Lab 4 - Predicting Breast Cancer

Submitted By Name: Harsha KG Register Number: 19112005 Class: 5 BSc Data Science

Objectives

- Understand about Logistic Regression,K-Nearest Neighbours and Decision Trees

Problem Definition

Compare and Contrast the Differences in Classification Result among Logistic Regression, K-Nearest Neighbors and Decision Trees with respect to the Breast Cancer Dataset.Download the dataset as available in the URL: https://www.kaggle.com/uciml/breast-cancer-wisconsin-data (https://www.kaggle.com/uciml/breast-cancer-wisconsin-data)

Demonstrate various evaluation metrices Check the effect of classification with respect to change in train-test dataset, classification parameters, hyper parameters etc

Use appropriate visualizations and interpretations on result cases Do the standard practices, as discussed in class

Approach

Imported the dataset using the required library.Did some preprocessing techniques before exploration to make the datset into standard format and then did some EDA.After that datset is splitted into training and testing set.Build models of Logistic, K Nearest Neighour and Decision tree by changing the parameter values of each algorithm.

References:

1. Scikit Documentation
2. https://towardsdatascience.com/machine-learning-with-python-classification-complete-tutorial-d2c99dc524ec (https://towardsdatascience.com/machine-learning-with-python-classification-complete-tutorial-d2c99dc524ec)
3. https://stackabuse.com/overview-of-classification-methods-in-python-with-scikit-learn/ (https://stackabuse.com/overview-of-classification-methods-in-python-with-scikit-learn/) [Evaluation Metrics]

In [1]:

```
# Importing the Libraries
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from sklearn import metrics
import hvplot.pandas
import itertools
import plotly.graph_objs as go
import plotly.tools as tls
import plotly.figure_factory as ff
import plotly.offline as py

from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split

import warnings
warnings.filterwarnings("ignore")
```

In [2]:

```
BC = pd.read_csv("Breast_Cancer_Ml4.csv")
```

In [3]:

```
BC.head()
```

Out[3]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothnes
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	

5 rows × 32 columns

In [4]:

```
BC.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 32 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
dtypes: float64(30), int64(1), object(1)
memory usage: 142.4+ KB
```

In [5]:

```
BC.isnull().sum()
BC.isna().sum()
```

Out[5]:

```
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
```

In [6]:

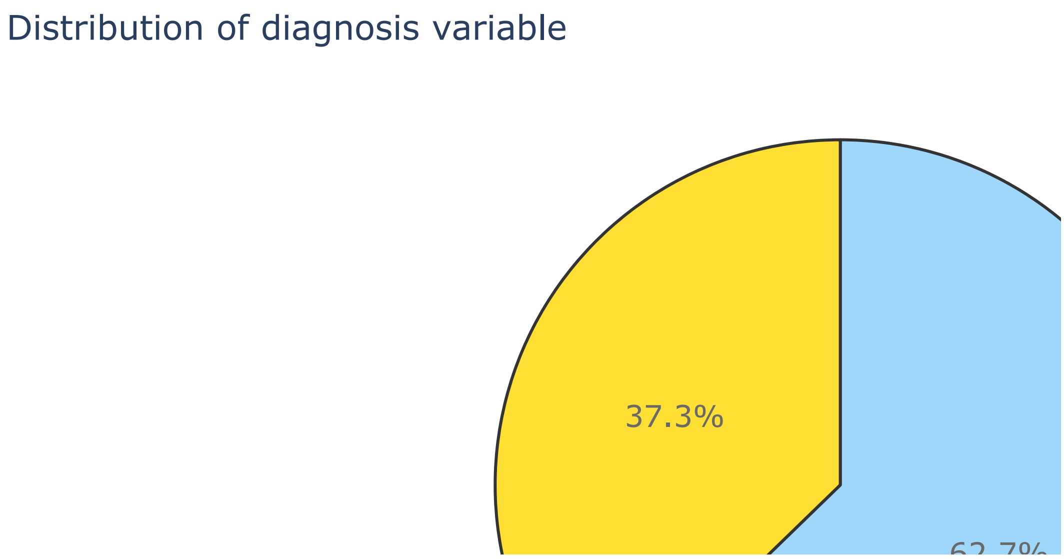
```
BC['diagnosis'].value_counts()
```

Out[6]:

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

In [7]:

```
#------------PERCENTAGE-------------------
trace = go.Pie(labels = ['benign','malignant'], values = BC['diagnosis'].value_counts
(),
               textfont=dict(size=15), opacity = 0.8,
               marker=dict(colors=['lightskyblue', 'gold'],
                           line=dict(color='#000000', width=1.5)))

layout = dict(title =  'Distribution of diagnosis variable')

fig = dict(data = [trace], layout=layout)
py.iplot(fig)
```



In [ ]:

```

```

In [8]:

```
#correlation
correlation = BC.corr()
#tick labels
matrix_cols = correlation.columns.tolist()
#convert to array
corr_array  = np.array(correlation)
```

In [9]:

```
#Plotting
trace = go.Heatmap(z = corr_array,
                   x = matrix_cols,
                   y = matrix_cols,
                   xgap = 2,
                   ygap = 2,
                   colorscale='Viridis',
                   colorbar   = dict() ,
                  )
layout = go.Layout(dict(title = 'Correlation Matrix for variables',
                        autosize = False,
                        height  = 720,
                        width   = 800,
                        margin  = dict(r = 0 ,l = 210,
                                       t = 25,b = 210,
                                     ),
                        yaxis   = dict(tickfont = dict(size = 9)),
                        xaxis   = dict(tickfont = dict(size = 9)),
                       )
                  )
fig = go.Figure(data = [trace],layout = layout)
py.iplot(fig)
```

Correlation Matrix for variables

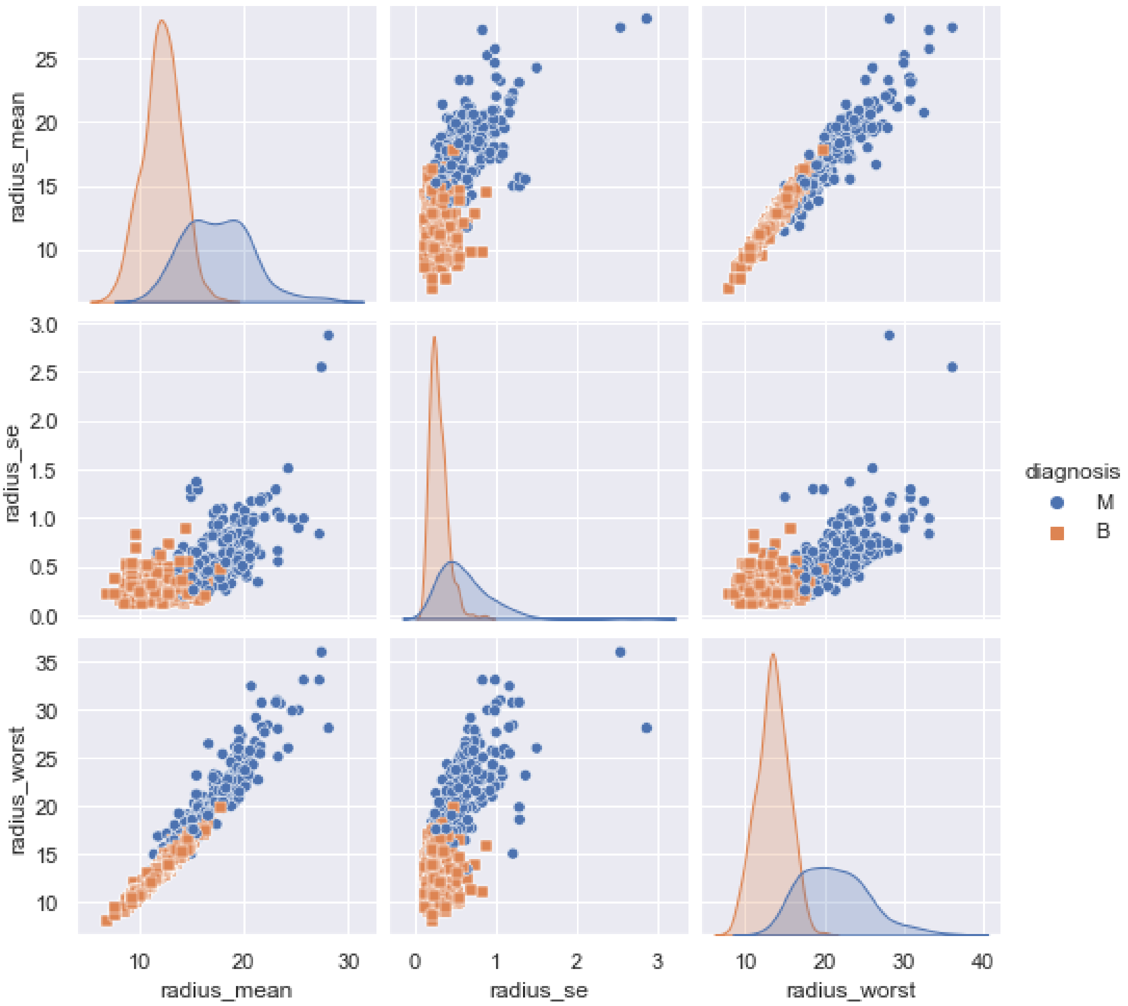
In [10]:

```
plt.figure(figsize = (20,10))
sns.set_theme(style="darkgrid")

radius = BC[['radius_mean','radius_se','radius_worst','diagnosis']]
sns.pairplot(radius, hue='diagnosis', markers=["o", "s"])
```

Out[10]:

```
<seaborn.axisgrid.PairGrid at 0x1a08f50ec10>
```

```
<Figure size 1440x720 with 0 Axes>
```



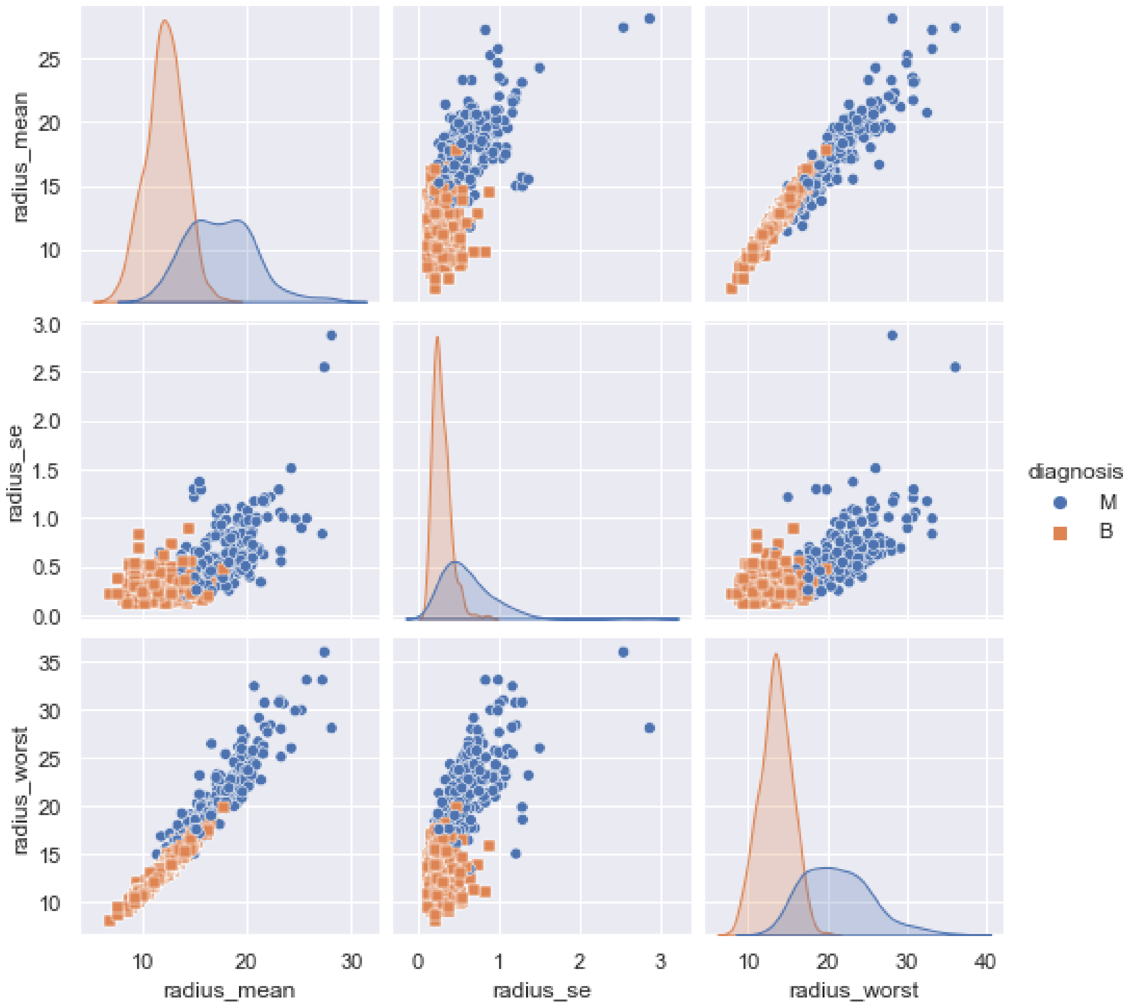
In [11]:

```
plt.figure(figsize = (20,10))
sns.set_theme(style="darkgrid")

radius = BC[['radius_mean','radius_se','radius_worst','diagnosis']]
sns.pairplot(radius, hue='diagnosis', markers=["o", "s"])
```

Out[11]:

```
<seaborn.axisgrid.PairGrid at 0x1a0b7b36970>
```

```
<Figure size 1440x720 with 0 Axes>
```



In [ ]:

In [ ]:

In [14]:

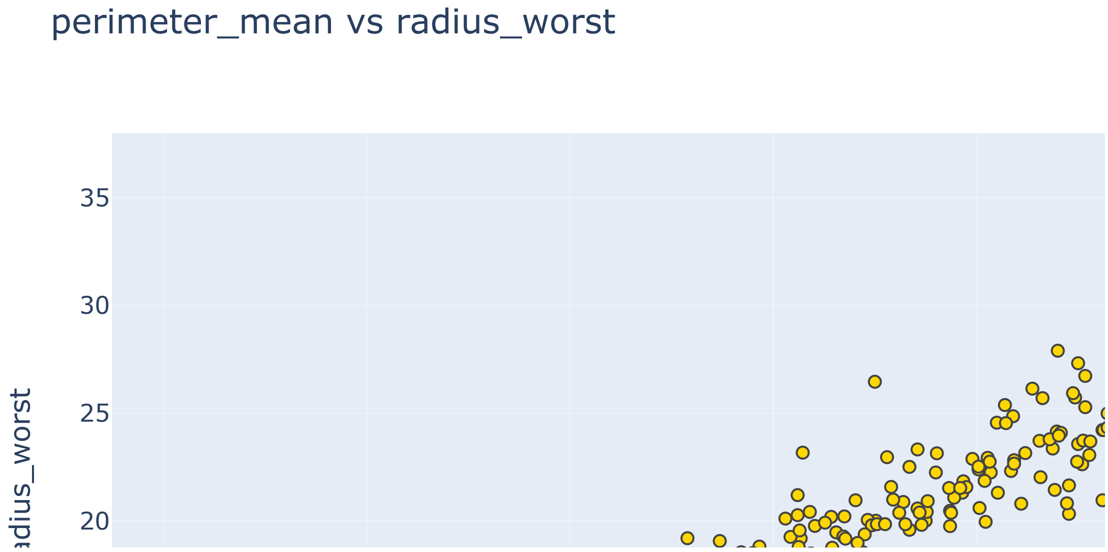
```
# 2 datasets
M = BC[(BC['diagnosis'] != 0)]
B = BC[(BC['diagnosis'] == 0)]
```

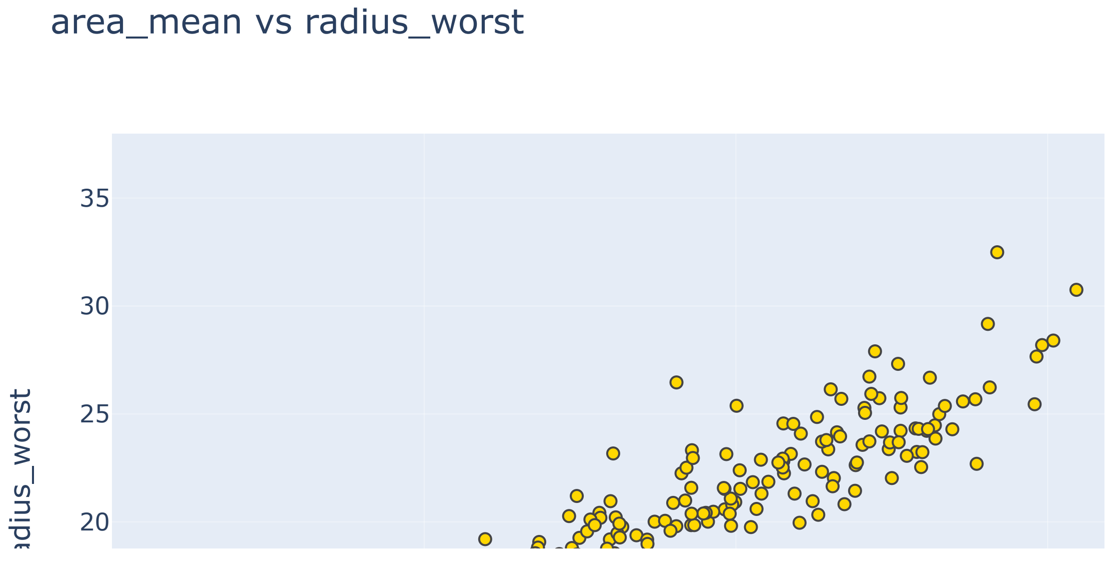
In [15]:

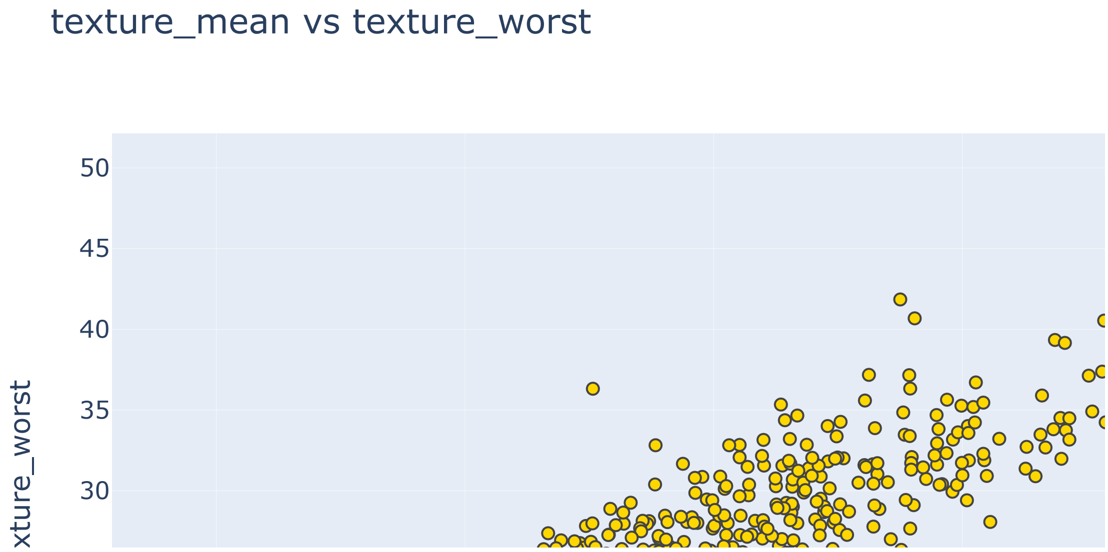
```
def plot_feat1_feat2(feat1, feat2) :
    trace0 = go.Scatter(
        x = M[feat1],
        y = M[feat2],
        name = 'malignant',
        mode = 'markers',
        marker = dict(color = '#FFD700',
            line = dict(
                width = 1)))

    trace1 = go.Scatter(
        x = B[feat1],
        y = B[feat2],
        name = 'benign',
        mode = 'markers',
        marker = dict(color = '#7EC0EE',
            line = dict(
                width = 1)))


    layout = dict(title = feat1 +" "+"vs"+" "+ feat2,
                  yaxis = dict(title = feat2,zeroline = False),
                  xaxis = dict(title = feat1, zeroline = False)
                 )

    plots = [trace0, trace1]

    fig = dict(data = plots, layout=layout)
    py.iplot(fig)
```

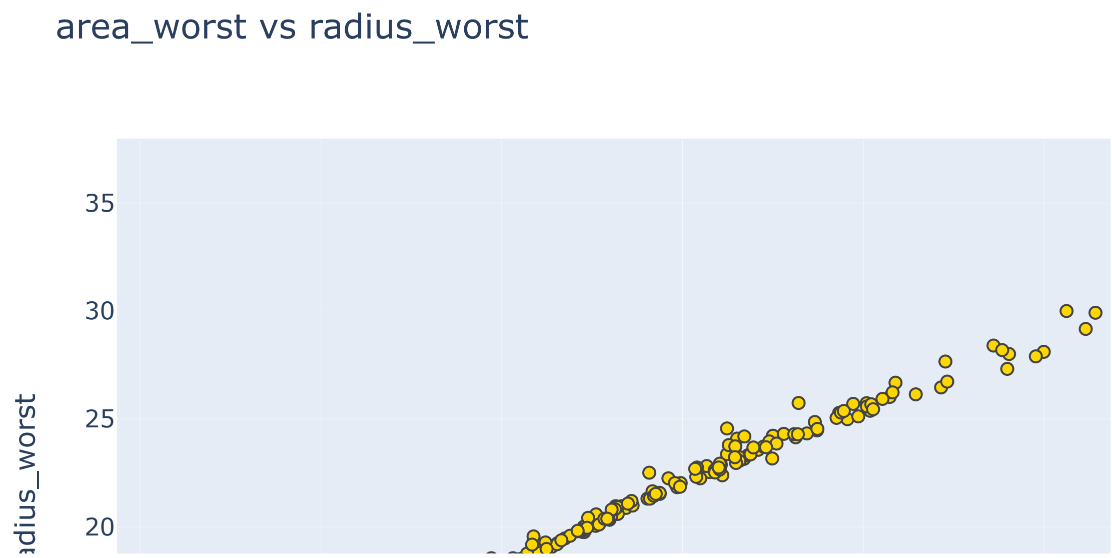
In [16]:

```
plot_feat1_feat2('perimeter_mean','radius_worst')
plot_feat1_feat2('area_mean','radius_worst')
plot_feat1_feat2('texture_mean','texture_worst')
plot_feat1_feat2('area_worst','radius_worst')
```


perimeter_mean vs radius_worst
35
30
25
20
adius_worst

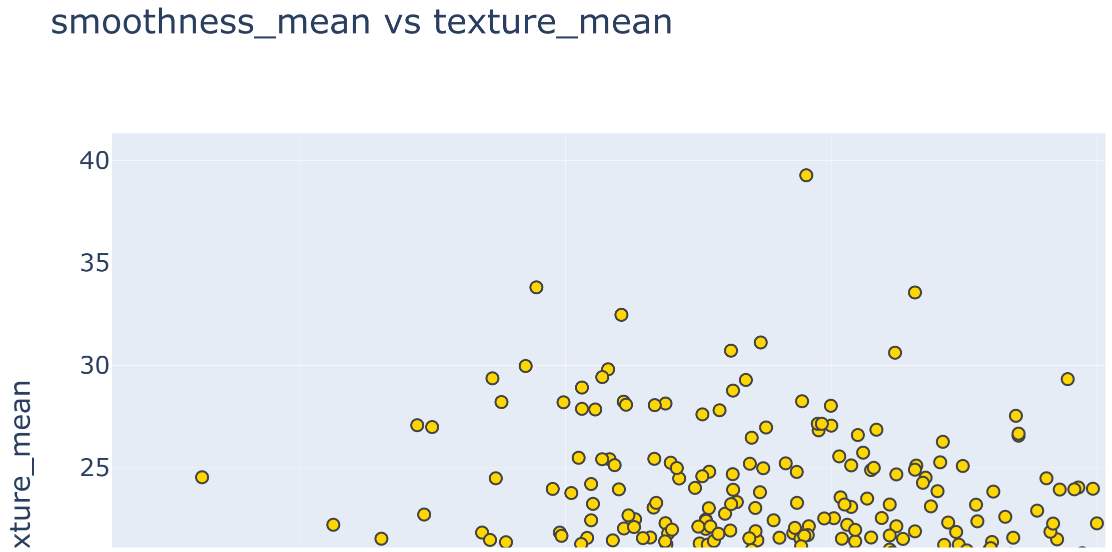

area_mean vs radius_worst
35
30
25
20
adius_worst

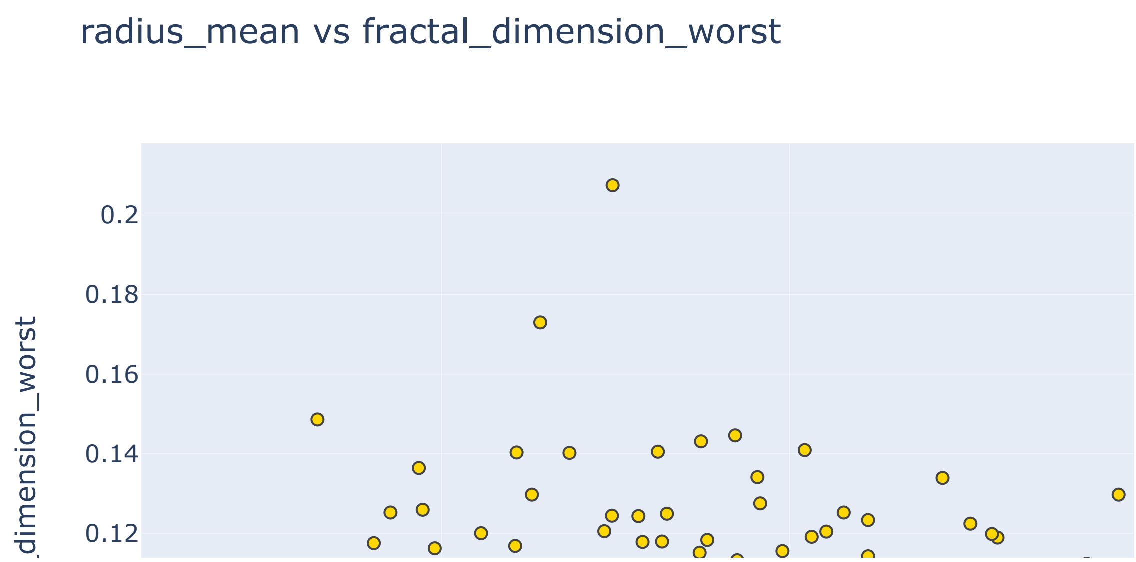

texture_mean vs texture_worst
50
45
40
35
30
xture_worst

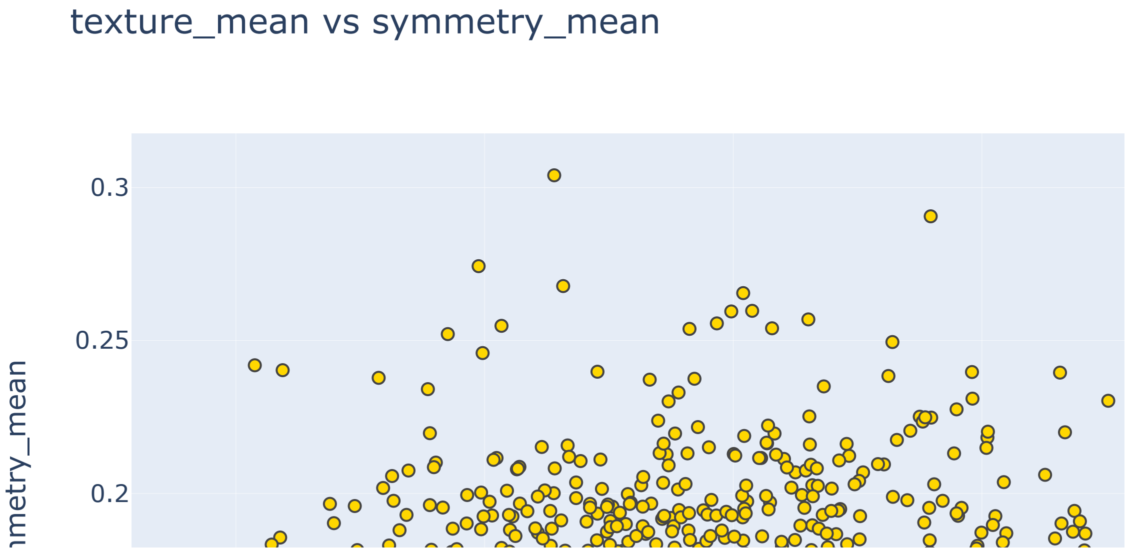

area_worst vs radius_worst
35
30
25
20
adius_worst

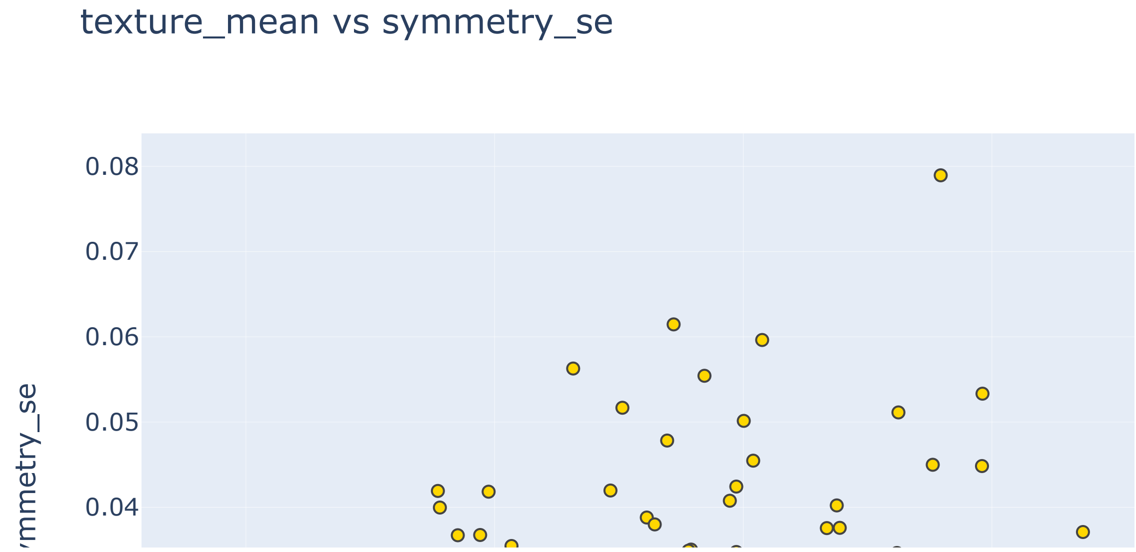
In [17]:

```
plot_feat1_feat2('smoothness_mean','texture_mean')
plot_feat1_feat2('radius_mean','fractal_dimension_worst')
plot_feat1_feat2('texture_mean','symmetry_mean')
plot_feat1_feat2('texture_mean','symmetry_se')
```


smoothness_mean vs texture_mean
40
35
30
25
xture_mean

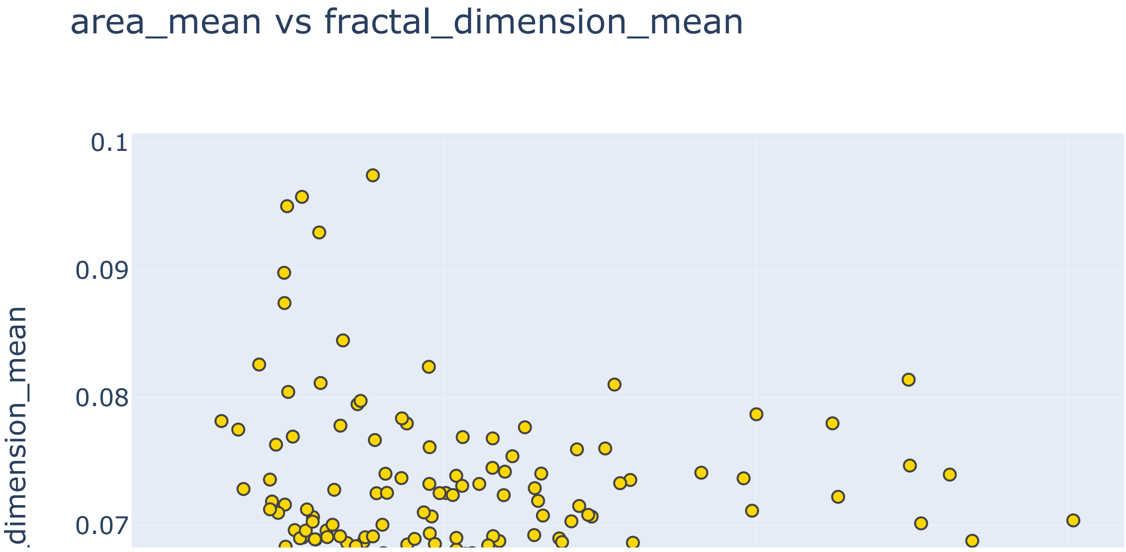

radius_mean vs fractal_dimension_worst
0.2
0.18
0.16
0.14
0.12
dimension_worst

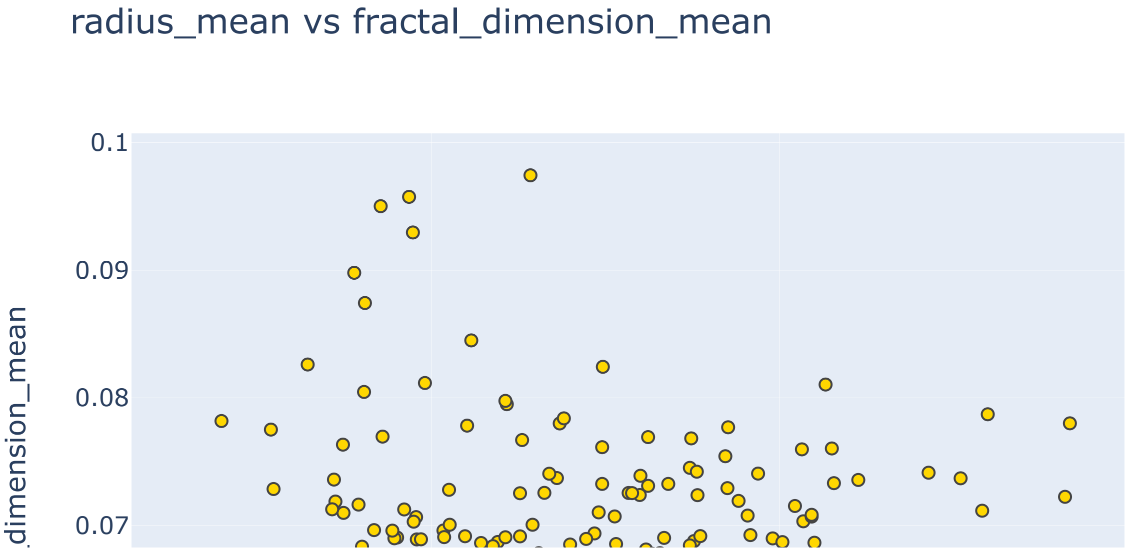

texture_mean vs symmetry_mean
0.3
0.25
0.2
mmetry_mean


texture_mean vs symmetry_se
0.08
0.07
0.06
0.05
0.04
ymmetry_se

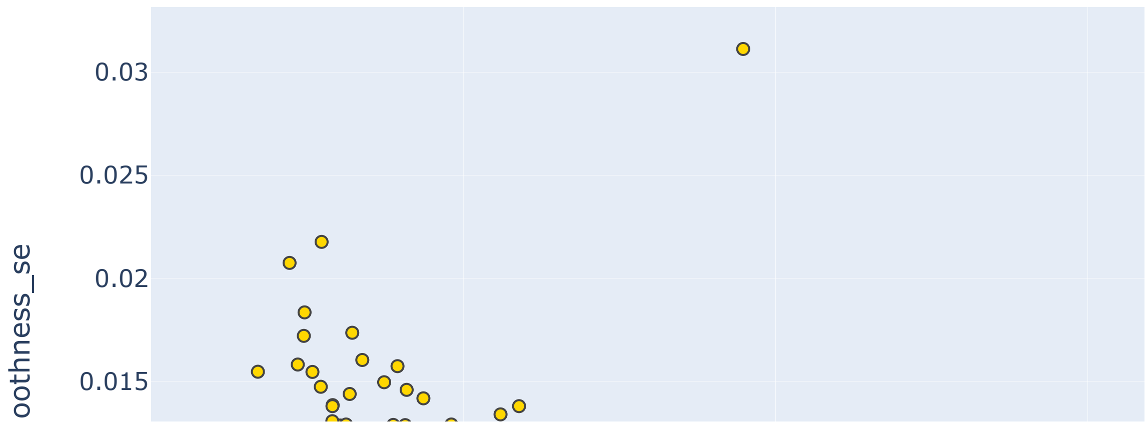
In [18]:

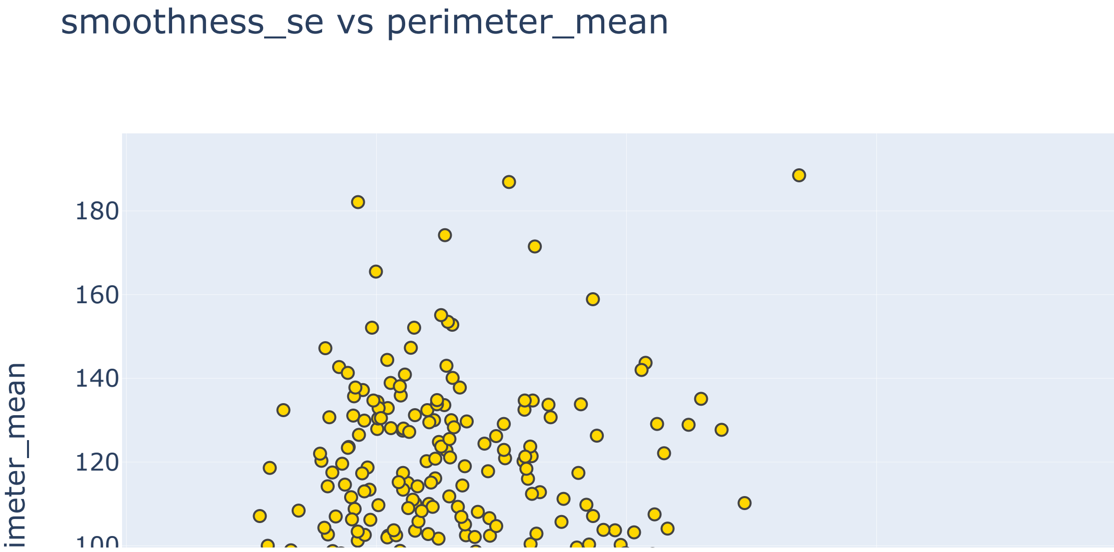
```
plot_feat1_feat2('area_mean','fractal_dimension_mean')
plot_feat1_feat2('radius_mean','fractal_dimension_mean')
plot_feat1_feat2('area_mean','smoothness_se')
plot_feat1_feat2('smoothness_se','perimeter_mean')
```


area_mean vs fractal_dimension_mean
0.1
0.09
0.08
0.07
dimension_mean


radius_mean vs fractal_dimension_mean
0.1
0.09
0.08
0.07
dimension_mean

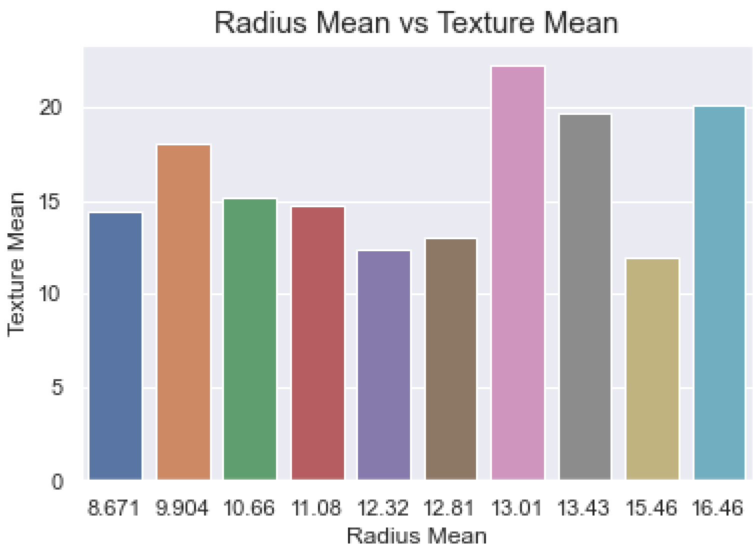
area_mean vs smoothness_se





In [19]:

```
sns.barplot(x="radius_mean", y="texture_mean", data=BC[170:180])
plt.title("Radius Mean vs Texture Mean",fontsize=15)
plt.xlabel("Radius Mean")
plt.ylabel("Texture Mean")
plt.show()
plt.style.use("ggplot")
```



In [20]:

```
BC.drop(columns='id',axis=1,inplace=True)
```

In [21]:

```
BC.head()
```

Out[21]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	c
0	M	17.99	10.38	122.80	1001.0	0.11840	
1	M	20.57	17.77	132.90	1326.0	0.08474	
2	M	19.69	21.25	130.00	1203.0	0.10960	
3	M	11.42	20.38	77.58	386.1	0.14250	
4	M	20.29	14.34	135.10	1297.0	0.10030	

5 rows × 31 columns

In [22]:

```
Y = BC['diagnosis']
X = BC.drop(['diagnosis'], axis=1)
```

In [23]:

```
Y.head()
```

Out[23]:

```
0    M
1    M
2    M
3    M
4    M
Name: diagnosis, dtype: object
```

In [24]:

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = .30, random_state
= 8)
```

In [25]:

```
X_train.head()
```

Out[25]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactne
547	10.26	16.58	65.85	320.8	0.08877	
331	12.98	19.35	84.52	514.0	0.09579	
421	14.69	13.98	98.22	656.1	0.10310	
95	20.26	23.03	132.40	1264.0	0.09078	
125	13.85	17.21	88.44	588.7	0.08785	

5 rows × 30 columns

In [26]:

```
X_test.head()
```

Out[26]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactne
325	12.670	17.30	81.25	489.9	0.10280	
557	9.423	27.88	59.26	271.3	0.08123	
475	12.830	15.73	82.89	506.9	0.09040	
308	13.500	12.71	85.69	566.2	0.07376	
553	9.333	21.94	59.01	264.0	0.09240	

5 rows × 30 columns

In [27]:

```
Y_test.head()
```

Out[27]:

```
325    B
557    B
475    B
308    B
553    B
Name: diagnosis, dtype: object
```

In [28]:

```
Y_train.head()
```

Out[28]:

```
547    B
331    B
421    B
95     M
125    B
Name: diagnosis, dtype: object
```

LogisticRegression

In [29]:

```
#Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [30]:

```
from sklearn.linear_model import LogisticRegression
```

In [31]:

```
logistic_regressor = LogisticRegression()
```

In [32]:

```
logistic_regressor.fit(X_train, Y_train)
```

Out[32]:

```
LogisticRegression()
```

In [33]:

```
y_pred = logistic_regressor.predict(X_test)
y_prob = logistic_regressor.predict_proba(X_test)
```

In [34]:

```
y_pred[0:5]
```

Out[34]:

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

In [35]:

```
data = pd.DataFrame({'Actual': Y_test, 'Predicted': y_pred})
data
```

Out[35]:

	Actual	Predicted
325	B	B
557	B	B
475	B	B
308	B	B
553	B	B
...	...	...
154	B	B
208	B	B
311	B	B
329	M	M
282	M	M

171 rows × 2 columns

In [36]:

```
y_prob[0:5]
```

Out[36]:

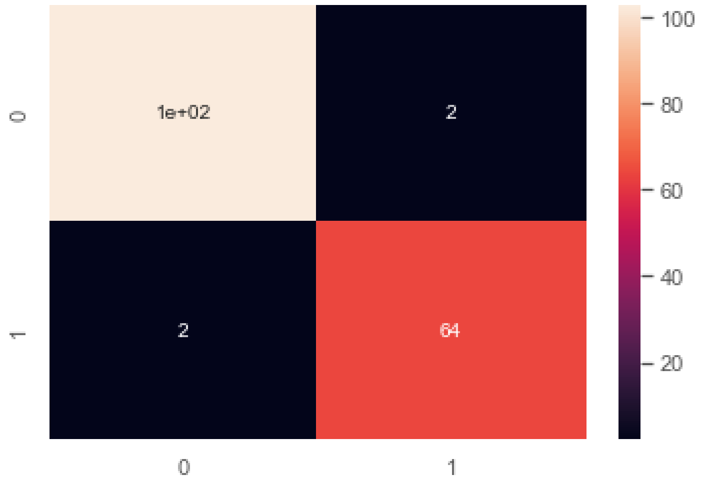
```
array([[9.98588993e-01, 1.41100703e-03],
       [9.99647709e-01, 3.52291383e-04],
       [9.92574976e-01, 7.42502412e-03],
       [9.99968116e-01, 3.18836820e-05],
       [9.99978480e-01, 2.15199463e-05]])
```

In [37]:

```
from sklearn.metrics import confusion_matrix
conf_matrix = confusion_matrix(Y_test, y_pred)
dataframe_conf_matrix = conf_matrix
sns.heatmap(dataframe_conf_matrix, annot=True)
```

Out[37]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a093de3070>
```



In [38]:

```
from sklearn.metrics import accuracy_score
acc_score1 = accuracy_score(Y_test, y_pred)
print(acc_score1)
```

```
0.9766081871345029
```

In [39]:

```
print("Training Score: ",logistic_regressor.score(X_train,Y_train)*100)
```

```
Training Score:  98.99497487437185
```

In [40]:

```
print("Testing Score: ",logistic_regressor.score(X_test,Y_test)*100)
```

```
Testing Score:  97.6608187134503
```

In [41]:

```
from sklearn.metrics import classification_report
class_report = classification_report(Y_test, y_pred)
print(class_report)
```

```
              precision    recall  f1-score   support

           B       0.98      0.98      0.98       105
           M       0.97      0.97      0.97        66

    accuracy                           0.98       171
   macro avg       0.98      0.98      0.98       171
weighted avg       0.98      0.98      0.98       171
```

In [42]:

```
from sklearn.metrics import confusion_matrix
Cm = confusion_matrix(Y_test,y_pred)
Cm
```

Out[42]:

```
array([[103,   2],
       [  2,  64]], dtype=int64)
```

In [43]:

```
Accuracy = (Cm[0][0] + Cm[1][1]) / (Cm[0][0] + Cm[1][1] + Cm[0][1] + Cm[1][0])
print("Accuracy",Accuracy)
Error_rate = (Cm[0][1] + Cm[1][0]) / (Cm[0][0] + Cm[1][1] + Cm[0][1] + Cm[1][0])
print("Error_rate",Error_rate)
Sensitivity = Cm[0][0]/(Cm[0][0] + Cm[1][0])
print("Sensitivity",Sensitivity)
Specificity = Cm[1][1]/(Cm[1][1] + Cm[0][1])
print("Specificity",Specificity)
Recall = Cm[0][0]/(Cm[0][0] + Cm[1][0])
print("Recall",Recall)
Precision = Cm[0][0]/(Cm[0][0] + Cm[0][1])
print("Precision",Precision)
F1Score = (2*(Precision*Recall))/(Precision + Recall)
print("F1Score",F1Score)
```

```
Accuracy 0.9766081871345029
Error_rate 0.023391812865497075
Sensitivity 0.9809523809523809
Specificity 0.9696969696969697
Recall 0.9809523809523809
Precision 0.9809523809523809
F1Score 0.9809523809523809
```

In [44]:

```
def doLogisticRegression(X, Y, test_size = 0.20, random_state = 42, penalty='l2', solve
r='lbfgs'):
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = test_size, ra
ndom_state = random_state)

    logistic_regressor = LogisticRegression(penalty=penalty, solver=solver)
    logistic_regressor.fit(X_train, Y_train)
    y_pred = logistic_regressor.predict(X_test)

    acc_score = accuracy_score(Y_test, y_pred)

    return acc_score
```

In [45]:

```
penalties = ['none', 'l2']
test_size = [0.30, 0.25, 0.20]
random_states = [10, 25, 55]
solvers = ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']

for t_size in test_size:
    for r_state in random_states:
        for penalty in penalties:
            for solver in solvers:
                accuracy = doLogisticRegression(X, Y, t_size, r_state, penalty)
                print("Test: {} | Random State: {} | Penalty: {} | Solver: {} | Accurac
y : {}".format(t_size, r_state, penalty, solver, accuracy))
```

```
Test: 0.3 | Random State: 10 | Penalty: none | Solver: newton-cg | Accurac
y : 0.9415204678362573
Test: 0.3 | Random State: 10 | Penalty: none | Solver: lbfgs | Accuracy :
0.9415204678362573
Test: 0.3 | Random State: 10 | Penalty: none | Solver: liblinear | Accurac
y : 0.9415204678362573
Test: 0.3 | Random State: 10 | Penalty: none | Solver: sag | Accuracy : 0.
9415204678362573
Test: 0.3 | Random State: 10 | Penalty: none | Solver: saga | Accuracy :
0.9415204678362573
Test: 0.3 | Random State: 10 | Penalty: l2 | Solver: newton-cg | Accuracy
: 0.9473684210526315
Test: 0.3 | Random State: 10 | Penalty: l2 | Solver: lbfgs | Accuracy : 0.
9473684210526315
Test: 0.3 | Random State: 10 | Penalty: l2 | Solver: liblinear | Accuracy
: 0.9473684210526315
Test: 0.3 | Random State: 10 | Penalty: l2 | Solver: sag | Accuracy : 0.94
73684210526315
Test: 0.3 | Random State: 10 | Penalty: l2 | Solver: saga | Accuracy : 0.9
473684210526315
Test: 0.3 | Random State: 25 | Penalty: none | Solver: newton-cg | Accurac
y : 0.9298245614035088
Test: 0.3 | Random State: 25 | Penalty: none | Solver: lbfgs | Accuracy :
0.9298245614035088
Test: 0.3 | Random State: 25 | Penalty: none | Solver: liblinear | Accurac
y : 0.9298245614035088
Test: 0.3 | Random State: 25 | Penalty: none | Solver: sag | Accuracy : 0.
9298245614035088
Test: 0.3 | Random State: 25 | Penalty: none | Solver: saga | Accuracy :
0.9298245614035088
Test: 0.3 | Random State: 25 | Penalty: l2 | Solver: newton-cg | Accuracy
: 0.9239766081871345
Test: 0.3 | Random State: 25 | Penalty: l2 | Solver: lbfgs | Accuracy : 0.
9239766081871345
Test: 0.3 | Random State: 25 | Penalty: l2 | Solver: liblinear | Accuracy
: 0.9239766081871345
Test: 0.3 | Random State: 25 | Penalty: l2 | Solver: sag | Accuracy : 0.92
39766081871345
Test: 0.3 | Random State: 25 | Penalty: l2 | Solver: saga | Accuracy : 0.9
239766081871345
Test: 0.3 | Random State: 55 | Penalty: none | Solver: newton-cg | Accurac
y : 0.9532163742690059
Test: 0.3 | Random State: 55 | Penalty: none | Solver: lbfgs | Accuracy :
0.9532163742690059
Test: 0.3 | Random State: 55 | Penalty: none | Solver: liblinear | Accurac
y : 0.9532163742690059
Test: 0.3 | Random State: 55 | Penalty: none | Solver: sag | Accuracy : 0.
9532163742690059
Test: 0.3 | Random State: 55 | Penalty: none | Solver: saga | Accuracy :
0.9532163742690059
Test: 0.3 | Random State: 55 | Penalty: l2 | Solver: newton-cg | Accuracy
: 0.9532163742690059
Test: 0.3 | Random State: 55 | Penalty: l2 | Solver: lbfgs | Accuracy : 0.
9532163742690059
Test: 0.3 | Random State: 55 | Penalty: l2 | Solver: liblinear | Accuracy
: 0.9532163742690059
Test: 0.3 | Random State: 55 | Penalty: l2 | Solver: sag | Accuracy : 0.95
32163742690059
Test: 0.3 | Random State: 55 | Penalty: l2 | Solver: saga | Accuracy : 0.9
532163742690059
Test: 0.25 | Random State: 10 | Penalty: none | Solver: newton-cg | Accura
```

```
cy : 0.9300699300699301
Test: 0.25 | Random State: 10 | Penalty: none | Solver: lbfgs | Accuracy :
0.9300699300699301
Test: 0.25 | Random State: 10 | Penalty: none | Solver: liblinear | Accura
cy : 0.9300699300699301
Test: 0.25 | Random State: 10 | Penalty: none | Solver: sag | Accuracy :
0.9300699300699301
Test: 0.25 | Random State: 10 | Penalty: none | Solver: saga | Accuracy :
0.9300699300699301
Test: 0.25 | Random State: 10 | Penalty: l2 | Solver: newton-cg | Accuracy
: 0.9300699300699301
Test: 0.25 | Random State: 10 | Penalty: l2 | Solver: lbfgs | Accuracy :
0.9300699300699301
Test: 0.25 | Random State: 10 | Penalty: l2 | Solver: liblinear | Accuracy
: 0.9300699300699301
Test: 0.25 | Random State: 10 | Penalty: l2 | Solver: sag | Accuracy : 0.9
300699300699301
Test: 0.25 | Random State: 10 | Penalty: l2 | Solver: saga | Accuracy : 0.
9300699300699301
Test: 0.25 | Random State: 25 | Penalty: none | Solver: newton-cg | Accura
cy : 0.916083916083916
Test: 0.25 | Random State: 25 | Penalty: none | Solver: lbfgs | Accuracy :
0.916083916083916
Test: 0.25 | Random State: 25 | Penalty: none | Solver: liblinear | Accura
cy : 0.916083916083916
Test: 0.25 | Random State: 25 | Penalty: none | Solver: sag | Accuracy :
0.916083916083916
Test: 0.25 | Random State: 25 | Penalty: none | Solver: saga | Accuracy :
0.916083916083916
Test: 0.25 | Random State: 25 | Penalty: l2 | Solver: newton-cg | Accuracy
: 0.916083916083916
Test: 0.25 | Random State: 25 | Penalty: l2 | Solver: lbfgs | Accuracy :
0.916083916083916
Test: 0.25 | Random State: 25 | Penalty: l2 | Solver: liblinear | Accuracy
: 0.916083916083916
Test: 0.25 | Random State: 25 | Penalty: l2 | Solver: sag | Accuracy : 0.9
16083916083916
Test: 0.25 | Random State: 25 | Penalty: l2 | Solver: saga | Accuracy : 0.
916083916083916
Test: 0.25 | Random State: 55 | Penalty: none | Solver: newton-cg | Accura
cy : 0.9300699300699301
Test: 0.25 | Random State: 55 | Penalty: none | Solver: lbfgs | Accuracy :
0.9300699300699301
Test: 0.25 | Random State: 55 | Penalty: none | Solver: liblinear | Accura
cy : 0.9300699300699301
Test: 0.25 | Random State: 55 | Penalty: none | Solver: sag | Accuracy :
0.9300699300699301
Test: 0.25 | Random State: 55 | Penalty: none | Solver: saga | Accuracy :
0.9300699300699301
Test: 0.25 | Random State: 55 | Penalty: l2 | Solver: newton-cg | Accuracy
: 0.9300699300699301
Test: 0.25 | Random State: 55 | Penalty: l2 | Solver: lbfgs | Accuracy :
0.9300699300699301
Test: 0.25 | Random State: 55 | Penalty: l2 | Solver: liblinear | Accuracy
: 0.9300699300699301
Test: 0.25 | Random State: 55 | Penalty: l2 | Solver: sag | Accuracy : 0.9
300699300699301
Test: 0.25 | Random State: 55 | Penalty: l2 | Solver: saga | Accuracy : 0.
9300699300699301
Test: 0.2 | Random State: 10 | Penalty: none | Solver: newton-cg | Accurac
y : 0.9385964912280702
```

