DATA COLLECTION

In [1]:

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

In [2]:

```
# To Import Dataset
sd=pd.read_csv(r"c:\Users\user\Downloads\8_dataset.csv")
sd
```

Out[2]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_
0	842302	M	17.99	10.38	122.80	1001.0	0.
1	842517	M	20.57	17.77	132.90	1326.0	0.(
2	84300903	M	19.69	21.25	130.00	1203.0	0.'
3	84348301	M	11.42	20.38	77.58	386.1	0.'
4	84358402	M	20.29	14.34	135.10	1297.0	0.'
...	...	...	...	...	...	...	
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.(

569 rows × 33 columns

In [3]:

```
# to display top 10 rows
sd.head(10)
```

Out[3]:

	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	843786	M	12.45	15.70	82.57	477.1	0.127
6	844359	M	18.25	19.98	119.60	1040.0	0.094
7	84458202	M	13.71	20.83	90.20	577.9	0.118
8	844981	M	13.00	21.82	87.50	519.8	0.127
9	84501001	M	12.46	24.04	83.97	475.9	0.118

10 rows × 33 columns

DATA CLEANING AND PRE_PROCESSING

In [4]:

```
sd.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 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
 32  Unnamed: 32              0 non-null      float64
dtypes: float64(31), int64(1), object(1)
memory usage: 146.8+ KB
```

In [5]:
```
# to display summary of statistics
sd.describe()
```

Out[5]:

	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mea
count	5.690000e+02	569.000000	569.000000	569.000000	569.000000	569.00000
mean	3.037183e+07	14.127292	19.289649	91.969033	654.889104	0.09636
std	1.250206e+08	3.524049	4.301036	24.298981	351.914129	0.01406
min	8.670000e+03	6.981000	9.710000	43.790000	143.500000	0.05263
25%	8.692180e+05	11.700000	16.170000	75.170000	420.300000	0.08637
50%	9.060240e+05	13.370000	18.840000	86.240000	551.100000	0.09587
75%	8.813129e+06	15.780000	21.800000	104.100000	782.700000	0.10530
max	9.113205e+08	28.110000	39.280000	188.500000	2501.000000	0.16340

8 rows × 32 columns

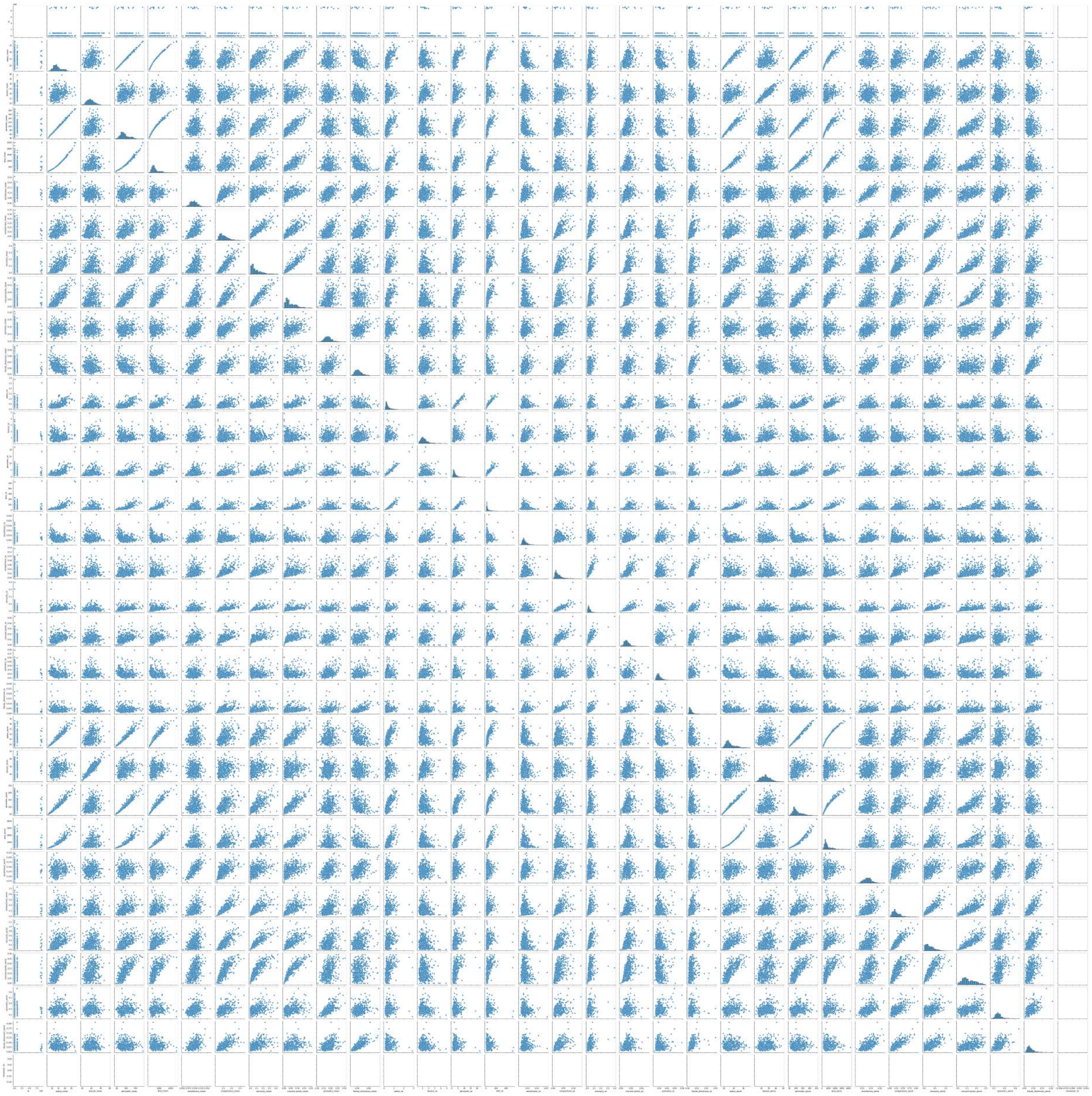
In [6]:
```
#to display colums heading
sd.columns
```

Out[6]:
```
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', 'Unnamed: 32'],
      dtype='object')
```

EDA and visualization

In [7]:
```
sns.pairplot(sd)
```

Out[7]:
```
<seaborn.axisgrid.PairGrid at 0x2655b112b20>
```



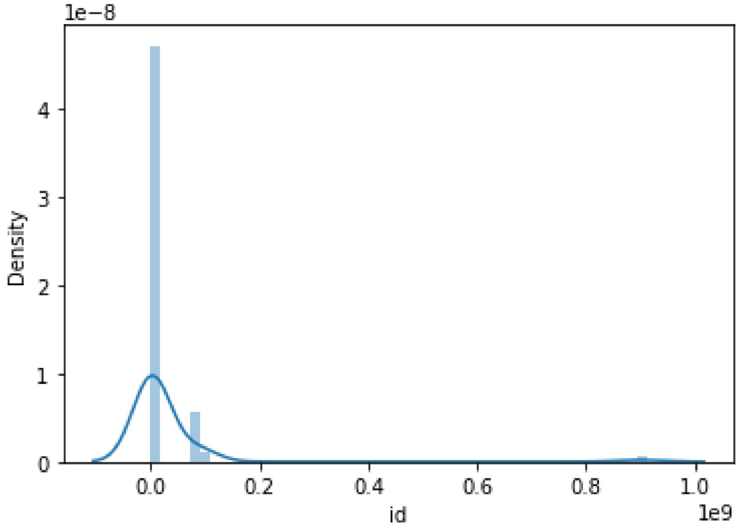
In [8]:

```
sns.distplot(sd['id'])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: Fut
ureWarning: `distplot` is a deprecated function and will be removed in a futu
re version. Please adapt your code to use either `displot` (a figure-level fu
nction with similar flexibility) or `histplot` (an axes-level function for hi
stograms).
  warnings.warn(msg, FutureWarning)
```

Out[8]:

```
<AxesSubplot:xlabel='id', ylabel='Density'>
```



In [9]:

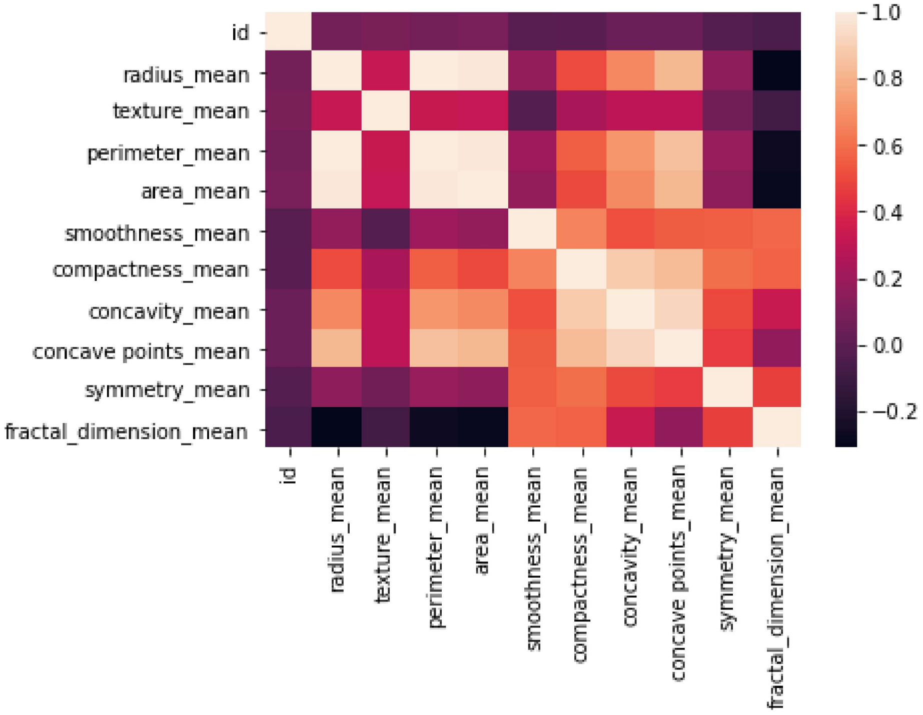
```
sd1=sd[['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean']]
```

In [10]:

```
sns.heatmap(sd1.corr())
```

Out[10]:

```
<AxesSubplot:>
```



TO TRAIN THE MODEL _MODEL BUILDING

we are goint train Liner Regression model; we need to split out the data into two varibles x and y where x is independent on x (output) and y is dependent on x(output) adress coloumn as it is not required our model

In [11]:

```
x= sd1[['id', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean']]
y=sd1['fractal_dimension_mean']
```

In [12]:

```
# To split my dataset  into training data and test data
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 [13]:

```
from sklearn.linear_model import LinearRegression

lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[13]:

```
LinearRegression()
```

In [14]:

```
print(lr.intercept_)
```

```
0.0792529984617628
```

In [15]:

```
coeff= pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[15]:

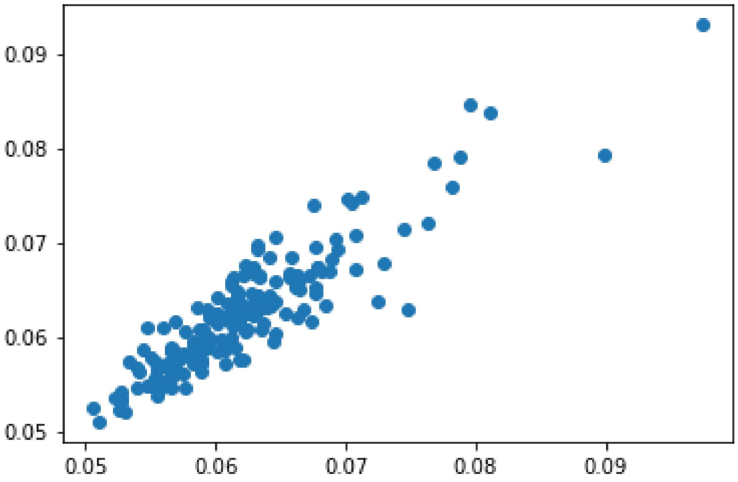
	Co-efficient
id	-9.842605e-13
radius_mean	2.908223e-03
texture_mean	-6.616331e-05
perimeter_mean	-9.844341e-04
area_mean	2.083285e-05
smoothness_mean	6.449293e-02
compactness_mean	1.413589e-01
concavity_mean	1.741398e-02
concave points_mean	-1.877573e-02
symmetry_mean	-4.636405e-03

In [16]:

```
prediction = lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[16]:

```
<matplotlib.collections.PathCollection at 0x2650c9a4910>
```



In [17]:

```
print(lr.score(x_test,y_test))
```

```
0.8127415733381201
```

In [18]:

```
lr.score(x_train,y_train)
```

Out[18]:

```
0.8563847506209721
```

In [19]:

```
from sklearn.linear_model import Ridge,Lasso
```

In [20]:

```
dr=Ridge(alpha=10)
dr.fit(x_train,y_train)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_ridge.py:14
7: LinAlgWarning: Ill-conditioned matrix (rcond=1.21314e-18): result may not
be accurate.
  return linalg.solve(A, Xy, sym_pos=True,
```

Out[20]:

```
Ridge(alpha=10)
```

In [21]:

```
dr.score(x_test,y_test)
```

Out[21]:

```
0.6057001770469257
```

In [22]:

```
dr.score(x_train,y_train)
```

Out[22]:

```
0.6464164180383427
```

In [23]:

```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[23]:

```
Lasso(alpha=10)
```

In [24]:

```
la.score(x_test,y_test)
```

Out[24]:

```
-0.01783058120755099
```

In [25]:

```
la.score(x_train,y_train)
```

Out[25]:

```
0.003404306103443777
```

ElasticNet

In [26]:

```
from sklearn.linear_model import ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[26]:

```
ElasticNet()
```

In [27]:

```
print(en.coef_)
```

```
[-2.37245640e-12 -0.00000000e+00 -0.00000000e+00 -0.00000000e+00
 -2.29497125e-06  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00]
```

In [28]:

```
print(en.intercept_)
```

```
0.06463532416367211
```

```
In [29]: prediction=en.predict(x_test)
```

```
In [30]: print(en.score(x_test,y_test))
```

```
0.018780250763528294
```

Evaluation metrics

```
In [31]: from sklearn import metrics
```

```
In [32]: print("mean Absolute Error:",metrics.mean_absolute_error(y_test,prediction))
```

```
mean Absolute Error: 0.004952724510297355
```

```
In [33]: print("mean squared Error:",metrics.mean_squared_error(y_test,prediction))
```

```
mean squared Error: 4.4778564914142446e-05
```

```
In [34]: print("Root mean Absolytre Error:",np.sqrt(metrics.mean_squared_error(y_test,pr
```

```
Root mean Absolytre Error: 0.006691678781452562
```

```
In [ ]:
```