In [1]:
```python
#importing all the necessary libaries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.metrics import accuracy_score, confusion_matrix, roc_curve, roc_auc_score
import seaborn as sns
sns.set()
```

In [2]:
```python
import os
```

In [3]:
```python
#Reading the data
data = pd.read_csv('C:\\Users\\nakkl\\OneDrive\\Desktop\\data.csv')
```

In [4]:
```python
data
```

Out[4]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	radius_worst	texture_worst	perimeter_worst	area_worst	smoothness
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	...	25.380	17.33	184.60	2019.0	(
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	...	24.990	23.41	158.80	1956.0	(
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	...	23.570	25.53	152.50	1709.0	(
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	...	14.910	26.50	98.87	567.7	(
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	...	22.540	16.67	152.20	1575.0	(
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
564	926424	M	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	...	25.450	26.40	166.10	2027.0	(
565	926682	M	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	...	23.690	38.25	155.00	1731.0	(
566	926954	M	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	...	18.980	34.12	126.70	1124.0	(
567	927241	M	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	...	25.740	39.42	184.60	1821.0	(
568	92751	B	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	...	9.456	30.37	59.16	268.6	(

569 rows × 32 columns

In [6]:
```python
#print the top 5 coloums
data.head(5)
```

Out[6]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	radius_worst	texture_worst	perimeter_worst	area_worst	smoothness_w
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	...	25.38	17.33	184.60	2019.0	0.
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	...	24.99	23.41	158.80	1956.0	0.
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	...	23.57	25.53	152.50	1709.0	0.
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	...	14.91	26.50	98.87	567.7	0.
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	...	22.54	16.67	152.20	1575.0	0.

5 rows × 32 columns

In [7]:
```python
#getting the basic information
data.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 [8]:
```python
#getting the statistical data
data.describe()
```

Out[8]:

	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	...	radius_worst	texture_worst	perimeter_worst	area_wors
count	5.690000e+02	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	...	569.000000	569.000000	569.000000	569.00000
mean	3.037183e+07	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.048919	0.181162	...	16.269190	25.677223	107.261213	880.58312
std	1.250206e+08	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.038803	0.027414	...	4.833242	6.146258	33.602542	569.35699
min	8.670000e+03	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.000000	0.106000	...	7.930000	12.020000	50.410000	185.20000
25%	8.692180e+05	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.020310	0.161900	...	13.010000	21.080000	84.110000	515.30000
50%	9.060240e+05	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.033500	0.179200	...	14.970000	25.410000	97.660000	686.50000
75%	8.813129e+06	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.074000	0.195700	...	18.790000	29.720000	125.400000	1084.00000
max	9.113205e+08	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.201200	0.304000	...	36.040000	49.540000	251.200000	4254.00000

8 rows × 31 columns

In [10]:
```python
#count the number of rows and coloums in data
data.shape
```

Out[10]:
```
(569, 32)
```

In [11]:
```python
#get the empty values(NaN,NAN,na) in each coloum
data.isna().sum()
```

Out[11]:
```
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
```

since there are no empty values so no need of filling the data we can use it as it is

In [12]:
```python
#Get the count of Maligant(M) or Benign(B) cells
data['diagnosis'].value_counts()
```

Out[12]:
```
B    357
M    212
Name: diagnosis, dtype: int64
```

In [13]:
```python
#visualise the count
sns.countplot(data['diagnosis'],label='count')
```

```
C:\Users\nakkl\OneDrive\Documents\Python Scripts\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[13]:
```
<AxesSubplot:xlabel='diagnosis', ylabel='count'>
```

In [14]:
```python
#looking at the data which are to be encoded(replaced)
data.dtypes
```

Out[14]:
```
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
```

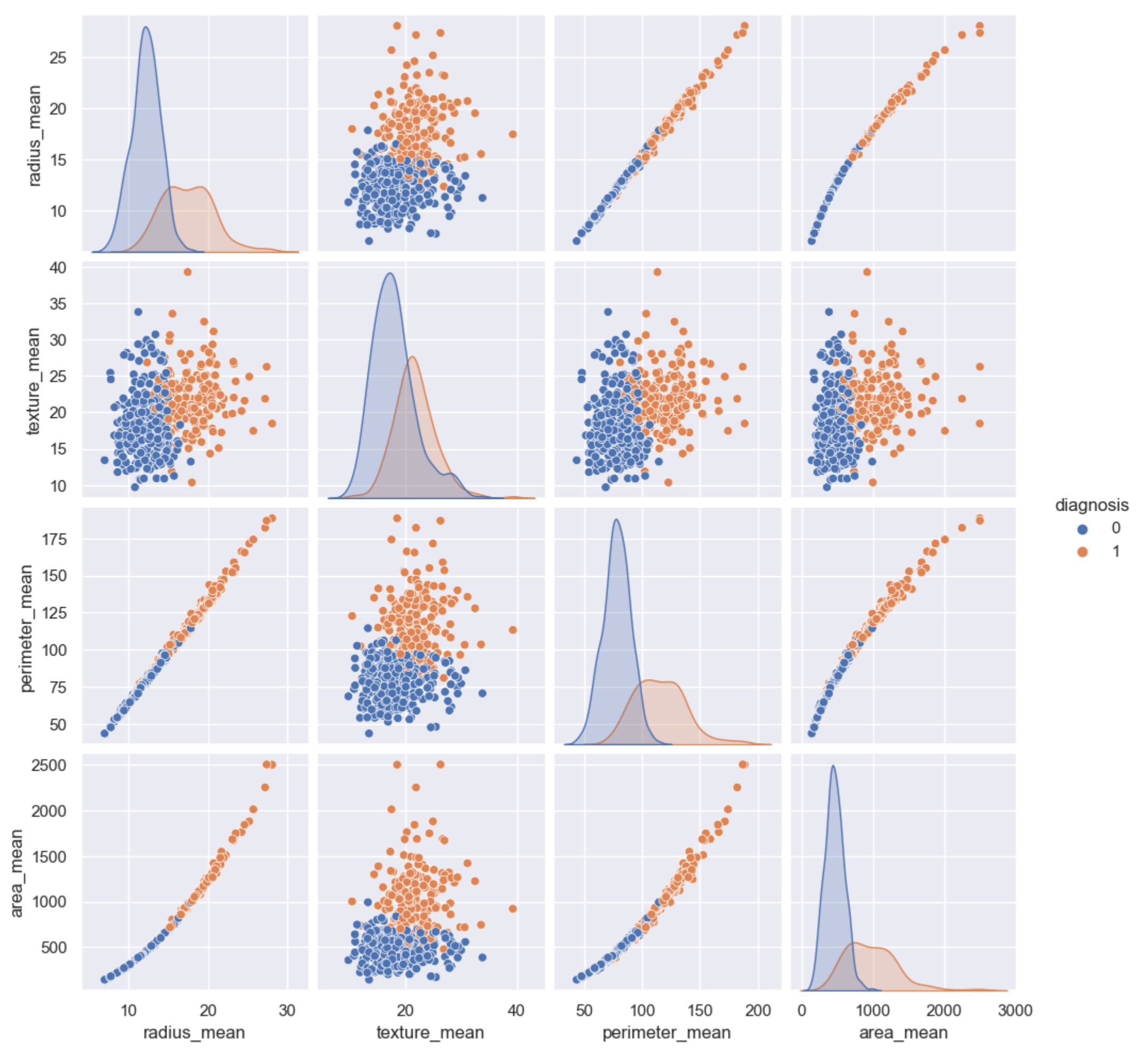
In [16]:
```python
#encoding the data
labelencoder_y = LabelEncoder()
data.iloc[:,1] = labelencoder_y.fit_transform(data.iloc[:,1].values)

data.iloc[:,1]
```

Out[16]:
```
0      1
1      1
2      1
3      1
4      1
      ..
564    1
565    1
566    1
567    1
568    0
Name: diagnosis, Length: 569, dtype: int32
```

In [21]:
```python
#create a pair plot
sns.pairplot(data.iloc[:,1:6], hue='diagnosis')
```

Out[21]:
```
<seaborn.axisgrid.PairGrid at 0x16dc87ddd00>
```



In [23]:
```python
#print the frist 5 rows of new data
data.head(5)
```

Out[23]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	radius_worst	texture_worst	perimeter_worst	area_worst	smoothness_v
0	842302	1	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	...	25.38	17.33	184.60	2019.0	0.
1	842517	1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	...	24.99	23.41	158.80	1956.0	0.
2	84300903	1	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	...	23.57	25.53	152.50	1709.0	0.
3	84348301	1	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	...	14.91	26.50	98.87	567.7	0.
4	84358402	1	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	...	22.54	16.67	152.20	1575.0	0.

5 rows × 32 columns

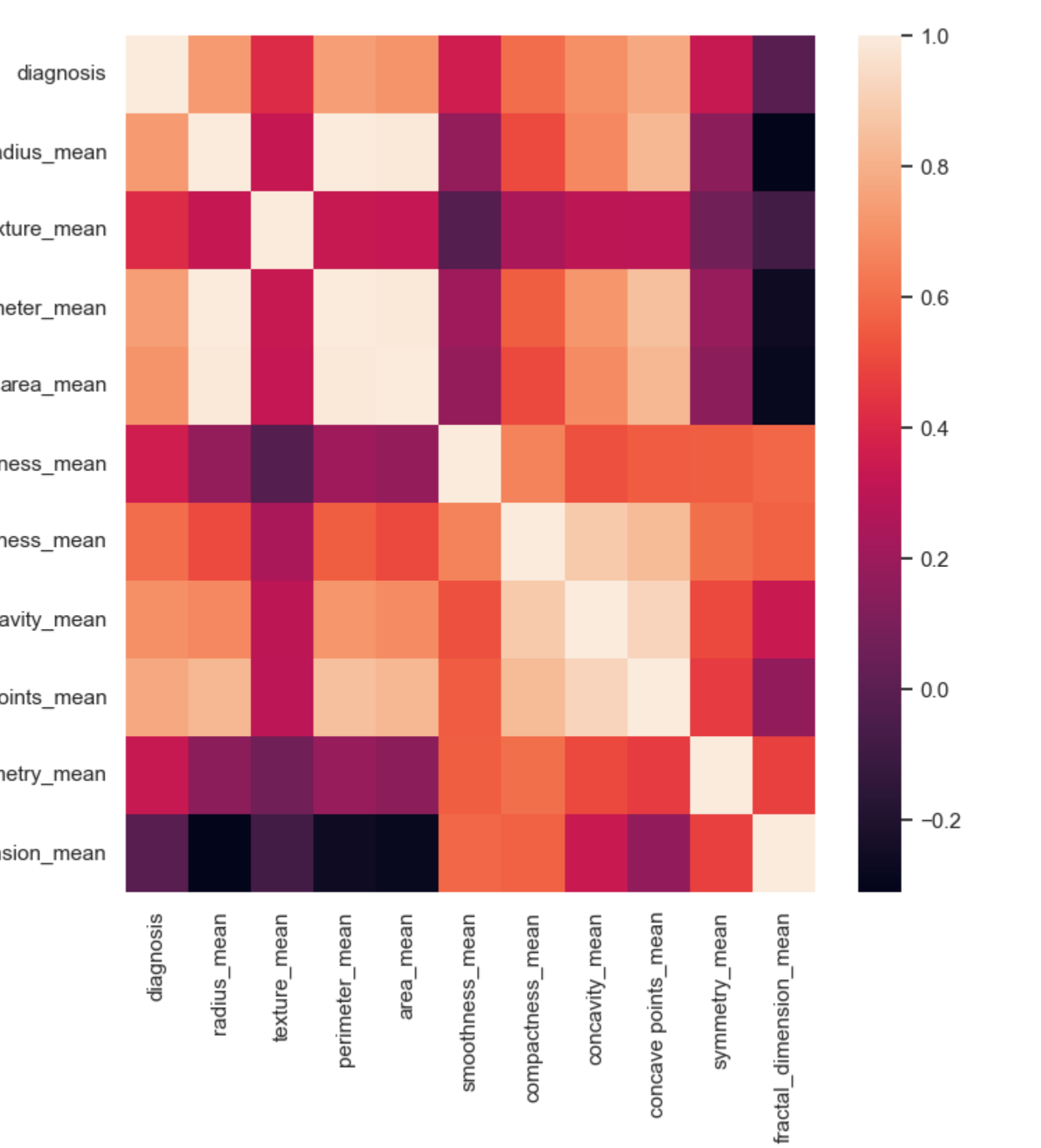
In [24]:
```python
#get the correlation of the coloums
data.iloc[:,1:12].corr()
```

Out[24]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	fractal_dimension_mean
diagnosis	1.000000	0.730029	0.415185	0.742636	0.708984	0.358560	0.596534	0.696360	0.776614	0.330499	-0.012838
radius_mean	0.730029	1.000000	0.323782	0.997855	0.987357	0.170581	0.506124	0.676764	0.822529	0.147741	-0.311631
texture_mean	0.415185	0.323782	1.000000	0.329533	0.321086	-0.023389	0.236702	0.302418	0.293464	0.071401	-0.076437
perimeter_mean	0.742636	0.997855	0.329533	1.000000	0.986507	0.207278	0.556936	0.716136	0.850977	0.183027	-0.261477
area_mean	0.708984	0.987357	0.321086	0.986507	1.000000	0.177028	0.498502	0.685983	0.823269	0.151293	-0.283110
smoothness_mean	0.358560	0.170581	-0.023389	0.207278	0.177028	1.000000	0.659123	0.521984	0.553695	0.557775	0.584792
compactness_mean	0.596534	0.506124	0.236702	0.556936	0.498502	0.659123	1.000000	0.883121	0.831135	0.602641	0.565369
concavity_mean	0.696360	0.676764	0.302418	0.716136	0.685983	0.521984	0.883121	1.000000	0.921391	0.500667	0.336783
concave points_mean	0.776614	0.822529	0.293464	0.850977	0.823269	0.553695	0.831135	0.921391	1.000000	0.462497	0.166917
symmetry_mean	0.330499	0.147741	0.071401	0.183027	0.151293	0.557775	0.602641	0.500667	0.462497	1.000000	0.479921
fractal_dimension_mean	-0.012838	-0.311631	-0.076437	-0.261477	-0.283110	0.584792	0.565369	0.336783	0.166917	0.479921	1.000000

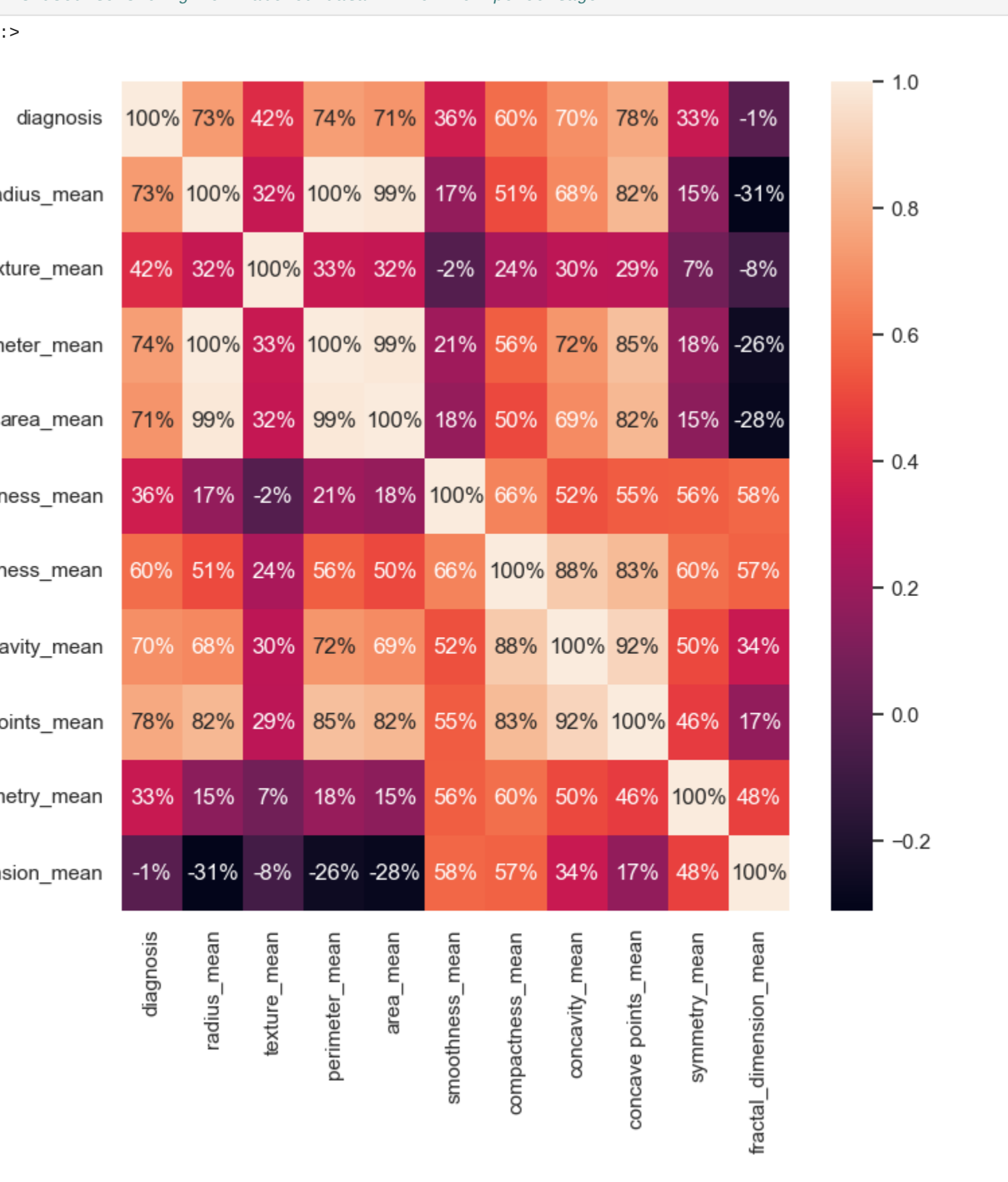
In [39]:
```python
#visualize the correlation
plt.figure(figsize=(8,8))
sns.heatmap(data.iloc[:,1:12].corr())
```

