```
cancer = dataMalignant.shape[0]
benign = dataBenign.shape[0] - cancer
print(cancer,benign)
cancerPatients = (cancer/(cancer+benign))*100
cancerPatients
nonCancerPatients = (benign/(cancer+benign))*100
cancerPatients, nonCancerPatients
```

```
212 145
```

[16]:
```
(59.38375350140056, 40.61624649859944)
```

The ratio shows that the dataset is a pretty balanced one.

[17]:
```
X = cData.drop('diagnosis',axis=1) #Independent Variables
Y = cData['diagnosis']   # Target or Dependent Variable

X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.
 ↪3,random_state=20)
X_train.head()
```

[17]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	\
435	13.98	19.62	91.12	599.5	0.10600	
72	17.20	24.52	114.20	929.4	0.10710	
266	10.60	18.95	69.28	346.4	0.09688	
468	17.60	23.33	119.00	980.5	0.09289	
456	11.63	29.29	74.87	415.1	0.09357	

	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	\
435	0.11330	0.11260	0.06463	0.1669	
72	0.18300	0.16920	0.07944	0.1927	
266	0.11470	0.06387	0.02642	0.1922	
468	0.20040	0.21360	0.10020	0.1696	
456	0.08574	0.07160	0.02017	0.1799	

	fractal_dimension_mean	…	radius_worst	texture_worst	\
435	0.06544	…	17.04	30.80	
72	0.06487	…	23.32	33.82	
266	0.06491	…	11.88	22.94	
468	0.07369	…	21.57	28.87	
456	0.06166	…	13.12	38.81	

	perimeter_worst	area_worst	smoothness_worst	compactness_worst	\
435	113.90	869.3	0.1613	0.3568	
72	151.60	1681.0	0.1585	0.7394	
266	78.28	424.8	0.1213	0.2515	
468	143.60	1437.0	0.1207	0.4785	
456	86.04	527.8	0.1406	0.2031	

```
     concavity_worst  concave points_worst  symmetry_worst  \
435           0.4069               0.18270          0.3179
72            0.6566               0.18990          0.3313
266           0.1916               0.07926          0.2940
468           0.5165               0.19960          0.2301
456           0.2923               0.06835          0.2884

     fractal_dimension_worst
435                  0.10550
72                   0.13390
266                  0.07587
468                  0.12240
456                  0.07220

[5 rows x 30 columns]
```

[18]:
```
print("{0:0.2f}% data is in training set".format((len(X_train)/len(cData.
 ↪index)) * 100))
print("{0:0.2f}% data is in test set".format((len(X_test)/len(cData.index)) *␣
 ↪100))
```

```
69.95% data is in training set
30.05% data is in test set
```

[19]:
```
#Below mentioned is a generic function to calculate accuracy score, confusion␣
 ↪matrix and  classification report
#for all the models. This function would print all the required values in a␣
 ↪matrix format.

def print_score(clf, X_train, y_train, X_test, y_test, train=True):
    if train: # for training data
        pred = clf.predict(X_train)
        clf_report = pd.DataFrame(classification_report(y_train, pred,␣
 ↪output_dict=True))
        print("Train Result:\n================================================")
        print(f"Accuracy Score: {accuracy_score(y_train, pred) * 100:.2f}%")
        print("_______________________________________________")
        print(f"CLASSIFICATION REPORT:\n{clf_report}")
        print("_______________________________________________")
        print(f"Confusion Matrix: \n {confusion_matrix(y_train, pred)}\n")

    elif train==False: # for testing data
        pred = clf.predict(X_test)
        clf_report = pd.DataFrame(classification_report(y_test, pred,␣
 ↪output_dict=True))
        print("Test Result:\n================================================")
```

```
        print(f"Accuracy Score: {accuracy_score(y_test, pred) * 100:.2f}%")
        print("_______________________________________________")
        print(f"CLASSIFICATION REPORT:\n{clf_report}")
        print("_______________________________________________")
        print(f"Confusion Matrix: \n {confusion_matrix(y_test, pred)}\n")
```

Logistic Regression

[20]:
```
# Fit the model on train
model = LogisticRegression(solver="liblinear")
model.fit(X_train, Y_train)
```

[20]:
```
LogisticRegression(solver='liblinear')
```

Accuracy Metrics

For classification models, we will find its accuracy using the following methods.

1. Confusion Matrix
2. Overall Accuracy
3. Recall
4. Precision Score
5. F1 Score

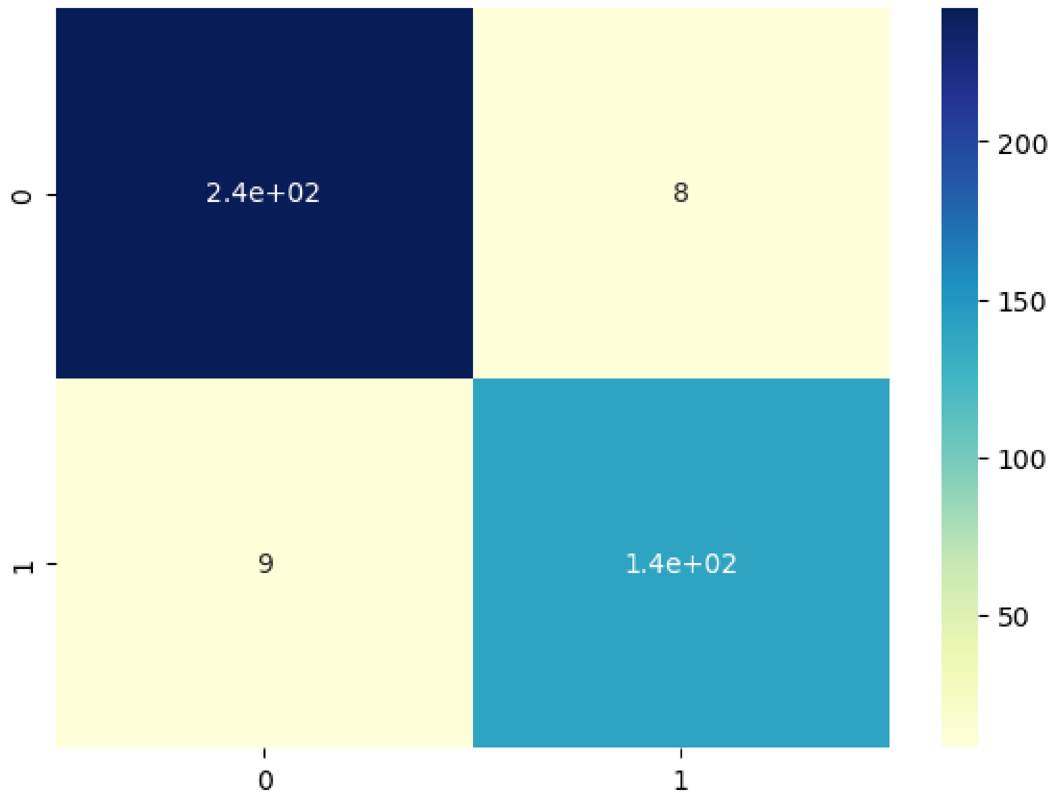
Though there are in-built functions available for the above said accuracy metrics, let us manually calculate in order to understand the logic

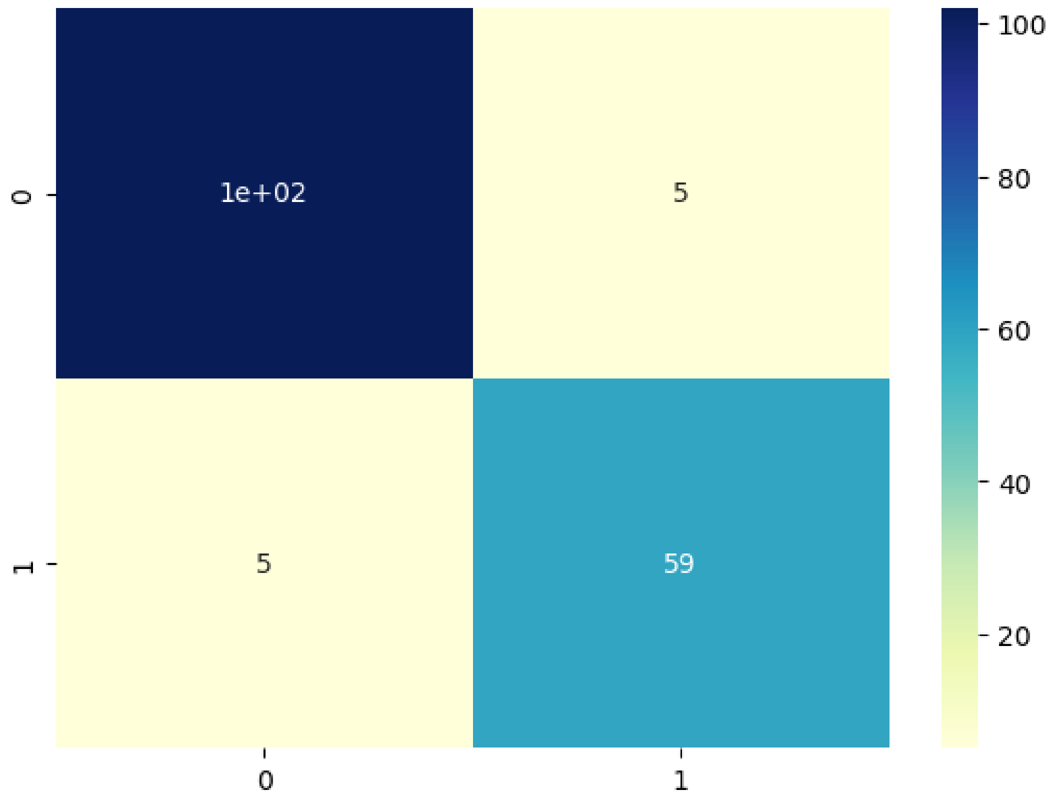
Confusion Matrix

[21]:
```
y_trainpredict = model.predict(X_train)
cmTr=metrics.confusion_matrix(Y_train, y_trainpredict, labels=[0, 1])
print('Confusion Matrix for Train Data:\n',cmTr)
plt.figure(figsize = (7,5))
sns.heatmap(cmTr, annot=True,cmap='YlGnBu')

y_testpredict = model.predict(X_test)
cmTs=metrics.confusion_matrix(Y_test, y_testpredict, labels=[0, 1])
print('Confusion Matrix for Test Data:\n',cmTs)
plt.figure(figsize = (7,5))
sns.heatmap(cmTs, annot=True,cmap='YlGnBu')
```

```
Confusion Matrix for Train Data:
 [[242   8]
 [  9 139]]
Confusion Matrix for Test Data:
 [[102   5]
 [  5  59]]
```

[21]:
```
<Axes: >
```


2.4e+02
8
9
1.4e+02
0
1
0
1
200
150
100
50



The confusion matrix

True Positives (TP): we correctly predicted that they do have cancer 59

True Negatives (TN): we correctly predicted that they don't have cancer 102

False Positives (FP): we incorrectly predicted that they do have cancer (a "Type I error") 5 Falsely predict positive Type I error

False Negatives (FN): we incorrectly predicted that they don't have cancer

(a "Type II error") 5 Falsely predict negative Type II error

Overall Accuracy

First We have seen the confusion matrix, which shows the True Positive, True Negative, False Positive and False Negative.

Now let us start with overall accuracy.

The formula to calculate overall accuracy is (TP + TN)/(TP+TN+FP+FN)

TP - True Positive TN - True Negative FP - False Positive FN - False Negative

[22]:
```
predTrain = model.predict(X_train)
numerator = cmTr[0][0]+cmTr[1][1]
```

```
accTrain = (numerator)/(Y_train.shape)
accTrain = accTrain * 100
print(f"Training Accuracy Score:", accTrain,"%")

predTest = model.predict(X_test)
numerator = cmTs[0][0]+cmTs[1][1]
accTest = (numerator)/(Y_test.shape)
accTest = accTest * 100
print(f"Testing Accuracy Score:", accTest,"%")
```

```
Training Accuracy Score: [95.72864322] %
Testing Accuracy Score: [94.15204678] %
```

Recall

The formula to find Recall or Type II error is True Positive / (True Positive + False Negative)

[23]:
```
trainRecall = cmTr[0][0]/(cmTr[0][0]+cmTr[1][0])
trainRecall = trainRecall * 100
print(f"Recall Score for Train Data:", trainRecall,"%")

testRecall = cmTs[0][0]/(cmTs[0][0]+cmTs[1][0])
testRecall = testRecall * 100
print(f"Recall Score for Test Data:", testRecall,"%")
```

```
Recall Score for Train Data: 96.41434262948208 %
Recall Score for Test Data: 95.32710280373831 %
```

Precision

The formula to find Recall or Type I error is True Positive / (True Positive + False Positive)

[24]:
```
trainPrecision = cmTr[0][0]/(cmTr[0][0]+cmTr[0][1])
trainPrecision = trainPrecision * 100
print(f"Precision Score for Train Data:", trainPrecision,"%")

testPrecision = cmTs[0][0]/(cmTs[0][0]+cmTs[0][1])
testPrecision = testPrecision * 100
print(f"Precision Score for Test Data:", testPrecision,"%")
```

```
Precision Score for Train Data: 96.8 %
Precision Score for Test Data: 95.32710280373831 %
```

F1 Score

The formula to find F1 Score is (2 * Precision * Recall) / (Precision + Recall)

[25]:
```
f1ScoreTrain = (2*trainPrecision*trainRecall) / (trainPrecision+trainRecall)
print(f"F1 Score for Train Data:", f1ScoreTrain,"%")

f1ScoreTest = (2*testPrecision*testRecall) / (testPrecision+testRecall)
print(f"F1 Score for Train Data:", f1ScoreTest,"%")
```

```
F1 Score for Train Data: 96.60678642714572 %
F1 Score for Train Data: 95.32710280373831 %
```

Decision-Tree Model

Gini Impurity

[26]:
```python
model_gini=DecisionTreeClassifier(criterion='gini')
```

[27]:
```python
model_gini.fit(X_train, Y_train)
```

[27]:
```
DecisionTreeClassifier()
```

[28]:
```python
model_gini.score(X_train, Y_train)
```

[28]:
```
1.0
```

[29]:
```python
model_gini.score(X_test,Y_test)
```

[29]:
```
0.935672514619883
```

Entropy

[30]:
```python
model_entropy=DecisionTreeClassifier(criterion='gini')
```

[31]:
```python
model_entropy.fit(X_train, Y_train)
```

[31]:
```
DecisionTreeClassifier()
```

[32]:
```python
model_entropy.score(X_train, Y_train)
```

[32]:
```
1.0
```

[33]:
```python
model_entropy.score(X_test,Y_test)
```

[33]:
```
0.9181286549707602
```

[34]:
```python
dtree = DecisionTreeClassifier(random_state=42)
dtree.fit(X_train, Y_train)

print_score(dtree, X_train, Y_train, X_test, Y_test, train=True)
print_score(dtree, X_train, Y_train, X_test, Y_test, train=False)
```

```
Train Result:
================================================
Accuracy Score: 100.00%
_______________________________________________
CLASSIFICATION REPORT:
             0    1  accuracy  macro avg  weighted avg
precision  1.0  1.0       1.0        1.0           1.0
recall     1.0  1.0       1.0        1.0           1.0
```

```
f1-score      1.0    1.0        1.0         1.0            1.0
support     250.0  148.0        1.0       398.0          398.0
_______________________________________________
Confusion Matrix:
 [[250   0]
 [  0 148]]

Test Result:
================================================
Accuracy Score: 91.81%
_______________________________________________
CLASSIFICATION REPORT:
                    0          1  accuracy   macro avg  weighted avg
precision    0.979381   0.837838  0.918129    0.908610      0.926406
recall       0.887850   0.968750  0.918129    0.928300      0.918129
f1-score     0.931373   0.898551  0.918129    0.914962      0.919088
support    107.000000  64.000000  0.918129  171.000000    171.000000
_______________________________________________
Confusion Matrix:
 [[95 12]
 [ 2 62]]
```

[35]:

```python
from sklearn.model_selection import GridSearchCV

params = {
    "criterion":("gini", "entropy"),
    "splitter":("best", "random"),
    "max_depth":(list(range(1, 20))),
    "min_samples_split":[2, 3, 4],
    "min_samples_leaf":list(range(1, 20)),
}


dtree = DecisionTreeClassifier(random_state=42)
dtreeXv = GridSearchCV(
    dtree,
    params,
    scoring="f1",
    n_jobs=-1,
    verbose=1,
    cv=5
)


dtreeXv.fit(X_train, Y_train)
best_params = dtreeXv.best_params_
```

```python
print(f"Best paramters: {best_params})")

dtree = DecisionTreeClassifier(**best_params)
dtree.fit(X_train, Y_train)
print_score(dtree, X_train, Y_train, X_test, Y_test, train=True)
print_score(dtree, X_train, Y_train, X_test, Y_test, train=False)
```

```
Fitting 5 folds for each of 4332 candidates, totalling 21660 fits
Best paramters: {'criterion': 'entropy', 'max_depth': 4, 'min_samples_leaf': 1,
'min_samples_split': 2, 'splitter': 'random'})
Train Result:
================================================
Accuracy Score: 94.97%
_______________________________________________
CLASSIFICATION REPORT:
                    0           1  accuracy   macro avg  weighted avg
precision    0.967480    0.921053  0.949749    0.944266      0.950215
recall       0.952000    0.945946  0.949749    0.948973      0.949749
f1-score     0.959677    0.933333  0.949749    0.946505      0.949881
support    250.000000  148.000000  0.949749  398.000000    398.000000
_______________________________________________
Confusion Matrix:
 [[238  12]
 [  8 140]]

Test Result:
================================================
Accuracy Score: 93.57%
_______________________________________________
CLASSIFICATION REPORT:
                    0          1  accuracy   macro avg  weighted avg
precision    0.989796   0.863014  0.935673    0.926405      0.942345
recall       0.906542   0.984375  0.935673    0.945459      0.935673
f1-score     0.946341   0.919708  0.935673    0.933025      0.936373
support    107.000000  64.000000  0.935673  171.000000    171.000000
_______________________________________________
Confusion Matrix:
 [[97 10]
 [ 1 63]]
```

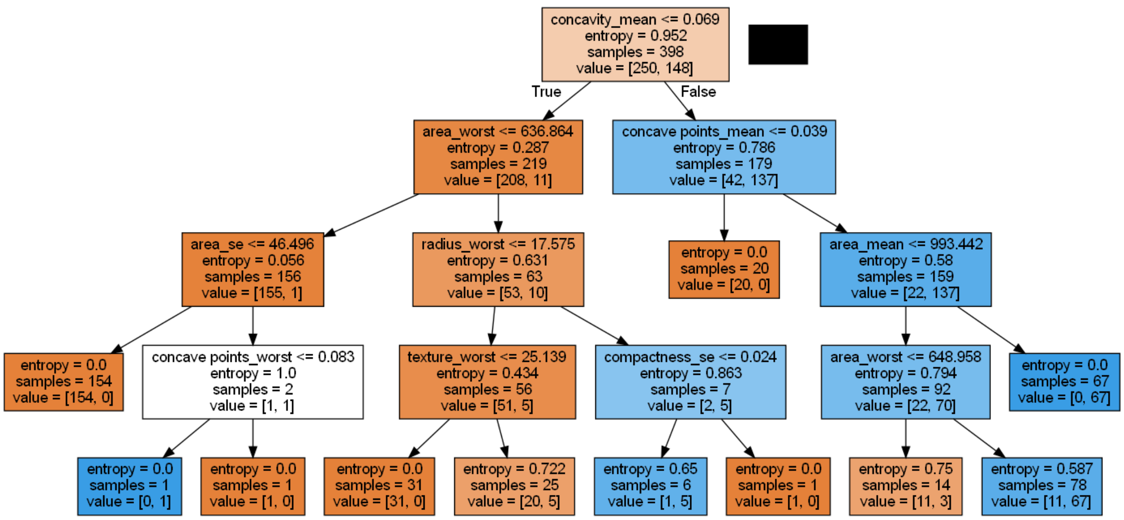
The suitable parameters to create best fit D-Tree model as per grid cross validation are : {'criterion': 'entropy', 'max_depth': 4, 'min_samples_leaf': 1, 'min_samples_split': 2, 'splitter': 'random'}

[36]:
```python
from IPython.display import Image
from six import StringIO
from sklearn.tree import export_graphviz
import pydot
```

```python
features = list(cData.columns)
features.remove("diagnosis")
```

[37]:
```python
import os
os.environ["PATH"] += os.pathsep + 'C:/Program Files (x86)/Graphviz2.38/bin/'
dot_data = StringIO()
export_graphviz(dtree, out_file=dot_data, feature_names=features, filled=True)
graph = pydot.graph_from_dot_data(dot_data.getvalue())
Image(graph[0].create_png())
```

[37]:



Random Forest Model

[38]:
```python
from sklearn.ensemble import RandomForestClassifier

ranFC = RandomForestClassifier(n_estimators=100)
ranFC.fit(X_train, Y_train)

print_score(ranFC, X_train, Y_train, X_test, Y_test, train=True)
print_score(ranFC, X_train, Y_train, X_test, Y_test, train=False)
```

```
Train Result:
================================================
Accuracy Score: 100.00%
_______________________________________________
CLASSIFICATION REPORT:
               0      1  accuracy  macro avg  weighted avg
precision    1.0    1.0       1.0        1.0           1.0
recall       1.0    1.0       1.0        1.0           1.0
f1-score     1.0    1.0       1.0        1.0           1.0
support    250.0  148.0       1.0      398.0         398.0
```

```
-----------------------------------------------
Confusion Matrix:
 [[250   0]
 [  0 148]]

Test Result:
================================================
Accuracy Score: 97.08%
-----------------------------------------------
CLASSIFICATION REPORT:
                    0          1  accuracy   macro avg  weighted avg
precision    0.990385   0.940299   0.97076    0.965342      0.971639
recall       0.962617   0.984375   0.97076    0.973496      0.970760
f1-score     0.976303   0.961832   0.97076    0.969068      0.970887
support    107.000000  64.000000   0.97076  171.000000    171.000000
-----------------------------------------------
Confusion Matrix:
 [[103   4]
 [  1  63]]
```

[39]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV

n_estimators = [int(x) for x in np.linspace(start=200, stop=2000, num=10)]
max_features = ['auto', 'sqrt']
max_depth = [int(x) for x in np.linspace(10, 110, num=11)]
max_depth.append(None)

min_samples_split = [2, 5, 10]
min_samples_leaf = [1, 2, 4]
bootstrap = [True, False]

random_grid = {
    'n_estimators': n_estimators,
    'max_features': max_features,
    'max_depth': max_depth,
    'min_samples_split': min_samples_split,
    'min_samples_leaf': min_samples_leaf,
    'bootstrap': bootstrap
}

ranFC = RandomForestClassifier(random_state=42)

randomizedSrchCV = RandomizedSearchCV(
    estimator=ranFC,
    scoring='f1',
```

```python
    param_distributions=random_grid,
    n_iter=200,
    cv=5,
    verbose=1,
    random_state=42,
    n_jobs=-1
)

randomizedSrchCV.fit(X_train, Y_train)
rf_best_params = randomizedSrchCV.best_params_
print(f"Best paramters: {rf_best_params})")

ranFC = RandomForestClassifier(**rf_best_params)
ranFC.fit(X_train, Y_train)

print_score(ranFC, X_train, Y_train, X_test, Y_test, train=True)
print_score(ranFC, X_train, Y_train, X_test, Y_test, train=False)
```

```
Fitting 5 folds for each of 200 candidates, totalling 1000 fits
Best paramters: {'n_estimators': 1000, 'min_samples_split': 2,
'min_samples_leaf': 1, 'max_features': 'auto', 'max_depth': 50, 'bootstrap':
False})
Train Result:
================================================
Accuracy Score: 100.00%
_______________________________________________
CLASSIFICATION REPORT:
               0      1  accuracy  macro avg  weighted avg
precision    1.0    1.0       1.0        1.0           1.0
recall       1.0    1.0       1.0        1.0           1.0
f1-score     1.0    1.0       1.0        1.0           1.0
support    250.0  148.0       1.0      398.0         398.0
_______________________________________________
Confusion Matrix:
 [[250   0]
 [  0 148]]

