In [2]:
```
#import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [3]:
```
df = pd.read_csv('cancer.csv')
df.head(7)
```

Out[3]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mea
0	842302	M	17.99	10.38	122.80	1001.0	0.1184
1	842517	M	20.57	17.77	132.90	1326.0	0.0847
2	84300903	M	19.69	21.25	130.00	1203.0	0.1096
3	84348301	M	11.42	20.38	77.58	386.1	0.1425
4	84358402	M	20.29	14.34	135.10	1297.0	0.1003
5	843786	M	12.45	15.70	82.57	477.1	0.1278
6	844359	M	18.25	19.98	119.60	1040.0	0.0946

7 rows × 32 columns

In [4]:
```
#Count the number of rows and columns in the data set
df.shape
```

Out[4]:
```
(569, 32)
```

In [5]:
```
#Count the empty (NaN, NAN, na) values in each column
df.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]:
```
#Drop the column with all missing values (na, NAN, NaN)
#NOTE: This drops the column Unnamed
df = df.dropna(axis=1)
```

In [7]:
```
#Get the new count of the number of rows and cols
df.shape
```

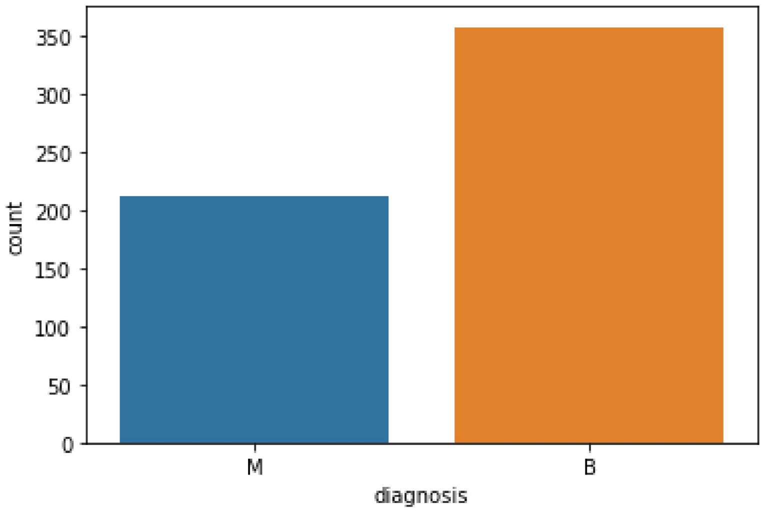
Out[7]:
```
(569, 32)
```

In [8]:
```
#Get a count of the number of Malignant (M) (harmful) or Benign (B) cells (not h
df['diagnosis'].value_counts()
```

Out[8]:
```
B    357
M    212
Name: diagnosis, dtype: int64
```

In [10]:
```
#Visualize this count
sns.countplot(df['diagnosis'],label="Count")
```

Out[10]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x29b8c08a248>
```



In [11]:
```
#Look at the data types to see which columns need to be transformed / encoded to
df.dtypes
```

Out[11]:
```
id                           int64
diagnosis                   object
radius_mean                float64
texture_mean               float64
perimeter_mean             float64
area_mean                  float64
smoothness_mean            float64
compactness_mean           float64
concavity_mean             float64
concave points_mean        float64
symmetry_mean              float64
fractal_dimension_mean     float64
radius_se                  float64
texture_se                 float64
perimeter_se               float64
area_se                    float64
smoothness_se              float64
compactness_se             float64
concavity_se               float64
concave points_se          float64
symmetry_se                float64
fractal_dimension_se       float64
radius_worst               float64
texture_worst              float64
perimeter_worst            float64
area_worst                 float64
smoothness_worst           float64
compactness_worst          float64
concavity_worst            float64
concave points_worst       float64
symmetry_worst             float64
fractal_dimension_worst    float64
dtype: object
```

In [12]:
```python
#Transform/ Encode the column diagnosis
#dictionary = {'M':1, 'B':0}#Create a dictionary file
#df.diagnosis = [dictionary[item] for item in df.diagnosis] #Change all 'M' to 1

#Encoding categorical data values (Transforming categorical data/ Strings to inte
from sklearn.preprocessing import LabelEncoder
labelencoder_Y = LabelEncoder()
df.iloc[:,1]= labelencoder_Y.fit_transform(df.iloc[:,1].values)
print(labelencoder_Y.fit_transform(df.iloc[:,1].values))
```

```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 0 1 1 1 1 1 1 1 1 0 1 0 0 0 0 0 1 1 0 1 1 0 0 0 0 1 0 1 1 0 0 0 0 1 0 1 1
 0 1 0 1 1 0 0 0 1 1 0 1 1 1 0 0 0 1 0 0 1 1 0 0 0 1 1 0 0 0 0 1 0 0 1 0 0
 0 0 0 0 0 0 1 1 1 0 1 1 0 0 0 1 1 0 1 0 1 1 0 1 1 0 0 1 0 0 1 0 0 0 0 1 0
 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 1 0 0 1 1 0 0 1 1 0 0 0 0 1 0 0 1 1 1 0 1
 0 1 0 0 0 1 0 0 1 1 0 1 1 1 1 0 1 1 1 0 1 0 1 0 0 1 0 1 1 1 1 0 0 1 1 0 0
 0 1 0 0 0 0 0 1 1 0 0 1 0 0 1 1 0 1 0 0 0 0 1 0 0 0 0 0 1 0 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 0 0 0 0 0 0 1 0 1 0 0 1 0 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0
 0 1 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 1 0 0 0 0 1 1 1 0 0
 0 0 1 0 1 0 1 0 0 0 1 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1
 1 0 1 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 1 0 0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0
 0 1 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 1 0 0 0 0 0 1 0 0
 1 0 1 0 0 1 0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0
 0 0 0 0 0 0 1 0 1 0 0 1 0 0 0 0 0 1 1 0 1 0 1 0 0 0 0 0 1 0 0 1 0 1 0 1 1
 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 1 1 1 1 1 1 0]
```

In [13]:
```python
#A “pairs plot” is also known as a scatterplot, in which one variable in the same
sns.pairplot(df, hue="diagnosis")
#sns.pairplot(df.iloc[:,1:6], hue="diagnosis") #plot a sample of the columns
```

```
C:\Users\DELL\Anaconda4\lib\site-packages\statsmodels\nonparametric\kde.py:48
7: RuntimeWarning: invalid value encountered in true_divide
  binned = fast_linbin(X, a, b, gridsize) / (delta * nobs)
C:\Users\DELL\Anaconda4\lib\site-packages\statsmodels\nonparametric\kdetools.
py:34: RuntimeWarning: invalid value encountered in double_scalars
  FAC1 = 2*(np.pi*bw/RANGE)**2
```

Out[13]:
```
<seaborn.axisgrid.PairGrid at 0x29b8c883f08>
```

In [14]:
```
#Print the first 5 rows of the new data set
df.head(5)
```

Out[14]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mea
0	842302	1	17.99	10.38	122.80	1001.0	0.1184
1	842517	1	20.57	17.77	132.90	1326.0	0.0847
2	84300903	1	19.69	21.25	130.00	1203.0	0.1096
3	84348301	1	11.42	20.38	77.58	386.1	0.1425
4	84358402	1	20.29	14.34	135.10	1297.0	0.1003

5 rows × 32 columns

In [15]:

```
#Get the correlation of the columns
df.corr()
#df.iloc[:,1:12].corr() #Get a sample of correlated column info
```

Out[15]:

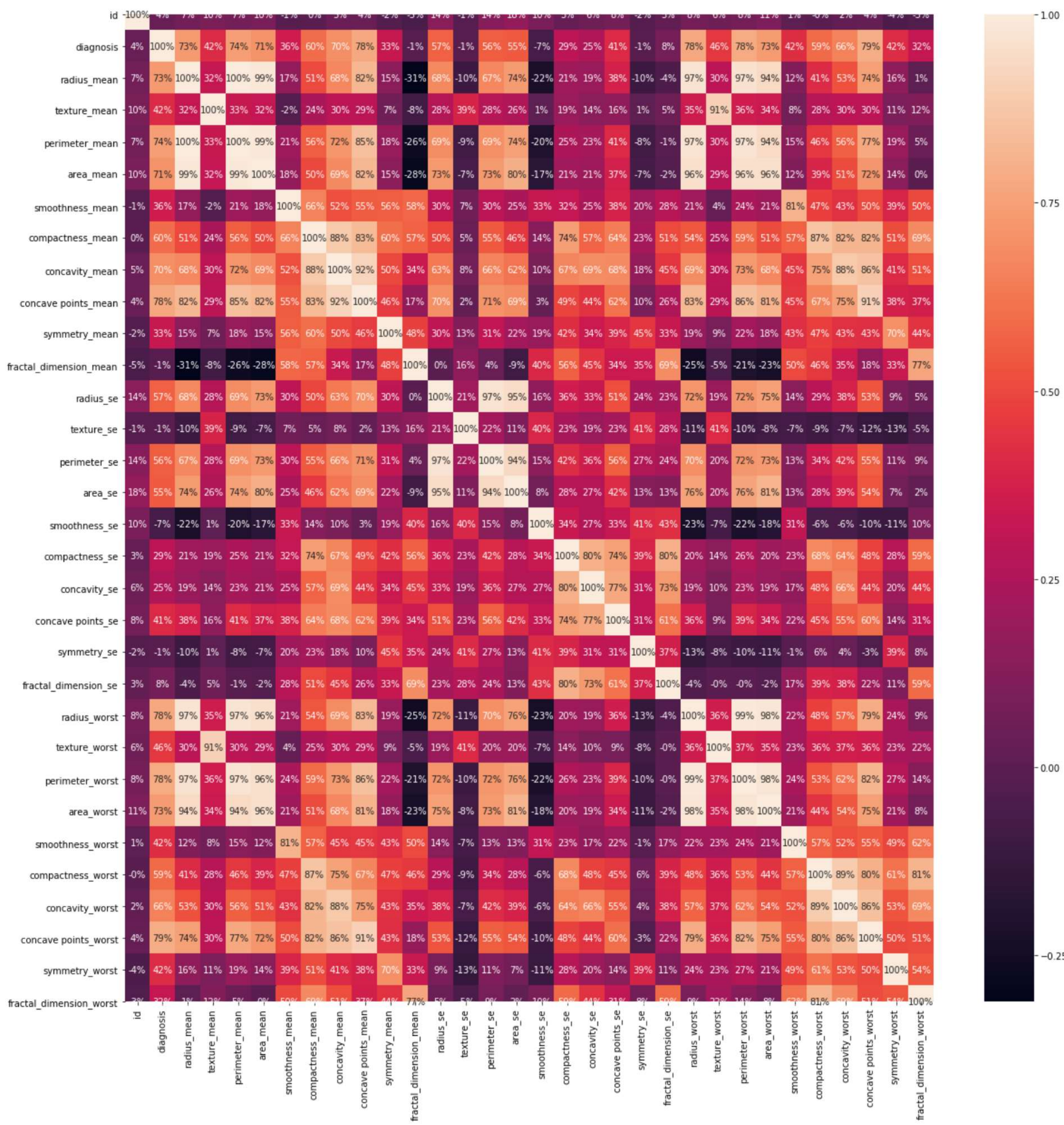
	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_
id	1.000000	0.039769	0.074626	0.099770	0.073159	0.09
diagnosis	0.039769	1.000000	0.730029	0.415185	0.742636	0.70
radius_mean	0.074626	0.730029	1.000000	0.323782	0.997855	0.98
texture_mean	0.099770	0.415185	0.323782	1.000000	0.329533	0.32
perimeter_mean	0.073159	0.742636	0.997855	0.329533	1.000000	0.98
area_mean	0.096893	0.708984	0.987357	0.321086	0.986507	1.00
smoothness_mean	-0.012968	0.358560	0.170581	-0.023389	0.207278	0.17
compactness_mean	0.000096	0.596534	0.506124	0.236702	0.556936	0.49
concavity_mean	0.050080	0.696360	0.676764	0.302418	0.716136	0.68
concave points_mean	0.044158	0.776614	0.822529	0.293464	0.850977	0.82
symmetry_mean	-0.022114	0.330499	0.147741	0.071401	0.183027	0.15
fractal_dimension_mean	-0.052511	-0.012838	-0.311631	-0.076437	-0.261477	-0.28
radius_se	0.143048	0.567134	0.679090	0.275869	0.691765	0.73
texture_se	-0.007526	-0.008303	-0.097317	0.386358	-0.086761	-0.06
perimeter_se	0.137331	0.556141	0.674172	0.281673	0.693135	0.72
area_se	0.177742	0.548236	0.735864	0.259845	0.744983	0.80
smoothness_se	0.096781	-0.067016	-0.222600	0.006614	-0.202694	-0.16
compactness_se	0.033961	0.292999	0.206000	0.191975	0.250744	0.21
concavity_se	0.055239	0.253730	0.194204	0.143293	0.228082	0.20
concave points_se	0.078768	0.408042	0.376169	0.163851	0.407217	0.37
symmetry_se	-0.017306	-0.006522	-0.104321	0.009127	-0.081629	-0.07
fractal_dimension_se	0.025725	0.077972	-0.042641	0.054458	-0.005523	-0.01
radius_worst	0.082405	0.776454	0.969539	0.352573	0.969476	0.96
texture_worst	0.064720	0.456903	0.297008	0.912045	0.303038	0.28
perimeter_worst	0.079986	0.782914	0.965137	0.358040	0.970387	0.95
area_worst	0.107187	0.733825	0.941082	0.343546	0.941550	0.95
smoothness_worst	0.010338	0.421465	0.119616	0.077503	0.150549	0.12
compactness_worst	-0.002968	0.590998	0.413463	0.277830	0.455774	0.39
concavity_worst	0.023203	0.659610	0.526911	0.301025	0.563879	0.51
concave points_worst	0.035174	0.793566	0.744214	0.295316	0.771241	0.72
symmetry_worst	-0.044224	0.416294	0.163953	0.105008	0.189115	0.14

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_
fractal_dimension_worst	-0.029866	0.323872	0.007066	0.119205	0.051019	0.0C

32 rows × 32 columns

In [16]:
```
#Visualize the correlation
#NOTE: To see the numbers within the cell ==>  sns.heatmap(df.corr(), annot=True,
plt.figure(figsize=(20,20))  #This is used to change the size of the figure/ heat
sns.heatmap(df.corr(), annot=True, fmt='.0%')
#plt.figure(figsize=(10,10)) #This is used to change the size of the figure/ heat
#sns.heatmap(df.iloc[:,1:12].corr(), annot=True, fmt='.0%') #Get a heap map of 11
```

Out[16]: `<matplotlib.axes._subplots.AxesSubplot at 0x29bb67504c8>`



In [17]:
```
#Split the data into independent 'X' and dependent 'Y' variables
X = df.iloc[:, 2:31].values #Notice I started from index  2 to 31, essentially re
Y = df.iloc[:, 1].values #Get the target variable 'diagnosis' located at index=1
```

In [18]:
```
# Split the dataset into 75% Training set and 25% Testing set
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.25, rand
```

In [19]:
```
# Scale the data to bring all features to the same level of magnitude
# This means the data will be within a specific range for example 0 -100 or 0 - 1

