Tumor detection using random forest classifier algorithm

Skills -

- EDA
- correlation
- scaling data , dividing data into train and test sets
- using random forest algorithm and calculating accuracy of prediction of the model

step 1: data cleaning and sorting

In [2]:
```
#case study - tumor detection

#preprocessing the data:

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

In [3]:
```
#importing the data
df = pd.read_csv(r"C:\Users\NISHAKA\Downloads\data+science\data science\Tumor_Detection.csv")
```

In [5]:
```
df.columns
```

Out[5]:
```
Index(['id', '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')
```

In [6]:
```
df.head()
```

printed the info about all the columns to check if any of them are unnecessary, then dropped id to prevent miscalculations.

In [7]:
```
df.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 [8]:
```
#drop unnamed columns
#dropping id
df.drop(['id'] , axis = 1, inplace = True)
```

Storing columns in list l and sorting based on types i.e. mean , se and worst

```
In [10]: type(df.columns)

Out[10]: pandas.core.indexes.base.Index

In [11]: #sorting the data based on the columns
         l = list(df.columns)
         print(l)

['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', 'con
cave points_worst', 'symmetry_worst', 'fractal_dimension_worst']

In [12]: #creating sub datas based on the categories they belong to i.e. mean , se and worst

         features_mean = l[1:11]
         features_se = l[11:21]
         features_worst = l[21:]

In [13]: print(features_mean)

['radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean', 'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean']

In [14]: print(features_se)

['radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se', 'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se', 'fractal_dimension_se']

In [15]: print(features_worst)

['radius_worst', 'texture_worst', 'perimeter_worst', 'area_worst', 'smoothness_worst', 'compactness_worst', 'concavity_worst', 'concave points_worst', 'symmetry_worst', 'fractal_dimension_worst']
```

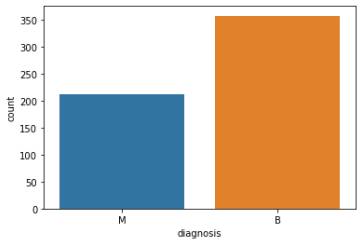
checking the number of different types of diagnostics and their count - since that will be the parameter to be tested.

```
In [15]: df['diagnosis'].unique()  #M = malignant , B- Begin

Out[15]: array(['M', 'B'], dtype=object)

In [16]: sns.countplot(x=df['diagnosis'],label="count")

Out[16]: <AxesSubplot:xlabel='diagnosis', ylabel='count'>
```



Malignant tumor - is cancerous and needs to be treated as soon as possible since it is life threatening

Benign tumor - is not cancerous and may or may not have to be removed by surgery.

Thus the aim was to check how some features of a tumor can be used to predict if it is of M type or B type.

Basic statistical summary of the data-

In [18]:
```
df.describe() #provides a statistical summary of the data
```

Out[18]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	fractal_dimension_mean	...	radius_worst	texture_worst	perimeter_worst	area_worst
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	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	0.181162	0.062798	...	16.269190	25.677223	107.261213	880.583128
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.038803	0.027414	0.007060	...	4.833242	6.146258	33.602542	569.356993
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.000000	0.106000	0.049960	...	7.930000	12.020000	50.410000	185.200000
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.020310	0.161900	0.057700	...	13.010000	21.080000	84.110000	515.300000
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.033500	0.179200	0.061540	...	14.970000	25.410000	97.660000	686.500000
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.074000	0.195700	0.066120	...	18.790000	29.720000	125.400000	1084.000000
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.201200	0.304000	0.097440	...	36.040000	49.540000	251.200000	4254.000000