Out[39]:
```
<AxesSubplot:>
```



In [40]:
```python
plt.figure(figsize=(8,8))
sns.heatmap(data.iloc[:,1:12].corr(), annot=True, fmt='.0%')
#annot(annotation) is used to label the data
#fmt(format) is used to show given labeled data in form of percentage
```

Out[40]:
```
<AxesSubplot:>
```



In [35]:
```python
#split the data set into independent(X) and dependent(Y) data sets
X = data.iloc[:,2:31].values
#x has the features of cancer or not
Y = data.iloc[:,1].values
#y has the diagnosis
```

In [36]:
```python
#split the data set into 75% training and 25% testing
X_train,X_test,Y_train,Y_test = train_test_split(X, Y, test_size = 0.25 , random_state = 0)
```

In [41]:
```python
#Scale the data(feature scaling)
from sklearn.preprocessing import StandardScaler
sc =  StandardScaler()
X_test = sc.fit_transform(X_test)
X_train = sc.fit_transform(X_train)
```

In [43]:
```python
#create a function for the models
def models(X_train,Y_train):

    #Logistic Regression
    log = LogisticRegression(random_state=0)
    log.fit(X_train, Y_train)

    #Decision Tree
    from sklearn.tree import DecisionTreeClassifier
    tree = DecisionTreeClassifier(criterion = 'entropy', random_state=0 )
    tree.fit(X_train, Y_train)

    #Random Forest Classifier
    forest = RandomForestClassifier(n_estimators = 10,criterion = 'entropy')
    forest.fit(X_train,Y_train)

    #Print the models accuarcy on the training data
    print('[0]Logistic Regression Training Accuracy:', log.score(X_train,Y_train))
    print('[1]Decision Tree Classifier Training Accuracy:', tree.score(X_train,Y_train))
    print('[2]Random Forest Classifier Training Accuracy:', forest.score(X_train,Y_train))

    return log,tree,forest
```

In [44]:
```python
#Getting all of the models
model = models(X_train, Y_train)
```

```
[0]Logistic Regression Training Accuracy: 0.9906103286384976
[1]Decision Tree Classifier Training Accuracy: 1.0
[2]Random Forest Classifier Training Accuracy: 0.9976525821596244
```

In [46]:
```python
#test model accuracy on test data on confusion matrix
for i in range( len(model) ):
    print('Model' , i)
    cm = confusion_matrix(Y_test,model[i].predict(X_test))

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

    print(cm)
    print('Testing Accuracy =', (TP + TN)/ (TP + TN + FN + FP))
    print()
```

```
Model 0
[[86  4]
 [ 3 50]]
Testing Accuracy = 0.951048951048951

Model 1
[[83  7]
 [ 2 51]]
Testing Accuracy = 0.9370629370629371

Model 2
[[89  1]
 [ 6 47]]
Testing Accuracy = 0.951048951048951
```

In [48]:
```python
#another way to find the accuracy
from sklearn.metrics import classification_report
for i in range( len(model) ):
    print('Model' , i)
    print( classification_report(Y_test,model[i].predict(X_test)))
    print( accuracy_score(Y_test, model[0].predict(X_test)))
    print()
```

```
Model 0
              precision    recall  f1-score   support

           0       0.97      0.96      0.96        90
           1       0.93      0.94      0.93        53

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

0.951048951048951

Model 1
              precision    recall  f1-score   support

           0       0.98      0.92      0.95        90
           1       0.88      0.96      0.92        53

    accuracy                           0.94       143
   macro avg       0.93      0.94      0.93       143
weighted avg       0.94      0.94      0.94       143

0.951048951048951

Model 2
              precision    recall  f1-score   support

           0       0.94      0.99      0.96        90
           1       0.98      0.89      0.93        53

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

0.951048951048951
```

In [49]:
```python
#print the prediction of Random Forest Classifier
pred = model[2].predict(X_test)
print(pred)
print()
print(Y_test)
```

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