Health Insurance Premium Price Prediction

Before starting, first let’s understand what Health Insurance is and what is Premium? **Health insurance** is a type of insurance that covers medical expenses that arise due to an illness. These expenses could be related to hospitalization costs, cost of medicines or doctor consultation fees. Insurance will cover a portion of the cost of a policyholder's medical costs. How much the insurance covers – and how much the policyholder pays via copays, deductibles, and coinsurance – depend on the details of the policy itself, with specific rules and regulations that apply to some plans. Now if you are wondering what are these copay, deductible and coinsurance? Let’s check them briefly because it is very important to know about these 3 words.

Copay - Copay refers to when policyholders have to bear a fixed part of their expenses towards medical treatment while the rest is borne by the insurer. This can either be as a fixed amount or a fixed percentage of the treatment costs.

For example, if your insurance policy comes with a copay clause of Rs. 2000 of your treatment expenses and the treatment costs you Rs. 10,000, you will be required to pay Rs. 2000 towards your treatment, while the rest of Rs. 8000 will be covered by the insurer.

Deductible: Deductibles is a fixed sum of money that policyholders are required to pay before their insurance policy starts contributing to their medical treatment. The term for paying deductibles is decided by the insurance provider - whether it is per year or per treatment.

For example, if your policy mandates a deductible of Rs. 5000, you will be required to pay for your treatment expenditures amounting up to Rs. 5000, after which your insurance policy will kick in.

Coinsurance: Coinsurance refers to the percentage of treatment costs that you have to bear after paying the deductibles. This amount is generally offered as a fixed percentage. It is similar to the copayment provision under health insurance.

For example, if your coinsurance is 20%, then you will be liable to bear 20% of the treatment cost while the rest 80% will be borne by your insurance provider.

What is Premium?

A health insurance premium is an upfront payment made on behalf of an individual or family to keep their health insurance policy active. Premiums are typically paid monthly when purchased on the individual market, although individuals who receive insurance through their employer usually pay their portion of the premium through payroll deductions In addition to the premium, consumers may have to pay out-of-pocket costs—deductible, copays, and coinsurance—when they seek medical care.

Why Health Insurance?

It’s important to have health insurance as a safety net. If you unexpectedly get sick or injured, health insurance is there to help cover costs that you likely can’t afford to pay on your own. Health care can be very expensive. It can be an enormous financial burden. Surgery, emergency care, prescription drugs, lab work, scans and examinations – these sorts of costs can add up very quickly.

Now it's quite clear about basic information regarding health insurance premium. Let's go for the dataset. This dataset is taken from the Kaggle.

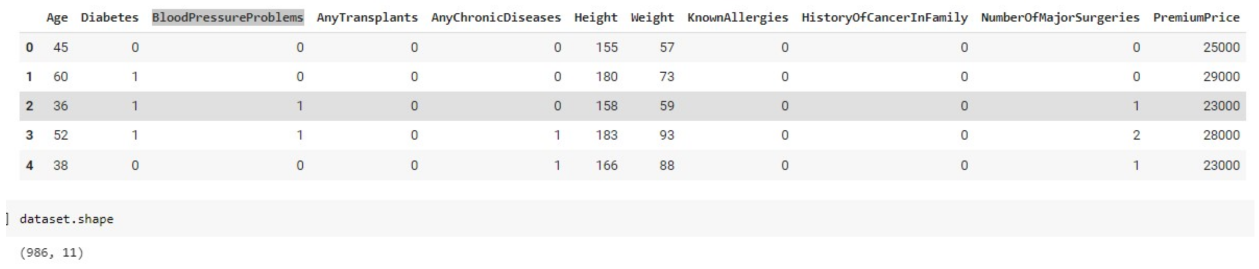
Problem Statement: Based on the given features need to predict the premium price. It is a Regression problem.

Why this Model should be used?

The model should be used because it makes the Insurer to predict the premium price for the the policy holder based on their health records.
Suppose if person wants to take health insurance and as per his health records his premium price to be opted is of Rs25000, but we suggest him to pay Rs 20000 which can be loss for the company. Hence using this model we get prediction of premium price such that we will get idea how much should be the premium price according to that we can continue the business towards individuals. This can be one of very good idea to increase the profit of business.

Drawback: If we have any advantages of using anything there will be disadvantage as well. This requires each employee to have the computer knowledge otherwise the person won't understand what is happening. The other one can be training cost is more. The data collection can be harder and time consuming because if we don't have sufficient data then it is difficult to build the Machine Learning Model.

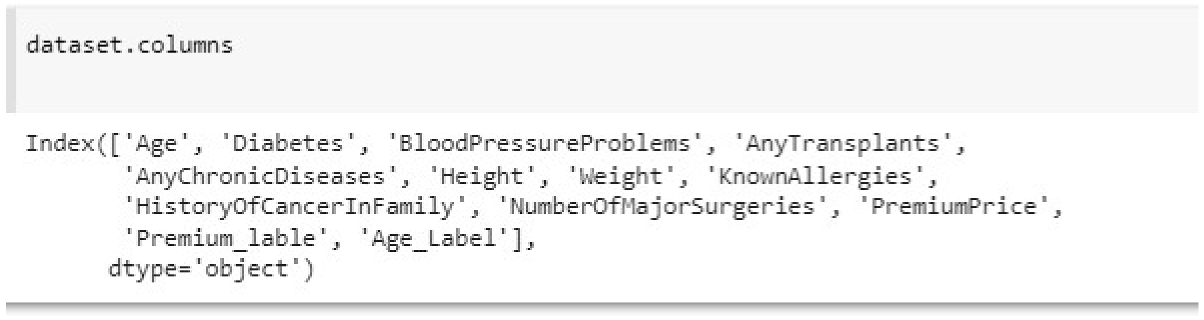
Dataset:



	Age	Diabetes	BloodPressureProblems	AnyTransplants	AnyChronicDiseases	Height	Weight	KnownAllergies	HistoryOfCancerInFamily	NumberOfMajorSurgeries	PremiumPrice
0	45	0	0	0	0	155	57	0	0	0	25000
1	60	1	0	0	0	180	73	0	0	0	29000
2	36	1	1	0	0	158	59	0	0	1	23000
3	52	1	1	0	1	183	93	0	0	2	28000
4	38	0	0	0	1	166	88	0	0	1	23000

```
] dataset.shape

(986, 11)
```

The dataset has total 986 rows and 11 columns or features.



```
dataset.columns
```

```
Index(['Age', 'Diabetes', 'BloodPressureProblems', 'AnyTransplants',
       'AnyChronicDiseases', 'Height', 'Weight', 'KnownAllergies',
       'HistoryOfCancerInFamily', 'NumberOfMajorSurgeries', 'PremiumPrice',
       'Premium_lable', 'Age_Label'],
      dtype='object')
```

Performance Matrices: Here I have used 2 Performance Matrices, first one is Mean absolute Percentage error and second one is R2_score. Let's understand what does these 2 matrices mean and how these are calculated and why these are used for this problem.

1. **MAPE: Mean Absolute Percentage Error** is a statistical measure to define the accuracy of a machine learning algorithm on a particular dataset. This is calculated as

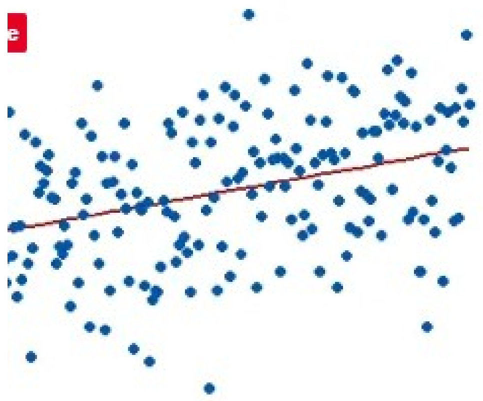
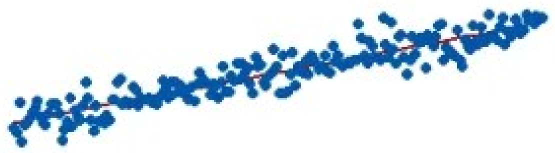
   $$M = \frac{1}{n}\sum_{t=1}^{n}\left|\frac{A_t - F_t}{A_t}\right|$$

   Where,
   M = MAPE
   N = Number of times the Summation Iteration Happens
   At = Actual Value
   Ft = Predicted or Forecast value

Here the objective of the problem is to predict the premium price, which is Target variable. This target range will not have any effect on the metric score values and the error will be in range from 0 to 1 in which 1 is worst and 0 is best model, MAPE is also very intuitive to interpretation of relative error. This is also makes easier to compare the results of model.

2. **R2_Score:** This statistic indicates the percentage of the variance in the dependent variable that the independent variables explain collectively. R-squared measures the strength of the relationship between your model and the dependent variable on a convenient 0 – 100% scale.
   This is used because r2 score will give the variability difference error by which we can get to know how the predicted value varies with respect to Actual values.

   Let's observe little about this r2_score because same plots are used for the models to check the variance of actual and predicted values.

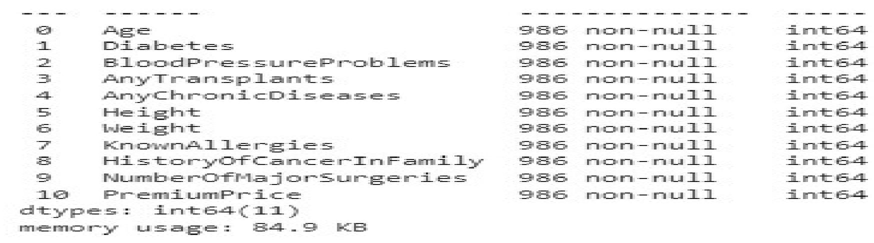



R-squared for the regression model on the left is 15%, and for the model on the right it is 85%. When a regression model accounts for more of the variance, the data points are closer to the regression line. Hence Scatter plot is suitable to observe these R2 score values.

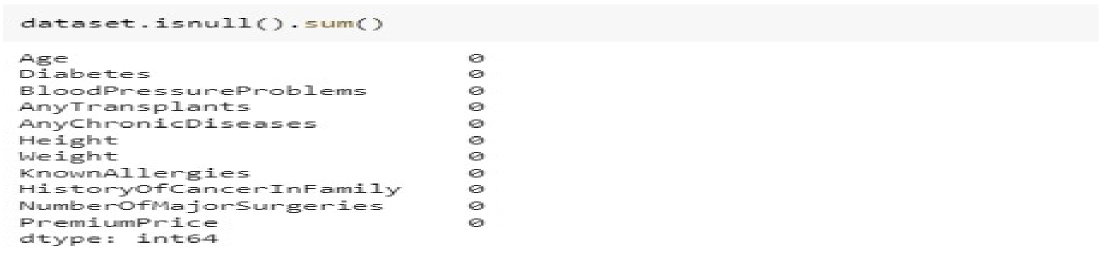
Data Cleaning:

This includes handling null values, removing duplicates and also converting categorical data into numerical data for further process.

But the good part of this dataset is that it does not have any null values, no duplicate values and all the features are numerical features.



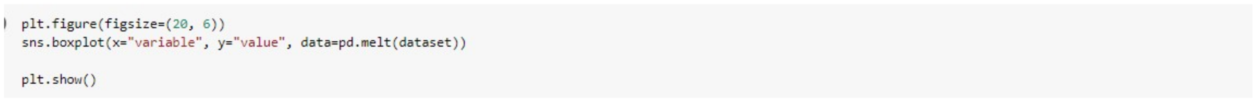
```
---  ------                   --------------  -----
 0   Age                      986 non-null    int64
 1   Diabetes                 986 non-null    int64
 2   BloodPressureProblems    986 non-null    int64
 3   AnyTransplants           986 non-null    int64
 4   AnyChronicDiseases       986 non-null    int64
 5   Height                   986 non-null    int64
 6   Weight                   986 non-null    int64
 7   KnownAllergies           986 non-null    int64
 8   HistoryOfCancerInFamily  986 non-null    int64
 9   NumberOfMajorSurgeries   986 non-null    int64
 10  PremiumPrice             986 non-null    int64
dtypes: int64(11)
memory usage: 84.9 KB
```



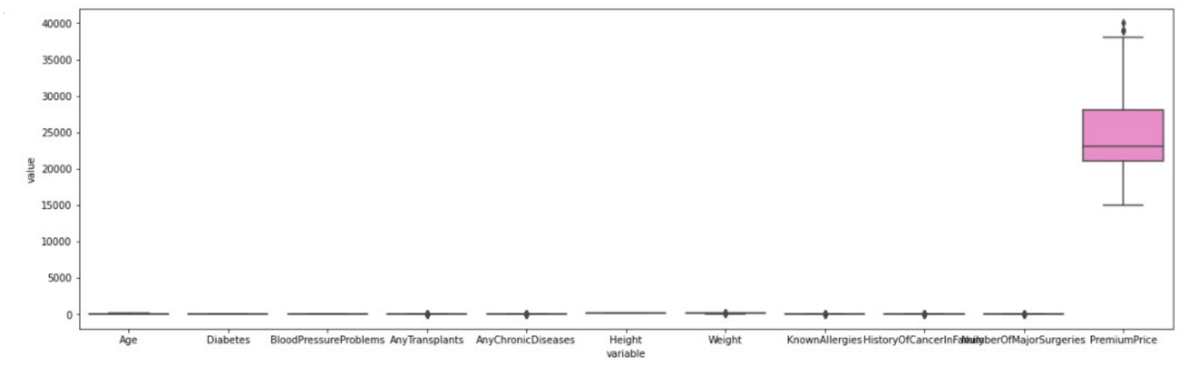
```
dataset.isnull().sum()
```

```
Age                        0
Diabetes                   0
BloodPressureProblems      0
AnyTransplants             0
AnyChronicDiseases         0
Height                     0
Weight                     0
KnownAllergies             0
HistoryOfCancerInFamily    0
NumberOfMajorSurgeries     0
PremiumPrice               0
dtype: int64
```

Checking for Outliers:



```
plt.figure(figsize=(20, 6))
sns.boxplot(x="variable", y="value", data=pd.melt(dataset))

plt.show()
```

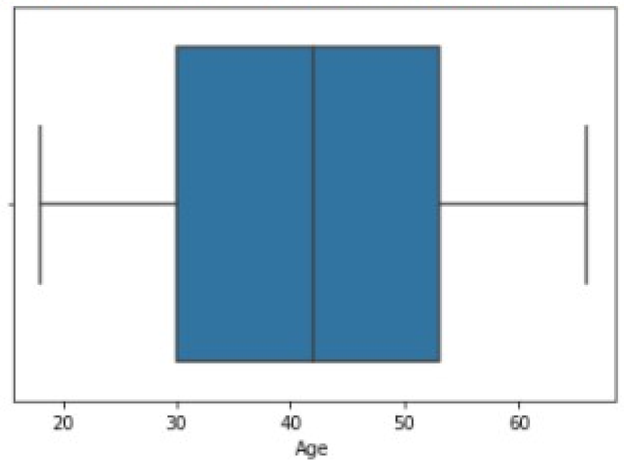


Observation: From above figure the box plot is not clear for all features except for Premium Price. Let's check for other 3 features i.e. for Age, Weight and Height because these have more value range as other features are just binary.

Code and Results:

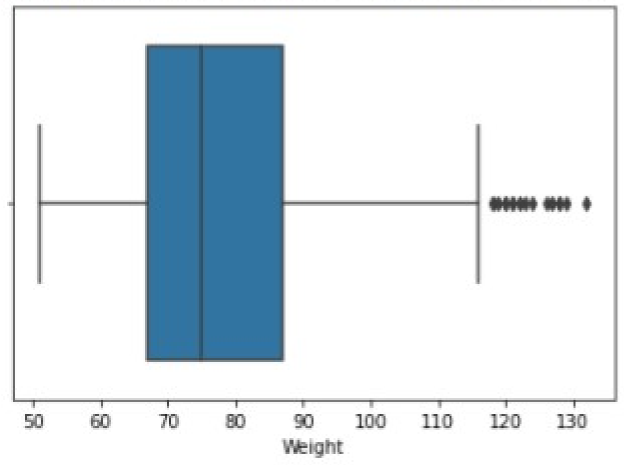
```
] sns.boxplot(x="Age", data=dataset)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff81eced350>
```



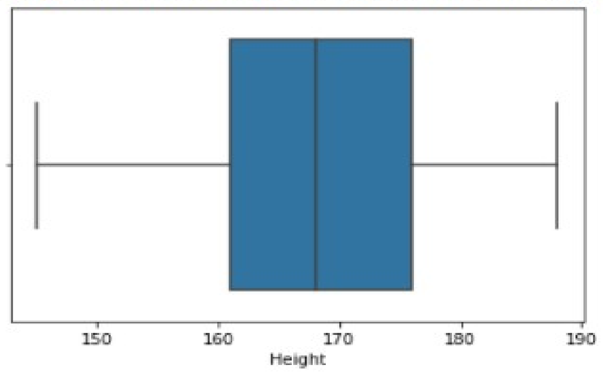
```
sns.boxplot(x="Weight", data=dataset)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff81e70e250>
```



```
sns.boxplot(x="Height", data=dataset)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff81e637390>
```



Observation:
Looking at above figures the weight feature has lot of outliers. Let's apply Log transformation. Because we have less number of rows hence we cannot remove these outliers which will affect shape of the dataset. This Log transformation will transform data to normal or close to normal.

Code and Results:

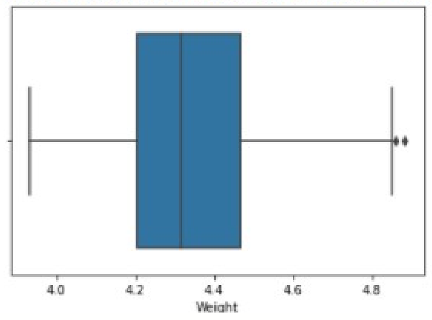
```
] dataset["Weight"]= np.log(dataset["Weight"])
```

```
] dataset["Weight"].head()
```

```
0    4.043051
1    4.290459
2    4.077537
3    4.532599
4    4.477337
Name: Weight, dtype: float64
```

```
]
  sns.boxplot(x="Weight", data=dataset)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff81e58fc50>
```



```
dataset["PremiumPrice"].value_counts()
```

```
23000    249
15000    202
28000    132
25000    103
29000     72
30000     47
35000     41
38000     34
31000     31
21000     26
19000     15
26000      7
39000      5
32000      4
24000      4
16000      3
36000      2
18000      2
34000      2
27000      1
22000      1
20000      1
17000      1
40000      1
Name: PremiumPrice, dtype: int64
```

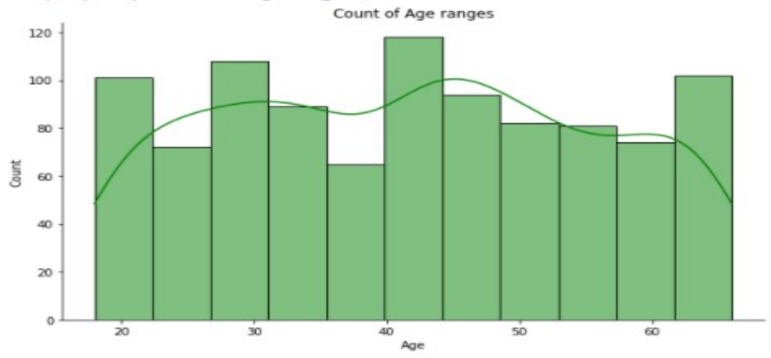
Observation:
From the above figure we can see that the number of outlier present in the weight feature is reduc ed , For premium price there few points which are not in the range because from above figure it is clear that for few these prices there are no sufficient number of samples to say about these prices. These cannot be altered. Only way is to increase the number of samples for these prices.

Data Visualization:

Code and Result:

```
sns.displot(x='Age',data=dataset,kde=True,aspect=10/6,color="g")
plt.title("Count of Age ranges")

#distplot is distribution plot
#x= variable That specifies x axis position
#data is the dataset name which has all the variables
#kde: A kernel density estimate which is analagous to a histogram.
#aspect is the size of figure
#color is parameter which mentions color  given to plot
```

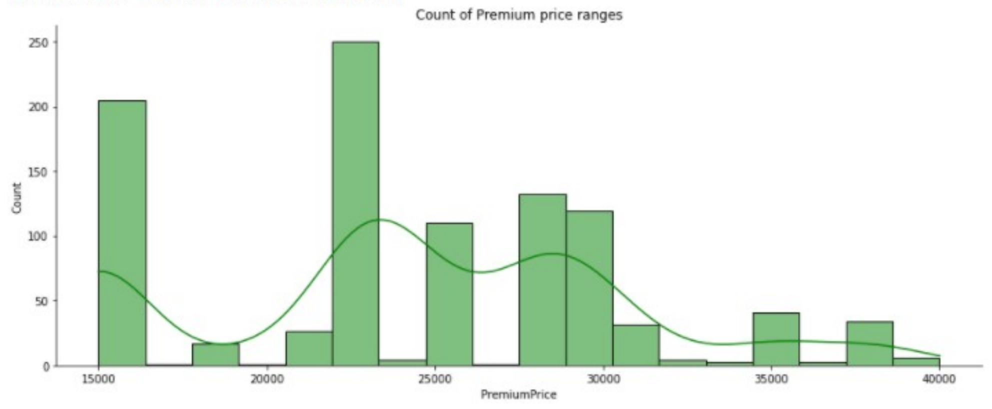
```
Text(0.5, 1.0, 'Count of Age ranges')
```



Observation:
From the above figure we can say the maximum nunmber of people who are paying premium are t hose who have age between 40 to 50 years. In a same way people who have age between 35 to 40 are those who are less number of premiums.

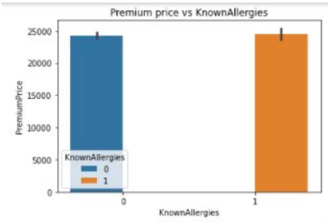
```
sns.displot(x='PremiumPrice',data=dataset,kde=True,aspect=15/6,color="g")
plt.title("Count of Premium price ranges")
#distplot is distribution plot
#x= variable That specifies x axis position
#data is the dataset name which has all the variables
#kde: A kernel density estimate which is analagous to a histogram.
#aspect is the size of figure
#color is parameter which mentions color  given to plot
```

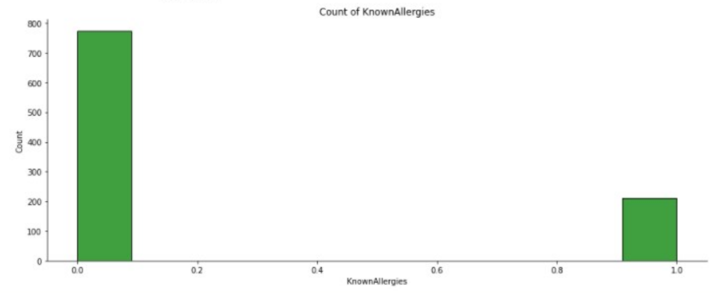
```
Text(0.5, 1.0, 'Count of Premium price ranges')
```



Observation:
From above fig we can approximately say that the more number of people are paying the premiu m amount between 23500 to 24500. And very few members are paying the premium amount of 20000 and around 27000.

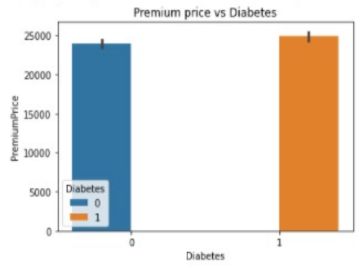
```
sns.barplot(x='KnownAllergies',y="PremiumPrice",data=dataset,hue='KnownAllergies')

plt.title("Premium price vs KnownAllergies")

sns.displot(x='KnownAllergies',data=dataset,aspect=15/6,color="g")
plt.title("Count of KnownAllergies")
```

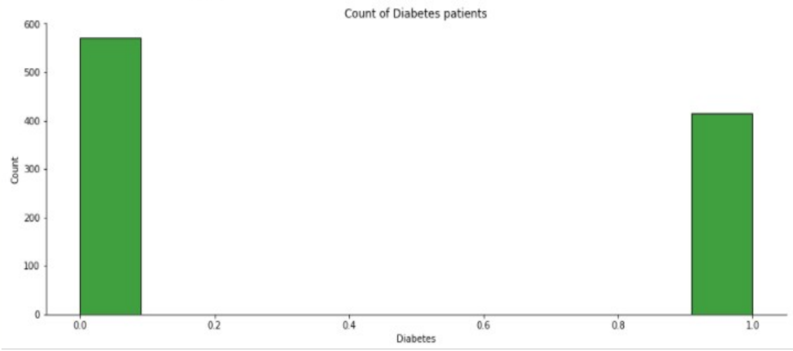




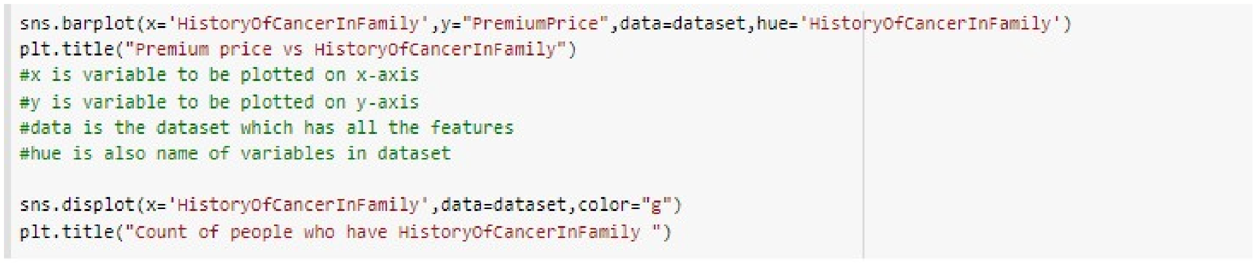
Observation: From above 2 figures we can say that the Knownallergy feature is not much effecting to decide the premium price as both like people who have allergy and who don't have allergy are paying almost same premium Price. But It is very clear More people who are doesn't have any Knownallergy are paying the premium amount.

```
sns.barplot(x='Diabetes',y="PremiumPrice",data=dataset,hue='Diabetes')
plt.title("Premium price vs Diabetes")
#x is variable to be plotted on x-axis
#y is variable to be plotted on y-axis
#data is the dataset which has all the features
#hue is also name of variables in dataset

sns.displot(x='Diabetes',data=dataset,aspect=15/6,color="g")
plt.title("Count of Diabetes patients")
```

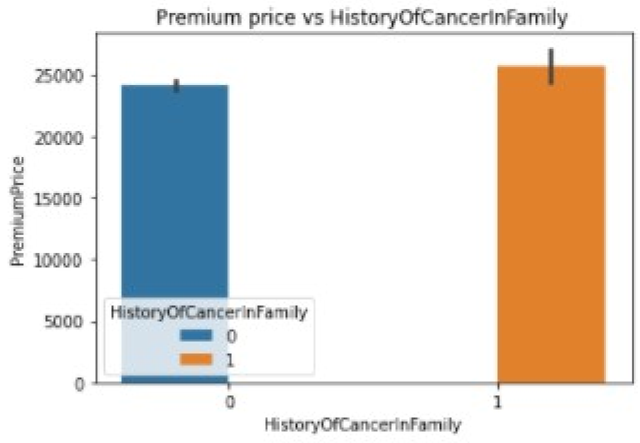


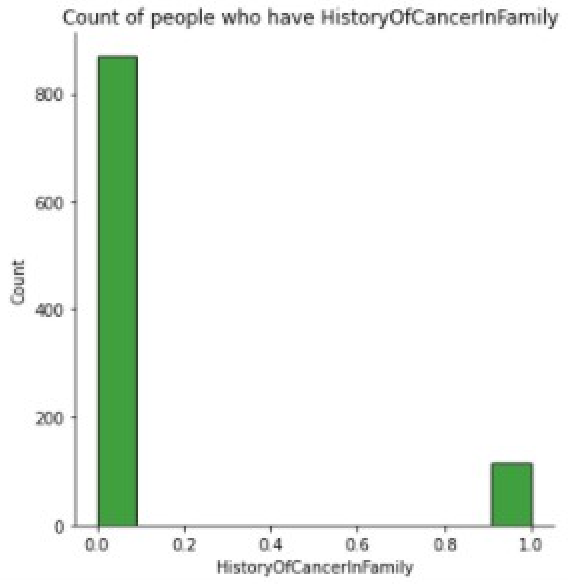


Observation:
From above figure we can say also that the Diabetes feature is not effecting very much to decide the premium price but there is very slight difference as people who have diabetes are taking littl e high premium amount as compared to people who doesn't have diabetes. But there is no much difference. But from second figure it is clear that more number of people who don't have diabetes are paying premiums.



```
sns.barplot(x='HistoryOfCancerInFamily',y="PremiumPrice",data=dataset,hue='HistoryOfCancerInFamily')
plt.title("Premium price vs HistoryOfCancerInFamily")
#x is variable to be plotted on x-axis
#y is variable to be plotted on y-axis
#data is the dataset which has all the features
#hue is also name of variables in dataset

sns.displot(x='HistoryOfCancerInFamily',data=dataset,color="g")
plt.title("Count of people who have HistoryOfCancerInFamily ")
```

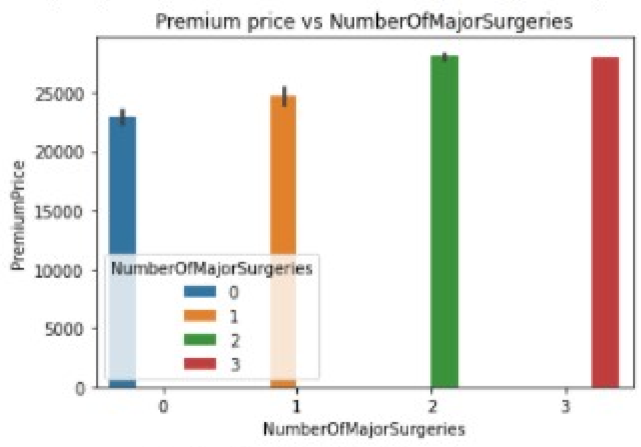


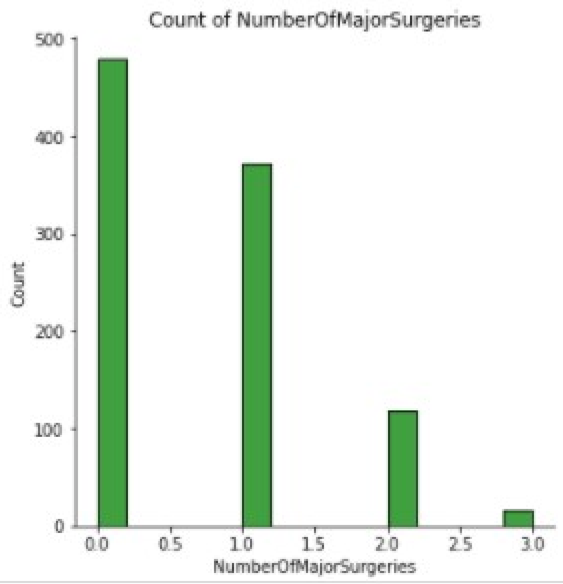


Observation:

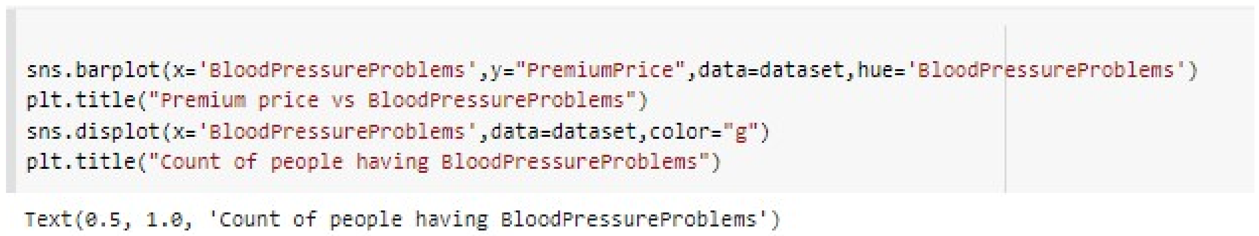
From above figures we can say also that the HistoryOfCancerInFamily feature is not effecting very much to decide the premium price but there is very slight difference as people who have HistoryOfCancerInFamily are taking little high premium amount as compared to people who doesn't have HistoryOfCancerInFamily. But there is no much difference. But from second figure it is clear that very high number of people who don't have HistoryOfCancerInFamily are paying premiums.

```
sns.barplot(x='NumberOfMajorSurgeries',y="PremiumPrice",data=dataset,hue='NumberOfMajorSurgeries')
plt.title("Premium price vs NumberOfMajorSurgeries")
#x is variable to be plotted on x-axis
#y is variable to be plotted on y-axis
#data is the dataset which has all the features
#hue is also name of variables in dataset
sns.displot(x='NumberOfMajorSurgeries',data=dataset,color="g")
plt.title("Count of NumberOfMajorSurgeries")
```



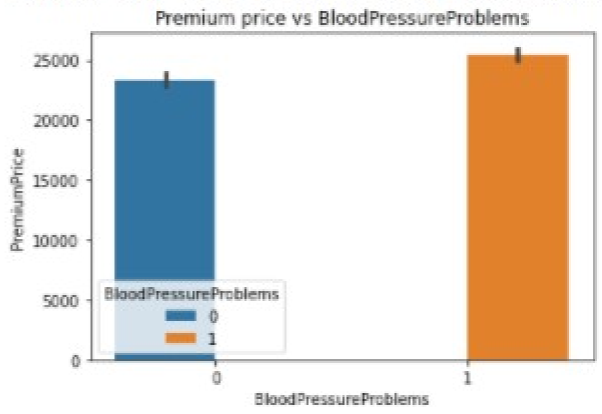


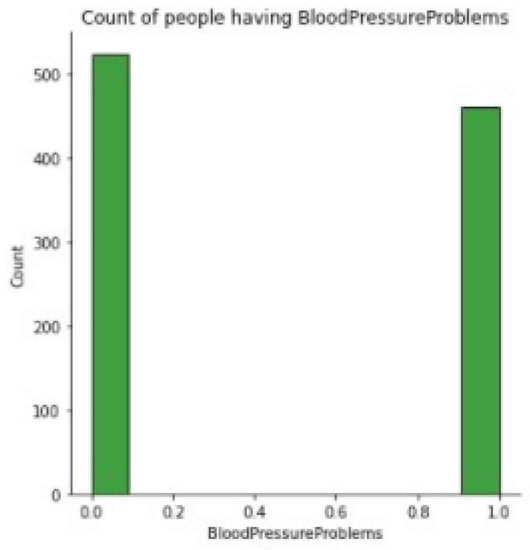
Observation:
From above figures we can say that the people who have 2 or 3 major surgeries are paying the premium amount of more than 25000. But from second figure it is very clear that most of the people who have 0 number surgeries are paying the premium.



```
sns.barplot(x='BloodPressureProblems',y="PremiumPrice",data=dataset,hue='BloodPressureProblems')
plt.title("Premium price vs BloodPressureProblems")
sns.displot(x='BloodPressureProblems',data=dataset,color="g")
plt.title("Count of people having BloodPressureProblems")
```

```
Text(0.5, 1.0, 'Count of people having BloodPressureProblems')
```



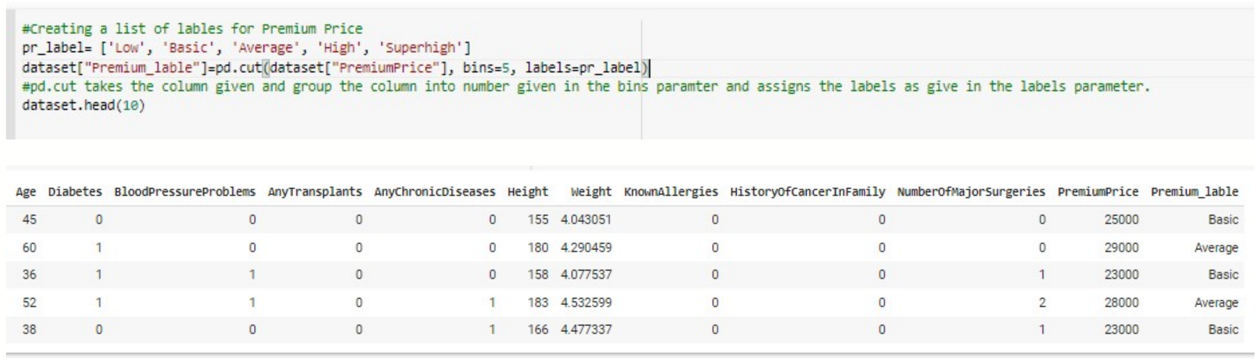


Observation:
From above figures we can say also that the BloodPressureProblems feature is not effecting very much to decide the premium price but there is very slight difference as people who have BloodPressureProblems are paying little high premium amount as compared to people who doesn't have BloodPressureProblems. But there is no much difference. But from second figure it is clear that slightly higher numbers of people who don't have BloodPressureProblems are paying premiums.

From above all the figures we can see that we create other features by using Age and Premium price. Because remaining features are not much effecting the premium price.

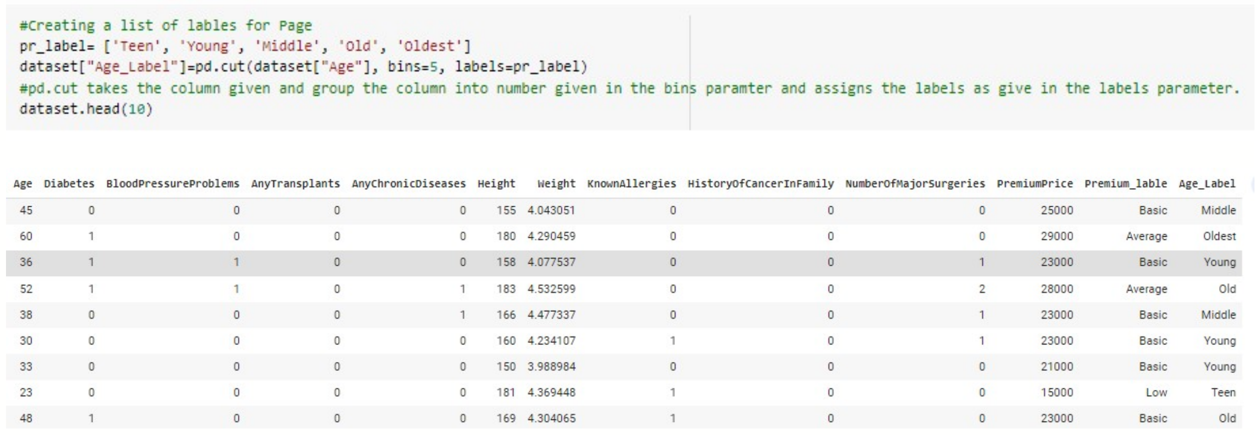
Features are created by dividing age and premium price columns into groups like Age columns is divided as Teen, Young, Middle, Old, and Oldest and Premium Price is divided like kind of salary bins such as Low, Basic, Average, High, Super high.

Code and Result:



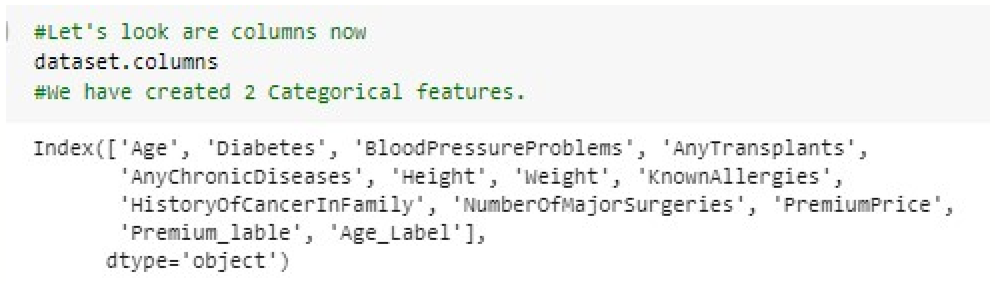
```
#Creating a list of lables for Premium Price
pr_label= ['Low', 'Basic', 'Average', 'High', 'Superhigh']
dataset["Premium_lable"]=pd.cut(dataset["PremiumPrice"], bins=5, labels=pr_label)
#pd.cut takes the column given and group the column into number given in the bins paramter and assigns the labels as give in the labels parameter.
dataset.head(10)
```

Age	Diabetes	BloodPressureProblems	AnyTransplants	AnyChronicDiseases	Height	Weight	KnownAllergies	HistoryOfCancerInFamily	NumberOfMajorSurgeries	PremiumPrice	Premium_lable
45	0	0	0	0	155	4.043051	0	0	0	25000	Basic
60	1	0	0	0	180	4.290459	0	0	0	29000	Average
36	1	1	0	0	158	4.077537	0	0	1	23000	Basic
52	1	1	0	1	183	4.532599	0	0	2	28000	Average
38	0	0	0	1	166	4.477337	0	0	1	23000	Basic



```
#Creating a list of lables for Page
pr_label= ['Teen', 'Young', 'Middle', 'Old', 'Oldest']
dataset["Age_Label"]=pd.cut(dataset["Age"], bins=5, labels=pr_label)
#pd.cut takes the column given and group the column into number given in the bins paramter and assigns the labels as give in the labels parameter.
dataset.head(10)
```

Age	Diabetes	BloodPressureProblems	AnyTransplants	AnyChronicDiseases	Height	Weight	KnownAllergies	HistoryOfCancerInFamily	NumberOfMajorSurgeries	PremiumPrice	Premium_lable	Age_Label
45	0	0	0	0	155	4.043051	0	0	0	25000	Basic	Middle
60	1	0	0	0	180	4.290459	0	0	0	29000	Average	Oldest
36	1	1	0	0	158	4.077537	0	0	1	23000	Basic	Young
52	1	1	0	1	183	4.532599	0	0	2	28000	Average	Old
38	0	0	0	1	166	4.477337	0	0	1	23000	Basic	Middle
30	0	0	0	0	160	4.234107	1	0	1	23000	Basic	Young
33	0	0	0	0	150	3.988984	0	0	0	21000	Basic	Young
23	0	0	0	0	181	4.369448	1	0	0	15000	Low	Teen
48	1	0	0	0	169	4.304065	1	0	0	23000	Basic	Old



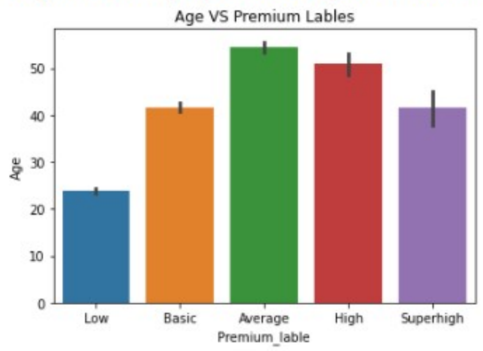
```
#Let's look are columns now
dataset.columns
#We have created 2 Categorical features.
```

```
Index(['Age', 'Diabetes', 'BloodPressureProblems', 'AnyTransplants',
       'AnyChronicDiseases', 'Height', 'Weight', 'KnownAllergies',
       'HistoryOfCancerInFamily', 'NumberOfMajorSurgeries', 'PremiumPrice',
       'Premium_lable', 'Age_Label'],
      dtype='object')
```

Before Going for Next process lets Visualize these bins which are created.

```
plt.title("Age VS Premium Lables")
sns.barplot(x='Premium_lable',y="Age",data=dataset)
#x is variable to be plotted on x-axis
#y is variable to be plotted on y-axis
#data is the dataset which has all the features
```

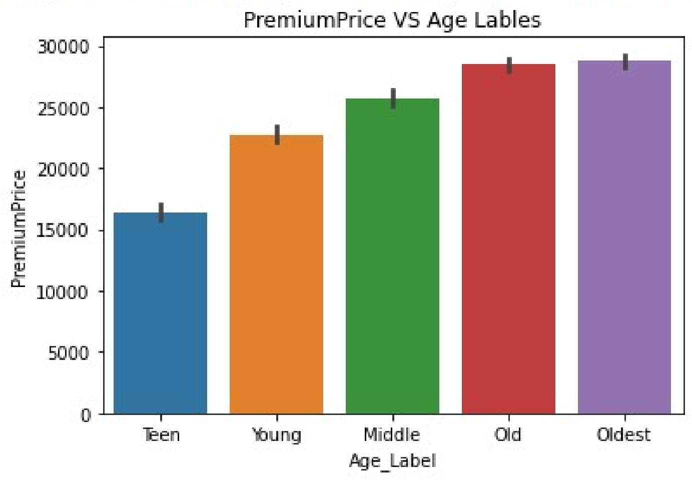
```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff81e4fb710>
```



Observation: From the above figure we can say that the premium price that is categorized in the Low label were taken by the people who have age around 22 and that is the least as compared to other labels. And Most of the people who have age around 50+ are paying the premium price that are grouped under Average label.

