Problem statement

predicting the house price in USA.To create a model to help him estimate of what the house would sell for.

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

In [2]:
```
df=pd.read_csv("BreastCancer  csv")
```

To display top 10 rows

In [3]:
```
1 df.head(10)
```

Out[3]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mea
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980
5	843786	M	12.45	15.70	82.57	477.1	0.12780	0.17000	0.1578
6	844359	M	18.25	19.98	119.60	1040.0	0.09463	0.10900	0.1127
7	84458202	M	13.71	20.83	90.20	577.9	0.11890	0.16450	0.0936
8	844981	M	13.00	21.82	87.50	519.8	0.12730	0.19320	0.1859
9	84501001	M	12.46	24.04	83.97	475.9	0.11860	0.23960	0.2273

10 rows × 33 columns

Data Cleaning And Pre-Processing

In [4]:
```
1 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]:
```
# Display the statistical summary
df.describe()
```

Out[5]:

	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean
count	5.690000e+02	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000
mean	3.037183e+07	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799
std	1.250206e+08	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720
min	8.670000e+03	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000
25%	8.692180e+05	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560
50%	9.060240e+05	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540
75%	8.813129e+06	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700
max	9.113205e+08	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800

8 rows × 32 columns

In [6]:
```
# To display the col headings
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]:
```
cols=df.dropna(axis=1)
cols
```

Out[7]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_m
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3(
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19
...	...	...	...	...	...	...	...	...	
564	926424	M	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24
565	926682	M	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14
566	926954	M	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09
567	927241	M	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35
568	92751	B	7.76	24.54	47.92	181.0	0.05263	0.04362	0.0(

569 rows × 32 columns

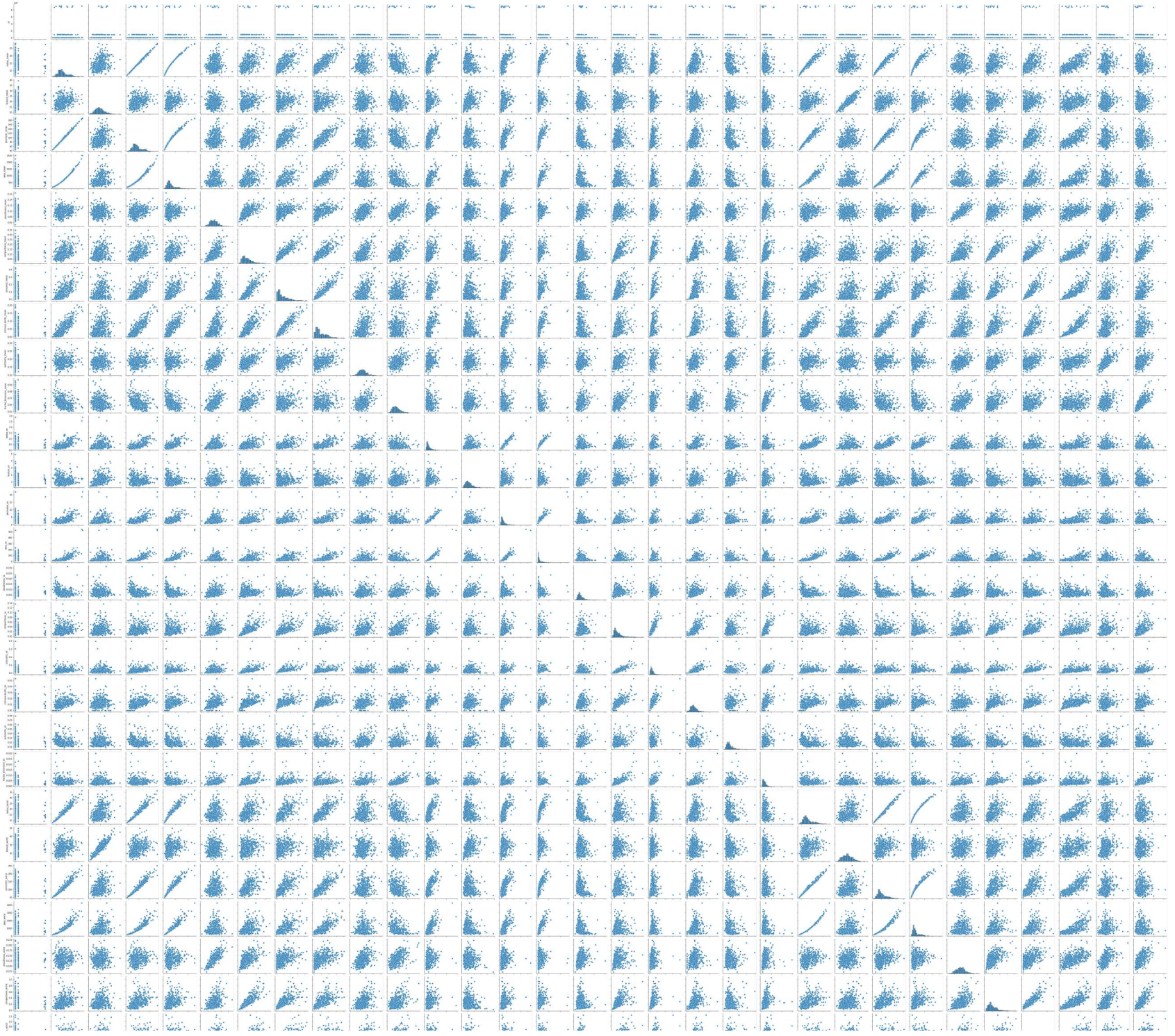
In [8]:
```
cols.columns
```

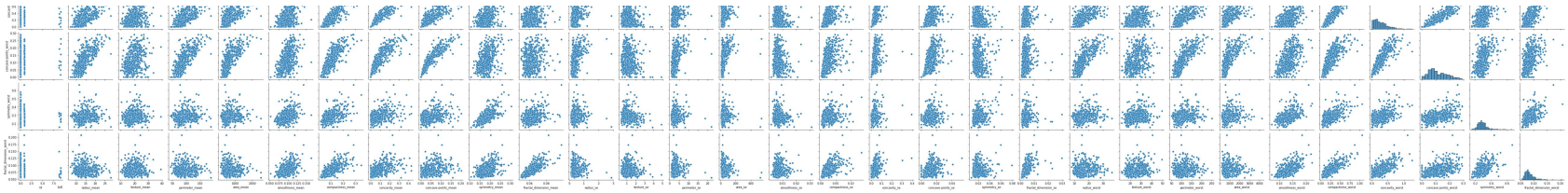
Out[8]:
```
Index(['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       '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 [9]:
```
1 sns.pairplot(cols)
```

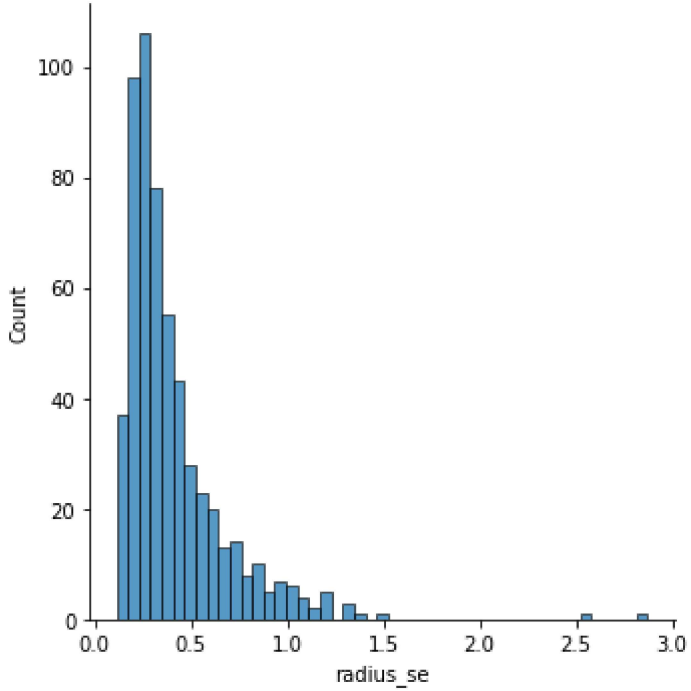
Out[9]:
```
<seaborn.axisgrid.PairGrid at 0x1a9f8be7d00>
```





In [10]:
```
1 sns.displot(df['radius_se'])
```

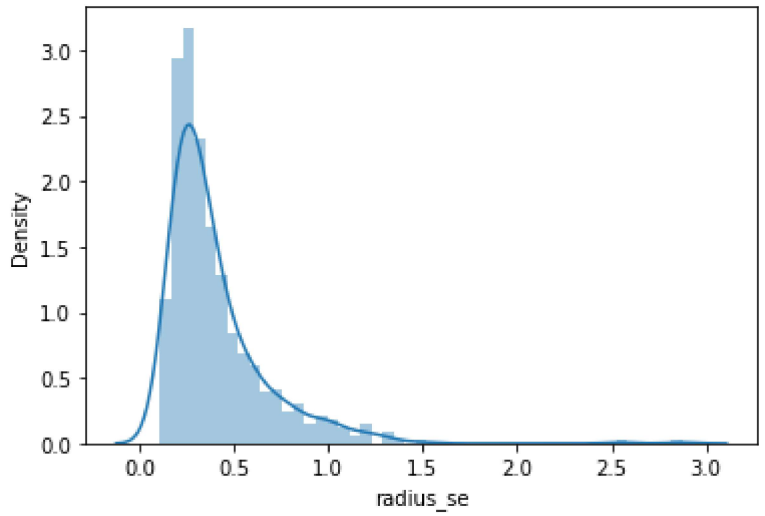
Out[10]: <seaborn.axisgrid.FacetGrid at 0x1a9f8bd7430>



In [11]:
```
# We use displot in older version we get distplot use displot
sns.distplot(df['radius_se'])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a dep
recated 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[11]: <AxesSubplot:xlabel='radius_se', ylabel='Density'>



In [12]:

```
df1=cols[['radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean']]
df1
```

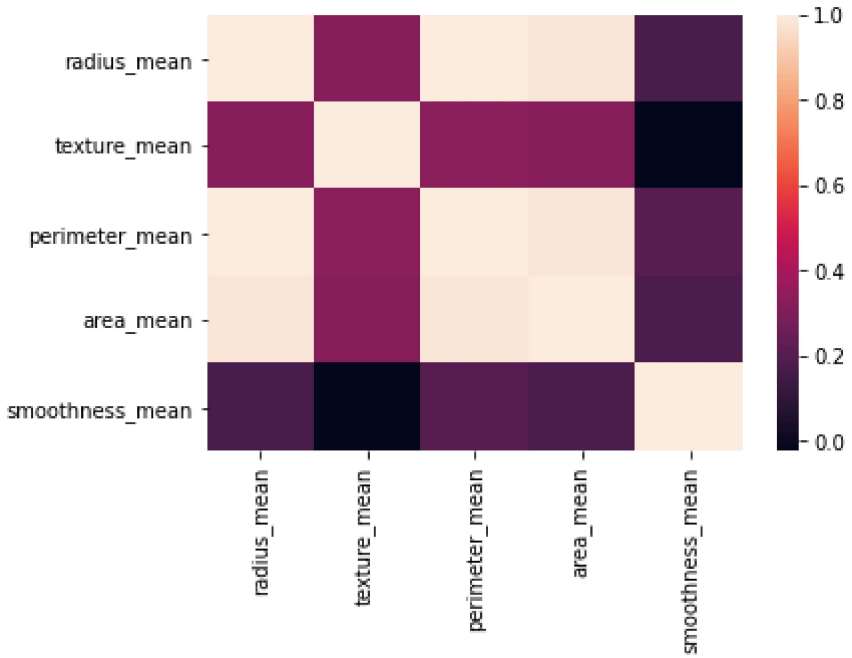
Out[12]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean
0	17.99	10.38	122.80	1001.0	0.11840
1	20.57	17.77	132.90	1326.0	0.08474
2	19.69	21.25	130.00	1203.0	0.10960
3	11.42	20.38	77.58	386.1	0.14250
4	20.29	14.34	135.10	1297.0	0.10030
...	...	...	...	...	...
564	21.56	22.39	142.00	1479.0	0.11100
565	20.13	28.25	131.20	1261.0	0.09780
566	16.60	28.08	108.30	858.1	0.08455
567	20.60	29.33	140.10	1265.0	0.11780
568	7.76	24.54	47.92	181.0	0.05263

569 rows × 5 columns

In [13]:
```
1 sns.heatmap(df1.corr())
```

Out[13]: <AxesSubplot:>



To train the model - MODEL BUILD

Going to train linear regression model;We split our data into 2 variables x and y where x is independent var(input) and y is dependent on x(output), we could ignore address col as it is not required for our model

In [14]:
```
1 x=df1[['radius_mean', 'texture_mean', 'perimeter_mean',
2        'area_mean', 'smoothness_mean']]
3 y=df1[['texture_mean']]
```

To split the dataset into test data

In [15]:
```
# importing lib for splitting test data
from sklearn.model_selection import train_test_split
```

In [16]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3)
```

In [17]:
```
from sklearn.linear_model import LinearRegression

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

Out[17]:
```
LinearRegression()
```

In [18]:
```
print(lr.intercept_)
```

```
[3.26849658e-13]
```

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

```
1.0
```

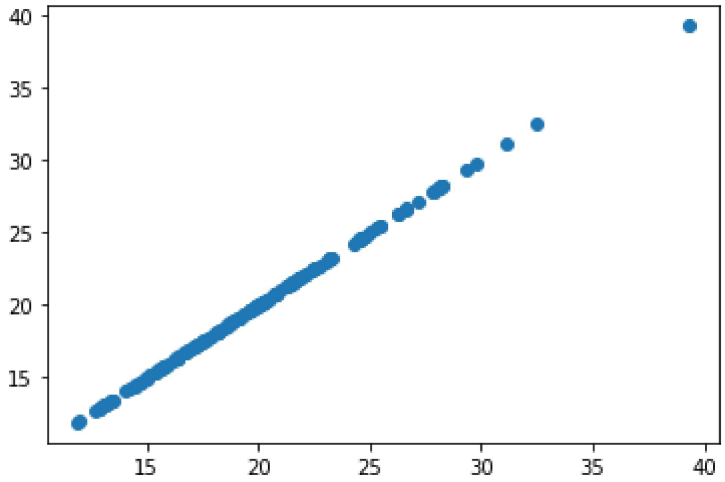
In [20]:
```
coeff=pd.DataFrame(lr.coef_)
coeff
```

Out[20]:

	0	1	2	3	4
0	5.162814e-14	1.0	-1.100310e-15	-1.492100e-15	6.352478e-15

In [21]:
```
pred = lr.predict(x_test)
plt.scatter(y_test,pred)
```

Out[21]: <matplotlib.collections.PathCollection at 0x1a9a9300d90>



In [22]:
```
from sklearn.linear_model import Ridge,Lasso
```

In [23]:
```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[23]: Ridge(alpha=10)

In [24]:
```
rr.score(x_test,y_test)
```

Out[24]: 0.999997631117829

In [25]:
```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[25]: Lasso(alpha=10)

In [26]:
```
la.score(x_test,y_test)
```

Out[26]: 0.6515090725116075

ELASTICNET

In [28]:
```
1 from sklearn.linear_model import ElasticNet
2 en=ElasticNet()
3 en.fit(x_train,y_train)
```

Out[28]:
```
ElasticNet()
```

In [29]:
```
1 print(en.coef_)
```

```
[ 0.00000000e+00  9.39099960e-01  0.00000000e+00  2.28203679e-04
 -0.00000000e+00]
```

In [30]:
```
1 print(en.intercept_)
```

```
[1.0150315]
```

In [31]:
```
prediction=en.predict(x_test)
prediction
```

Out[31]:
```
array([15.81310857, 13.50341205, 19.10868513, 24.85005626, 26.72403023,
       27.63531467, 19.04174463, 18.14868339, 38.11296225, 17.06478748,
       15.87120242, 15.79688413, 23.02177548, 13.38560997, 21.75121744,
       19.7426211 , 16.89288845, 22.45357221, 17.9903448 , 14.49578909,
       18.24804662, 20.1502992 , 16.48186507, 15.15887765, 19.93880099,
       21.34182864, 17.56730871, 28.84751099, 20.03051772, 20.24199813,
       17.78873796, 20.64948101, 12.32189479, 12.34909332, 26.25592486,
       14.66757819, 20.26309264, 15.86777685, 21.96344674, 24.6067228 ,
       21.10860376, 16.29623751, 18.65248172, 17.51046054, 22.10363226,
       18.78080221, 20.64805767, 22.19489163, 13.15800225, 19.37560136,
       18.43439595, 27.48904125, 14.90258222, 24.33207895, 23.03835656,
       17.32245799, 27.43240205, 30.56364329, 20.04300548, 17.17605947,
       24.41670787, 17.3804124 , 17.28776768, 17.53613429, 24.20565937,
       19.91079108, 19.48124604, 22.17478688, 14.85659583, 24.35304669,
       22.38005179, 22.96112993, 13.2694721 , 15.37489096, 13.36774246,
       13.67739275, 24.56098325, 20.080834  , 17.23307742, 24.1018494 ,
       24.10879611, 19.65494787, 13.41699048, 22.93095304, 21.61809189,
       17.54837951, 18.04418558, 16.44345923, 18.74891881, 18.95520645,
       13.6598912 , 13.25132656, 20.09040255, 21.90036371, 27.83237022,
       20.62037573, 21.38914207, 13.63904019, 20.71188457, 21.42804399,
       20.31823597, 17.8704062 , 19.80128211, 19.75394   , 17.03656454,
       21.55673557, 21.71971362, 27.25905006, 13.81103217, 21.06699136,
       22.77511526, 20.43587484, 15.16134142, 15.99420671, 20.61448724,
       17.60213414, 24.25385503, 15.54380386, 18.22295353, 18.23796001,
       21.46615066, 14.6368738 , 16.53319059, 16.81068782, 20.14962559,
       16.24831653, 22.06660591, 16.44983627, 19.72765858, 16.49041172,
       27.58077997, 13.17260561, 15.6200399 , 19.93799379, 19.11129932,
       18.53276942, 24.9446204 , 27.54397322, 15.74842048, 15.90275032,
       21.43605979, 25.96229602, 21.397239  , 19.24242253, 15.11814079,
       18.88199023, 20.01558054, 22.39736743, 20.28146052, 18.18929349,
       22.97866613, 17.75770728, 19.57859008, 20.0212223 , 13.08020092,
       26.25478623, 15.55901572, 16.87554916, 19.83148707, 29.17379387,
       19.57698083, 16.8427496 , 18.7010688 , 21.14950204, 31.78670032,
       19.05678062, 24.99349231, 26.44535064, 27.56403483, 14.34344461,
       18.33673505])
```

In [32]:
```
print(en.score(x_test,y_test))
```

```
0.9967130173840267
```

EVALUATION METRICS

In [33]:
```
from sklearn import metrics
```

In [34]:
```
print("Mean Absolute Error:",metrics.mean_absolute_error(y_test,prediction))
```

```
Mean Absolute Error: 0.1972575544531973
```

In [35]:
```
print("Mean Squared Error:",metrics.mean_squared_error(y_test,prediction))
```

```
Mean Squared Error: 0.06762033326424403
```

In [37]:
```
print("Root Mean Squared Error:",np.sqrt(metrics.mean_squared_error(y_test,prediction)))
```

```
Root Mean Squared Error: 0.2600390994912958
```

In [ ]:
```

```