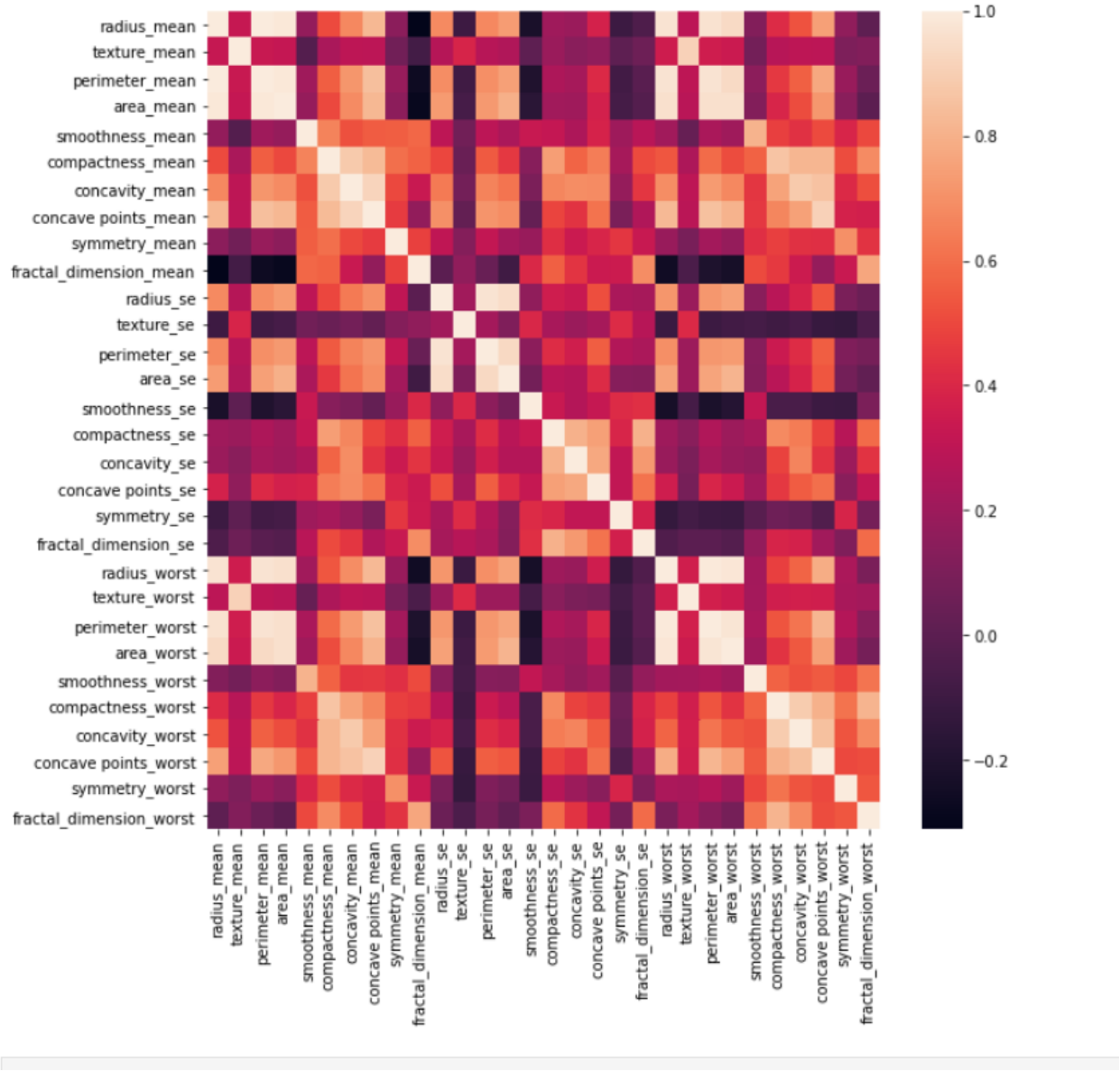
8 rows × 30 columns

checking the correlation between the parameters

In [19]:
```
correlation = df.corr()
print(correlation)
```

```
                         radius_mean  texture_mean  perimeter_mean  area_mean  \
radius_mean                 1.000000      0.323782        0.997855   0.987357
texture_mean                0.323782      1.000000        0.329533   0.321086
perimeter_mean              0.997855      0.329533        1.000000   0.986507
area_mean                   0.987357      0.321086        0.986507   1.000000
smoothness_mean             0.170581     -0.023389        0.207278   0.177028
compactness_mean            0.506124      0.236702        0.556936   0.498502
concavity_mean              0.676764      0.302418        0.716136   0.685983
concave points_mean         0.822529      0.293464        0.850977   0.823269
symmetry_mean               0.147741      0.071401        0.183027   0.151293
fractal_dimension_mean     -0.311631     -0.076437       -0.261477  -0.283110
radius_se                   0.679090      0.275869        0.691765   0.732562
texture_se                 -0.097317      0.386358       -0.086761  -0.066280
perimeter_se                0.674172      0.281673        0.693135   0.726628
area_se                     0.735864      0.259845        0.744983   0.800086
smoothness_se              -0.222600      0.006614       -0.202694  -0.166777
compactness_se              0.206000      0.191975        0.250744   0.212583
concavity_se                0.194204      0.143293        0.228082   0.207660
concave points_se           0.376169      0.163851        0.407217   0.372320
symmetry_se                -0.104321      0.009127       -0.081629  -0.072497
fractal_dimension_se       -0.042641      0.054458       -0.005523  -0.019887
radius_worst                0.969539      0.352573        0.969476   0.962746
texture_worst               0.297008      0.912045        0.303038   0.287489
perimeter_worst             0.965137      0.358040        0.970387   0.959120
area_worst                  0.941082      0.343546        0.941550   0.959213
smoothness_worst            0.119616      0.077503        0.150549   0.123523
compactness_worst           0.413463      0.277830        0.455774   0.390410
concavity_worst             0.526911      0.301025        0.563879   0.512606
concave points_worst        0.744214      0.295316        0.771241   0.722017
symmetry_worst              0.163953      0.105008        0.189115   0.143570
fractal_dimension_worst     0.007066      0.119205        0.051019   0.003738
```

Since there are too many parameters plotting a heat map for better visualization.



Mapping the diagnostic values to 1 and 0 in order to make them numeric so that they can be used as the parameter to be tested in the ML model.

```
In [21]:
df["diagnosis"] = df["diagnosis"].map({'M' : 1 , 'B' : 0 })
df.head()
```

Out[21]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	...	radius_worst	texture_worst	perimeter_worst	area_worst	smoothness_worst	compactness_worst
0	1	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	...	25.38	17.33	184.60	2019.0	0.1622	0.6656
1	1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	...	24.99	23.41	158.80	1956.0	0.1238	0.1866
2	1	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	...	23.57	25.53	152.50	1709.0	0.1444	0.4245
3	1	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	...	14.91	26.50	98.87	567.7	0.2098	0.8663
4	1	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	...	22.54	16.67	152.20	1575.0	0.1374	0.2050

5 rows × 31 columns

```
In [22]:
df["diagnosis"].unique()
```

```
Out[22]: array(['M', 'B'], dtype=object)
```

```
In [23]:
X= df.drop('diagnosis' , axis = 1)
X.head()
```

dropped it from X in order to assign y variable as the diagnostics.

```
In [24]:
y = df["diagnosis"]
y.head()
```

```
Out[24]: 0    M
         1    M
         2    M
         3    M
         4    M
         Name: diagnosis, dtype: object
```

Scaling the data - to fit within a certain scale to avoid errors originating from different units while training

Splitting - to have a training set and a test set for the algorithm to calculate the accuracy. test_size = 0.3 implies 70 percent data is used to train.

In [25]:
```
#divide the dataset into train and test set
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
X_train , X_test , y_train , y_test = train_test_split(X,y,test_size = 0.3)
```

In [26]:
```
df.shape
```

Out[26]: (569, 31)

In [27]:
```
X_train.shape
```

Out[27]: (398, 30)

In [28]:
```
X_test.shape
```

Out[28]: (171, 30)

In [29]:
```
y_test.shape
```

Out[29]: (171,)

In [30]:
```
y_train.shape
```

Out[30]: (398,)

In [31]:
```
ss = StandardScaler()
X_train = ss.fit_transform(X_train)
X_test = ss.transform(X_test)

X_train
```

```
array([[-0.47569249, -1.52380144, -0.54229856, ..., -1.34661963,
        -1.00175526, -0.743829  ],
       [-0.36224007,  2.19865163, -0.39850193, ..., -0.77255582,
        -0.85472895, -0.62961369],
       [ 0.83171155, -0.5405556 ,  0.7965382 , ...,  1.04568929,
         0.36942488, -0.16474672],
       ...,
       [ 3.56537454,  1.58556893,  3.69206165, ...,  2.20194684,
        -0.42483681, -0.53087616],
       [-0.68098734, -0.07091112, -0.71861595, ..., -0.88577271,
         0.92716599, -0.78118914],
       [-0.47839374, -0.85982131, -0.39575921, ...,  0.87104622,
         1.72302579,  2.14464409]])
```

Finally using the random forest algorithm from sklearn.ensemble library to predict the outcome and used accuracy_score from sklearn.metrics library to check the accuracy of the algorithm.

In [32]:

```
#applying random forest classifier algorithm
#checking accuracy score

from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier

rfc = RandomForestClassifier()
rfc.fit(X_train,y_train)

y_pred = rfc.predict(X_test)
print(accuracy_score(y_test,y_pred))
```

```
0.9473684210526315
```

model accuracy - 94.736