Problem Statement

A real estate agent want help to predict the house price for regions in USA.He gave us the dataset to work on to use linear regression model.Create a model that helps him to estimate of what the house would sell for

Import libraries

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

In [2]:
```
# To import dataset
df=pd.read_csv('BreastCancer csv')
df
```

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
df.head(10)
```

Out[3]:

	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
5	843786	M	12.45	15.70	82.57	477.1	0.12
6	844359	M	18.25	19.98	119.60	1040.0	0.09
7	84458202	M	13.71	20.83	90.20	577.9	0.11
8	844981	M	13.00	21.82	87.50	519.8	0.12
9	84501001	M	12.46	24.04	83.97	475.9	0.11

10 rows × 33 columns

Data Cleaning and Pre-Processing

In [4]:
```
df.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]:
```
df.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]:
```
df.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')
```

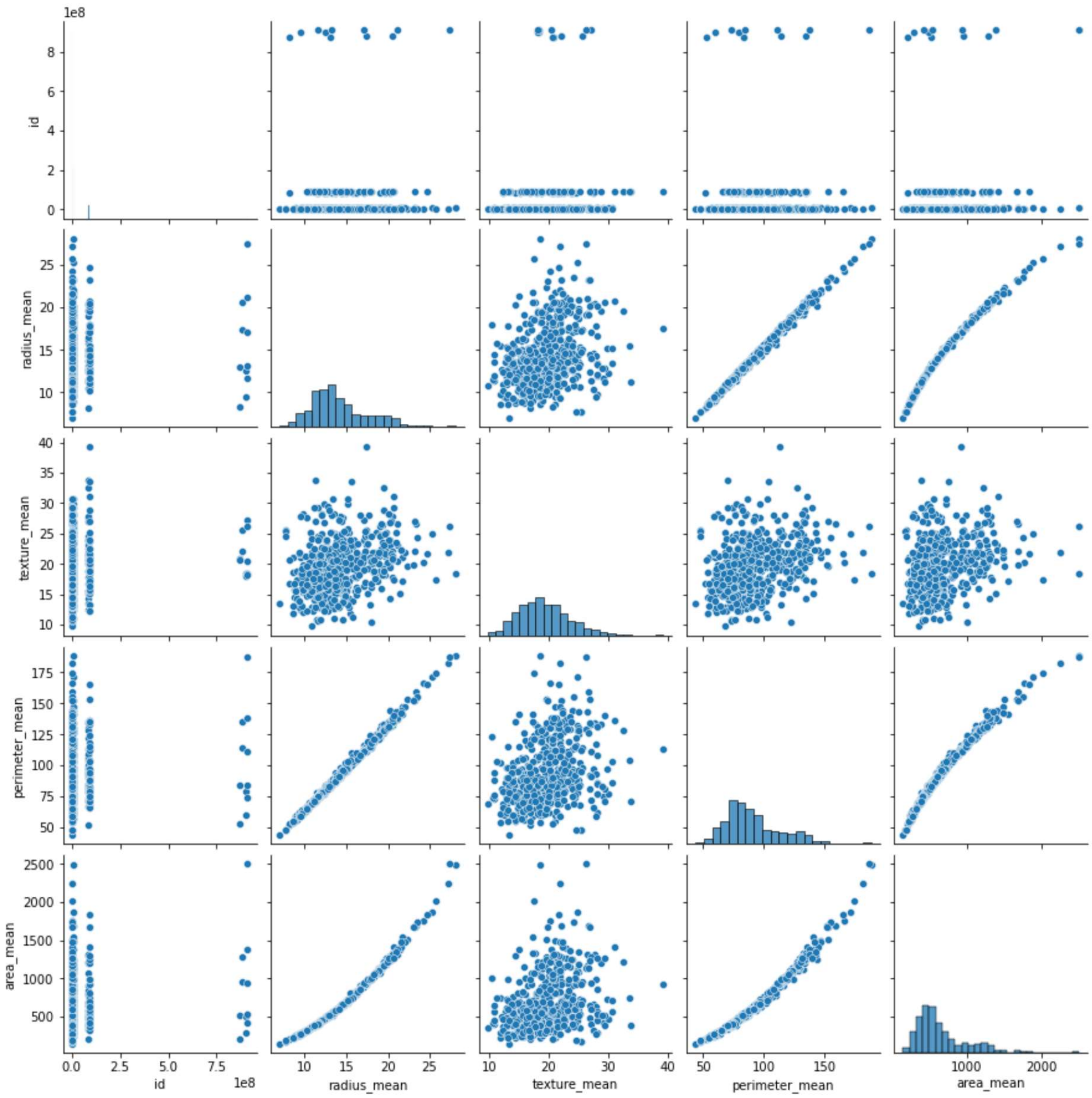
In [7]:
```
a = df.dropna(axis='columns')
a.columns
```

Out[7]:
```
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 Visualization

In [8]:
```
sns.pairplot(a[['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_me
       'area_mean']])
```

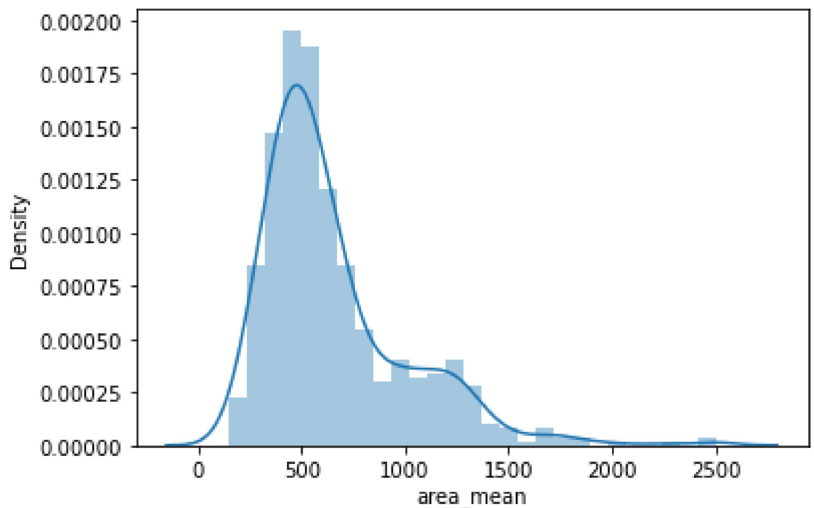
Out[8]: <seaborn.axisgrid.PairGrid at 0x1a9004a1160>



In [9]:
```
sns.distplot(a['area_mean'])
```

```
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 function for histograms).
  warnings.warn(msg, FutureWarning)
```

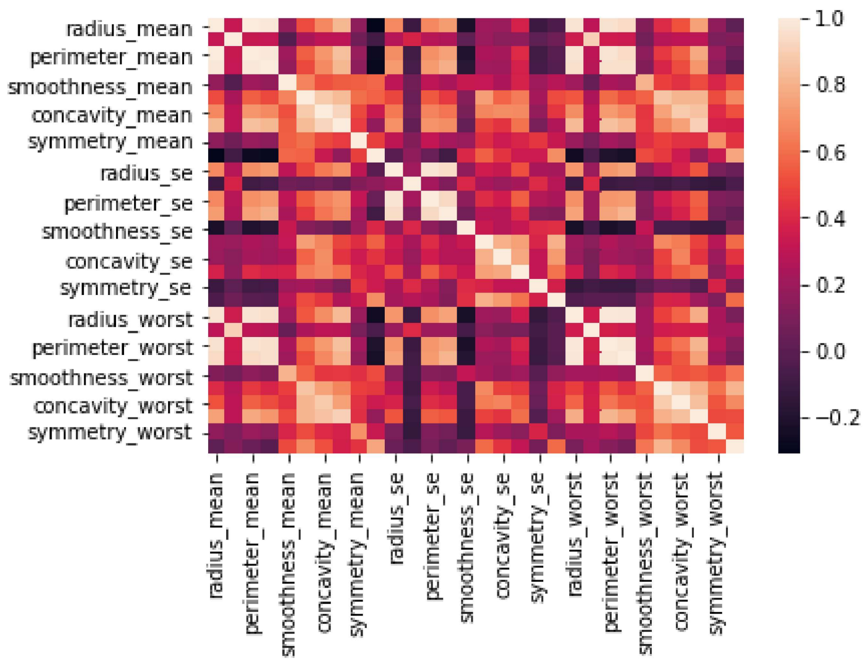
Out[9]: <AxesSubplot:xlabel='area_mean', ylabel='Density'>



In [10]:
```
a1=a[['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 [11]:
```
sns.heatmap(a1.corr())
```

Out[11]: <AxesSubplot:>



To Train the Model - Model Building

We are going to train Linear Regression model;We need to split out data into two variables x and y where x is independent variable (input) and y is dependent on x(output). We could ignore address column as it is not required for our model.

In [12]:
```
x=a1[['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']]
y=a1['area_mean']
```

To split my dataset into training and test data

In [13]:
```
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 [14]:
```
from sklearn.linear_model import LinearRegression

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

Out[14]:
```
LinearRegression()
```

In [15]:
```
print(lr.intercept_)
```

```
2.2737367544323206e-13
```

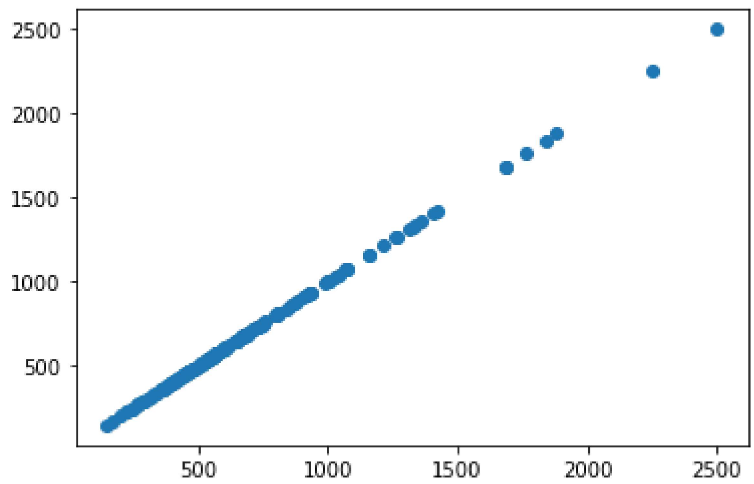
In [16]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[16]:

	Co-efficient
radius_mean	-7.948633e-14
texture_mean	2.477203e-15
perimeter_mean	1.180114e-14
area_mean	1.000000e+00
smoothness_mean	2.466150e-13
compactness_mean	6.737845e-13
concavity_mean	-1.844173e-13
concave points_mean	-2.292592e-14
symmetry_mean	-2.298415e-13
fractal_dimension_mean	-1.666690e-12
radius_se	6.379513e-14
texture_se	1.692779e-14
perimeter_se	-1.161653e-14
area_se	-6.645494e-16
smoothness_se	-8.254036e-13
compactness_se	5.914487e-13
concavity_se	6.301441e-13
concave points_se	-4.072886e-12
symmetry_se	-1.274531e-12
fractal_dimension_se	-9.833008e-13
radius_worst	-1.687976e-14
texture_worst	-2.535485e-15
perimeter_worst	2.897057e-15
area_worst	-2.653879e-16
smoothness_worst	6.298029e-14
compactness_worst	-9.376073e-14
concavity_worst	-7.809966e-14
concave points_worst	2.586427e-13
symmetry_worst	1.218439e-13
fractal_dimension_worst	1.392947e-13

In [17]:
```
prediction=lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[17]: <matplotlib.collections.PathCollection at 0x1a903a9eca0>



In [18]:
```
print(lr.score(x_test,y_test))
```

```
1.0
```

In [19]:
```
from sklearn.linear_model import ElasticNet
en = ElasticNet()
en.fit(x_train,y_train)
```

Out[19]: ElasticNet()

In [20]:
```
print(en.coef_)
```

```
[ 0.00000000e+00  0.00000000e+00  0.00000000e+00  9.99985318e-01
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00 -0.00000000e+00  0.00000000e+00 -0.00000000e+00
  0.00000000e+00  0.00000000e+00 -0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00 -0.00000000e+00 -0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  7.08117175e-06
 -0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
 -0.00000000e+00 -0.00000000e+00]
```

In [21]:
```
print(en.intercept_)
```

```
0.0033697282995035494
```

In [22]:
```
print(en.predict(x_test))
```

```
[ 566.30009169  321.4011833   504.80020364  566.20000465  357.60105121
  143.50257436  449.3008988   529.40012431 1263.9972043   668.59940552
  386.80102543  451.10025348  337.70126406  463.70043317  595.90020095
  526.39995238  602.40037461  565.40210463  355.30127759  545.20178939
  534.60089841  537.8999542   260.90173518  234.30217457  241.00196999
  644.19922204 1260.9971138   689.39931612  285.70175417  394.10088851
  904.29866142  384.60093451  466.50093165  492.10006578  571.09984632
  378.40090587  920.59982418  289.7018137   465.4004946   415.10101286
 1156.99696952  712.80019833  805.0988503   736.90035434  555.10017257
  264.00158841  476.30028571  372.7007056   680.89892187  657.09891652
  560.99981757  404.90073632  928.29990941  407.40099207  458.70051437
  464.10044429  298.30154794 1076.99852643 1067.99901971 1325.99775276
  705.60026863  477.30014357  674.49886712  507.90012414 1760.99219478
  644.80025417  538.90025392  271.20193306  998.89805147  280.50175546
  409.00094946  402.90069629  682.50110345 1040.99807069 1006.00000077
  838.09821002  933.09829524  201.90212058  422.90107183  512.20037541
 1001.00297038  600.39955919 1840.99504945  480.40005132  728.19856237
  642.70132664  358.90116737 1877.99393974  618.40015741  537.29999133
  538.39992349  562.09993667  371.10156113  869.49930688  813.00016471
 1684.99269452 2500.99677447  568.90001812  678.10029282  221.80205289
  561.0000817   359.90155207  453.10036645  553.4997195   747.19844912
 1154.99644656  731.29907273  461.40022901  584.79966005  646.09908581
  798.80095385  491.90029532  446.00052804  527.20004897  329.60133058
  536.90036684 1681.99837518  602.90039913  412.60094122 1419.00683919
  572.30020117  546.30002419  799.99833673  271.30172766  664.69897985
  420.50108936  552.399958    438.60105872  680.6987591   793.19930118
  392.00080109  435.60031959 1406.9974487   461.00112215  684.49904321
  508.30009207  433.80070362 1075.99866857  744.70205969 1363.99757725
 1214.0013515   984.60066902 1026.99880731  389.40100142  593.69957046
  224.50202033  662.6993314   857.59845482  361.60164112 2249.99310929
  514.29989776  599.39966026  758.5987137   611.19970724  918.59952781
  930.90137244  311.70125278  396.6007902   674.79933645 1334.99754982
  725.49932287  713.29947437  551.70115154  441.00076219  668.30010884
 1310.99891478  170.40245134  516.40004113  880.1977548   481.90109501
  462.00033916]
```

In [23]:
```
print(en.score(x_test,y_test))
```

```
0.9999999999785036
```

Evaluation Metrics

In [24]:
```
from sklearn import metrics
print("Mean Absolytre Error:",metrics.mean_absolute_error(y_test,prediction))
print("Mean Squared Error:",metrics.mean_squared_error(y_test,prediction))
print("Root Mean Squared Error:",metrics.mean_squared_error(y_test,prediction)
```

```
Mean Absolytre Error: 8.493271063705525e-14
Mean Squared Error: 1.611335588803441e-26
Root Mean Squared Error: 1.611335588803441e-26
```