In [1]:
```python
#data annalysis libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```python
data=pd.read_csv(r"C:\Users\91852\Downloads\data.csv")
data.head()
```

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 rows × 33 columns

In [3]:
```python
data.shape
```

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

In [4]:
```python
data.isnull().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]:
```
data=data.dropna(axis=1)
data.shape
```

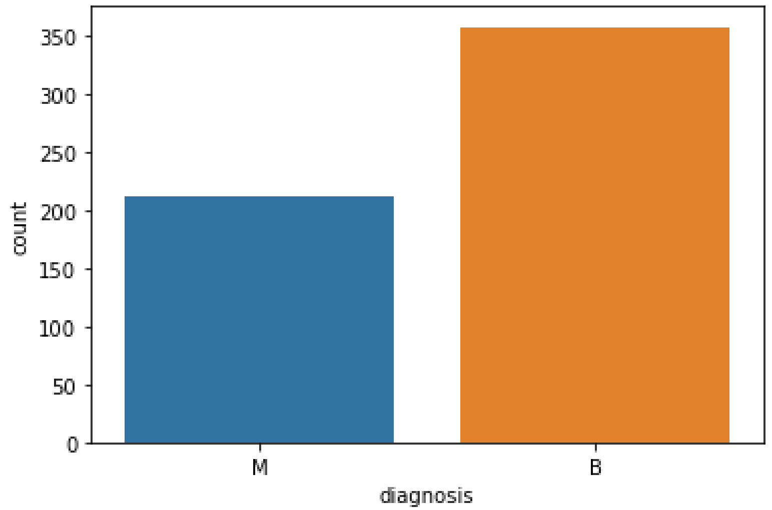
Out[5]:
```
(569, 32)
```

In [6]:
```
data["diagnosis"].value_counts()
```

Out[6]:
```
B    357
M    212
Name: diagnosis, dtype: int64
```

In [7]:
```
sns.countplot(data["diagnosis"],label="count")
```

Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x188dc2bad30>



In [8]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 32 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   id                       569 non-null    int64  
 1   diagnosis                569 non-null    object 
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
```

```
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
 18  concavity_se             569 non-null    float64
 19  concave points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_se     569 non-null    float64
 22  radius_worst             569 non-null    float64
 23  texture_worst            569 non-null    float64
 24  perimeter_worst          569 non-null    float64
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_worst        569 non-null    float64
 28  concavity_worst          569 non-null    float64
 29  concave points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
dtypes: float64(30), int64(1), object(1)
memory usage: 142.4+ KB
```

In [9]:

```python
from sklearn.preprocessing import LabelEncoder
labelencoder_y=LabelEncoder()
labelencoder_y.fit_transform(data.iloc[:,1].values)
```

Out[9]:

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
1,
       1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1,
1,
       0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1,
1,
       0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1,
0,
       0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0,
```

```
1,
        1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
0,
        0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0,
0,
        0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1,
1,
        1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1,
1,
        0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0,
0,
        0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1,
        1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0,
0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0,
0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1,
1,
        1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
1,
        1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1,
1,
        0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0,
0,
        0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0,
1,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0,
0,
        0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
1,
        0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
0,
        0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0,
0,
        0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
0,
        0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0])
```

In [10]:
```
data.corr()
```

Out[10]:

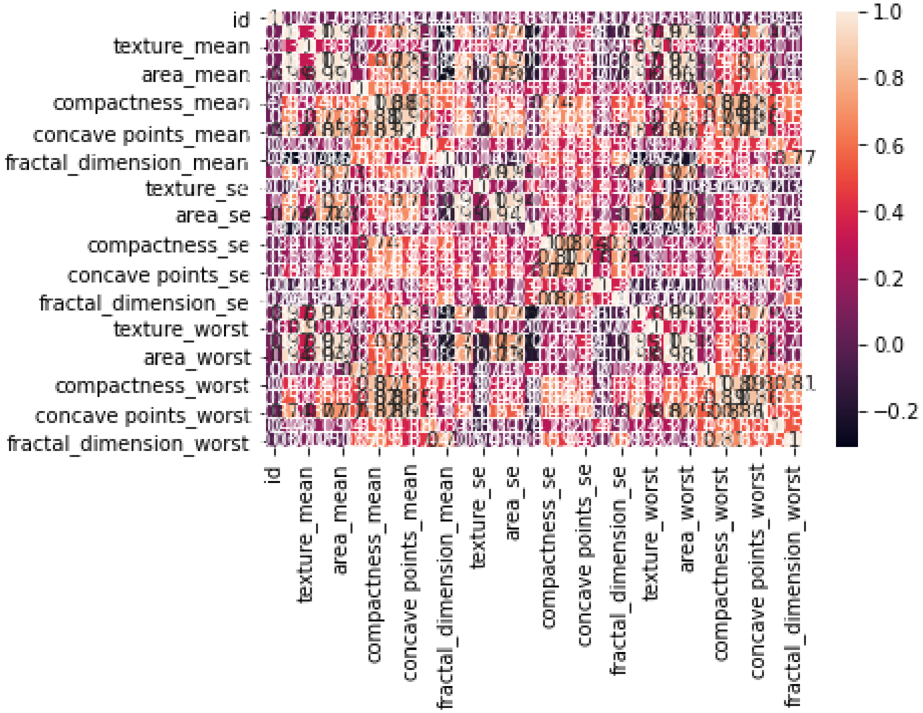
	id	radius_mean	texture_mean	perimeter_mean	area_mean	smo
id	1.000000	0.074626	0.099770	0.073159	0.096893	
radius_mean	0.074626	1.000000	0.323782	0.997855	0.987357	
texture_mean	0.099770	0.323782	1.000000	0.329533	0.321086	
perimeter_mean	0.073159	0.997855	0.329533	1.000000	0.986507	
area_mean	0.096893	0.987357	0.321086	0.986507	1.000000	
smoothness_mean	-0.012968	0.170581	-0.023389	0.207278	0.177028	
compactness_mean	0.000096	0.506124	0.236702	0.556936	0.498502	
concavity_mean	0.050080	0.676764	0.302418	0.716136	0.685983	
concave points_mean	0.044158	0.822529	0.293464	0.850977	0.823269	
symmetry_mean	-0.022114	0.147741	0.071401	0.183027	0.151293	
fractal_dimension_mean	-0.052511	-0.311631	-0.076437	-0.261477	-0.283110	
radius_se	0.143048	0.679090	0.275869	0.691765	0.732562	
texture_se	-0.007526	-0.097317	0.386358	-0.086761	-0.066280	
perimeter_se	0.137331	0.674172	0.281673	0.693135	0.726628	
area_se	0.177742	0.735864	0.259845	0.744983	0.800086	
smoothness_se	0.096781	-0.222600	0.006614	-0.202694	-0.166777	
compactness_se	0.033961	0.206000	0.191975	0.250744	0.212583	
concavity_se	0.055239	0.194204	0.143293	0.228082	0.207660	
concave points_se	0.078768	0.376169	0.163851	0.407217	0.372320	
symmetry_se	-0.017306	-0.104321	0.009127	-0.081629	-0.072497	
fractal_dimension_se	0.025725	-0.042641	0.054458	-0.005523	-0.019887	
radius_worst	0.082405	0.969539	0.352573	0.969476	0.962746	

	id	radius_mean	texture_mean	perimeter_mean	area_mean	smo
texture_worst	0.064720	0.297008	0.912045	0.303038	0.287489	
perimeter_worst	0.079986	0.965137	0.358040	0.970387	0.959120	
area_worst	0.107187	0.941082	0.343546	0.941550	0.959213	
smoothness_worst	0.010338	0.119616	0.077503	0.150549	0.123523	
compactness_worst	-0.002968	0.413463	0.277830	0.455774	0.390410	
concavity_worst	0.023203	0.526911	0.301025	0.563879	0.512606	
concave points_worst	0.035174	0.744214	0.295316	0.771241	0.722017	
symmetry_worst	-0.044224	0.163953	0.105008	0.189115	0.143570	
fractal_dimension_worst	-0.029866	0.007066	0.119205	0.051019	0.003738	

31 rows × 31 columns

In [11]:
```
sns.heatmap(data.corr(),annot=True)
plt.figure(figsize=(40,40))
```

Out[11]:
```
<Figure size 2880x2880 with 0 Axes>
```




```
<Figure size 2880x2880 with 0 Axes>
```

In [12]:
```
x=data.iloc[:,2:31].values
y=data.iloc[:,1].values
```

In [22]:
```
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,rando
m_state=0)
```

In [14]:
```
#Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
```

In [20]:
```
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 SVC linear
  from sklearn.svm import SVC
  svc_lin = SVC(kernel = 'linear', random_state = 0)
  svc_lin.fit(x_train, y_train)

  #Using SVC rbf
  from sklearn.svm import SVC
  svc_rbf = SVC(kernel = 'rbf', random_state = 0)
  svc_rbf.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]Support Vector Machine (Linear Classifier) Training Accuracy:', svc_lin.score(x_train, y_train))
  print('[3]Support Vector Machine (RBF Classifier) Training Accuracy:', svc_rbf.score(x_train, y_train))
  print('[4]Gaussian Naive Bayes Training Accuracy:', gauss.score(x_train, y_train))
  print('[5]Decision Tree Classifier Training Accuracy:', tree.score(x_train, y_train))
  print('[6]Random Forest Classifier Training Accuracy:', forest.score(x_train, y_train))
  return log, knn, svc_lin, svc_rbf, gauss, tree, forest
```

In [23]:

```
model=models(x_train,y_train)
```

```
C:\Users\91852\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:762: 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(
```

```
[0]Logistic Regression Training Accuracy: 0.9577464788732394
[1]K Nearest Neighbor Training Accuracy: 0.9413145539906104
[2]Support Vector Machine (Linear Classifier) Training Accuracy: 0.9647887323943662
[3]Support Vector Machine (RBF Classifier) Training Accuracy: 0.903755868544601
[4]Gaussian Naive Bayes Training Accuracy: 0.9507042253521126
```

```
[5]Decision Tree Classifier Training Accuracy: 1.0
[6]Random Forest Classifier Training Accuracy: 0.9953051643192489
```

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
```

```
[[85  5]
 [ 1 52]]
Model[0] Testing Accuracy = "0.958041958041958!"

[[85  5]
 [ 4 49]]
Model[1] Testing Accuracy = "0.9370629370629371!"

[[86  4]
 [ 1 52]]
Model[2] Testing Accuracy = "0.965034965034965!"

[[89  1]
 [ 8 45]]
Model[3] Testing Accuracy = "0.9370629370629371!"

[[86  4]
 [ 5 48]]
Model[4] Testing Accuracy = "0.9370629370629371!"

[[84  6]
 [ 1 52]]
Model[5] Testing Accuracy = "0.951048951048951!"
```

```
[[87  3]
 [ 2 51]]
Model[6] 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

           B       0.99      0.94      0.97        90
           M       0.91      0.98      0.95        53

    accuracy                           0.96       143
   macro avg       0.95      0.96      0.96       143
weighted avg       0.96      0.96      0.96       143

0.958041958041958

Model  1
              precision    recall  f1-score   support

           B       0.96      0.94      0.95        90
           M       0.91      0.92      0.92        53

    accuracy                           0.94       143
   macro avg       0.93      0.93      0.93       143
```

```
weighted avg       0.94      0.94      0.94       143

0.9370629370629371

Model  2
              precision    recall  f1-score   support

           B       0.99      0.96      0.97        90
           M       0.93      0.98      0.95        53

    accuracy                           0.97       143
   macro avg       0.96      0.97      0.96       143
weighted avg       0.97      0.97      0.97       143

0.965034965034965

Model  3
              precision    recall  f1-score   support

           B       0.92      0.99      0.95        90
           M       0.98      0.85      0.91        53

    accuracy                           0.94       143
   macro avg       0.95      0.92      0.93       143
weighted avg       0.94      0.94      0.94       143

0.9370629370629371

Model  4
              precision    recall  f1-score   support

           B       0.95      0.96      0.95        90
           M       0.92      0.91      0.91        53

    accuracy                           0.94       143
   macro avg       0.93      0.93      0.93       143
weighted avg       0.94      0.94      0.94       143

0.9370629370629371
```

```
Model  5
              precision    recall  f1-score   support

           B       0.99      0.93      0.96        90
           M       0.90      0.98      0.94        53

    accuracy                           0.95       143
   macro avg       0.94      0.96      0.95       143
weighted avg       0.95      0.95      0.95       143

0.951048951048951

Model  6
              precision    recall  f1-score   support

           B       0.98      0.97      0.97        90
           M       0.94      0.96      0.95        53

    accuracy                           0.97       143
   macro avg       0.96      0.96      0.96       143
weighted avg       0.97      0.97      0.97       143

0.965034965034965
```

In [26]:
```
#conclusion for selectiong best model
```

In [ ]: