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

Data Collection

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

In [203]:
```
#import the dataset
data=pd.read_csv(r"C:\Users\user\Desktop\Vicky\11_winequality-red.csv")[0:500]
```

In [204]:
```
#to display top 10 rows
data.head()
```

Out[204]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4

In [205]:
```
#to display null values
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed acidity         500 non-null    float64
 1   volatile acidity      500 non-null    float64
 2   citric acid           500 non-null    float64
 3   residual sugar        500 non-null    float64
 4   chlorides             500 non-null    float64
 5   free sulfur dioxide   500 non-null    float64
 6   total sulfur dioxide  500 non-null    float64
 7   density               500 non-null    float64
 8   pH                    500 non-null    float64
 9   sulphates             500 non-null    float64
 10  alcohol               500 non-null    float64
 11  quality               500 non-null    int64  
dtypes: float64(11), int64(1)
memory usage: 47.0 KB
```

In [206]:
```
data.shape
```

Out[206]: (500, 12)

In [207]:
```
#to display summary of statistics
data.describe()
```

Out[207]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	d
count	500.00000	500.000000	500.000000	500.000000	500.000000	500.000000	500.000000	500.0
mean	8.68640	0.533370	0.302460	2.586800	0.093962	15.041000	51.444000	0.9
std	1.88393	0.176169	0.216569	1.382229	0.060240	9.783673	33.716947	0.0
min	4.60000	0.180000	0.000000	1.200000	0.039000	3.000000	8.000000	0.9
25%	7.40000	0.400000	0.107500	1.900000	0.073000	7.000000	25.000000	0.9
50%	8.10000	0.530000	0.275000	2.200000	0.082000	12.000000	42.000000	0.9
75%	9.82500	0.645000	0.480000	2.700000	0.093000	20.000000	67.000000	0.9
max	15.60000	1.330000	1.000000	15.500000	0.611000	68.000000	165.000000	1.0

In [208]:
```
#to display columns name
data.columns
```

Out[208]:
```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'quality'],
      dtype='object')
```

In [209]:
```
data.fillna(value=5)
```

Out[209]:

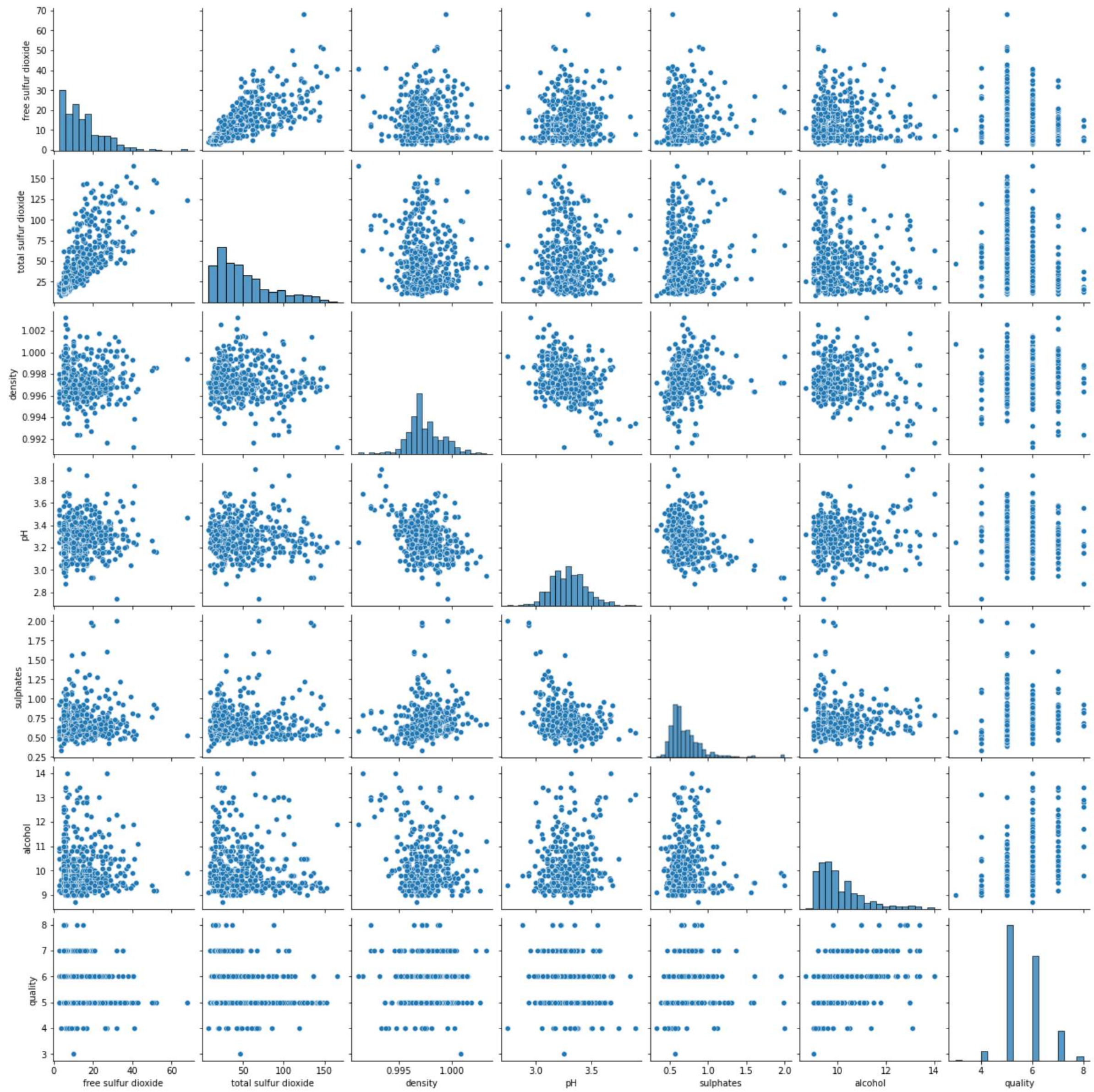
	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcoh
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9
...	...	...	...	...	...	...	...	...	...	...	
495	10.7	0.35	0.53	2.6	0.070	5.0	16.0	0.9972	3.15	0.65	11
496	7.8	0.52	0.25	1.9	0.081	14.0	38.0	0.9984	3.43	0.65	9
497	7.2	0.34	0.32	2.5	0.090	43.0	113.0	0.9966	3.32	0.79	11
498	10.7	0.35	0.53	2.6	0.070	5.0	16.0	0.9972	3.15	0.65	11
499	8.7	0.69	0.31	3.0	0.086	23.0	81.0	1.0002	3.48	0.74	11