Test Result:
================================================
Accuracy Score: 96.49%
_______________________________________________
CLASSIFICATION REPORT:
                    0          1  accuracy   macro avg  weighted avg
precision    0.990291   0.926471  0.964912    0.958381      0.966405
recall       0.953271   0.984375  0.964912    0.968823      0.964912
f1-score     0.971429   0.954545  0.964912    0.962987      0.965110
support    107.000000  64.000000  0.964912  171.000000    171.000000
_______________________________________________
```

```
Confusion Matrix:
 [[102   5]
 [  1  63]]
```

[40]:
```python
n_estimators = [100, 500, 1000, 1500]
max_features = ['auto', 'sqrt']
max_depth = [2, 3, 5]
max_depth.append(None)
min_samples_split = [2, 5, 10]
min_samples_leaf = [1, 2, 4, 10]
bootstrap = [True, False]


params_grid = {
    'n_estimators': n_estimators,
    'max_features': max_features,
    'max_depth': max_depth,
    'min_samples_split': min_samples_split,
    'min_samples_leaf': min_samples_leaf,
    'bootstrap': bootstrap
}

ranGS = RandomForestClassifier(random_state=42)

ranGSCV = GridSearchCV(
    ranGS,
    params_grid,
    scoring="f1",
    cv=5,
    verbose=1,
    n_jobs=-1
)

ranGSCV.fit(X_train, Y_train)
best_params = ranGSCV.best_params_
print(f"Best parameters: {best_params}")

ranGS = RandomForestClassifier(**best_params)
ranGS.fit(X_train, Y_train)

print_score(ranGS, X_train, Y_train, X_test, Y_test, train=True)
print_score(ranGS, X_train, Y_train, X_test, Y_test, train=False)
```

```
Fitting 5 folds for each of 768 candidates, totalling 3840 fits
Best parameters: {'bootstrap': False, 'max_depth': None, 'max_features': 'auto',
'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 500}
Train Result:
```

```
================================================
Accuracy Score: 100.00%
_______________________________________________
CLASSIFICATION REPORT:
               0      1  accuracy  macro avg  weighted avg
precision    1.0    1.0       1.0        1.0           1.0
recall       1.0    1.0       1.0        1.0           1.0
f1-score     1.0    1.0       1.0        1.0           1.0
support    250.0  148.0       1.0      398.0         398.0
_______________________________________________
Confusion Matrix:
 [[250   0]
 [  0 148]]

Test Result:
================================================
Accuracy Score: 96.49%
_______________________________________________
CLASSIFICATION REPORT:
                    0          1  accuracy   macro avg  weighted avg
precision    0.990291   0.926471  0.964912    0.958381      0.966405
recall       0.953271   0.984375  0.964912    0.968823      0.964912
f1-score     0.971429   0.954545  0.964912    0.962987      0.965110
support    107.000000  64.000000  0.964912  171.000000    171.000000
_______________________________________________
Confusion Matrix:
 [[102   5]
 [  1  63]]
```

The suitable parameters to create best fit random forest model as per randomized search cross validation are : {'n_estimators': 1000, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_features': 'auto', 'max_depth': 50, 'bootstrap': False}

The suitable parameters to create best fit random forest model as per grid cross validation are : {'bootstrap': False, 'max_depth': None, 'max_features': 'auto', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 500}

Looking at the hyper parameter tuning results, both provided same hyperparameters variables except for the n_estimators. However, the overfitting issue is handled well in Randomized Search Cross Validation comparatively.

Summary

The result of how likely a person have the chances of developing cancer always depends on numerous factors like food, lifestyle, environment, kind of medication they take etc. However, we tried to do our best to predict with the diagnostic results.

As per the data visualisation results, the value range for beningn tumors were high compared to the malignant tumors. This shows that the size of the tumor have nothing to do with the final result.

With the limited & uncertain information we are able to predict a person with cancer with 98.4% accuracy and non cancer patients with 95.3% accuracy (based on the recall method) in this well balanced dataset. Here we have taken Recall results because, more than False Positives, False Negatives are pretty dangerous in this case.

Though we tried Logistic regression, Decision Tree and Random Forest, as per the accuracy range and the percentage of type I and Type II errors, we could conclude that the Random Forest regressor as our Final Model. In this we used Randomized search cross validation method to arrive at the best parameters to be passed in order to attain maximum accuracy / minimum error along with handling overfitting.