In [3]:
```
#import libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
#loading the data
df = pd.read_csv('data.csv')
df.head(7)
```

Out[3]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	con
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	
5	843786	M	12.45	15.70	82.57	477.1	0.12780	0.17000	
6	844359	M	18.25	19.98	119.60	1040.0	0.09463	0.10900	

7 rows × 33 columns

In [4]:
```
#Counting number of rows and columns in the dataset
df.shape
```

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

In [5]:
```
#Count the number of empty values(NaN) 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
Unnamed: 32                569
dtype: int64
```

In [6]:
```
#Drop the column with all missing values
df = df.dropna(axis=1)
#Get the new count of number of rows and columns
df.shape
```

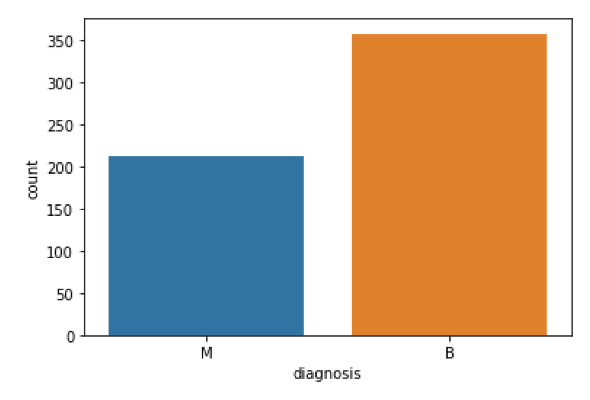
Out[6]:
```
(569, 32)
```

In [8]:
```
#Get a count of number of Malignant(M) or Benign(B) cells
df['diagnosis'].value_counts()
```

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

In [9]:
```
#Visualize the count
sns.countplot(df['diagnosis'],label = 'count')
```

Out[9]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x24c4c3c1a88>
```



In [10]:
```
#Look at the datatypes to see which columns need to be encoded
df.dtypes
```

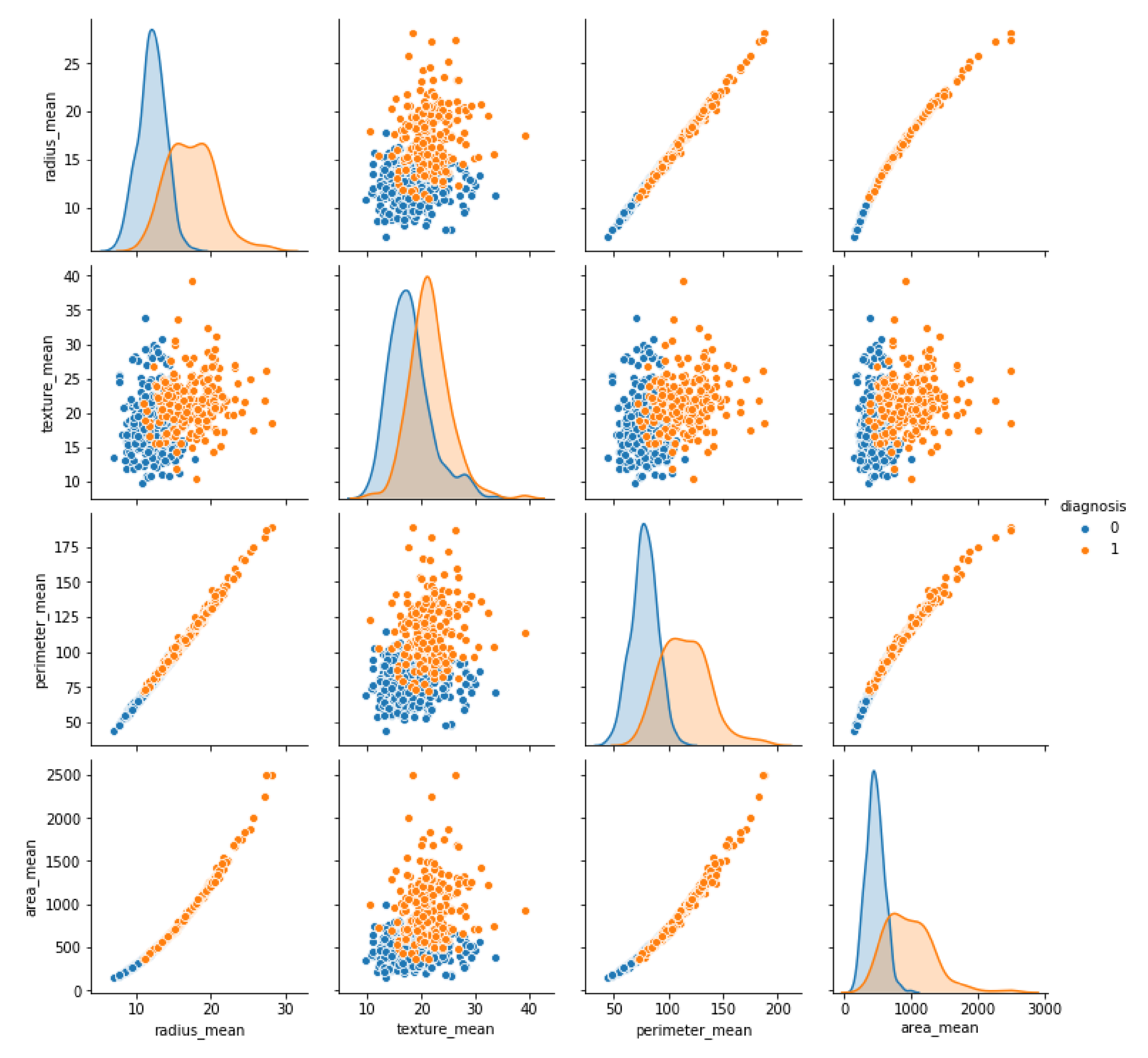
Out[10]:
```
id                           int64
diagnosis                   object
radius_mean                float64
texture_mean               float64
perimeter_mean             float64
area_mean                  float64
smoothness_mean            float64
compactness_mean           float64
concavity_mean             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 [11]:
```
#Encode the categorical data values
from sklearn.preprocessing import LabelEncoder
labelencoder_Y = LabelEncoder()
labelencoder_Y.fit_transform(df.iloc[:,1].values)
df.iloc[:,1] = labelencoder_Y.fit_transform(df.iloc[:,1].values)
df.iloc[:,1]
```

Out[11]:
```
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 [12]:
```
#Create a pairplot using seaborn
sns.pairplot(df.iloc[:,1:6], hue = 'diagnosis')
```

Out[12]:
```
<seaborn.axisgrid.PairGrid at 0x24c4c5c4808>
```



In [13]:
```
#Printing the first 5 rows of new data
df.head(5)
```

Out[13]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	con
0	842302	1	17.99	10.38	122.80	1001.0	0.11840	0.27760	
1	842517	1	20.57	17.77	132.90	1326.0	0.08474	0.07864	
2	84300903	1	19.69	21.25	130.00	1203.0	0.10960	0.15990	
3	84348301	1	11.42	20.38	77.58	386.1	0.14250	0.28390	
4	84358402	1	20.29	14.34	135.10	1297.0	0.10030	0.13280	

5 rows × 32 columns

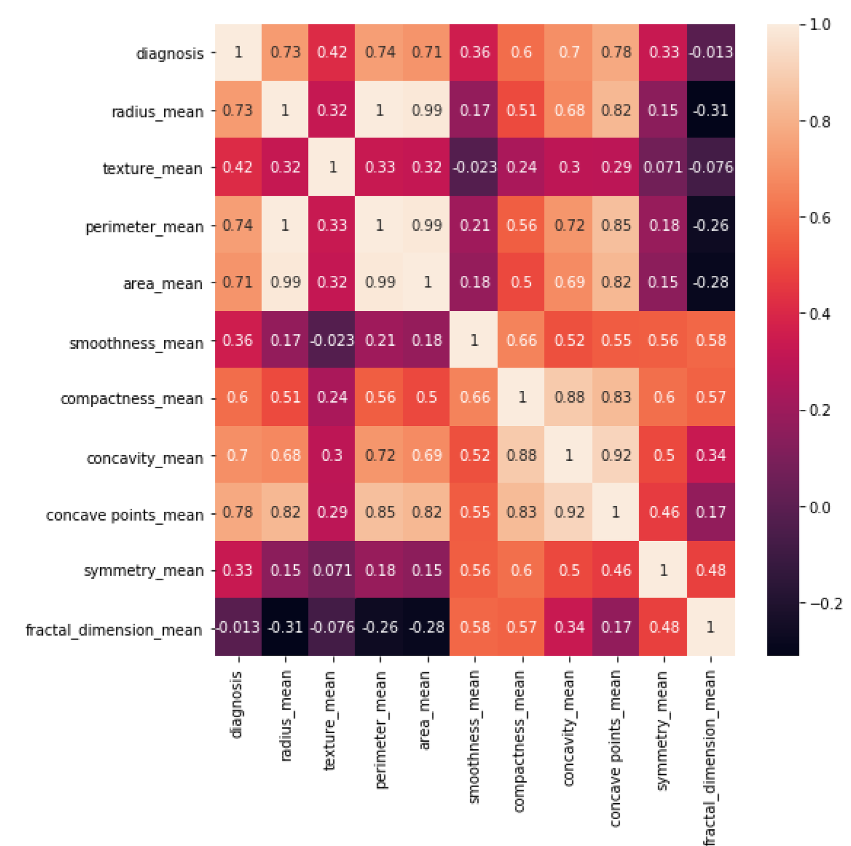
In [14]:
```
#Get the correlation of columns
df.iloc[:,1:12].corr()
```

Out[14]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_
diagnosis	1.000000	0.730029	0.415185	0.742636	0.708984	0.358560	0.5
radius_mean	0.730029	1.000000	0.323782	0.997855	0.987357	0.170581	0.5
texture_mean	0.415185	0.323782	1.000000	0.329533	0.321086	-0.023389	0.2
perimeter_mean	0.742636	0.997855	0.329533	1.000000	0.986507	0.207278	0.5
area_mean	0.708984	0.987357	0.321086	0.986507	1.000000	0.177028	0.4
smoothness_mean	0.358560	0.170581	-0.023389	0.207278	0.177028	1.000000	0.6
compactness_mean	0.596534	0.506124	0.236702	0.556936	0.498502	0.659123	1.0
concavity_mean	0.696360	0.676764	0.302418	0.716136	0.685983	0.521984	0.8
concave points_mean	0.776614	0.822529	0.293464	0.850977	0.823269	0.553695	0.8
symmetry_mean	0.330499	0.147741	0.071401	0.183027	0.151293	0.557775	0.6
fractal_dimension_mean	-0.012838	-0.311631	-0.076437	-0.261477	-0.283110	0.584792	0.5

In [32]:
```
#Visualize the correlation
plt.figure(figsize=(8,8))
sns.heatmap(df.iloc[:,1:12].corr(), annot = True)
```

Out[32]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1b80ca3ba88>
```



In [15]:
```
#Split the dataset into independent(X) and dependent(Y) datasets
X = df.iloc[:,2:31].values
Y = df.iloc[:,1].values
```

In [17]:
```
#Split the dataset into 75% training and 25% testing
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 [18]:
```
#Scale the data (Feature Scaling)
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.fit_transform(X_test)
```

In [19]:
```
#Create a function for models
def models(X_train,Y_train):

    #Logistic Regression
    from sklearn.linear_model import LogisticRegression
    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)

    #Print the models accuracy on training data
    print('[0]Logistic Regression Accuracy:',log.score(X_train ,Y_train))
    print('[1]Decisiontree Classifier Accuracy:',tree.score(X_train ,Y_train))

    return log, tree
```

In [20]:
```
#Getting all models
model = models(X_train , Y_train)
```

```
[0]Logistic Regression Accuracy: 0.9906103286384976
[1]Decisiontree Classifier Accuracy: 1.0
```