500 rows × 12 columns

In [212]:
```
data1=data[['free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'quality']]
```

In [213]:
```
sns.pairplot(data1)
```

Out[213]: <seaborn.axisgrid.PairGrid at 0x12533b239a0>

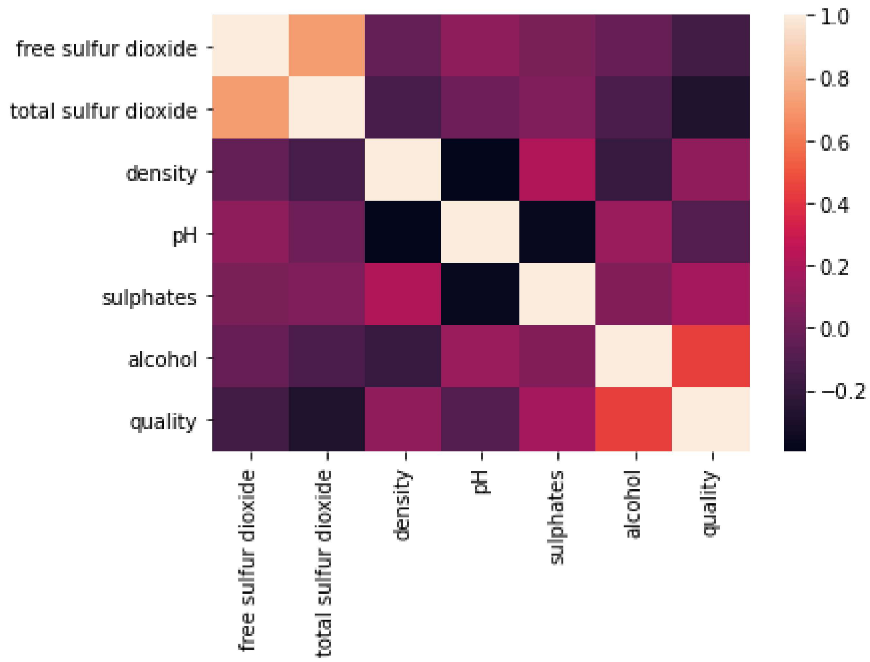


EDA and Visualization

In [214]:
```
#sns.distplot(data['Co2-Emissions'])
```

In [215]:
```
sns.heatmap(data1.corr())
```

Out[215]: <AxesSubplot:>



In [174]:
```
data1.fillna(value=5)
```

Out[174]:

	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.
...	...	...	...	...	...	...	
495	914333	B	14.87	20.21	96.12	680.9	0.
496	914366	B	12.65	18.17	82.69	485.6	0.
497	914580	B	12.47	17.31	80.45	480.1	0.
498	914769	M	18.49	17.52	121.30	1068.0	0.
499	91485	M	20.59	21.24	137.80	1320.0	0.

