# Initializing the libs and data

In [4]:

```
import pandas as pd #for data frame operations
import numpy as np #for mathematical operations
import matplotlib.pyplot as plt #for data visualization
import seaborn as sns #for data visualization
```

In [5]:

```
import warnings#to remove error messages
warnings.filterwarnings('ignore')
```

In [6]:

```
df = pd.read_csv(r"C:\Users\Aditya Singh\Downloads\data.csv")
#importing the csv file
```

# Knowing the data

In [5]:

```
df.head() #top 5 rows
```

Out[5]:

| texture_worst | perimeter_worst | area_worst | smoothness_worst | compactness_worst | concavity_wor |
|---|---|---|---|---|---|
| 17.33 | 184.60 | 2019.0 | 0.1622 | 0.6656 | 0.71 |
| 23.41 | 158.80 | 1956.0 | 0.1238 | 0.1866 | 0.24 |
| 25.53 | 152.50 | 1709.0 | 0.1444 | 0.4245 | 0.45 |
| 26.50 | 98.87 | 567.7 | 0.2098 | 0.8663 | 0.68 |
| 16.67 | 152.20 | 1575.0 | 0.1374 | 0.2050 | 0.40 |

In [6]:

```
df.tail()#last 5 columns
```

Out[6]:

| | id | diagnosis | radius_mean | texture_mean | perimeter_mean | area_mean | smoothness_ |
|---|---|---|---|---|---|---|---|
| **564** | 926424 | M | 21.56 | 22.39 | 142.00 | 1479.0 | 0. |
| **565** | 926682 | M | 20.13 | 28.25 | 131.20 | 1261.0 | 0.( |
| **566** | 926954 | M | 16.60 | 28.08 | 108.30 | 858.1 | 0.( |
| **567** | 927241 | M | 20.60 | 29.33 | 140.10 | 1265.0 | 0. |
| **568** | 92751 | B | 7.76 | 24.54 | 47.92 | 181.0 | 0.( |

5 rows × 33 columns

In [8]:

```
df.shape#no. of rows and col
```

Out[8]:

(569, 33)

In [12]:

```
df.describe().T#for diff stats of data and T for transpose

#for the mean , std etc column e get data as e+ so need to standardize it
```

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **area_se** | 569.0 | 4.033708e+01 | 4.549101e+01 | 6.802000 | 17.850000 | 24.530000 | 4.51 |
| **smoothness_se** | 569.0 | 7.040979e-03 | 3.002518e-03 | 0.001713 | 0.005169 | 0.006380 | 8.1 |
| **compactness_se** | 569.0 | 2.547814e-02 | 1.790818e-02 | 0.002252 | 0.013080 | 0.020450 | 3.2 |
| **concavity_se** | 569.0 | 3.189372e-02 | 3.018606e-02 | 0.000000 | 0.015090 | 0.025890 | 4.2( |
| **concave points_se** | 569.0 | 1.179614e-02 | 6.170285e-03 | 0.000000 | 0.007638 | 0.010930 | 1.4 |
| **symmetry_se** | 569.0 | 2.054230e-02 | 8.266372e-03 | 0.007882 | 0.015160 | 0.018730 | 2.3 |
| **fractal_dimension_se** | 569.0 | 3.794904e-03 | 2.646071e-03 | 0.000895 | 0.002248 | 0.003187 | 4.5 |
| **radius_worst** | 569.0 | 1.626919e+01 | 4.833242e+00 | 7.930000 | 13.010000 | 14.970000 | 1.87 |
| **texture_worst** | 569.0 | 2.567722e+01 | 6.146258e+00 | 12.020000 | 21.080000 | 25.410000 | 2.97 |
| **perimeter_worst** | 569.0 | 1.072612e+02 | 3.360254e+01 | 50.410000 | 84.110000 | 97.660000 | 1.25 |
| **area_worst** | 569.0 | 8.805831e+02 | 5.693570e+02 | 185.200000 | 515.300000 | 686.500000 | 1.08 |
| **smoothness_worst** | 569.0 | 1.323686e-01 | 2.283243e-02 | 0.071170 | 0.116600 | 0.131300 | 1.4( |

In [7]:

```
df.diagnosis.unique()#finds the no. of types of a value
```

Out[7]:

```
array(['M', 'B'], dtype=object)
```

In [9]:

```
df['diagnosis'].value_counts()#gives the count of values
```

Out[9]:

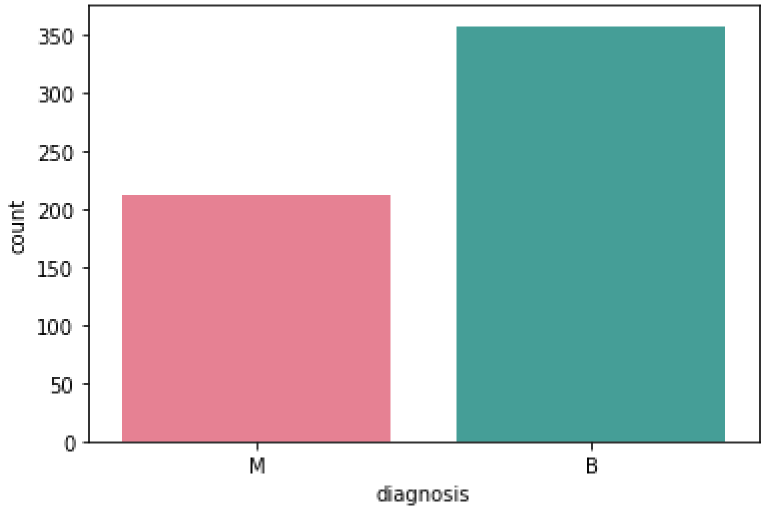
```
B    357
M    212
Name: diagnosis, dtype: int64
```

In [10]:

```
sns.countplot(df['diagnosis'],palette='husl')
```

Out[10]:

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




# Cleaning the data

In [11]:

```
df.drop('id',axis=1,inplace=True)#dropped the insignifacnt column which are not needed
df.drop('Unnamed: 32',axis=1,inplace=True)
```

In [12]:

```
df.head()
```

Out[12]:

| | diagnosis | radius_mean | texture_mean | perimeter_mean | area_mean | smoothness_mean | com |
|---|---|---|---|---|---|---|---|
| **0** | M | 17.99 | 10.38 | 122.80 | 1001.0 | 0.11840 | |
| **1** | M | 20.57 | 17.77 | 132.90 | 1326.0 | 0.08474 | |
| **2** | M | 19.69 | 21.25 | 130.00 | 1203.0 | 0.10960 | |
| **3** | M | 11.42 | 20.38 | 77.58 | 386.1 | 0.14250 | |
| **4** | M | 20.29 | 14.34 | 135.10 | 1297.0 | 0.10030 | |

5 rows × 31 columns

In [14]:

```
#as we cant have textual data so we will map it to numeric as we have two values and bo

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

df.head()
```

Out[14]:

| | diagnosis | radius_mean | texture_mean | perimeter_mean | area_mean | smoothness_mean | com |
|---|---|---|---|---|---|---|---|
| **0** | 1 | 17.99 | 10.38 | 122.80 | 1001.0 | 0.11840 | |
| **1** | 1 | 20.57 | 17.77 | 132.90 | 1326.0 | 0.08474 | |
| **2** | 1 | 19.69 | 21.25 | 130.00 | 1203.0 | 0.10960 | |
| **3** | 1 | 11.42 | 20.38 | 77.58 | 386.1 | 0.14250 | |
| **4** | 1 | 20.29 | 14.34 | 135.10 | 1297.0 | 0.10030 | |

5 rows × 31 columns

In [20]:

```
#now we check if any cell is empty/NULL

df.isnull().sum()
```

```
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
```

In [19]:

```
#def diagnosis_value(diagnosis):
#    if diagnosis == 'M':
#        return 1
#    else:
#        return 0


#df['diagnosis'] = df['diagnosis'].apply(diagnosis_value)


```

In [21]:

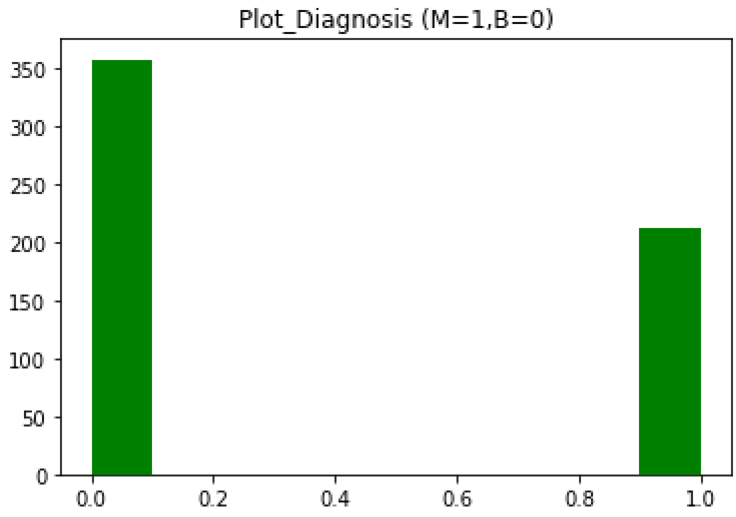
```
#knowing the correlation
df.corr()
```

Out[21]:

| worst | texture_worst | perimeter_worst | area_worst | smoothness_worst | compactness_worst | concav |
|---|---|---|---|---|---|---|
| 6454 | 0.456903 | 0.782914 | 0.733825 | 0.421465 | 0.590998 | |
| 9539 | 0.297008 | 0.965137 | 0.941082 | 0.119616 | 0.413463 | |
| 2573 | 0.912045 | 0.358040 | 0.343546 | 0.077503 | 0.277830 | |
| 9476 | 0.303038 | 0.970387 | 0.941550 | 0.150549 | 0.455774 | |
| 2746 | 0.287489 | 0.959120 | 0.959213 | 0.123523 | 0.390410 | |
| 3120 | 0.036072 | 0.238853 | 0.206718 | 0.805324 | 0.472468 | |
| 5315 | 0.248133 | 0.590210 | 0.509604 | 0.565541 | 0.865809 | |
| 8236 | 0.299879 | 0.729565 | 0.675987 | 0.448822 | 0.754968 | |
| 0318 | 0.292752 | 0.855923 | 0.809630 | 0.452753 | 0.667454 | |
| 5728 | 0.090651 | 0.219169 | 0.177193 | 0.426675 | 0.473200 | |
| 3691 | -0.051269 | -0.205151 | -0.231854 | 0.504942 | 0.458798 | |
| 5065 | 0.194799 | 0.719684 | 0.751548 | 0.141919 | 0.287103 | |
| 1690 | 0.409003 | -0.102242 | -0.083195 | -0.073658 | -0.092439 | |
| 7201 | 0.200371 | 0.721031 | 0.730713 | 0.130054 | 0.341919 | |
| 7373 | 0.196497 | 0.761213 | 0.811408 | 0.125389 | 0.283257 | |
| 0691 | -0.074743 | -0.217304 | -0.182195 | 0.314457 | -0.055558 | |
| 4607 | 0.143003 | 0.260516 | 0.199371 | 0.227394 | 0.678780 | |
| 6904 | 0.100241 | 0.226680 | 0.188353 | 0.168481 | 0.484858 | |
| 8127 | 0.086741 | 0.394999 | 0.342271 | 0.215351 | 0.452888 | |
| 8121 | -0.077473 | -0.103753 | -0.110343 | -0.012662 | 0.060255 | |
| 7488 | -0.003195 | -0.001000 | -0.022736 | 0.170568 | 0.390159 | |
| 0000 | 0.359921 | 0.993708 | 0.984015 | 0.216574 | 0.475820 | |
| 9921 | 1.000000 | 0.365098 | 0.345842 | 0.225429 | 0.360832 | |
| 3708 | 0.365098 | 1.000000 | 0.977578 | 0.236775 | 0.529408 | |
| 4015 | 0.345842 | 0.977578 | 1.000000 | 0.209145 | 0.438296 | |
| 6574 | 0.225429 | 0.236775 | 0.209145 | 1.000000 | 0.568187 | |
| 5820 | 0.360832 | 0.529408 | 0.438296 | 0.568187 | 1.000000 | |
| 3975 | 0.368366 | 0.618344 | 0.543331 | 0.518523 | 0.892261 | |
| 7424 | 0.359755 | 0.816322 | 0.747419 | 0.547691 | 0.801080 | |
| 3529 | 0.233027 | 0.269493 | 0.209146 | 0.493838 | 0.614441 | |
| 3492 | 0.219122 | 0.138957 | 0.079647 | 0.617624 | 0.810455 | |

In [22]:

```
#plotting the histo M,B

plt.hist(df['diagnosis'],color='g')
plt.title('Plot_Diagnosis (M=1,B=0)')
plt.show()
```

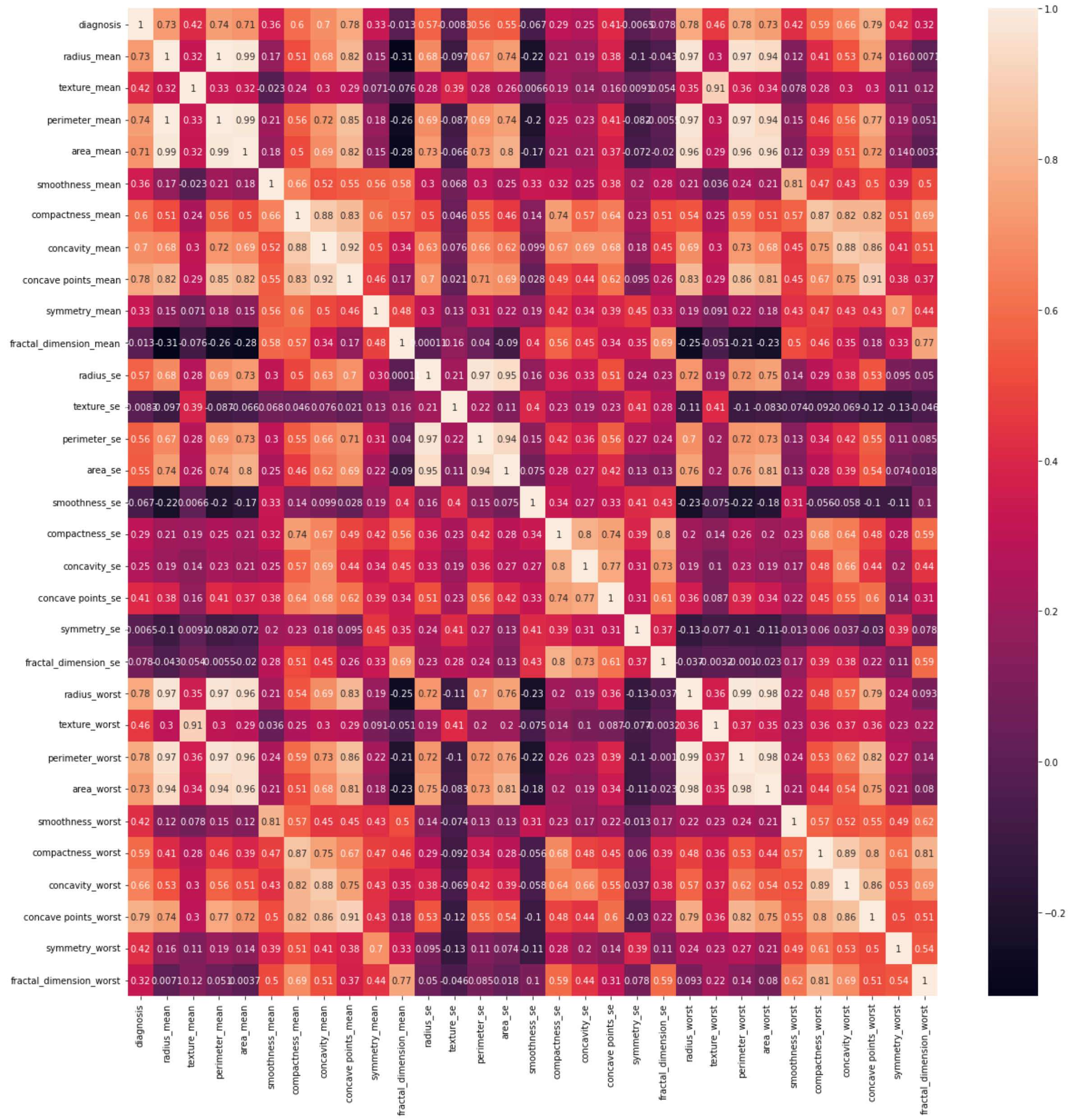


In [25]:

```
#plotting corr heatmap

plt.figure(figsize=(20,20))
sns.heatmap(df.corr(),annot=True)
```

Out[25]:

<AxesSubplot:>



In [29]:

```
# generate a scatter plot matrix with the "mean" columns
cols = ['diagnosis',
        'radius_mean',
        'texture_mean',
        'perimeter_mean',
        'area_mean',
        'smoothness_mean',
        'compactness_mean',
        'concavity_mean',
        'concave points_mean',
        'symmetry_mean',
        'fractal_dimension_mean']

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

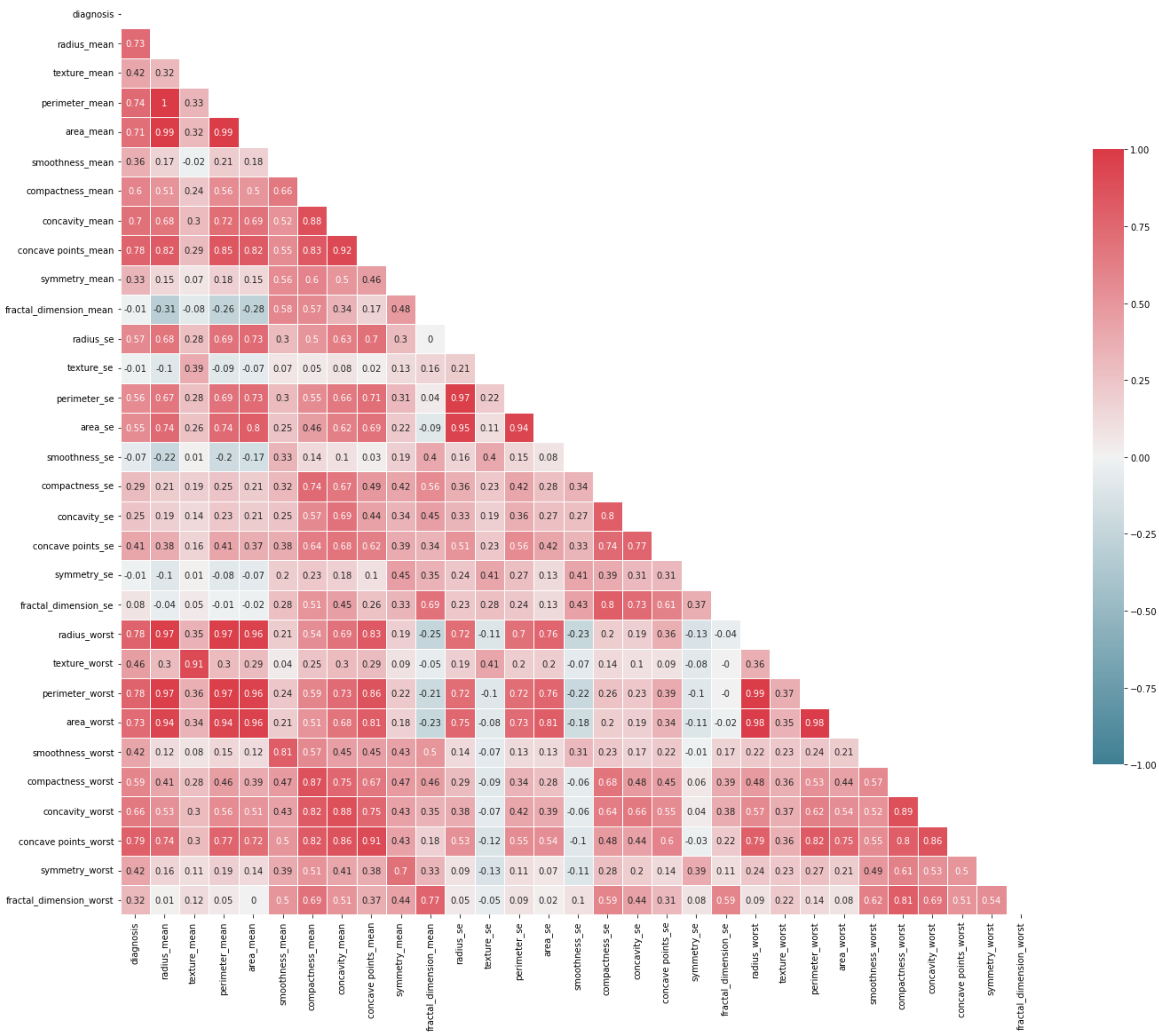
Out[29]:

```
<seaborn.axisgrid.PairGrid at 0x25fa3618160>
```

almost perfectly linear patterns between the radius, perimeter and area attributes are hinting at the presence of multicollinearity between these variables. (they are highly linearly related) Another set of variables that possibly imply multicollinearity are the concavity, concave_points and compactness.

In [28]:

```
# Generate and visualize the correlation matrix
corr = df.corr().round(2)

# Mask for the upper triangle
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True

# Set figure size
f, ax = plt.subplots(figsize=(20, 20))

# Define custom colormap
cmap = sns.diverging_palette(220, 10, as_cmap=True)

# Draw the heatmap
sns.heatmap(corr, mask=mask, cmap=cmap, vmin=-1, vmax=1, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5}, annot=True)

plt.tight_layout()
```



also there is multicollinearity between the attributes compactness, concavity, and concave points. So we can choose just ONE out of these, I am going for Compactness.

In [30]:

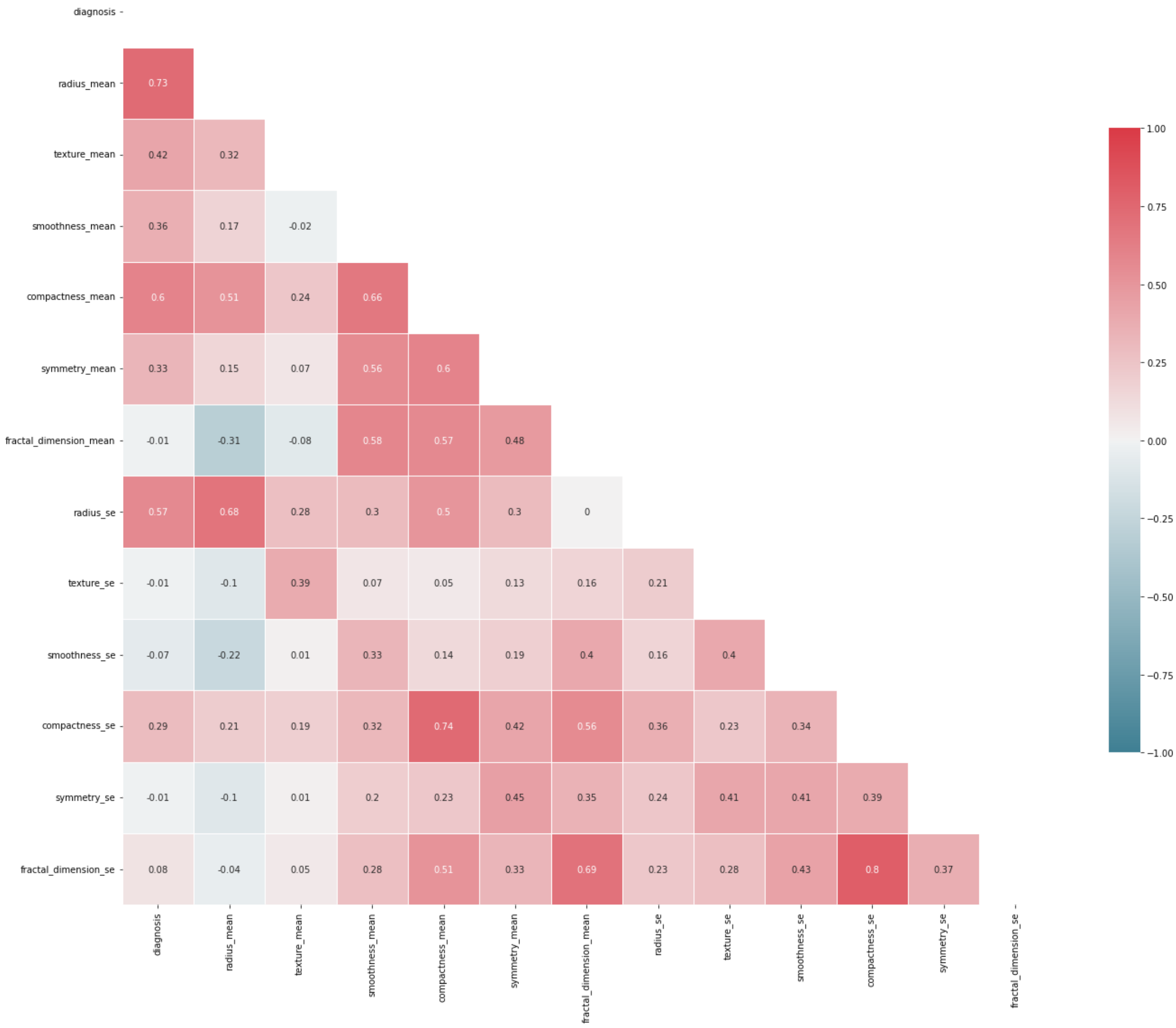
```
# first, drop all "worst" columns
cols = ['radius_worst',
        'texture_worst',
        'perimeter_worst',
        'area_worst',
        'smoothness_worst',
        'compactness_worst',
        'concavity_worst',
        'concave points_worst',
        'symmetry_worst',
        'fractal_dimension_worst']
df = df.drop(cols, axis=1)

# then, drop all columns related to the "perimeter" and "area" attributes
cols = ['perimeter_mean',
        'perimeter_se',
        'area_mean',
        'area_se']
df = df.drop(cols, axis=1)

# lastly, drop all columns related to the "concavity" and "concave points" attributes
cols = ['concavity_mean',
        'concavity_se',
        'concave points_mean',
        'concave points_se']
df = df.drop(cols, axis=1)

# verify remaining columns
df.columns
```

Out[30]:

```
Index(['diagnosis', 'radius_mean', 'texture_mean', 'smoothness_mean',
       'compactness_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'smoothness_se', 'compactness_se',
       'symmetry_se', 'fractal_dimension_se'],
      dtype='object')
```

In [31]:

```
# Draw the heatmap again, with the new correlation matrix
corr = df.corr().round(2)
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True

f, ax = plt.subplots(figsize=(20, 20))
sns.heatmap(corr, mask=mask, cmap=cmap, vmin=-1, vmax=1, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5}, annot=True)
plt.tight_layout()
```



# Building Model

In [32]:

```
X=df.drop(['diagnosis'],axis=1)
y=df['diagnosis']

```

In [33]:

```
from sklearn.model_selection import train_test_split
```

In [34]:

```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=40)
```

# Feature Scaling

In [35]:

```
from sklearn.preprocessing import StandardScaler
ss=StandardScaler()

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

# Models and finding out the Best one

# #Logistic Regression

In [36]:

```
from sklearn.linear_model import LogisticRegression
lr=LogisticRegression()

model1=lr.fit(X_train,y_train)
prediction1=model1.predict(X_test)
```

In [37]:

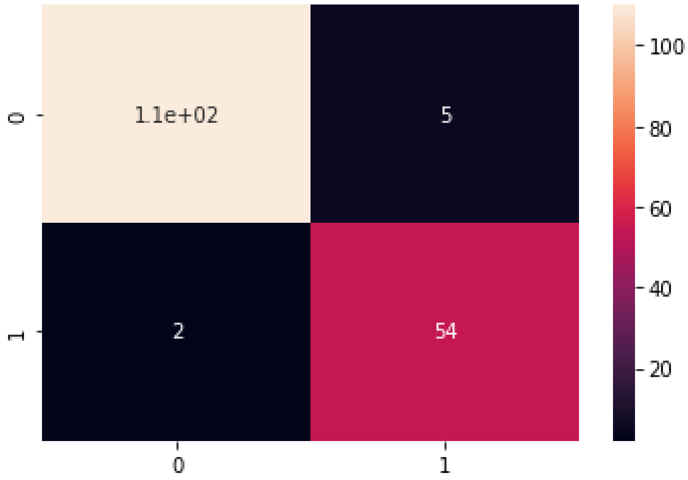
```
from sklearn.metrics import confusion_matrix

cm=confusion_matrix(y_test,prediction1)
cm
```

Out[37]:

```
array([[110,   5],
       [  2,  54]], dtype=int64)
```

In [38]:

```
sns.heatmap(cm,annot=True)
plt.savefig('h.png')
```




In [39]:

```
from sklearn.metrics import accuracy_score
```

In [40]:

```
accuracy_score(y_test,prediction1)
```

Out[40]:

```
0.9590643274853801
```

# Decision Tree

In [41]:

```
from sklearn.tree import DecisionTreeClassifier

dtc=DecisionTreeClassifier()
model2=dtc.fit(X_train,y_train)
prediction2=model2.predict(X_test)
cm2= confusion_matrix(y_test,prediction2)
```

In [42]:

```
accuracy_score(y_test,prediction2)
```

Out[42]:

```
0.9122807017543859
```

# Random Forest

In [43]:

```
from sklearn.ensemble import RandomForestClassifier

rfc=RandomForestClassifier()
model3 = rfc.fit(X_train, y_train)
prediction3 = model3.predict(X_test)
confusion_matrix(y_test, prediction3)
```

Out[43]:

```
array([[109,   6],
       [  5,  51]], dtype=int64)
```

In [44]:

```
accuracy_score(y_test, prediction3)
```

Out[44]:

```
0.935672514619883
```

# K Nearest Neighbor (K NN)

# Support Vector Machine

# Naive Bayes

In [45]:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
```

In [47]:

```
clf = GaussianNB()
clf.fit(X_train,y_train)
pred = clf.predict(X_test)
accuracy_score(pred,y_test)

```

Out[47]:

```
0.9298245614035088
```

In [48]:

```
clf = SVC(kernel="linear")
clf.fit(X_train,y_train)
pred1 = clf.predict(X_test)
accuracy_score(y_test,pred1)
```

Out[48]:

```
0.9590643274853801
```

In [52]:

```
clf_knn = KNeighborsClassifier(n_neighbors=4)
clf_knn.fit(X_train,y_train)
pred11 = clf_knn.predict(X_test)
accuracy_score(y_test,pred11)
```

Out[52]:

```
0.9590643274853801
```

We get the highest accuracy of approx 96% from SVM and KNN

In [ ]:

```

```