```
Test: 0.2 | Random State: 10 | Penalty: none | Solver: lbfgs | Accuracy :
0.9385964912280702
Test: 0.2 | Random State: 10 | Penalty: none | Solver: liblinear | Accurac
y : 0.9385964912280702
Test: 0.2 | Random State: 10 | Penalty: none | Solver: sag | Accuracy : 0.
9385964912280702
Test: 0.2 | Random State: 10 | Penalty: none | Solver: saga | Accuracy :
0.9385964912280702
Test: 0.2 | Random State: 10 | Penalty: l2 | Solver: newton-cg | Accuracy
: 0.9298245614035088
Test: 0.2 | Random State: 10 | Penalty: l2 | Solver: lbfgs | Accuracy : 0.
9298245614035088
Test: 0.2 | Random State: 10 | Penalty: l2 | Solver: liblinear | Accuracy
: 0.9298245614035088
Test: 0.2 | Random State: 10 | Penalty: l2 | Solver: sag | Accuracy : 0.92
98245614035088
Test: 0.2 | Random State: 10 | Penalty: l2 | Solver: saga | Accuracy : 0.9
298245614035088
Test: 0.2 | Random State: 25 | Penalty: none | Solver: newton-cg | Accurac
y : 0.9210526315789473
Test: 0.2 | Random State: 25 | Penalty: none | Solver: lbfgs | Accuracy :
0.9210526315789473
Test: 0.2 | Random State: 25 | Penalty: none | Solver: liblinear | Accurac
y : 0.9210526315789473
Test: 0.2 | Random State: 25 | Penalty: none | Solver: sag | Accuracy : 0.
9210526315789473
Test: 0.2 | Random State: 25 | Penalty: none | Solver: saga | Accuracy :
0.9210526315789473
Test: 0.2 | Random State: 25 | Penalty: l2 | Solver: newton-cg | Accuracy
: 0.9385964912280702
Test: 0.2 | Random State: 25 | Penalty: l2 | Solver: lbfgs | Accuracy : 0.
9385964912280702
Test: 0.2 | Random State: 25 | Penalty: l2 | Solver: liblinear | Accuracy
: 0.9385964912280702
Test: 0.2 | Random State: 25 | Penalty: l2 | Solver: sag | Accuracy : 0.93
85964912280702
Test: 0.2 | Random State: 25 | Penalty: l2 | Solver: saga | Accuracy : 0.9
385964912280702
Test: 0.2 | Random State: 55 | Penalty: none | Solver: newton-cg | Accurac
y : 0.9210526315789473
Test: 0.2 | Random State: 55 | Penalty: none | Solver: lbfgs | Accuracy :
0.9210526315789473
Test: 0.2 | Random State: 55 | Penalty: none | Solver: liblinear | Accurac
y : 0.9210526315789473
Test: 0.2 | Random State: 55 | Penalty: none | Solver: sag | Accuracy : 0.
9210526315789473
Test: 0.2 | Random State: 55 | Penalty: none | Solver: saga | Accuracy :
0.9210526315789473
Test: 0.2 | Random State: 55 | Penalty: l2 | Solver: newton-cg | Accuracy
: 0.9210526315789473
Test: 0.2 | Random State: 55 | Penalty: l2 | Solver: lbfgs | Accuracy : 0.
9210526315789473
Test: 0.2 | Random State: 55 | Penalty: l2 | Solver: liblinear | Accuracy
: 0.9210526315789473
Test: 0.2 | Random State: 55 | Penalty: l2 | Solver: sag | Accuracy : 0.92
10526315789473
Test: 0.2 | Random State: 55 | Penalty: l2 | Solver: saga | Accuracy : 0.9
210526315789473
```

In [46]:

```
BC_1= pd.DataFrame(columns = ['Test Size', 'Random States', 'Penalty', 'Solvers', 'Accu
racy'])
BC_1
```

Out[46]:

Test Size	Random States	Penalty	Solvers	Accuracy

In [48]:

```
penalties = ['none', 'l2']
test_size = [0.30, 0.25, 0.20]
random_states = [10, 25, 55]
solvers = ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']
for t_size in test_size:
    for r_state in random_states:
        for penalty in penalties:
            for solver in solvers:
                accuracy = doLogisticRegression(X, Y, t_size, r_state, penalty)


 #print("Test: {} | Random State: {} | Penalty: {} | Solver: {} | Accuracy : {}".format
(t_size, r_state, penalty, solver, accuracy))

                BCEvaluation = {}
                BCEvaluation['Test Size'] = t_size
                BCEvaluation['Random States'] = r_state
                BCEvaluation['Penalty'] = penalty
                BCEvaluation['Solvers'] = solver
                BCEvaluation['Accuracy'] = accuracy
                BC_1= BC_1.append(BCEvaluation, ignore_index = True)
```

In [49]:

```
BC_1
```

Out[49]:

	Test Size	Random States	Penalty	Solvers	Accuracy
0	0.3	10	none	newton-cg	0.941520
1	0.3	10	none	lbfgs	0.941520
2	0.3	10	none	liblinear	0.941520
3	0.3	10	none	sag	0.941520
4	0.3	10	none	saga	0.941520
...	...	...	...	...	...
175	0.2	55	l2	newton-cg	0.921053
176	0.2	55	l2	lbfgs	0.921053
177	0.2	55	l2	liblinear	0.921053
178	0.2	55	l2	sag	0.921053
179	0.2	55	l2	saga	0.921053

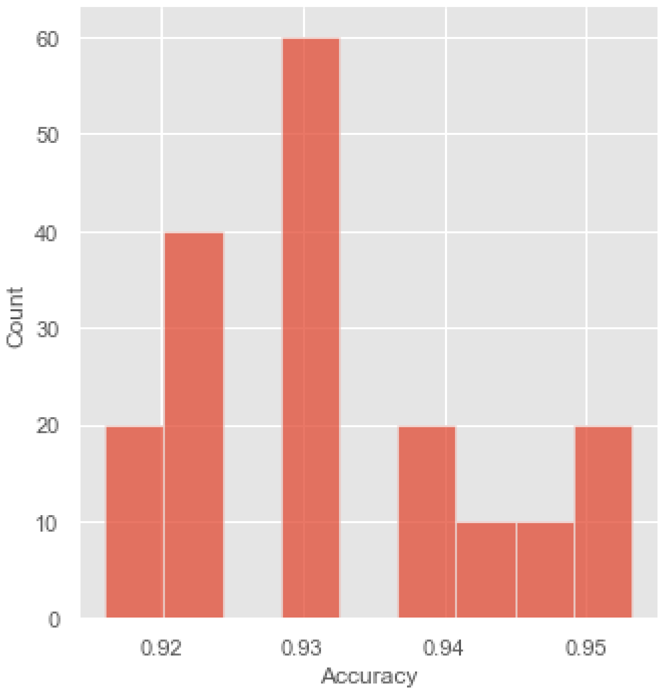
180 rows × 5 columns

In [50]:

```
sns.displot(x = 'Accuracy', data = BC_1)
```

Out[50]:

```
<seaborn.axisgrid.FacetGrid at 0x1a093ed4c10>
```



K-Nearest Neighbors

In [80]:

```
from sklearn.neighbors import KNeighborsClassifier
classifier= KNeighborsClassifier()
classifier.fit(X_train, Y_train)
```

Out[80]:

```
KNeighborsClassifier()
```

In [81]:

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

In [82]:

```
from sklearn.metrics import accuracy_score
acc_score2 = accuracy_score(Y_test, y_pred)
print(acc_score2)
```

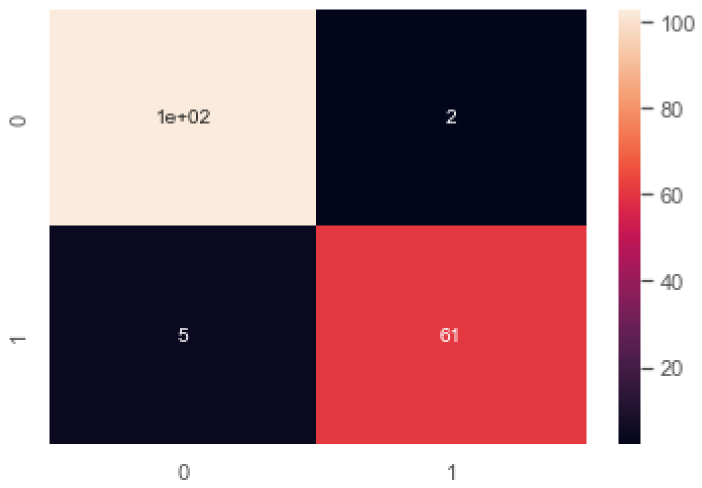
```
0.9590643274853801
```

In [83]:

```
from sklearn.metrics import confusion_matrix
conf_matrix = confusion_matrix(Y_test, y_pred)
dataframe_conf_matrix = conf_matrix
sns.heatmap(dataframe_conf_matrix, annot=True)
```

Out[83]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a0939d98b0>
```



In [84]:

```
Accuracy = (Cm[0][0] + Cm[1][1]) / (Cm[0][0] + Cm[1][1] + Cm[0][1] + Cm[1][0])
print("Accuracy",Accuracy)
Error_rate = (Cm[0][1] + Cm[1][0]) / (Cm[0][0] + Cm[1][1] + Cm[0][1] + Cm[1][0])
print("Error_rate",Error_rate)
Sensitivity = Cm[0][0]/(Cm[0][0] + Cm[1][0])
print("Sensitivity",Sensitivity)
Specificity = Cm[1][1]/(Cm[1][1] + Cm[0][1])
print("Specificity",Specificity)
Recall = Cm[0][0]/(Cm[0][0] + Cm[1][0])
print("Recall",Recall)
Precision = Cm[0][0]/(Cm[0][0] + Cm[0][1])
print("Precision",Precision)
F1Score = (2*(Precision*Recall))/(Precision + Recall)
print("F1Score",F1Score)
```

```
Accuracy 0.9766081871345029
Error_rate 0.023391812865497075
Sensitivity 0.9809523809523809
Specificity 0.9696969696969697
Recall 0.9809523809523809
Precision 0.9809523809523809
F1Score 0.9809523809523809
```

In [85]:

```
error_rate = []

# Will take some time
for i in range(1, 40):

    knn = KNeighborsClassifier(n_neighbors = i)
    knn.fit(X_train, Y_train)
    pred_i = knn.predict(X_test)
    error_rate.append(np.mean(pred_i != Y_test))

plt.figure(figsize =(10, 6))
plt.plot(range(1, 40), error_rate, color ='blue',
                linestyle ='dashed', marker ='o',
         markerfacecolor ='red', markersize = 10)

plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
```

Out[85]:

```
Text(0, 0.5, 'Error Rate')
```

In [128]:

```
BC_2=pd.DataFrame(columns=['n_neighbour','leaf_size','Accuracy'])
```

In [129]:

```
BC_2
```

Out[129]:

n_neighbour	leaf_size	Accuracy

In [154]:

```
def doKNeighborsClassifier(X, Y, test_size = 0.20, random_state = 42):
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = test_size, ra
ndom_state = random_state)

    KNeighbors_Classifier =  KNeighborsClassifier()
    KNeighbors_Classifier.fit(X_train, Y_train)
    y_pred = KNeighbors_Classifier.predict(X_test)

    acc_score = accuracy_score(Y_test, y_pred)
```

In [156]:

```
n_neighbour = [5, 25, 35]
leaf_size = [10, 25, 55]

for neighbour in n_neighbour:
    for leaf in leaf_size:
        accuracy = doKNeighborsClassifier(X, Y,neighbour,leaf)

        BCEvaluation_KNN = {}
        BCEvaluation_KNN['n_neighbour'] = neighbour
        BCEvaluation_KNN['leaf_size'] = leaf
        BCEvaluation_KNN['Accuracy'] = accuracy
        BC_2= BC_2.append(BCEvaluation_KNN, ignore_index = True)
```

Decision Tree

In [98]:

```
#Fitting Decision Tree classifier to the training set
from sklearn.tree import DecisionTreeClassifier
classifier= DecisionTreeClassifier()
classifier.fit(X_train, Y_train)
```

Out[98]:

```
DecisionTreeClassifier()
```

In [99]:

```
#Predicting the test set result
y_pred= classifier.predict(X_test)
y_pred
```

Out[99]:

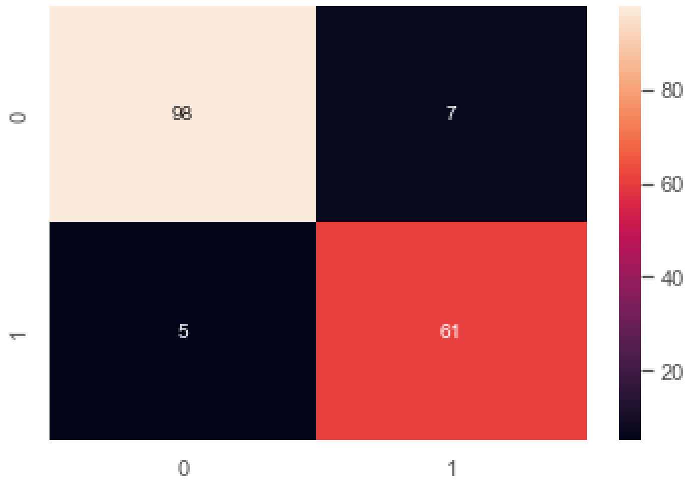
```
array(['B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
       'M', 'M', 'B', 'B', 'M', 'B', 'M', 'M', 'M', 'M', 'B', 'B', 'M',
       'M', 'M', 'B', 'B', 'B', 'B', 'M', 'M', 'M', 'B', 'M', 'M', 'B',
       'M', 'B', 'M', 'M', 'B', 'M', 'B', 'M', 'M', 'M', 'M', 'B', 'M',
       'B',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

In [135]:

```
from sklearn.metrics import confusion_matrix
conf_matrix = confusion_matrix(Y_test, y_pred)
dataframe_conf_matrix = conf_matrix
sns.heatmap(dataframe_conf_matrix, annot=True)
```

Out[135]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a0942b3b20>
```



In [100]:

```
#Creating the Confusion matrix
from sklearn.metrics import confusion_matrix
cm= confusion_matrix(Y_test, y_pred)
cm
```

Out[100]:

```
array([[98,  7],
       [ 5, 61]], dtype=int64)
```

In [101]:

```
from sklearn.metrics import accuracy_score
acc_score3 = accuracy_score(Y_test, y_pred)
print(acc_score3)
```

0.9298245614035088

In [139]:

```
def doDecisionTreeClassifier(X, Y, test_size = 0.20, random_state = 42):
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = test_size, ra
ndom_state = random_state)
    classifier= DecisionTreeClassifier()
    classifier.fit(X_train, Y_train)
    y_pred= classifier.predict(X_test)

    acc_score = accuracy_score(Y_test, y_pred)
```

In [143]:

```
BC_3= pd.DataFrame(columns = ['Test Size', 'Random States'])
BC_3
```

Out[143]:

Test Size	Random States

In [144]:

```
test_size = [0.30, 0.25, 0.20]
random_states = [10, 25, 55]
for t_size in test_size:
    for r_state in random_states:
                accuracy = doDecisionTreeClassifier(X, Y, t_size, r_state)


 #print("Test: {} | Random State: {} | Penalty: {} | Solver: {} | Accuracy : {}".format
(t_size, r_state, penalty, solver, accuracy))

                BCEvaluation = {}
                BCEvaluation['Test Size'] = t_size
                BCEvaluation['Random States'] = r_state
                BCEvaluation['Accuracy'] = accuracy
                BC_3= BC_3.append(BCEvaluation, ignore_index = True)
```

Accuracy Score of Models

In [102]:

```
prediction_columns = ["NAME OF MODEL", "ACCURACY SCORE"]
df_pred = {"NAME OF MODEL" : ["LOGISTIC REGRESSION", "K-NN","DECISION TREE"],
           "ACCURACY SCORE " : [acc_score1, acc_score2, acc_score3]}
df_predictions = pd.DataFrame (df_pred)
df_predictions
```

Out[102]:

	NAME OF MODEL	ACCURACY SCORE
0	LOGISTIC REGRESSION	0.976608
1	K-NN	0.959064
2	DECISION TREE	0.929825

Hyperparameter Tuning

In [138]:

```
parameters = [{'penalty': ['l1', 'l2'], 'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000],
                'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']}]
grid_search = GridSearchCV(estimator = logistic_regressor,
                           param_grid = parameters,
                           scoring = 'accuracy',
                           cv = 10,
                           n_jobs = -1)
grid_search.fit(X_train, Y_train)
best_accuracy_log = grid_search.best_score_
best_parameters = grid_search.best_params_
print(best_accuracy_log)
print(best_parameters)
```

```
0.9824999999999999
{'C': 0.1, 'penalty': 'l2', 'solver': 'liblinear'}
```

In [106]:

```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
oHe = OneHotEncoder()
```

In [120]:

```
parameters = [{'criterion':['gini','entropy'],'max_depth':[4,5,6,7,8,9,10,11,12,15,20,3
0,40,50,70,90,120,150],
                'max_leaf_nodes': [2,4,6,10,15,30,40,50,100], 'min_samples_split': [2,
3, 4]}]
oHe = OneHotEncoder()
grid_search = GridSearchCV(estimator = classifier,
                           param_grid = parameters,
                           scoring = 'accuracy',
                           cv = 10,
                           n_jobs = -1)
grid_search.fit(X_train, Y_train)
best_accuracy_dtc = grid_search.best_score_
best_parameters = grid_search.best_params_
print(best_accuracy_dtc)
print(best_parameters)
```

```
0.9246153846153847
{'criterion': 'gini', 'max_depth': 7, 'max_leaf_nodes': 30, 'min_samples_s
plit': 4}
```

Comparing models before and after Parameter Tuning

In [116]:

```
prediction_columns = ["NAME OF MODEL", "ACCURACY SCORE", "BEST ACCURACY (AFTER HYPER-PA
RAMETER TUNING)"]
df_pred = {"NAME OF MODEL" : ["LOGISTIC REGRESSION", "K-NN", "DECISION TREE", ],
           "ACCURACY SCORE " : [acc_score1, acc_score2, acc_score3],
           "BEST ACCURACY (AFTER HYPER-PARAMETER TUNING)" : [best_accuracy_log, best_ac
curacy_knn, best_accuracy_dtc]}
df_predictions = pd.DataFrame (df_pred)
df_predictions
```

Out[116]:

	NAME OF MODEL	ACCURACY SCORE	BEST ACCURACY (AFTER HYPER-PARAMETER TUNING)
0	LOGISTIC REGRESSION	0.976608	0.982500
1	K-NN	0.959064	0.969872
2	DECISION TREE	0.929825	0.926987

Conclusion

In [ ]:

```
To conclude this notebook, it is fairly evident that it is the Logistic Regression mode
l that has come out
triumphant with the highest accuracies both before and after the hyper-parameter tuning
. It ended up with an
accuracy of 97% before hyper-parameter tuning and  98% after and is hence, the best
suited model out of the rest for the given dataset.
```