500 rows × 27 columns

To train the model

we are going to train the linear regression model ;We need to split the two variable x and y

In [183]:
```
x=data[['fixed acidity', 'volatile acidity'
       ]]

y=data1['alcohol']
```

In [184]:
```

#To split test and train data
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.1)
```

In [185]:
```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[185]: LinearRegression()

In [186]:
```
lr.intercept_
```

Out[186]: 38835336.15720841

In [187]:
```
coeff = pd.DataFrame(lr.coef_,x.columns,columns=["Co-efficient"])
coeff
```

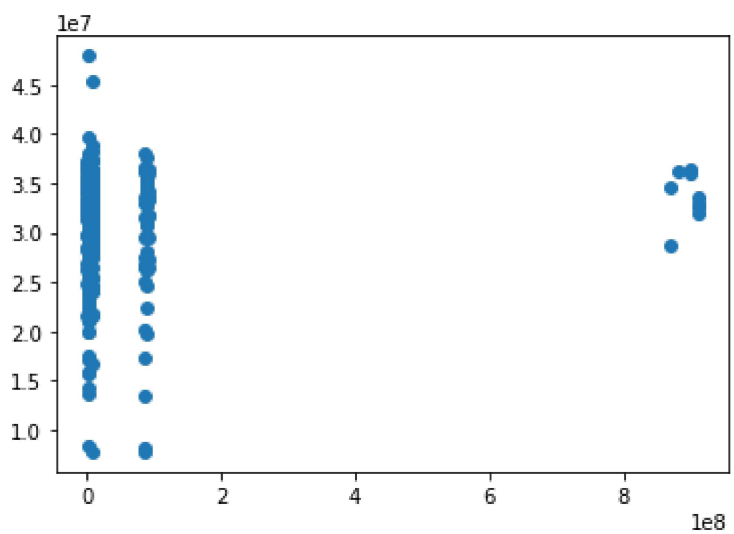
Out[187]:

	Co-efficient
compactness_worst	-5.751864e+07
concavity_worst	2.734553e+07

In [ ]:

In [188]:
```
prediction = lr.predict(x_train)
plt.scatter(y_train,prediction)
```

Out[188]: <matplotlib.collections.PathCollection at 0x1252332bdf0>



In [189]:
```
lr.score(x_test,y_test)
```
Out[189]:
```
-0.004073209588378202
```

In [190]:
```
lr.score(x_train,y_train)
```
Out[190]:
```
0.0015226742955708472
```

In [191]:
```
from sklearn.linear_model import Ridge,Lasso
```

In [192]:
```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
rr.score(x_test,y_test)
```
Out[192]:
```
-0.009582880189396903
```

In [193]:
```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
la.score(x_test,y_test)
```
Out[193]:
```
-0.0040737802210928376
```

In [194]:
```
from sklearn.linear_model import ElasticNet
en= ElasticNet()
en.fit(x_train,y_train)
```
Out[194]:
```
ElasticNet()
```

In [195]:
```
print(en.coef_)
```
```
[-1241747.15166715  -916017.96199218]
```

In [196]:
```
print(en.intercept_)
```
```
32089776.5923149
```

In [197]:
```
prediction = en.predict(x_test)
prediction
```

Out[197]:
```
array([31393740.13355368, 31088477.23434921, 31253514.61656909,
       31542670.42800784, 31365755.09598579, 31882295.855237  ,
       31913180.90717359, 31534008.63013534, 31710076.15057814,
       31780477.09711697, 31991828.95134632, 31496373.89934802,
       31741856.92519902, 31944222.95010149, 31769883.86167683,
       31373511.88890291, 31168714.43934447, 31044309.26365758,
       31874848.90308605, 31368514.77333577, 31781323.68778142,
       31934699.37516704, 31284579.70265878, 31228892.23275906,
       31627703.65606768, 31960955.92049683, 31496299.88949108,
       31547727.64488157, 31849175.58418468, 30784912.09903306,
       31439275.55225564, 31609501.3545823 , 31918403.4321233 ,
       31493836.73418531, 31848553.36347367, 31356747.691898  ,
       31664347.58357762, 31226737.5240613 , 30935373.9311361 ,
       31996899.68451488, 31771755.96513853, 32002412.79677851,
       31819043.74666099, 31906922.55968469, 31666012.16821847,
       31571940.18819558, 31242794.29848333, 31353086.52385158,
       31404467.02644319, 31614317.40728684])
```

In [216]:
```
#print(en.score(x_test,y_train))
```

In [217]:
```
from sklearn import metrics
```

In [218]:
```
print("Mean Absolute error:",metrics.mean_absolute_error(y_test,prediction))
```

```
Mean Absolute error: 62586502.49745239
```

In [219]:
```
print("Mean Absolute Square error:",metrics.mean_squared_error(y_test,predicti
```

```
Mean Absolute Square error: 3.080001673006421e+16
```

In [220]:
```
print("Root mean Square error:",np.sqrt(metrics.mean_squared_error(y_test,pred
```

```
Root mean Square error: 175499335.41203
```

In [ ]:

In [ ]: