Breast Cancer Data Analysis

In [2]:
```
#importing the dataset
df = pd.read_csv('C:/Users/zizhe/Desktop/Leo Zhao/Breast Cancer Prediction/data.csv')
df.head()
```

nean	concavity_mean	concave points_mean	...	texture_worst	perimeter_worst	area_worst	smoothness_worst	compactness_worst	concavity_worst	concave points_worst
7760	0.3001	0.14710	...	17.33	184.60	2019.0	0.1622	0.6656	0.7119	0.2654
7864	0.0869	0.07017	...	23.41	158.80	1956.0	0.1238	0.1866	0.2416	0.1860
5990	0.1974	0.12790	...	25.53	152.50	1709.0	0.1444	0.4245	0.4504	0.2430
8390	0.2414	0.10520	...	26.50	98.87	567.7	0.2098	0.8663	0.6869	0.2575
3280	0.1980	0.10430	...	16.67	152.20	1575.0	0.1374	0.2050	0.4000	0.1625

Data Preprocessing Part 1

In [3]:
```
# dropping unnecessary columns
df.drop([32','id'],axis=1,inplace=True)
```

In [4]:
```
df.head()
```

Out[4]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean
0	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710
1	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017
2	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790
3	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520
4	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430

5 rows × 31 columns

In [5]:

```
#checking for the missing values
df.isnull().sum()
```

Out[5]:
```
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
```

In [6]:
```
#checking the data types of the columns
df.dtypes
```

Out[6]:
```
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 [19]:
```
# checking the data description
df.describe()
```

Out[19]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symm
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.048919	
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.038803	
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.000000	
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.020310	
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.033500	
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.074000	
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.201200	

8 rows × 30 columns

In [11]:
```
# Remove any leading or trailing spaces from column names
df.columns = df.columns.strip()
```

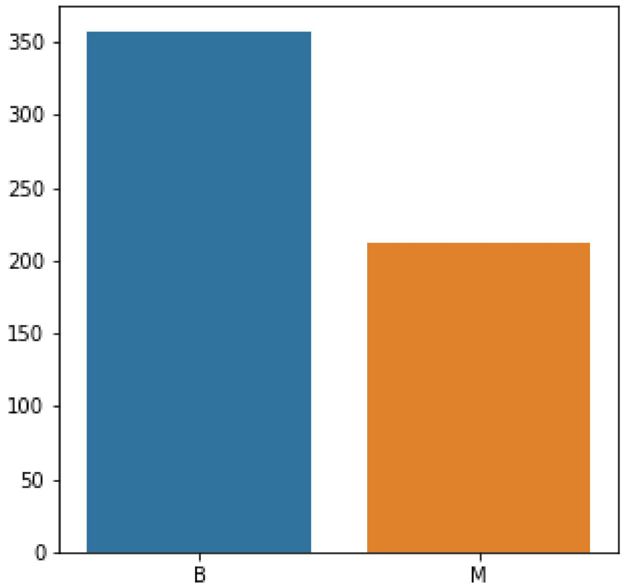
In [12]:
```
# Check the corrected column names
print(df.columns)
```

```
Index(['diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst'],
      dtype='object')
```

Exploratory Data Analysis

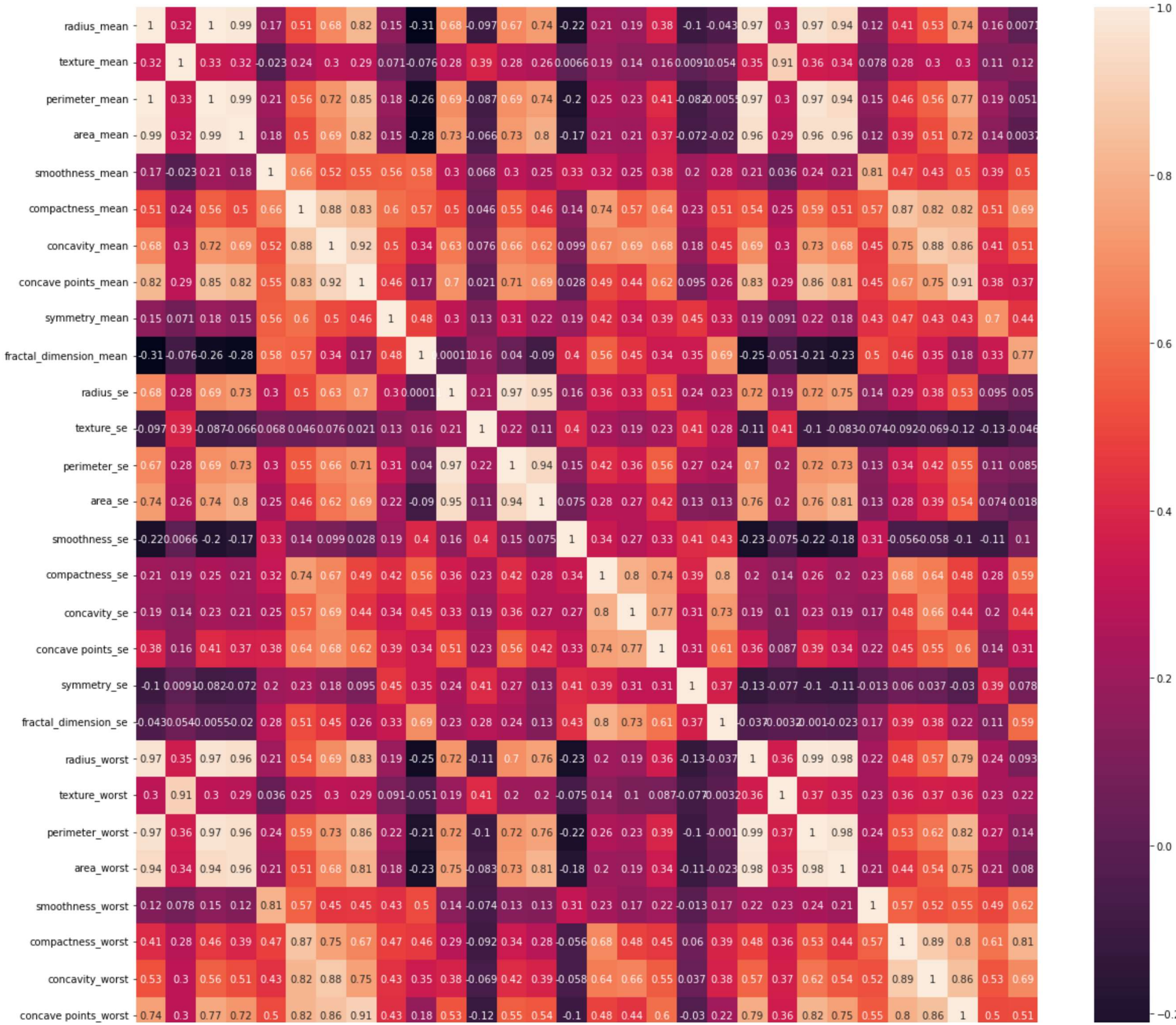
In [17]:
```
# bar plot for the number of diagnosis
plt.figure(figsize=(5,5)) #plt.figure(figsize=(5, 5)):
sns.barplot(x=df['diagnosis'].value_counts(),y=df['diagnosis'].value_counts().values)
```

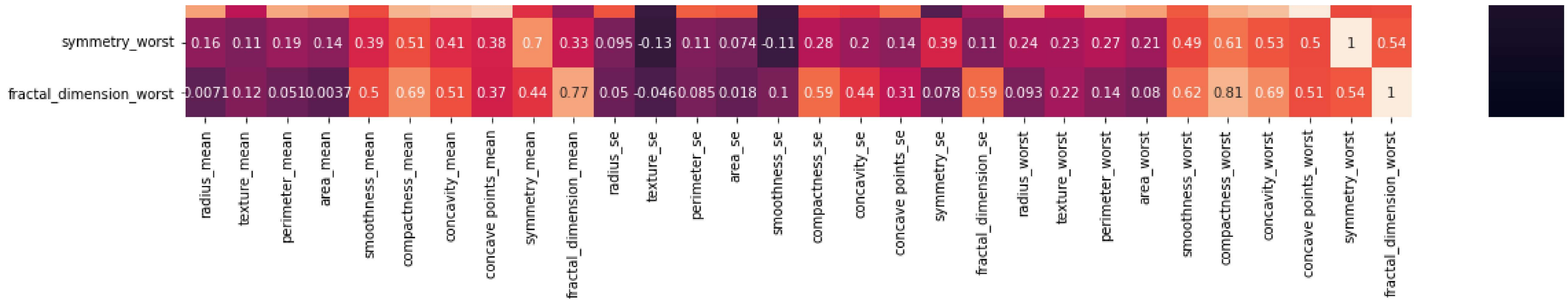
Out[17]: <AxesSubplot:>



In [21]:
```
# create a heatmap to check the correlation.
plt.figure(figsize=(20,20))
sns.heatmap(df.cor(),annot=True)
```

Out[21]: <AxesSubplot:>


radius_mean
texture_mean
perimeter_mean
area_mean
smoothness_mean
compactness_mean
concavity_mean
concave points_mean
symmetry_mean
fractal_dimension_mean
radius_se
texture_se
perimeter_se
area_se
smoothness_se
compactness_se
concavity_se
concave points_se
symmetry_se
fractal_dimension_se
radius_worst
texture_worst
perimeter_worst
area_worst
smoothness_worst
compactness_worst
concavity_worst
concave points_worst
1.0
0.8
0.6
0.4
0.2
0.0
−0.2



Train Test Split

In [22]:
```
#split a DataFrame 'df' into training and testing datasets for machine learning purposes.

from sklearn.model_selection import train_test_split

X_train,X_test,y_train,y_test = train_test_split(df.drop(['diagnosis']),df['diagnosis'],test_size,random_state=42)
```

Using Decision Tree Classifier

In [23]:
```
from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier()
dtree.fit(X_train,y_train)
```

Out[23]:
```
DecisionTreeClassifier()
```

In [24]:
```
#predicting the diagnosis
y_pred = dtree.predict(X_test)
```

Model Evaluation

In [25]:
```
# printing samples from predicted and actual values
print('Predicted values: ',y_pred[:10])
print('Actual values:y_test[:10])
```

```
Predicted values:  ['B' 'M' 'M' 'B' 'B' 'M' 'M' 'M' 'B' 'B']
Actual values:  204    B
70     M
131    M
431    B
540    B
567    M
369    M
29     M
81     B
477    B
Name: diagnosis, dtype: object
```

In [26]:
```
# model evaluation
print(dtree.score(X_test,y_test))
```

```
0.9415204678362573
```

In [ ]:
```
# 0.94 means that the model is making correct predictions for approximately 94% of the samples in the test data.
```

Using logistic regression

In [28]:
```
from sklearn.linear_model import LogisticRegression
logmodel = LogisticRegression()
logmodel(X_train,y_train)
```

```
C:\Users\zizhe\New folder\lib\site-packages\sklearn\linear_model\_logistic.py:814: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Out[28]:
```
LogisticRegression()
```

In [30]:
```
yhat = logmodel.predict(X_test)

#yhat: This variable stores the predicted labels for the test data. After running this line of code, yhat will be an ar
```

Model Evaluation

In [31]:
```
# printing samples from predicted and actual values
print('Predicted values: ',yhat[:10])
print('Actual values: ',y_test[:10])
# If the predicted values closely match the actual values, it's a positive sign that your model is making accurate pred
```

```
Predicted values:  ['B' 'M' 'M' 'B' 'B' 'M' 'M' 'M' 'B' 'B']
Actual values:  204    B
70     M
131    M
431    B
540    B
567    M
369    M
29     M
81     B
477    B
Name: diagnosis, dtype: object
```

In [ ]:
```
#The numbers "204" and "477" you see in the output are actually indices or row numbers from your test dataset. In the c
```

In [32]:
```
# model evaluation
print(logmodel.score(X_test,y_test))

#logmodel.score(X_test, y_test), calculates and prints the accuracy of the logistic regression model (logmodel) on the
```

```
0.9707602339181286
```

In [ ]:
```
# Conclusion
```

In [ ]:
```
# From both the models we can see that the accuracy is 94% and 97% respectively. But we can see that the recall value f
```