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

In [20]:
```
#Create a function within many Machine Learning Models
def models(X_train,Y_train):

  #Using Logistic Regression Algorithm to the Training Set
  from sklearn.linear_model import LogisticRegression
  log = LogisticRegression(random_state = 0)
  log.fit(X_train, Y_train)

  #Using KNeighborsClassifier Method of neighbors class to use Nearest Neighbor a
  from sklearn.neighbors import KNeighborsClassifier
  knn = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
  knn.fit(X_train, Y_train)

  #Using SVC method of svm class to use Support Vector Machine Algorithm
  from sklearn.svm import SVC
  svc_lin = SVC(kernel = 'linear', random_state = 0)
  svc_lin.fit(X_train, Y_train)

  #Using SVC method of svm class to use Kernel SVM Algorithm
  from sklearn.svm import SVC
  svc_rbf = SVC(kernel = 'rbf', random_state = 0)
  svc_rbf.fit(X_train, Y_train)

  #Using GaussianNB method of naïve_bayes class to use Naïve Bayes Algorithm
  from sklearn.naive_bayes import GaussianNB
  gauss = GaussianNB()
  gauss.fit(X_train, Y_train)

  #Using DecisionTreeClassifier of tree class to use Decision Tree Algorithm
  from sklearn.tree import DecisionTreeClassifier
  tree = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
  tree.fit(X_train, Y_train)

  #Using RandomForestClassifier method of ensemble class to use Random Forest Cla
  from sklearn.ensemble import RandomForestClassifier
  forest = RandomForestClassifier(n_estimators = 10, criterion = 'entropy', rand
  forest.fit(X_train, Y_train)

  #print model accuracy on the training data.
  print('[0]Logistic Regression Training Accuracy:', log.score(X_train, Y_train)
  print('[1]K Nearest Neighbor Training Accuracy:', knn.score(X_train, Y_train))
  print('[2]Support Vector Machine (Linear Classifier) Training Accuracy:', svc_
  print('[3]Support Vector Machine (RBF Classifier) Training Accuracy:', svc_rbf
  print('[4]Gaussian Naive Bayes Training Accuracy:', gauss.score(X_train, Y_tra
  print('[5]Decision Tree Classifier Training Accuracy:', tree.score(X_train, Y_
  print('[6]Random Forest Classifier Training Accuracy:', forest.score(X_train, Y

  return log, knn, svc_lin, svc_rbf, gauss, tree, forest
```

In [24]:
```
model = models(X_train,Y_train)
```

```
[0]Logistic Regression Training Accuracy: 0.9906103286384976
[1]K Nearest Neighbor Training Accuracy: 0.9765258215962441
[2]Support Vector Machine (Linear Classifier) Training Accuracy: 0.988262910798
1221
[3]Support Vector Machine (RBF Classifier) Training Accuracy: 0.983568075117370
9
[4]Gaussian Naive Bayes Training Accuracy: 0.9507042253521126
[5]Decision Tree Classifier Training Accuracy: 1.0
[6]Random Forest Classifier Training Accuracy: 0.9953051643192489

C:\Users\DELL\Anaconda4\lib\site-packages\sklearn\linear_model\logistic.py:432:
FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a sol
ver to silence this warning.
  FutureWarning)
```

In [26]:

```
#Show the confusion matrix and accuracy for all of the models on the test data
#Classification accuracy is the ratio of correct predictions to total predictions
from sklearn.metrics import confusion_matrix
for i in range(len(model)):
  cm = confusion_matrix(Y_test, model[i].predict(X_test))

  TN = cm[1][1]
  TP = cm[0][0]
  FN = cm[1][0]
  FP = cm[0][1]

  print(cm)
  print('Model[{}] Testing Accuracy = "{}!"'.format(i,  (TP + TN) / (TP + TN + FN
  print()# Print a new line
```

```
[[86  4]
 [ 4 49]]
Model[0] Testing Accuracy = "0.9440559440559441!"

[[89  1]
 [ 5 48]]
Model[1] Testing Accuracy = "0.958041958041958!"

[[87  3]
 [ 2 51]]
Model[2] Testing Accuracy = "0.965034965034965!"

[[88  2]
 [ 3 50]]
Model[3] Testing Accuracy = "0.965034965034965!"

[[85  5]
 [ 6 47]]
Model[4] Testing Accuracy = "0.9230769230769231!"

[[84  6]
 [ 1 52]]
Model[5] Testing Accuracy = "0.951048951048951!"

[[87  3]
 [ 2 51]]
Model[6] Testing Accuracy = "0.965034965034965!"
```

In [27]:
```
#Show other ways to get the classification accuracy & other metrics

from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score

for i in range(len(model)):
  print('Model ',i)
  #Check precision, recall, f1-score
  print( classification_report(Y_test, model[i].predict(X_test)) )
  #Another way to get the models accuracy on the test data
  print( accuracy_score(Y_test, model[i].predict(X_test)))
  print()#Print a new line
```

```
Model  0
              precision    recall  f1-score   support

           0       0.96      0.96      0.96        90
           1       0.92      0.92      0.92        53

    accuracy                           0.94       143
   macro avg       0.94      0.94      0.94       143
weighted avg       0.94      0.94      0.94       143

0.9440559440559441

Model  1
              precision    recall  f1-score   support

           0       0.95      0.99      0.97        90
           1       0.98      0.91      0.94        53

    accuracy                           0.96       143
   macro avg       0.96      0.95      0.95       143
weighted avg       0.96      0.96      0.96       143

0.958041958041958

Model  2
              precision    recall  f1-score   support

           0       0.98      0.97      0.97        90
           1       0.94      0.96      0.95        53

    accuracy                           0.97       143
   macro avg       0.96      0.96      0.96       143
weighted avg       0.97      0.97      0.97       143

0.965034965034965

Model  3
              precision    recall  f1-score   support

           0       0.97      0.98      0.97        90
           1       0.96      0.94      0.95        53

    accuracy                           0.97       143
```

```
   macro avg       0.96      0.96      0.96       143
weighted avg       0.96      0.97      0.96       143

0.965034965034965

Model  4
              precision    recall  f1-score   support

           0       0.93      0.94      0.94        90
           1       0.90      0.89      0.90        53

    accuracy                           0.92       143
   macro avg       0.92      0.92      0.92       143
weighted avg       0.92      0.92      0.92       143

0.9230769230769231

Model  5
              precision    recall  f1-score   support

           0       0.99      0.93      0.96        90
           1       0.90      0.98      0.94        53

    accuracy                           0.95       143
   macro avg       0.94      0.96      0.95       143
weighted avg       0.95      0.95      0.95       143

0.951048951048951

Model  6
              precision    recall  f1-score   support

           0       0.98      0.97      0.97        90
           1       0.94      0.96      0.95        53

    accuracy                           0.97       143
   macro avg       0.96      0.96      0.96       143
weighted avg       0.97      0.97      0.97       143

0.965034965034965
```

In [28]:
```
#Print Prediction of Random Forest Classifier model
pred = model[6].predict(X_test)
print(pred)

#Print a space
print()

#Print the actual values
print(Y_test)
```

```
[1 0 0 0 0 0 0 0 0 0 1 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 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 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 1 1 0 0 0 1]

[1 0 0 0 0 0 0 0 0 0 0 0 0 0 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 1
 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 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 1 1 0 0 0 1]
```

In [ ]: