In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
df=pd.read_csv('breastcancer.csv')
df.head(7)
```

Out[2]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_me
0	842302	M	17.99	10.38	122.80	1001.0	0.118
1	842517	M	20.57	17.77	132.90	1326.0	0.084
2	84300903	M	19.69	21.25	130.00	1203.0	0.109
3	84348301	M	11.42	20.38	77.58	386.1	0.142
4	84358402	M	20.29	14.34	135.10	1297.0	0.100
5	843786	M	12.45	15.70	82.57	477.1	0.127
6	844359	M	18.25	19.98	119.60	1040.0	0.094

7 rows × 33 columns

In [3]:
```
#Number of rows and columns
df.shape
```

Out[3]:
```
(569, 33)
```

In [4]:
```
df.isna().sum()
```

Out[4]:
```
id                         0
diagnosis                  0
radius_mean                0
```

```
texture_mean                0
perimeter_mean              0
area_mean                   0
smoothness_mean             0
compactness_mean            0
concavity_mean              0
concave points_mean         0
symmetry_mean               0
fractal_dimension_mean      0
radius_se                   0
texture_se                  0
perimeter_se                0
area_se                     0
smoothness_se               0
compactness_se              0
concavity_se                0
concave points_se           0
symmetry_se                 0
fractal_dimension_se        0
radius_worst                0
texture_worst               0
perimeter_worst             0
area_worst                  0
smoothness_worst            0
compactness_worst           0
concavity_worst             0
concave points_worst        0
symmetry_worst              0
fractal_dimension_worst     0
Unnamed: 32               569
dtype: int64
```

In [5]:
```
#Drop the column with missing values
df=df.dropna(axis=1)
```

In [6]:
```
#Get the new count of number of rows and columns
df.shape
```

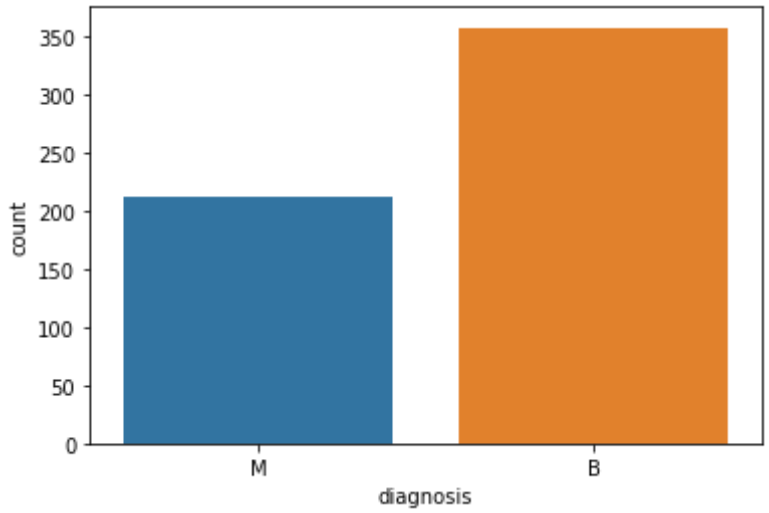
Out[6]: (569, 32)

In [7]:
```
#Get the count of number of Melignant and Belign patient present:
df['diagnosis'].value_counts()
```

Out[7]:
```
B    357
M    212
Name: diagnosis, dtype: int64
```

In [8]:
```
sns.countplot(df['diagnosis'],label='count')
```

Out[8]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1891b2e3048>
```



In [9]:
```
#Looks at the data type which needs to be encoded
df.dtypes
```

Out[9]:
```
id                         int64
diagnosis                 object
radius_mean              float64
texture_mean             float64
perimeter_mean           float64
area_mean                float64
smoothness_mean          float64
compactness_mean         float64
```

