Problem Statement

Linear Regression

Import Libraries

In [1]:

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

In [2]:

```
a=pd.read_csv("cancer.csv")
a
```

Out[2]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	cor
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	
...	...	...	...	...	...	...	...	
564	926424	M	21.56	22.39	142.00	1479.0	0.11100	
565	926682	M	20.13	28.25	131.20	1261.0	0.09780	
566	926954	M	16.60	28.08	108.30	858.1	0.08455	

To display top 10 rows

In [3]:

```
c=a.head(10)
c
```

Out[3]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness
0	842302	M	17.99	10.38	122.80	1001.0	
1	842517	M	20.57	17.77	132.90	1326.0	
2	84300903	M	19.69	21.25	130.00	1203.0	
3	84348301	M	11.42	20.38	77.58	386.1	
4	84358402	M	20.29	14.34	135.10	1297.0	
5	843786	M	12.45	15.70	82.57	477.1	
6	844359	M	18.25	19.98	119.60	1040.0	
7	84458202	M	13.71	20.83	90.20	577.9	
8	844981	M	13.00	21.82	87.50	519.8	
9	84501001	M	12.46	24.04	83.97	475.9	

10 rows × 32 columns

To find Missing values

In [4]:

```
c.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 32 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   id                       10 non-null     int64  
 1   diagnosis                10 non-null     object 
 2   radius_mean              10 non-null     float64
 3   texture_mean             10 non-null     float64
 4   perimeter_mean           10 non-null     float64
 5   area_mean                10 non-null     float64
 6   smoothness_mean          10 non-null     float64
 7   compactness_mean         10 non-null     float64
 8   concavity_mean           10 non-null     float64
 9   concave points_mean      10 non-null     float64
 10  symmetry_mean            10 non-null     float64
 11  fractal_dimension_mean   10 non-null     float64
 12  radius_se                10 non-null     float64
 13  texture_se               10 non-null     float64
 14  perimeter_se             10 non-null     float64
 15  area_se                  10 non-null     float64
 16  smoothness_se            10 non-null     float64
 17  compactness_se           10 non-null     float64
 18  concavity_se             10 non-null     float64
 19  concave points_se        10 non-null     float64
 20  symmetry_se              10 non-null     float64
 21  fractal_dimension_se     10 non-null     float64
 22  radius_worst             10 non-null     float64
 23  texture_worst            10 non-null     float64
 24  perimeter_worst          10 non-null     float64
 25  area_worst               10 non-null     float64
 26  smoothness_worst         10 non-null     float64
 27  compactness_worst        10 non-null     float64
 28  concavity_worst          10 non-null     float64
 29  concave points_worst     10 non-null     float64
 30  symmetry_worst           10 non-null     float64
 31  fractal_dimension_worst  10 non-null     float64
dtypes: float64(30), int64(1), object(1)
memory usage: 2.6+ KB
```

To display summary of statistics

In [5]:

```
a.describe()
```

Out[5]:

	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_
count	5.690000e+02	569.000000	569.000000	569.000000	569.000000	569.0(
mean	3.037183e+07	14.127292	19.289649	91.969033	654.889104	0.09
std	1.250206e+08	3.524049	4.301036	24.298981	351.914129	0.0'
min	8.670000e+03	6.981000	9.710000	43.790000	143.500000	0.05
25%	8.692180e+05	11.700000	16.170000	75.170000	420.300000	0.08
50%	9.060240e+05	13.370000	18.840000	86.240000	551.100000	0.09
75%	8.813129e+06	15.780000	21.800000	104.100000	782.700000	0.1(
max	9.113205e+08	28.110000	39.280000	188.500000	2501.000000	0.16

8 rows × 31 columns

To display column heading

In [6]:

```
a.columns
```

Out[6]:

```
Index(['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mea
n',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_s
e',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_s
e',
       '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')
```

Pairplot

In [7]:

```
s=a.dropna(axis=1)
s
```

Out[7]:

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

569 rows × 32 columns

In [8]:

```
s.columns
```

Out[8]:

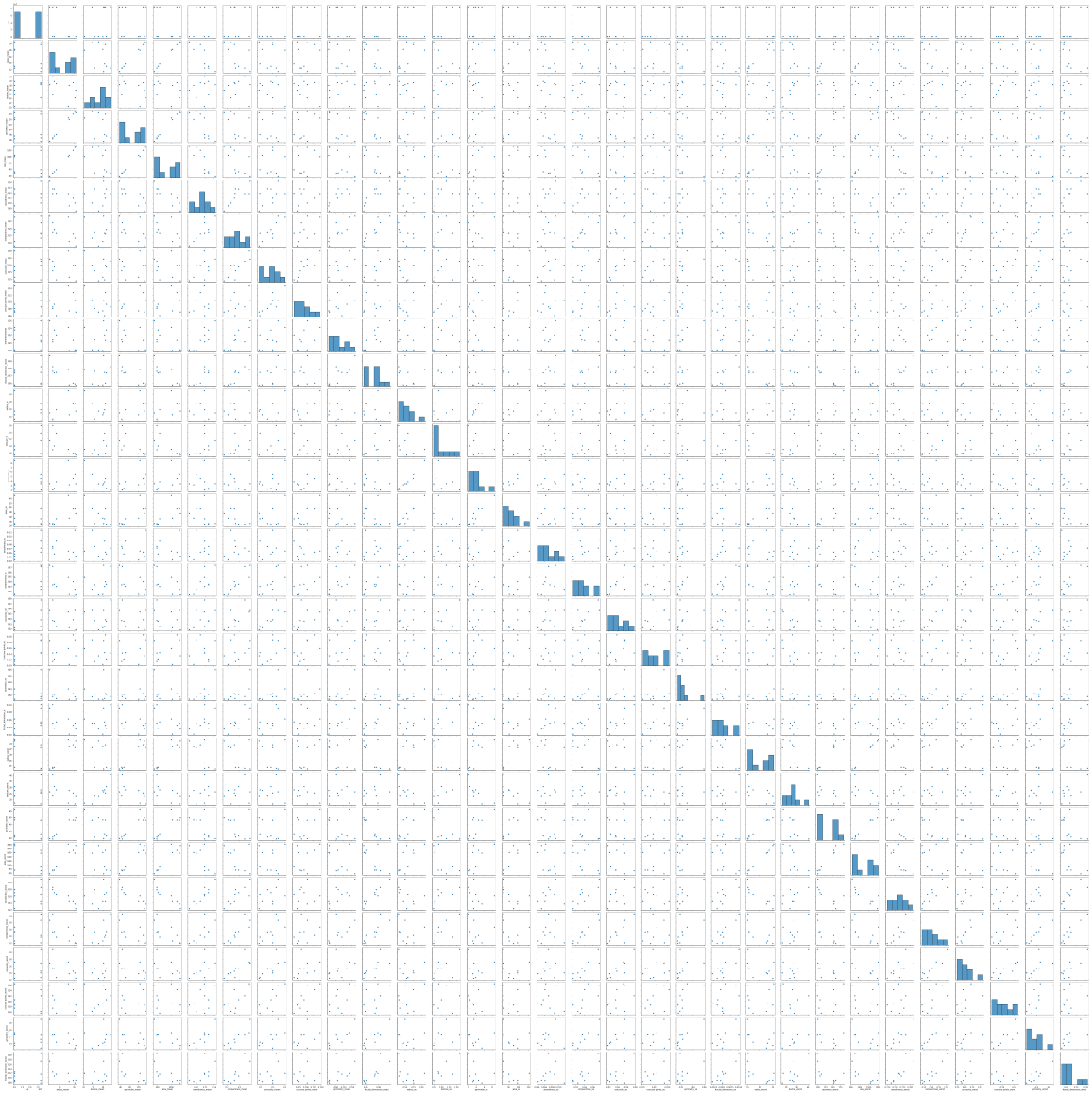
```
Index(['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mea
n',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_s
e',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_s
e',
       '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 [9]:

```
sns.pairplot(c)
```

Out[9]:

```
<seaborn.axisgrid.PairGrid at 0x20fed37f520>
```



Distribution Plot

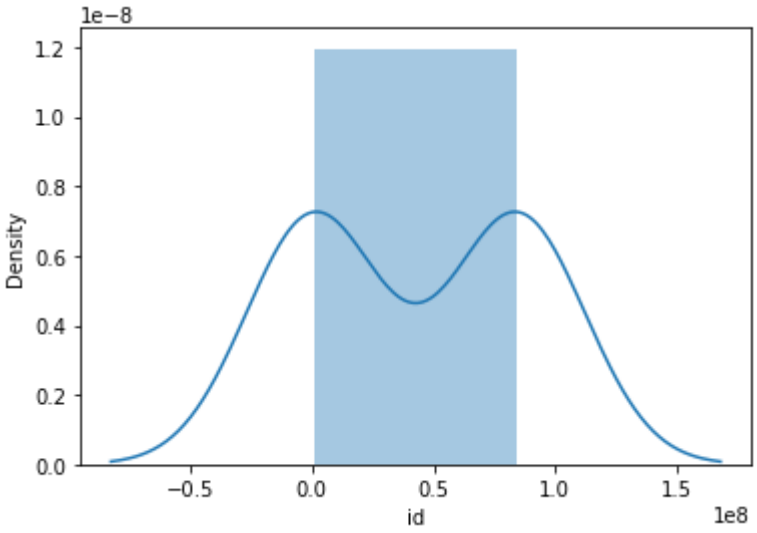
In [10]:

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

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

Out[10]:

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



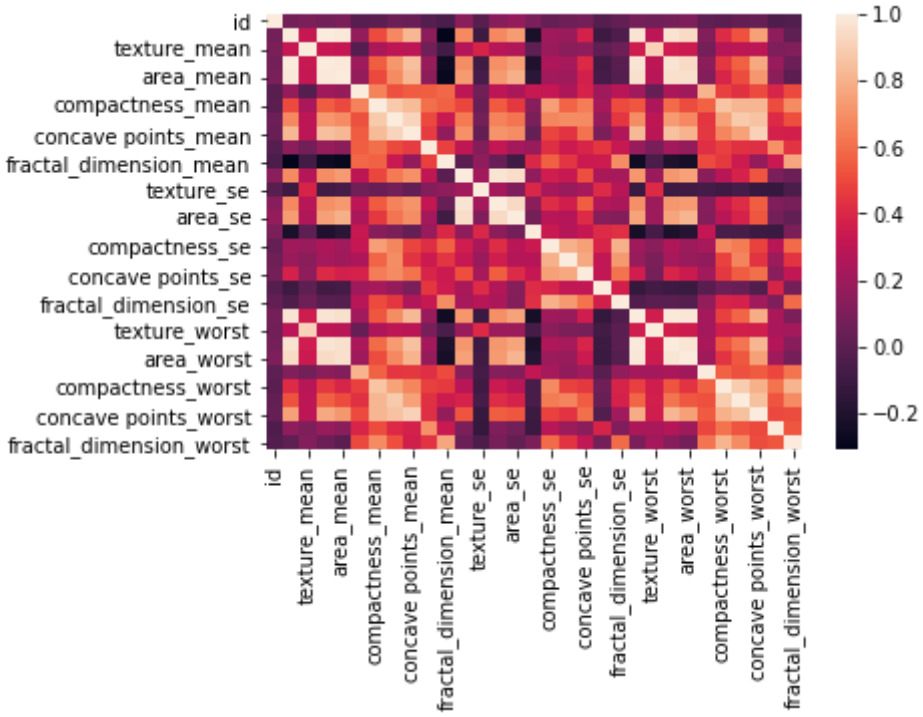
Correlation

In [11]:

```
b=a[['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']]
sns.heatmap(b.corr())
```

Out[11]:

<AxesSubplot:>



Train the model - Model Building

In [12]:

```
g=c[['id']]
h=c['id']
```

To split dataset into training end test

In [13]:

```
from sklearn.model_selection import train_test_split
g_train,g_test,h_train,h_test=train_test_split(g,h,test_size=0.6)
```

To run the model

In [14]:

```
from sklearn.linear_model import LinearRegression
```

In [15]:

```
lr=LinearRegression()
lr.fit(g_train,h_train)
```

Out[15]:

```
LinearRegression()
```

In [16]:

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

```
0.0
```

Coeffecient

In [17]:

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

Out[17]:

	Co-effecient
id	1.0

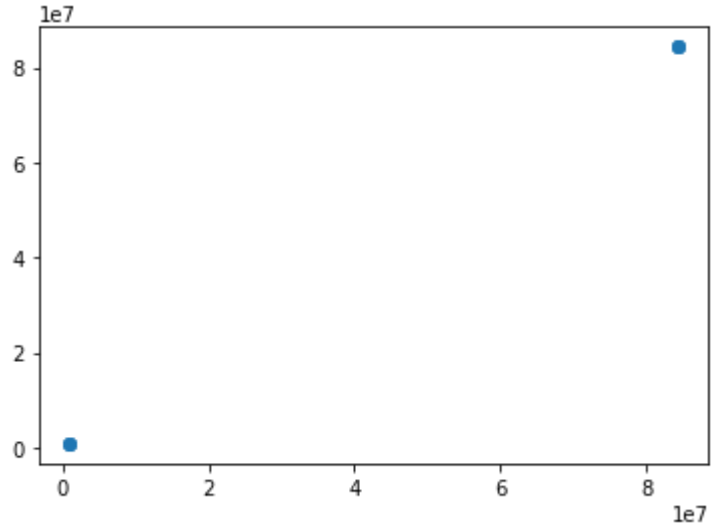
Best Fit line

In [18]:

```
prediction=lr.predict(g_test)
plt.scatter(h_test,prediction)
```

Out[18]:

```
<matplotlib.collections.PathCollection at 0x20f9c720070>
```



To find score

In [19]:

```
print(lr.score(g_test,h_test))
```

```
1.0
```

Import Lasso and ridge

In [20]:

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

Ridge

In [21]:

```
ri=Ridge(alpha=5)
ri.fit(g_train,h_train)
```

Out[21]:

```
Ridge(alpha=5)
```

In [22]:

```
ri.score(g_test,h_test)
```

Out[22]:

```
1.0
```

In [23]:

```
ri.score(g_train,h_train)
```

Out[23]:

```
1.0
```

Lasso

In [24]:

```
l=Lasso(alpha=6)
l.fit(g_train,h_train)
```

Out[24]:

```
Lasso(alpha=6)
```

In [25]:

```
l.score(g_test,h_test)
```

Out[25]:

```
1.0
```

In [27]:

```
ri.score(g_train,h_train)
```

Out[27]:

```
1.0
```

ElasticNet

In [28]:

```
from sklearn.linear_model import ElasticNet
e=ElasticNet()
e.fit(g_train,h_train)
```

Out[28]:

```
ElasticNet()
```

Coeffecient,intercept

In [29]:

```
print(e.coef_)
```

```
[1.]
```

In [30]:

```
print(e.intercept_)
```

```
2.2351741790771484e-08
```

Prediction

In [31]:

```
c=e.predict(g_test)
```

In [32]:

```
print(e.score(g_test,h_test))
```

```
1.0
```

Evaluation

In [33]:

```
from sklearn import metrics
print("Mean Absolute Error",metrics.mean_absolute_error(h_test,c))
```

```
Mean Absolute Error 2.584420144557953e-08
```

In [34]:

```
print("Mean Squared Error",metrics.mean_squared_error(h_test,c))
```

```
Mean Squared Error 6.835894697521105e-16
```

In [35]:

```
print("Root Mean Squared Error",np.sqrt(metrics.mean_squared_error(h_test,c)))
```

```
Root Mean Squared Error 2.6145543975066012e-08
```

In [ ]:

```

```