kaviyadevi 20106064

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

In [2]:
```
#to import dataset
data=pd.read_csv(r"C:\Users\user\Downloads\8_BreastCancerPrediction - 8_BreastCar
data
```

Out[2]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_m
0	842302	M	17.99	10.38	122.80	1001.0	0.11
1	842517	M	20.57	17.77	132.90	1326.0	0.08
2	84300903	M	19.69	21.25	130.00	1203.0	0.10
3	84348301	M	11.42	20.38	77.58	386.1	0.14
4	84358402	M	20.29	14.34	135.10	1297.0	0.10
...	...	...	...	...	...	...	
564	926424	M	21.56	22.39	142.00	1479.0	0.11
565	926682	M	20.13	28.25	131.20	1261.0	0.09
566	926954	M	16.60	28.08	108.30	858.1	0.08
567	927241	M	20.60	29.33	140.10	1265.0	0.11
568	92751	B	7.76	24.54	47.92	181.0	0.05

569 rows × 32 columns

In [3]:
```
data.head()
```

Out[3]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mea
0	842302	M	17.99	10.38	122.80	1001.0	0.1184
1	842517	M	20.57	17.77	132.90	1326.0	0.0847
2	84300903	M	19.69	21.25	130.00	1203.0	0.1096
3	84348301	M	11.42	20.38	77.58	386.1	0.1425
4	84358402	M	20.29	14.34	135.10	1297.0	0.1003

5 rows × 32 columns

DATA CLEANING AND PREPROCESSING

In [4]:
```
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 [5]:
```
data.describe()
```

Out[5]:

	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean
count	5.690000e+02	569.000000	569.000000	569.000000	569.000000	569.000000
mean	3.037183e+07	14.127292	19.289649	91.969033	654.889104	0.096360
std	1.250206e+08	3.524049	4.301036	24.298981	351.914129	0.014064
min	8.670000e+03	6.981000	9.710000	43.790000	143.500000	0.052630
25%	8.692180e+05	11.700000	16.170000	75.170000	420.300000	0.086370
50%	9.060240e+05	13.370000	18.840000	86.240000	551.100000	0.095870
75%	8.813129e+06	15.780000	21.800000	104.100000	782.700000	0.105300
max	9.113205e+08	28.110000	39.280000	188.500000	2501.000000	0.163400

8 rows × 31 columns

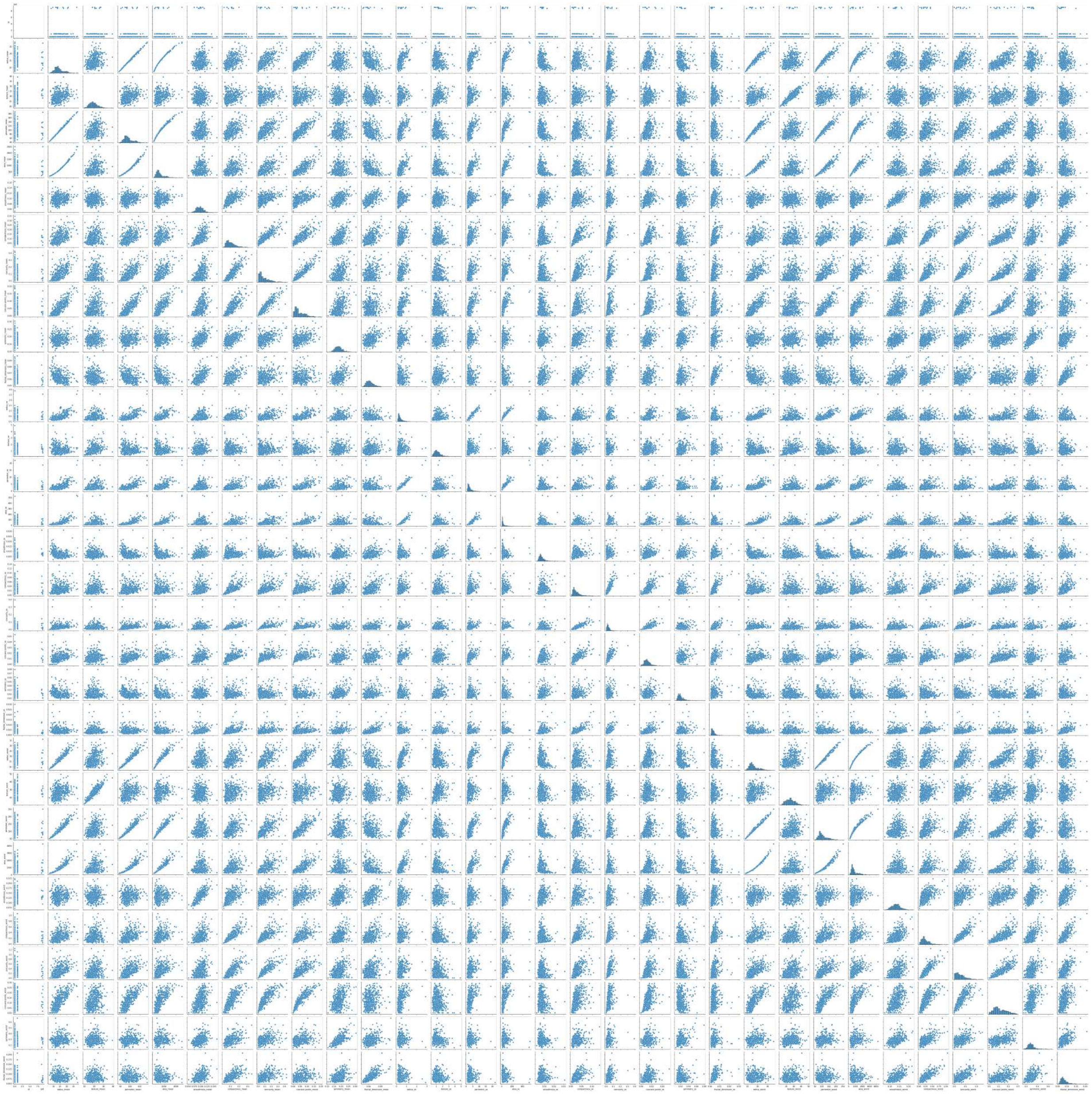
In [6]:
```
data.columns
```

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'],
      dtype='object')
```

EDA and DATA VISUALIZATION

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

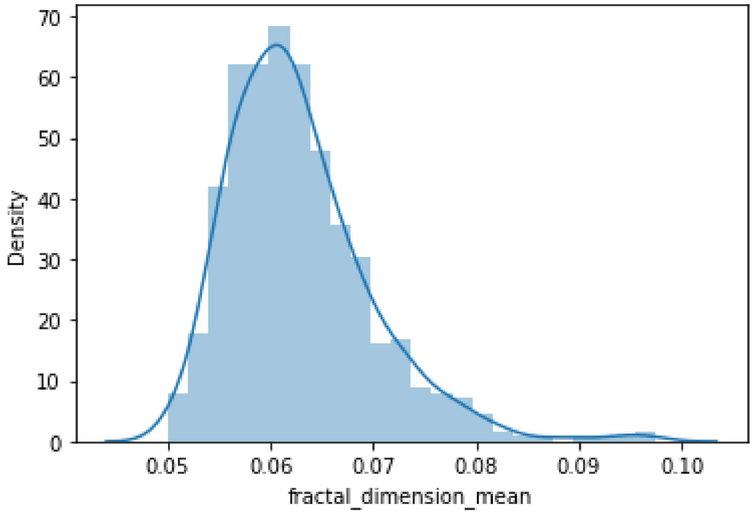
Out[7]:
```
<seaborn.axisgrid.PairGrid at 0x136bedc7fd0>
```



```
In [8]: sns.distplot(data["fractal_dimension_mean"])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: Futur
eWarning: `distplot` is a deprecated function and will be removed in a future v
ersion. Please adapt your code to use either `displot` (a figure-level function
with similar flexibility) or `histplot` (an axes-level function for histogram
s).
  warnings.warn(msg, FutureWarning)
```

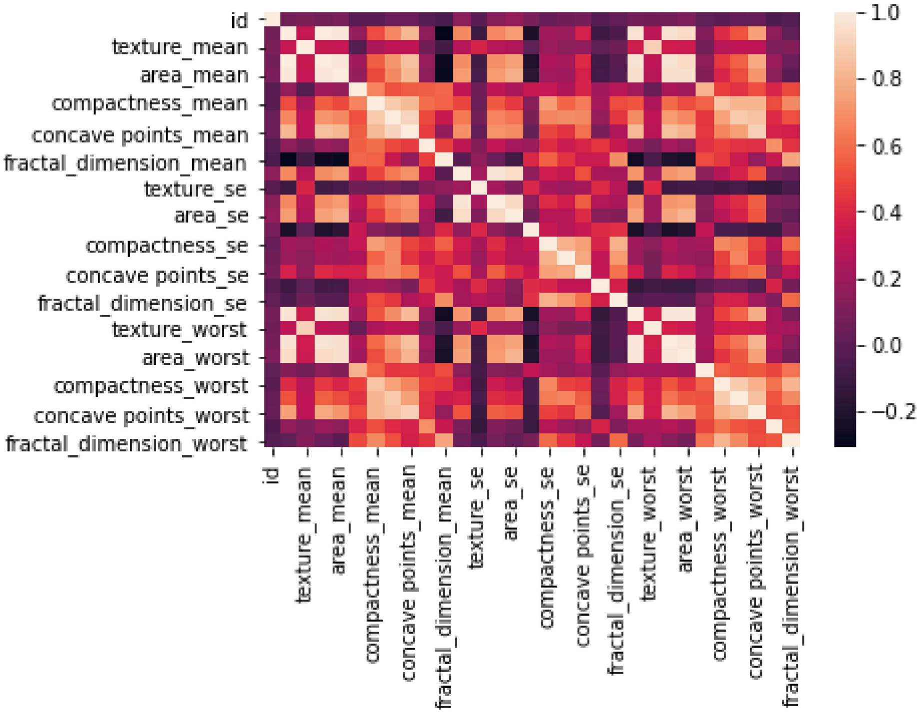
```
Out[8]: <AxesSubplot:xlabel='fractal_dimension_mean', ylabel='Density'>
```



```
In [9]: df=data[['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']]
```

In [10]:
```
sns.heatmap(df.corr())
```

Out[10]:
```
<AxesSubplot:>
```



TRAINNING MODEL

In [11]:
```
x=df[['radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean',
       'radius_se', 'texture_se', 'perimeter_se']]
y=df[["fractal_dimension_mean"]]
```

In [12]:

```
#to split my dataset into trainning and test

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]:

```
#to find intercept
print(lr.intercept_)
```

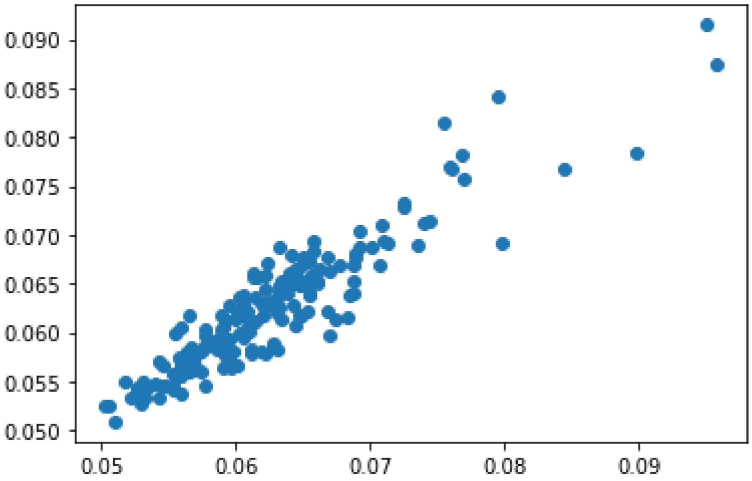
```
[0.07481545]
```

In [15]:

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

Out[15]:

```
<matplotlib.collections.PathCollection at 0x136ee403490>
```



In [16]:

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

```
0.8567728054030949
```

RIDGE AND LASSO REGRESSION

In [17]:

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

In [18]:

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

Out[18]:

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

```
In [19]: rr.score(x_test,y_test)

Out[19]: 0.5936419382876517
```

```
In [20]: la=Lasso(alpha=10)
         la.fit(x_train,y_train)

Out[20]: Lasso(alpha=10)
```

```
In [21]: la.score(x_test,y_test)

Out[21]: -0.0002766339454336464
```

```
In [22]: from sklearn.linear_model import ElasticNet
         en=ElasticNet()
         en.fit(x_train,y_train)

Out[22]: ElasticNet()
```

```
In [23]: print(en.coef_)

         [-0.00000000e+00 -0.00000000e+00 -0.00000000e+00 -1.06076681e-06
           0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
           0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
```

In [24]:
```
print(en.predict(x_test))
```

```
[0.06219157 0.06299447 0.06324916 0.06259551 0.06322359 0.06254449
 0.06303552 0.0630997  0.06298588 0.06322317 0.0630946  0.06305981
 0.06290961 0.06303976 0.06311115 0.06231462 0.0627978  0.06318827
 0.06283217 0.06318848 0.06281997 0.06267019 0.06276598 0.06317034
 0.06305589 0.0627245  0.06317459 0.06311837 0.06213005 0.06306904
 0.06308315 0.06277309 0.06327037 0.06300412 0.06288224 0.06290451
 0.06284522 0.06282941 0.06303658 0.0623963  0.06318806 0.06283323
 0.06292573 0.06331503 0.06287524 0.06282623 0.06313332 0.06301961
 0.06321818 0.06317957 0.06284384 0.06267518 0.06272164 0.06294461
 0.0630962  0.06307593 0.06267295 0.0630457  0.06303159 0.06314043
 0.06280311 0.06222446 0.0630229  0.06263773 0.06254173 0.06260686
 0.06295448 0.0629915  0.0622531  0.06211308 0.06314563 0.06210035
 0.06256995 0.06224992 0.06243343 0.06277415 0.06230189 0.06324141
 0.06304857 0.06275346 0.06188925 0.06288966 0.06314563 0.06290335
 0.06293199 0.06229659 0.06277256 0.06297739 0.06300539 0.06264569
 0.06257525 0.06223188 0.06295331 0.06280989 0.06300264 0.06309556
 0.06298428 0.06295469 0.06291332 0.06205898 0.06314679 0.06303435
 0.06326539 0.06319241 0.06292944 0.06292849 0.06238782 0.06238675
 0.06220324 0.06280692 0.06305695 0.06288086 0.06307986 0.06293294
 0.0629234  0.06252359 0.06310277 0.06226583 0.06292265 0.06281987
 0.06318456 0.06317268 0.06307657 0.06248753 0.06308771 0.06157633
 0.06296678 0.06261949 0.06175772 0.06306501 0.0630596  0.06306108
 0.06318806 0.06279706 0.06329456 0.06323166 0.06307519 0.06289019
 0.06289327 0.06291618 0.06304263 0.06266383 0.06270074 0.06236872
 0.06279791 0.06291512 0.06223825 0.06275229 0.06326793 0.06301833
 0.06289338 0.06229129 0.06264017 0.06290886 0.06291576 0.06324014
 0.06247904 0.06295755 0.06300614 0.06217778 0.06306395 0.06280056
 0.06292859 0.062426   0.06296211 0.06287322 0.06233053 0.06258904
 0.06310861 0.06303297 0.06298588]
```

In [25]:
```
print(en.score(x_test,y_test))
```

```
0.029748357908262357
```

Evaluation metrics

In [35]:
```
from sklearn import metrics
```

In [36]:
```
print("Mean Absolute error",metrics.mean_absolute_error(y_test,prediction))
```

```
Mean Absolute error 0.002056058444403791
```

In [37]:
```
print("Mean Squared error",metrics.mean_squared_error(y_test,prediction))
```

```
Mean Squared error 7.982044185108945e-06
```

In [38]:
```
print("Root Mean Absolute error",np.sqrt(metrics.mean_squared_error(y_test,predic
```

```
Root Mean Absolute error 0.0028252511720392127
```