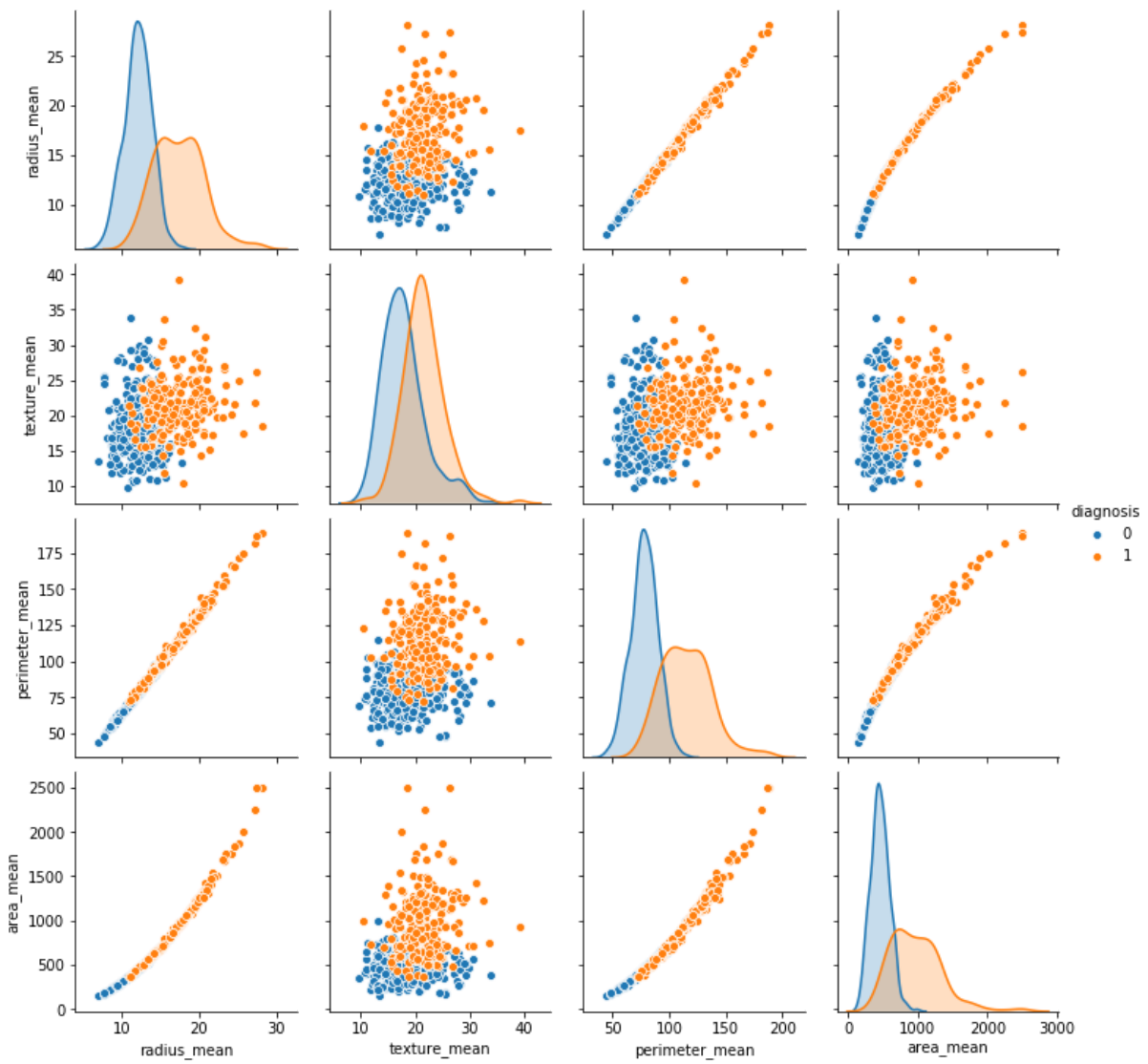
```
concavity_mean             float64
concave points_mean        float64
symmetry_mean              float64
fractal_dimension_mean     float64
radius_se                  float64
texture_se                 float64
perimeter_se               float64
area_se                    float64
smoothness_se              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 [10]:
```
#Encode the categorical data values
from sklearn.preprocessing import LabelEncoder
labelencoder_Y=LabelEncoder()
df.iloc[:,1]=labelencoder_Y.fit_transform(df.iloc[:,1].values)

#df.iloc[:,1]
```

In [11]:
```
sns.pairplot(df.iloc[:,1:6], hue='diagnosis')
```

Out[11]:
```
<seaborn.axisgrid.PairGrid at 0x1891ccbad30>
```



In [12]:
```
df.head(5)
```

Out[12]:

id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_me

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_me
0	842302	1	17.99	10.38	122.80	1001.0	0.118
1	842517	1	20.57	17.77	132.90	1326.0	0.084
2	84300903	1	19.69	21.25	130.00	1203.0	0.109
3	84348301	1	11.42	20.38	77.58	386.1	0.142
4	84358402	1	20.29	14.34	135.10	1297.0	0.100

5 rows × 32 columns

In [13]:
```
df.iloc[:,1:12].corr()
```

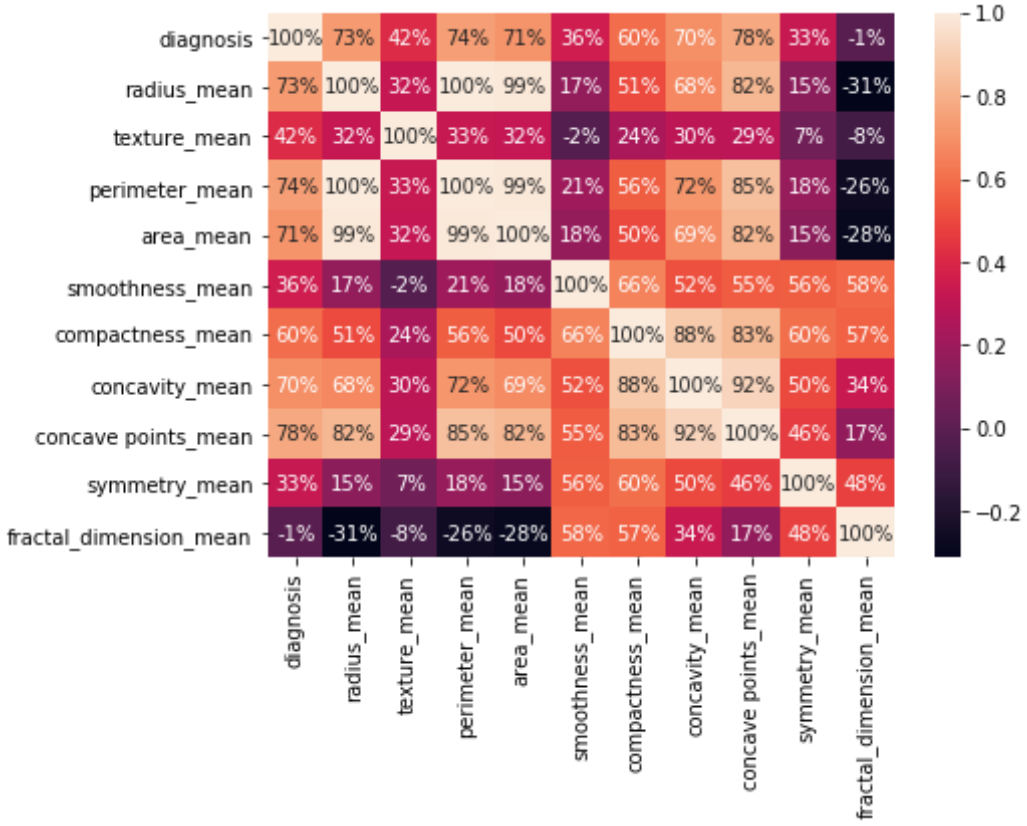
Out[13]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smo
diagnosis	1.000000	0.730029	0.415185	0.742636	0.708984	
radius_mean	0.730029	1.000000	0.323782	0.997855	0.987357	
texture_mean	0.415185	0.323782	1.000000	0.329533	0.321086	
perimeter_mean	0.742636	0.997855	0.329533	1.000000	0.986507	
area_mean	0.708984	0.987357	0.321086	0.986507	1.000000	
smoothness_mean	0.358560	0.170581	-0.023389	0.207278	0.177028	
compactness_mean	0.596534	0.506124	0.236702	0.556936	0.498502	
concavity_mean	0.696360	0.676764	0.302418	0.716136	0.685983	
concave points_mean	0.776614	0.822529	0.293464	0.850977	0.823269	
symmetry_mean	0.330499	0.147741	0.071401	0.183027	0.151293	
fractal_dimension_mean	-0.012838	-0.311631	-0.076437	-0.261477	-0.283110	

In [14]:
```
#visulaise the correlation
```

```
plt.figure(figsize=(7,5))
sns.heatmap(df.iloc[:,1:12].corr(), annot=True,fmt='.0%')
```

Out[14]: <matplotlib.axes._subplots.AxesSubplot at 0x1891b9e0d68>



In [19]:
```
X = df.iloc[:, 2:31].values
Y = df.iloc[:, 1].values
```

In [20]:
```
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, random_state = 0)
```

In [21]:
```
#Feature Scaling
```

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

In [22]:

```
def models(X_train,Y_train):

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

  #Using KNeighborsClassifier
  from sklearn.neighbors import KNeighborsClassifier
  knn = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p =
 2)
  knn.fit(X_train, Y_train)









  #Using GaussianNB
  from sklearn.naive_bayes import GaussianNB
  gauss = GaussianNB()
  gauss.fit(X_train, Y_train)

  #Using DecisionTreeClassifier
  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 Classification algorithm
  from sklearn.ensemble import RandomForestClassifier
  forest = RandomForestClassifier(n_estimators = 10, criterion = 'entro
py', random_state = 0)
  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]Gaussian Naive Bayes Training Accuracy:', gauss.score(X_tra
in, Y_train))
  print('[3]Decision Tree Classifier Training Accuracy:', tree.score(X_
train, Y_train))
  print('[4]Random Forest Classifier Training Accuracy:', forest.score(
X_train, Y_train))

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

In [23]:
```
model = models(X_train,Y_train)
```

```
[0]Logistic Regression Training Accuracy: 0.9906103286384976
[1]K Nearest Neighbor Training Accuracy: 0.9765258215962441
[2]Gaussian Naive Bayes Training Accuracy: 0.9507042253521126
[3]Decision Tree Classifier Training Accuracy: 1.0
[4]Random Forest Classifier Training Accuracy: 0.9953051643192489
```

```
C:\Users\KIIT\Anaconda3\lib\site-packages\sklearn\linear_model\logisti
c.py:433: FutureWarning: Default solver will be changed to 'lbfgs' in
0.22. Specify a solver to silence this warning.
  FutureWarning)
```

In [24]:
```
from sklearn.metrics import confusion_matrix
for i in range(len(model)):
  cm = confusion_matrix(Y_test, model[i].predict(X_test))

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

  print(cm)
```

```
  print('Model[{}] Testing Accuracy = "{}!"'.format(i,  (TP + TN) / (TP
 + TN + FN + FP)))
  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!"

[[85  5]
 [ 6 47]]
Model[2] Testing Accuracy = "0.9230769230769231!"

[[84  6]
 [ 1 52]]
Model[3] Testing Accuracy = "0.951048951048951!"

[[87  3]
 [ 2 51]]
Model[4] Testing Accuracy = "0.965034965034965!"
```

In [25]:

```
#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

   micro avg       0.94      0.94      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

   micro avg       0.96      0.96      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       0.93      0.94      0.94        90
           1       0.90      0.89      0.90        53

   micro avg       0.92      0.92      0.92       143
   macro avg       0.92      0.92      0.92       143
weighted avg       0.92      0.92      0.92       143

0.9230769230769231

Model  3
              precision    recall  f1-score   support

           0       0.99      0.93      0.96        90
```

```
              1       0.90      0.98      0.94        53

   micro avg       0.95      0.95      0.95       143
   macro avg       0.94      0.96      0.95       143
weighted avg       0.95      0.95      0.95       143

0.951048951048951

Model  4
              precision    recall  f1-score   support

           0       0.98      0.97      0.97        90
           1       0.94      0.96      0.95        53

   micro avg       0.97      0.97      0.97       143
   macro avg       0.96      0.96      0.96       143
weighted avg       0.97      0.97      0.97       143

0.965034965034965
```

In [27]:
```
#Print Prediction of Random Forest Classifier model
pred = model[4].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 [ ]: