In [27]:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
```

In [28]:

```
df = pd.read_csv(r"C:\Users\Praneal\OneDrive\Desktop\breast cancer.csv")
print(df)
```

```
           id diagnosis  radius_mean  texture_mean  perimeter_mean  area_mean  \
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   
..        ...       ...          ...           ...             ...        ...   
564    926424         M        21.56         22.39          142.00     1479.0   
565    926682         M        20.13         28.25          131.20     1261.0   
566    926954         M        16.60         28.08          108.30      858.1   
567    927241         M        20.60         29.33          140.10     1265.0   
568     92751         B         7.76         24.54           47.92      181.0   

     smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
0            0.11840           0.27760         0.30010              0.14710   
1            0.08474           0.07864         0.08690              0.07017   
2            0.10960           0.15990         0.19740              0.12790   
3            0.14250           0.28390         0.24140              0.10520   
4            0.10030           0.13280         0.19800              0.10430   
..               ...               ...             ...                  ...   
564          0.11100           0.11590         0.24390              0.13890   
565          0.09780           0.10340         0.14400              0.09791   
566          0.08455           0.10230         0.09251              0.05302   
567          0.11780           0.27700         0.35140              0.15200   
568          0.05263           0.04362         0.00000              0.00000   

     ...  texture_worst  perimeter_worst  area_worst  smoothness_worst  \
0    ...          17.33           184.60      2019.0           0.16220   
1    ...          23.41           158.80      1956.0           0.12380   
2    ...          25.53           152.50      1709.0           0.14440   
3    ...          26.50            98.87       567.7           0.20980   
4    ...          16.67           152.20      1575.0           0.13740   
..   ...            ...              ...         ...               ...   
564  ...          26.40           166.10      2027.0           0.14100   
565  ...          38.25           155.00      1731.0           0.11660   
566  ...          34.12           126.70      1124.0           0.11390   
567  ...          39.42           184.60      1821.0           0.16500   
568  ...          30.37            59.16       268.6           0.08996   

     compactness_worst  concavity_worst  concave points_worst  symmetry_worst  \
0              0.66560           0.7119                0.2654          0.4601   
```

```
1              0.18660        0.2416             0.1860         0.2750
2              0.42450        0.4504             0.2430         0.3613
3              0.86630        0.6869             0.2575         0.6638
4              0.20500        0.4000             0.1625         0.2364
..                 ...           ...                ...            ...
564            0.21130        0.4107             0.2216         0.2060
565            0.19220        0.3215             0.1628         0.2572
566            0.30940        0.3403             0.1418         0.2218
567            0.86810        0.9387             0.2650         0.4087
568            0.06444        0.0000             0.0000         0.2871

     fractal_dimension_worst  Unnamed: 32
0                    0.11890          NaN
1                    0.08902          NaN
2                    0.08758          NaN
3                    0.17300          NaN
4                    0.07678          NaN
..                       ...          ...
564                  0.07115          NaN
565                  0.06637          NaN
566                  0.07820          NaN
567                  0.12400          NaN
568                  0.07039          NaN

[569 rows x 33 columns]
```

In [29]:

```
df.head()
```

Out[29]:

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

5 rows × 33 columns

In [30]:

```
# for analysis purposes, will look to determine the count of the number of 'empty'
#(for example NA, NaN)
#values in this data set


df.isna().sum()
```

Out[30]:

```
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 [31]:

```
#drop the empty columns with missing values from the dataset.
#Will look to drop the column with all missing values


df=df.dropna(axis=1)
df
```

Out[31]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean
0	842302	M	17.99	10.38	122.80	1001.0	0.11840
1	842517	M	20.57	17.77	132.90	1326.0	0.08474
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960
3	84348301	M	11.42	20.38	77.58	386.1	0.14250
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030
...	...	...	...	...	...	...	...
564	926424	M	21.56	22.39	142.00	1479.0	0.11100
565	926682	M	20.13	28.25	131.20	1261.0	0.09780
566	926954	M	16.60	28.08	108.30	858.1	0.08455
567	927241	M	20.60	29.33	140.10	1265.0	0.11780
568	92751	B	7.76	24.54	47.92	181.0	0.05263

569 rows × 32 columns

In [32]:

```
#checking on the new number of rows and columns
#after any empty values/columns have been dropped
#from the dataset.

df.shape
```

Out[32]:

```
(569, 32)
```

In [33]:

```
#after the data has been cleaned,
#the next step is to determine
#what is the number of 'Benign' and 'Malignant' cases in the dataset.

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

Out[33]:

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

In [34]:

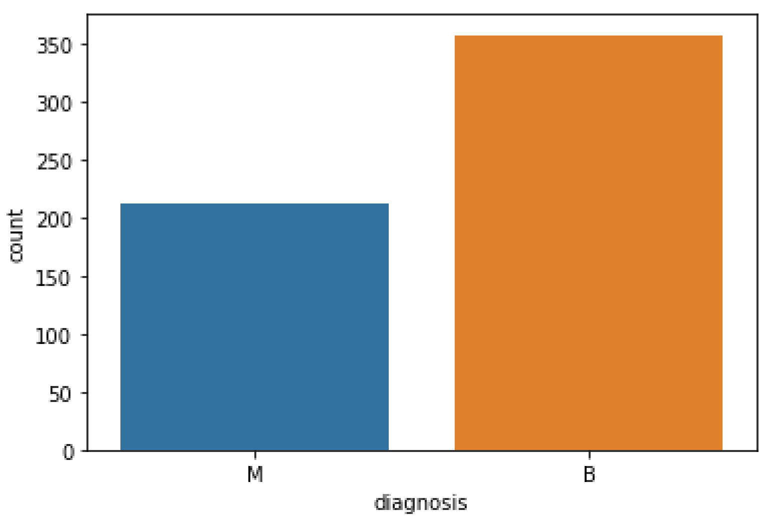
```
#after determing the number values of the Benign and Malignant cases in the dataset,
#the next step will be to visualize these statistics to further understand difference
#in the dataset between number of patients with a Malignant and Benign diagnosis.

sns.countplot(df['diagnosis'], label='Number of cases')
```

```
C:\Users\Praneal\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: P
ass the following variable as a keyword arg: x. From version 0.12, the only valid positi
onal argument will be `data`, and passing other arguments without an explicit keyword wi
ll result in an error or misinterpretation.
  warnings.warn(
```

Out[34]:

```
<AxesSubplot:xlabel='diagnosis', ylabel='count'>
```



In [35]:

```
#another useful step will be to view the data type of each category
#and decide if there needs to be any transformation
#for any of the categories to make it easier for further data analysis

df.dtypes
```

Out[35]:

```
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 [36]:

```
#to make it easier for data analysis purposes,
#the diagnosis category will be converted to numerical values as well.

df['diagnosis']= df['diagnosis'].map({'M':1, 'B':0})

df
```

Out[36]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean
0	842302	1	17.99	10.38	122.80	1001.0	0.11840
1	842517	1	20.57	17.77	132.90	1326.0	0.08474
2	84300903	1	19.69	21.25	130.00	1203.0	0.10960
3	84348301	1	11.42	20.38	77.58	386.1	0.14250
4	84358402	1	20.29	14.34	135.10	1297.0	0.10030
...	...	...	...	...	...	...	...
564	926424	1	21.56	22.39	142.00	1479.0	0.11100
565	926682	1	20.13	28.25	131.20	1261.0	0.09780
566	926954	1	16.60	28.08	108.30	858.1	0.08455

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean
567	927241	1	20.60	29.33	140.10	1265.0	0.11780
568	92751	0	7.76	24.54	47.92	181.0	0.05263

569 rows × 32 columns

In [37]:

```
#after cleaning up the data,
#the next step is to determine the level of correlation between the different categorie

df.corr()

#based on the correlation values,
#it appears that categories such as radius mean, perimeter mean
#and area means are correlated with diagnosis.
```

Out[37]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean
id	1.000000	0.039769	0.074626	0.099770	0.073159	0.096893
diagnosis	0.039769	1.000000	0.730029	0.415185	0.742636	0.708984
radius_mean	0.074626	0.730029	1.000000	0.323782	0.997855	0.987357
texture_mean	0.099770	0.415185	0.323782	1.000000	0.329533	0.321086
perimeter_mean	0.073159	0.742636	0.997855	0.329533	1.000000	0.986507
area_mean	0.096893	0.708984	0.987357	0.321086	0.986507	1.000000
smoothness_mean	-0.012968	0.358560	0.170581	-0.023389	0.207278	0.177028
compactness_mean	0.000096	0.596534	0.506124	0.236702	0.556936	0.498502
concavity_mean	0.050080	0.696360	0.676764	0.302418	0.716136	0.685983
concave points_mean	0.044158	0.776614	0.822529	0.293464	0.850977	0.823269
symmetry_mean	-0.022114	0.330499	0.147741	0.071401	0.183027	0.151293
fractal_dimension_mean	-0.052511	-0.012838	-0.311631	-0.076437	-0.261477	-0.283110
radius_se	0.143048	0.567134	0.679090	0.275869	0.691765	0.732562
texture_se	-0.007526	-0.008303	-0.097317	0.386358	-0.086761	-0.066280
perimeter_se	0.137331	0.556141	0.674172	0.281673	0.693135	0.726628
area_se	0.177742	0.548236	0.735864	0.259845	0.744983	0.800086
smoothness_se	0.096781	-0.067016	-0.222600	0.006614	-0.202694	-0.166777
compactness_se	0.033961	0.292999	0.206000	0.191975	0.250744	0.212583
concavity_se	0.055239	0.253730	0.194204	0.143293	0.228082	0.207660
concave points_se	0.078768	0.408042	0.376169	0.163851	0.407217	0.372320

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean
symmetry_se	-0.017306	-0.006522	-0.104321	0.009127	-0.081629	-0.072497
fractal_dimension_se	0.025725	0.077972	-0.042641	0.054458	-0.005523	-0.019887
radius_worst	0.082405	0.776454	0.969539	0.352573	0.969476	0.962746
texture_worst	0.064720	0.456903	0.297008	0.912045	0.303038	0.287489
perimeter_worst	0.079986	0.782914	0.965137	0.358040	0.970387	0.959120
area_worst	0.107187	0.733825	0.941082	0.343546	0.941550	0.959213
smoothness_worst	0.010338	0.421465	0.119616	0.077503	0.150549	0.123523
compactness_worst	-0.002968	0.590998	0.413463	0.277830	0.455774	0.390410
concavity_worst	0.023203	0.659610	0.526911	0.301025	0.563879	0.512606
concave points_worst	0.035174	0.793566	0.744214	0.295316	0.771241	0.722017
symmetry_worst	-0.044224	0.416294	0.163953	0.105008	0.189115	0.143570
fractal_dimension_worst	-0.029866	0.323872	0.007066	0.119205	0.051019	0.003738

32 rows × 32 columns

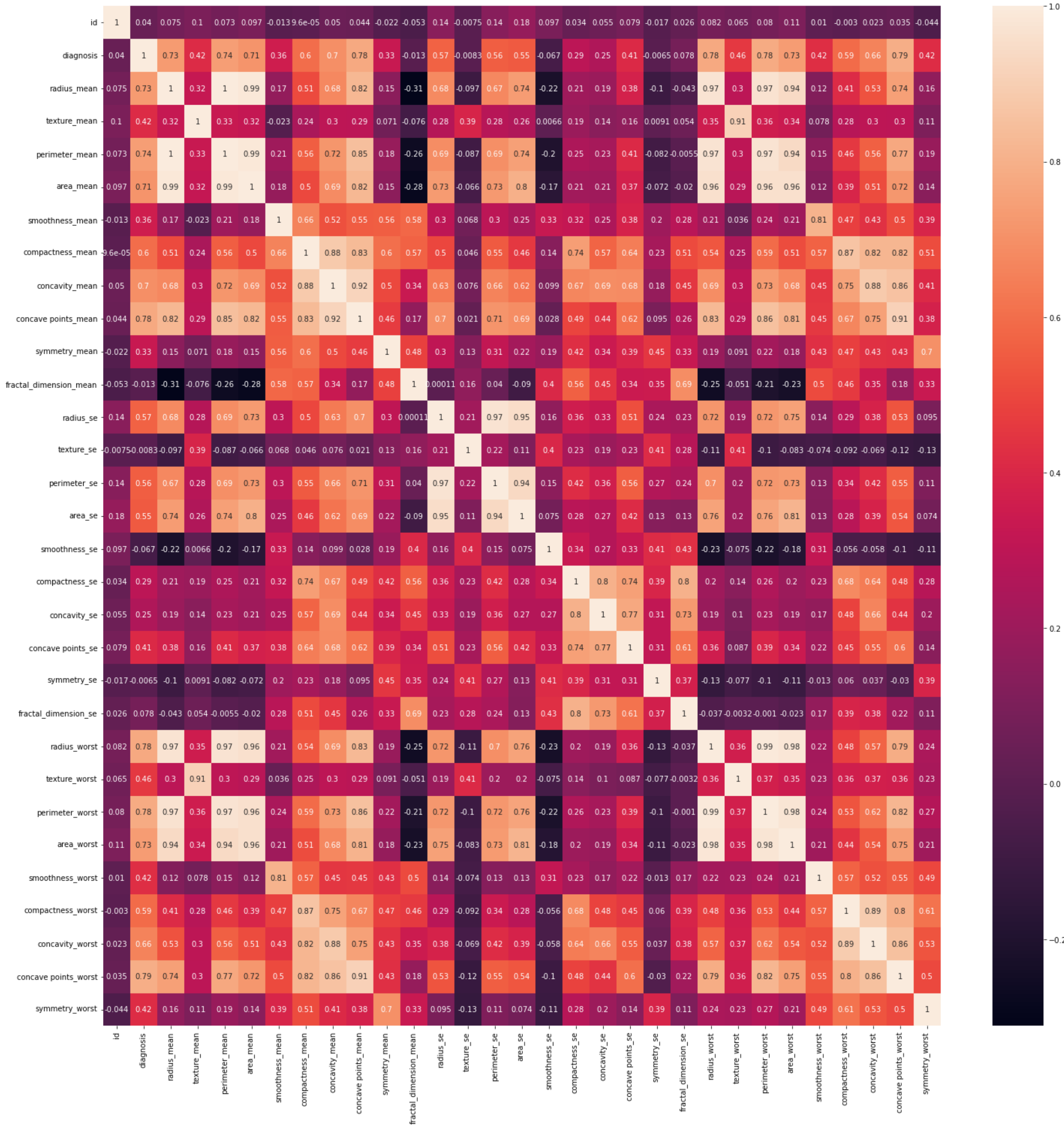
In [38]:

```
#to further understand the levels of correlation of individual characteritics,
#it may be helpful to visualize the correlations as well.

plt.figure(figsize=(25,25))
sns.heatmap(df.iloc[:,0:31].corr(), annot=True)
```

Out[38]:

```
<AxesSubplot:>
```



In [39]:

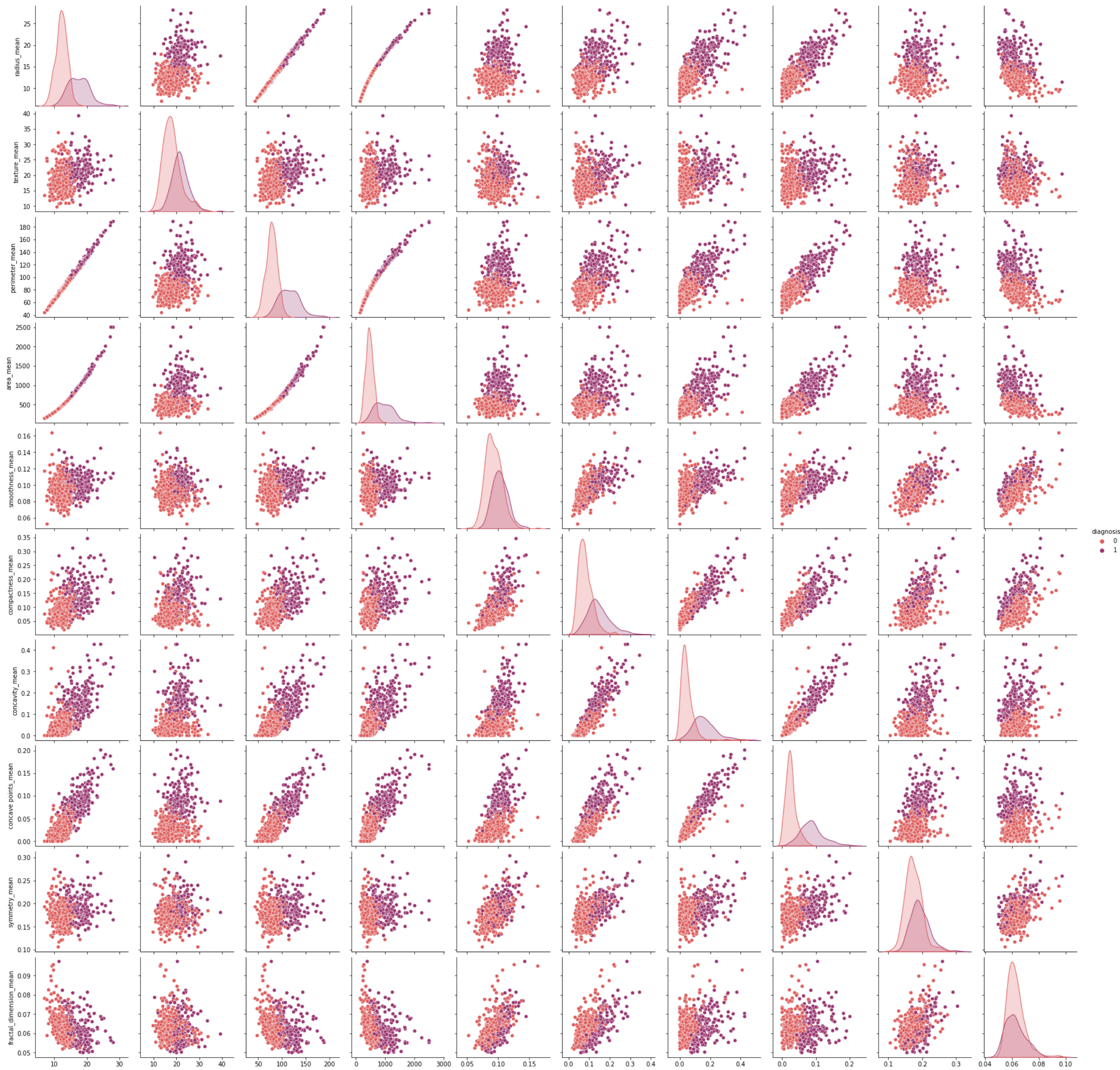
```
#after seeing the correlation values displayed on a correlation plot,
#it is useful to plot the coreelations graphically
#to further understand the strength of correlation of different characteristics with di
#Plotting just the 'mean' characteristics displayed the following:

cols= ['diagnosis',
       'radius_mean',
       'texture_mean',
       'perimeter_mean',
       'area_mean',
       'smoothness_mean',
       'compactness_mean',
       'concavity_mean',
       'concave points_mean',
       'symmetry_mean',
       'fractal_dimension_mean']
```

```
#based on the scatterplots displayed, it can be said that
#radius mean, perimeter mean and area mean
#have a highly linear correlation with the diagnosis of breast cancer.

sns.pairplot(data=df[cols], hue='diagnosis', palette='flare')
```

Out[39]: <seaborn.axisgrid.PairGrid at 0x24fbe6ce3a0>



In [40]:

```
#the next step as a part of the data analysis
#will be to split the data set into 70% training and 30% testing.

X=df.drop(['diagnosis'],axis=1) #X will be independent variables from the data set, whi
Y= df['diagnosis'] #Y will be the dependent variable ('diagnosis')
X_train, X_test, Y_train, Y_test =train_test_split(X, Y, test_size=0.30, random_state=0
```

In [41]:

```
#After establishing the train and testing sets, the next step will be to scale the data
#Scaling is done to bring all the values to a consistent standard
#which makes further data analysis easier.
```

```
ss=StandardScaler()

X_train=ss.fit_transform(X_train)
X_test=ss.fit_transform(X_test)

X_train
```

Out[41]:
```
array([[-0.232028  , -0.74998027, -1.09978744, ..., -0.6235968 ,
         0.07754241,  0.45062841],
       [-0.23217735, -1.02821446, -0.1392617 , ..., -0.7612376 ,
        -1.07145262, -0.29541379],
       [-0.17081111, -0.53852228, -0.29934933, ..., -0.50470441,
         0.34900827, -0.13371556],
       ...,
       [ 6.83303935, -1.3214733 , -0.20855336, ..., -0.98621857,
        -0.69108476, -0.13148524],
       [-0.23231516, -1.24245479, -0.23244704, ..., -1.7562754 ,
        -1.55125275, -1.01078909],
       [-0.2319212 , -0.74441558,  1.13188181, ..., -0.28490593,
        -1.2308599 ,  0.20083251]])
```

In [42]:
```
#Logistic Regression

LR= LogisticRegression()
model1=LR.fit(X_train,Y_train)
prediction1=model1.predict(X_test)

model1
prediction1
```

Out[42]:
```
array([1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 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, 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, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0,
       1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,
       0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0], dtype=int64)
```

In [62]:
```
CM=confusion_matrix(Y_test,prediction1)
CM
#The confusion matrix determines that this model
#is currently predicting 104 cases correctly (True Positive),
#4 cases incorrectly identified (False Positive),
#2 cases that are False Negative,
#and 61 cases which are True Negative.
```

Out[62]:
```
array([[103,   5],
       [  2,  61]], dtype=int64)
```

In [44]:
```
TP=CM[0][0]
TN=CM[1][1]
FN=CM[1][0]
FP=CM[0][1]
print('Testing accuracy:',(TP+TN)/(TP+TN+FN+FP))
```

```
Testing accuracy: 0.9590643274853801
```

In [45]:
```
#create a classification report to display the metrics for the model
#for the basis of comparison
#the classification report will provide precision and recall
#for both benign (0) and malignant(1) cases in the dataset.
#The precision value represents all positive cases, which includes false positives.
#The recall value represents all of the correct positive predictions
#that have been made out of all the positive predictions.

from sklearn.metrics import classification_report
print(classification_report(Y_test,prediction1))
```

```
              precision    recall  f1-score   support

           0       0.98      0.95      0.97       108
           1       0.92      0.97      0.95        63

    accuracy                           0.96       171
   macro avg       0.95      0.96      0.96       171
weighted avg       0.96      0.96      0.96       171
```

In [46]:
```
dtc=DecisionTreeClassifier()
model2= dtc.fit(X_train,Y_train)
prediction2=model2.predict(X_test)
cm2=confusion_matrix(Y_test,prediction2)
cm2
```

Out[46]:
```
array([[97, 11],
       [ 4, 59]], dtype=int64)
```

In [47]:
```
TP=cm2[0][0]
TN=cm2[1][1]
FN=cm2[1][0]
FP=cm2[0][1]
print('Testing accuracy:',(TP+TN)/(TP+TN+FN+FP))
```

```
Testing accuracy: 0.9122807017543859
```

In [48]:
```
#create a classification report
#to display the metrics for the model for the basis of comparison

from sklearn.metrics import classification_report
print(classification_report(Y_test,prediction2))
```

```
              precision    recall  f1-score   support

           0       0.96      0.90      0.93       108
           1       0.84      0.94      0.89        63

    accuracy                           0.91       171
   macro avg       0.90      0.92      0.91       171
weighted avg       0.92      0.91      0.91       171
```

In [49]:
```
rfc=RandomForestClassifier()
model3=rfc.fit(X_train,Y_train)
prediction3=model3.predict(X_test)
```

```
cm3=confusion_matrix(Y_test,prediction3)
cm3
```

Out[49]:
```
array([[102,   6],
       [  0,  63]], dtype=int64)
```

In [50]:
```
TP=cm3[0][0]
TN=cm3[1][1]
FN=cm3[1][0]
FP=cm3[0][1]
print('Testing accuracy:',(TP+TN)/(TP+TN+FN+FP))
```

```
Testing accuracy: 0.9649122807017544
```

In [51]:
```
#create a classification report
#to display the metrics for the model for the basis of comparison

from sklearn.metrics import classification_report
print(classification_report(Y_test,prediction3))
```

```
              precision    recall  f1-score   support

           0       1.00      0.94      0.97       108
           1       0.91      1.00      0.95        63

    accuracy                           0.96       171
   macro avg       0.96      0.97      0.96       171
weighted avg       0.97      0.96      0.97       171
```

In [52]:
```
knn=KNeighborsClassifier()
model4=knn.fit(X_train,Y_train)
prediction4=model4.predict(X_test)
cm4=confusion_matrix(Y_test,prediction4)
cm4
```

Out[52]:
```
array([[105,   3],
       [  5,  58]], dtype=int64)
```

In [53]:
```
TP=cm4[0][0]
TN=cm4[1][1]
FN=cm4[1][0]
FP=cm4[0][1]
print('Testing accuracy:',(TP+TN)/(TP+TN+FN+FP))
```

```
Testing accuracy: 0.9532163742690059
```

In [54]:
```
from sklearn.metrics import classification_report
print(classification_report(Y_test,prediction4))
```

```
              precision    recall  f1-score   support

           0       0.95      0.97      0.96       108
           1       0.95      0.92      0.94        63

    accuracy                           0.95       171
```

```
   macro avg       0.95      0.95      0.95       171
weighted avg       0.95      0.95      0.95       171
```

In [55]:

```
svm=SVC()
model5=svm.fit(X_train,Y_train)
prediction5=model5.predict(X_test)
cm5=confusion_matrix(Y_test,prediction4)
cm5
```

Out[55]:

```
array([[105,   3],
       [  5,  58]], dtype=int64)
```

In [56]:

```
TP=cm5[0][0]
TN=cm5[1][1]
FN=cm5[1][0]
FP=cm5[0][1]
print('Testing accuracy:',(TP+TN)/(TP+TN+FN+FP))
```

```
Testing accuracy: 0.9532163742690059
```

In [57]:

```
#create a classification report
#to display the metrics for the model for the basis of comparison


from sklearn.metrics import classification_report
print(classification_report(Y_test,prediction5))
```

```
              precision    recall  f1-score   support

           0       0.97      0.98      0.98       108
           1       0.97      0.95      0.96        63

    accuracy                           0.97       171
   macro avg       0.97      0.97      0.97       171
weighted avg       0.97      0.97      0.97       171
```

In [58]:

```
#As a part of this initial data analysis,
#it appears as though the Support Vector Machine model
#is the best model to work with this data set to help determine the characteristics
#that predict benign and malignant cancer diagnosis in cancer patients.
#With the highest precision, recall and accuracy scores,
#this may be the model to use for further data analysis.
#Further data analysis may be needed to confirm that the SVM
#is the best model to be using for this data set.
```