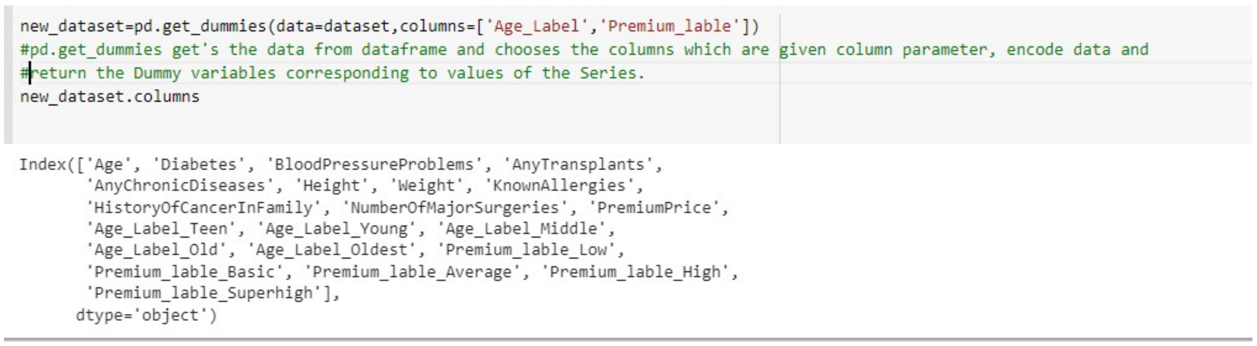
```
plt.title("PremiumPrice VS Age Lables")

sns.barplot(x='Age_Label',y="PremiumPrice",data=dataset)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff81e4d22d0>
```



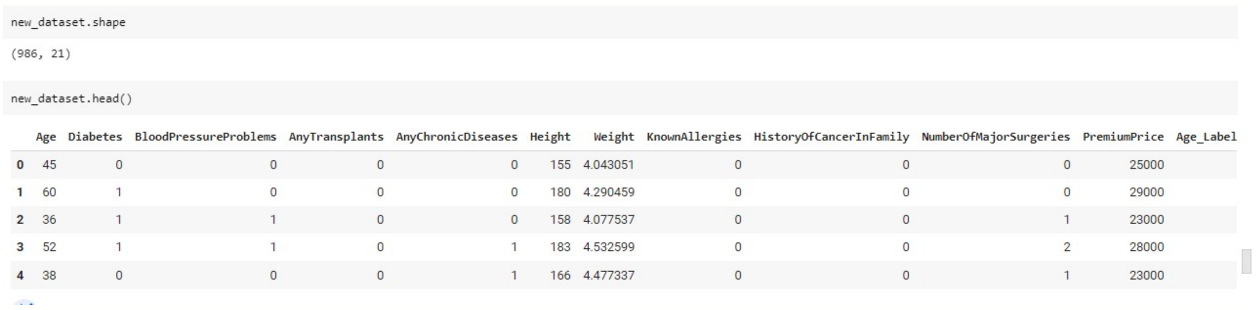
Observation:
From above figure it's very clear that most of the old and oldest people are paying more premium price.

Now let’s convert these categorical data into numerical data as it is required for the further process.



```
new_dataset=pd.get_dummies(data=dataset,columns=['Age_Label','Premium_lable'])
#pd.get_dummies get's the data from dataframe and chooses the columns which are given column parameter, encode data and
#return the Dummy variables corresponding to values of the Series.
new_dataset.columns
```

```
Index(['Age', 'Diabetes', 'BloodPressureProblems', 'AnyTransplants',
       'AnyChronicDiseases', 'Height', 'Weight', 'KnownAllergies',
       'HistoryOfCancerInFamily', 'NumberOfMajorSurgeries', 'PremiumPrice',
       'Age_Label_Teen', 'Age_Label_Young', 'Age_Label_Middle',
       'Age_Label_Old', 'Age_Label_Oldest', 'Premium_lable_Low',
       'Premium_lable_Basic', 'Premium_lable_Average', 'Premium_lable_High',
       'Premium_lable_Superhigh'],
      dtype='object')
```



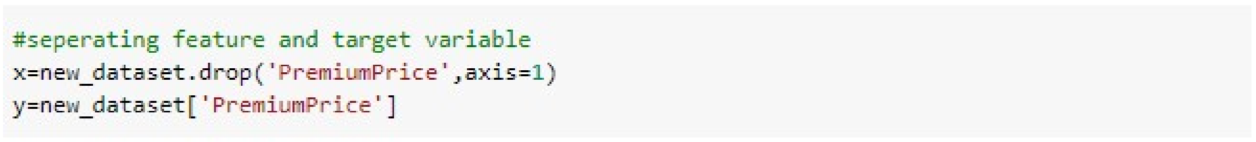
```
new_dataset.shape
```

```
(986, 21)
```

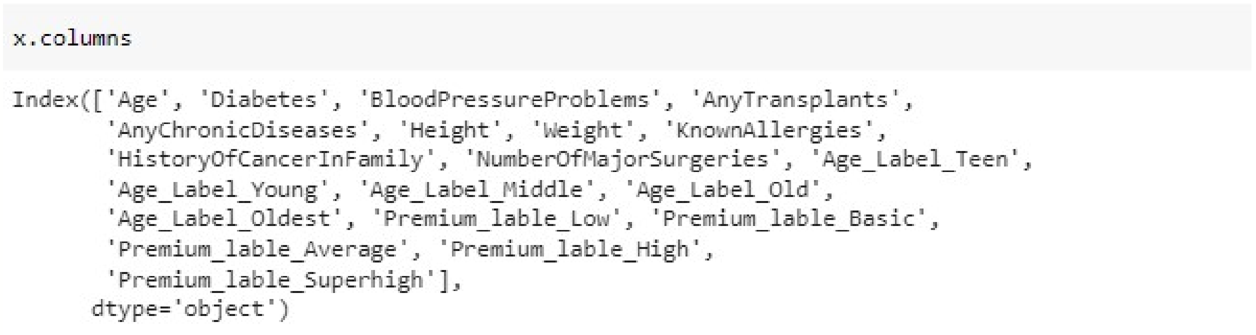
```
new_dataset.head()
```

	Age	Diabetes	BloodPressureProblems	AnyTransplants	AnyChronicDiseases	Height	Weight	KnownAllergies	HistoryOfCancerInFamily	NumberOfMajorSurgeries	PremiumPrice	Age_Label
0	45	0	0	0	0	155	4.043051	0	0	0	25000	
1	60	1	0	0	0	180	4.290459	0	0	0	29000	
2	36	1	1	0	0	158	4.077537	0	0	1	23000	
3	52	1	1	0	1	183	4.532599	0	0	2	28000	
4	38	0	0	0	1	166	4.477337	0	0	1	23000	

Data Splitting and Scaling



```
#seperating feature and target variable
x=new_dataset.drop('PremiumPrice',axis=1)
y=new_dataset['PremiumPrice']
```

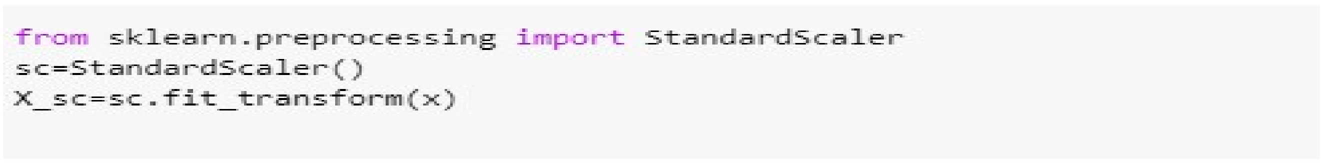


```
x.columns
```

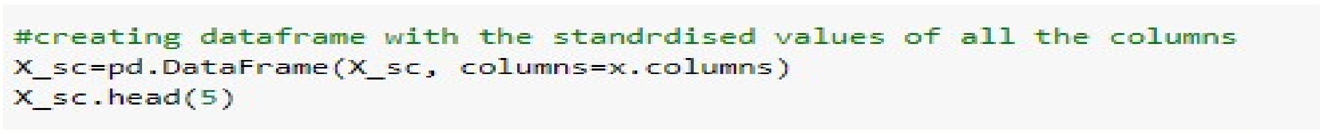
```
Index(['Age', 'Diabetes', 'BloodPressureProblems', 'AnyTransplants',
       'AnyChronicDiseases', 'Height', 'Weight', 'KnownAllergies',
       'HistoryOfCancerInFamily', 'NumberOfMajorSurgeries', 'Age_Label_Teen',
       'Age_Label_Young', 'Age_Label_Middle', 'Age_Label_Old',
       'Age_Label_Oldest', 'Premium_lable_Low', 'Premium_lable_Basic',
       'Premium_lable_Average', 'Premium_lable_High',
       'Premium_lable_Superhigh'],
      dtype='object')
```

As we can observe from the above columns that the measuring scale of all the columns are differ ent like age is measures as years, weight is measured as kg's but these are replaced by log values as it had outliers. And height is measured as centi meters. If we consider as it is this may effect th e modelling.
The **standardising** the data means it is the process of rescaling the attributes so that they have m ean as 0 and variance as 1.The ultimate goal to perform standardization is to bring down all the features to a common scale without distorting the differences in the range of the values.

Code and Result:



```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
X_sc=sc.fit_transform(x)
```



```
#creating dataframe with the standrdised values of all the columns
X_sc=pd.DataFrame(X_sc, columns=x.columns)
X_sc.head(5)
```

Modeling:

Initially data is trained and tested with 7 Regression models by giving default parameters.

What is **Regression** and when it is used?

Regression analysis is a form of predictive modeling technique which is used to find the relationship between the features and target variable. In this dataset the target variable has continuous values hence regression technique is used.

In this project total 6 Models are used because to compare the results of all these regression models so that we can test the data set on all the models instead just checking with anyone model. The hyper parameter tuning is done to that model which has good results and no overfitting.

Models are:

1. Linear Regression

2. XGBRegressor

3. RandomForestRegressor

4. ExtraTreeRegressor

5. SVR

6. KNeighborsRegressor

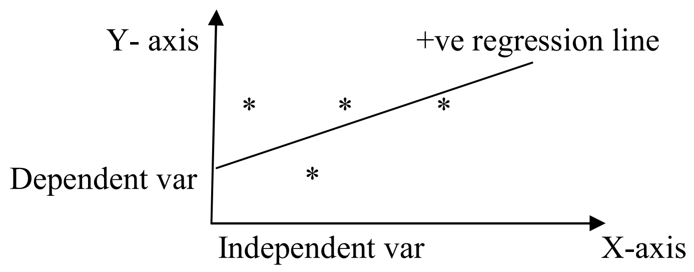
Before going to coding part let’s just understand little theory behind all the models.

Linear Regression

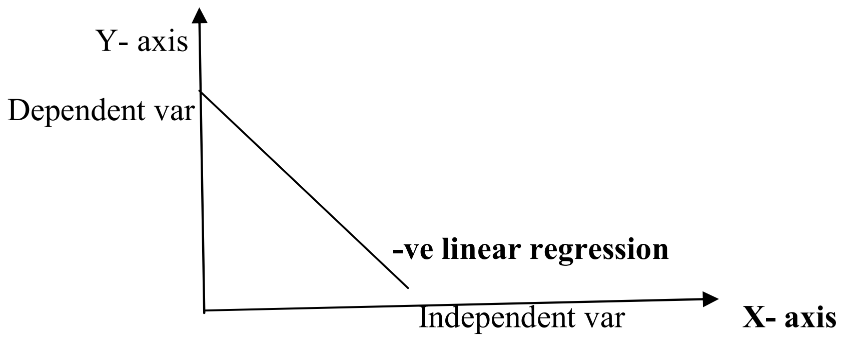
Theory: We can say that linear regression is linear model for example a model that assumes a linear relationship between the input variables or features and Target variables. If the data is free from the outliers the models works well and linear regression is not computationally expensive as compared to Decision Tree or any clustering algorithm.

Understanding of Linear Regression

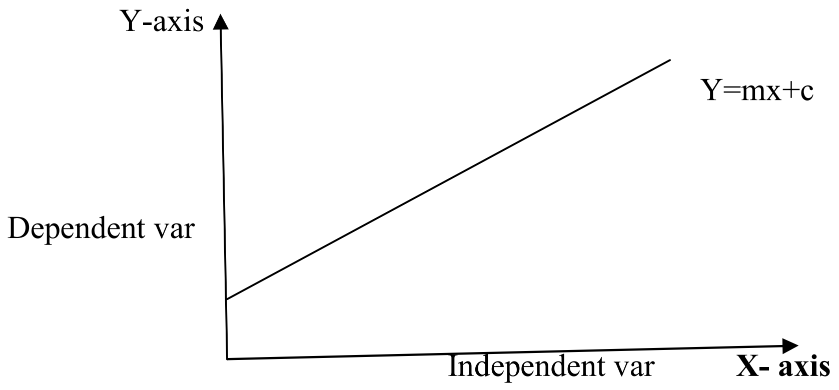
- Suppose we have a graph where on X- axis we have Independent variable and on Y- axis we have dependent variable. Suppose we have data point on X- axis which is increasing and if a data point on the Y- axis is also increasing we get a positive Linear Regression



- But If in case we have data point on X- axis which is increasing and if a data point on the Y- axis is i.e. on dependent variable is decreasing then we get a Negative Linear Regression

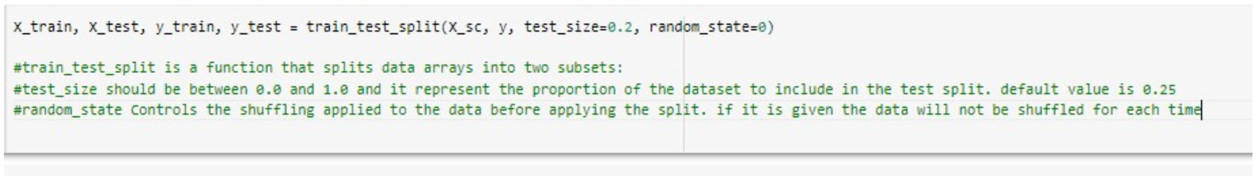


- The line y=mx+c which is the line of regression. This line is found such a way the it fits all the data point.

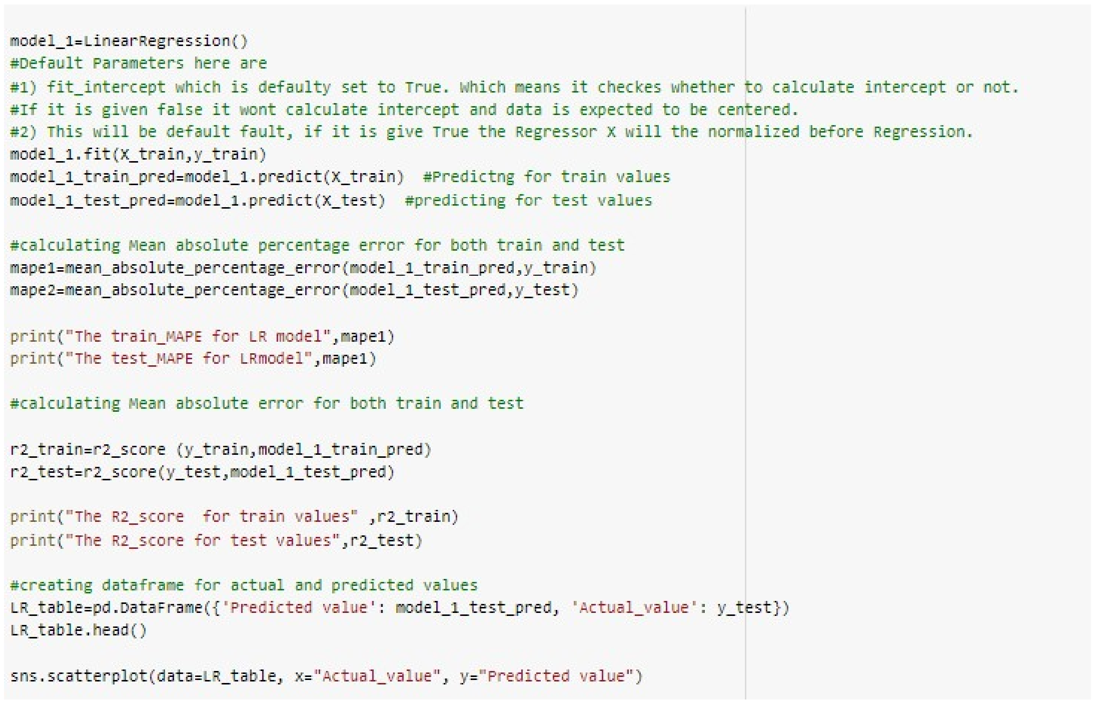


Linear regression will try to find the line which will best fit all the points and reduce the error between the actual and predicted values. And as per our data set the correlation between features and target variable is quite good. Let's look at results later.

Code and Results:

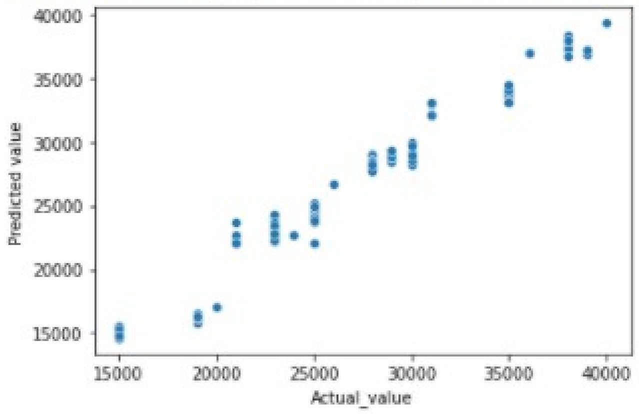


```
X_train, X_test, y_train, y_test = train_test_split(X_sc, y, test_size=0.2, random_state=0)

#train_test_split is a function that splits data arrays into two subsets:
#test_size should be between 0.0 and 1.0 and it represent the proportion of the dataset to include in the test split. default value is 0.25
#random_state Controls the shuffling applied to the data before applying the split. if it is given the data will not be shuffled for each time
```



```
model_1=LinearRegression()
#Default Parameters here are
#1) fit_intercept which is defaulty set to True. Which means it checkes whether to calculate intercept or not.
#If it is given false it wont calculate intercept and data is expected to be centered.
#2) This will be default fault, if it is give True the Regressor X will the normalized before Regression.
model_1.fit(X_train,y_train)
model_1_train_pred=model_1.predict(X_train)  #Predictng for train values
model_1_test_pred=model_1.predict(X_test)  #predicting for test values

#calculating Mean absolute percentage error for both train and test
mape1=mean_absolute_percentage_error(model_1_train_pred,y_train)
mape2=mean_absolute_percentage_error(model_1_test_pred,y_test)

print("The train_MAPE for LR model",mape1)
print("The test_MAPE for LRmodel",mape1)

#calculating Mean absolute error for both train and test

r2_train=r2_score (y_train,model_1_train_pred)
r2_test=r2_score(y_test,model_1_test_pred)

print("The R2_score  for train values" ,r2_train)
print("The R2_score for test values",r2_test)

#creating dataframe for actual and predicted values
LR_table=pd.DataFrame({'Predicted value': model_1_test_pred, 'Actual_value': y_test})
LR_table.head()

sns.scatterplot(data=LR_table, x="Actual_value", y="Predicted value")
```

```
The train_MAPE for LR model 0.026659463324349087
The test_MAPE for LRmodel 0.026659463324349087
The R2_score  for train values 0.9799012205433837
The R2_score for test values 0.9778505834282073
<matplotlib.axes._subplots.AxesSubplot at 0x7fcda047d390>
```



```
LR_table.head()
```

	Predicted value	Actual_value
231	22801.639556	23000
688	23198.621342	23000
27	28347.240925	28000
366	23996.151979	23000
715	37236.015958	38000

From above output we can say that the result is not good because for the few prices the predicting value is not even closer to it. This is not right model for this dataset.

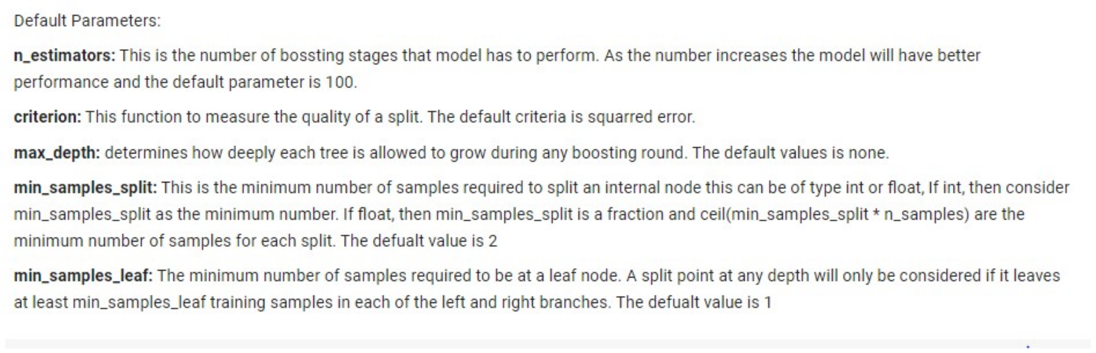
Let's Move on to next model

RandomForestRegressor

Theory:

Random forest is a type of supervised learning algorithm that uses ensemble methods (bagging) **to** solve both Regressor and classification problem. The algorithm operates by constructing a multiple of decision trees at training time and outputting the mean/mode of prediction of the individual trees. The difference between Random forest and Extra tree Regressor is that Random Forest chooses the optimum split while Extra Trees chooses it randomly. Which means Random forest uses bootstrap replicas**,** that is to say, it subsamples the input data with replacement, whereas Extra Trees use the whole original sample. However, once the split points are selected, the two algorithms choose the best one between all the subset of features. In terms of computational and execution time Extra Tree Regressor algorithm is faster.

Parameters:



Default Parameters:

n_estimators: This is the number of bossting stages that model has to perform. As the number increases the model will have better performance and the default parameter is 100.

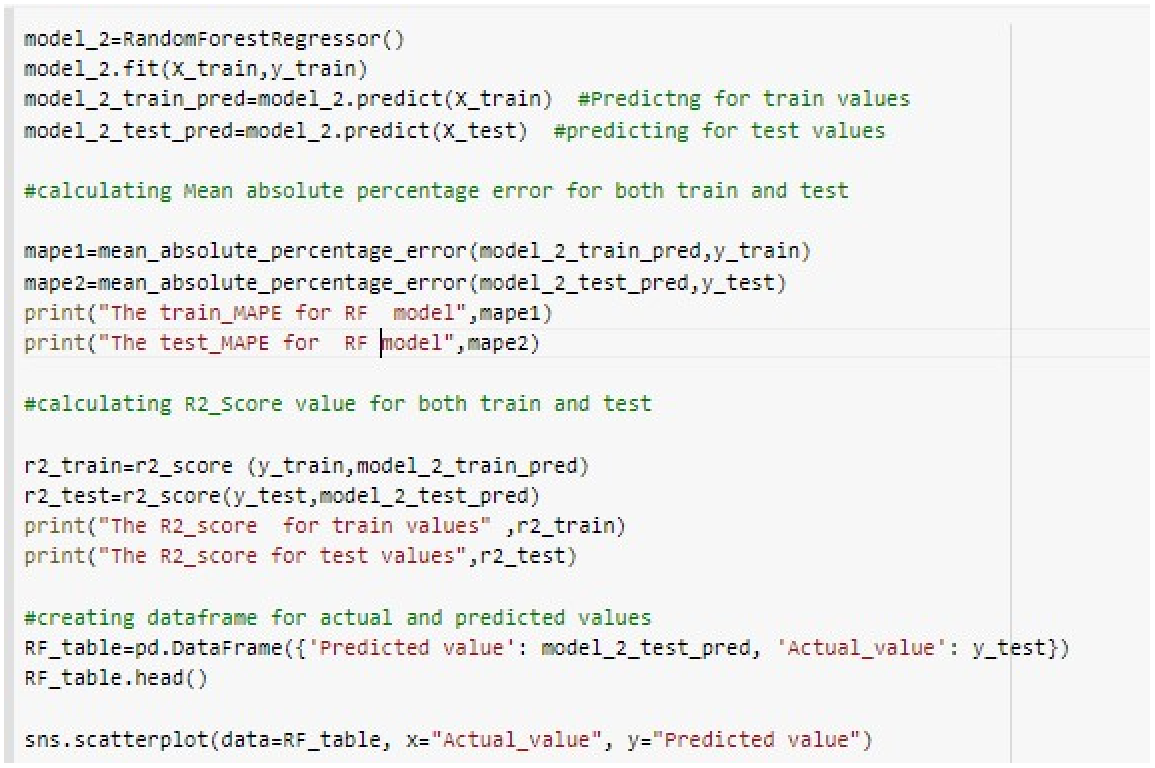
criterion: This function to measure the quality of a split. The default criteria is squarred error.

max_depth: determines how deeply each tree is allowed to grow during any boosting round. The default values is none.

min_samples_split: This is the minimum number of samples required to split an internal node this can be of type int or float, If int, then consider min_samples_split as the minimum number. If float, then min_samples_split is a fraction and ceil(min_samples_split * n_samples) are the minimum number of samples for each split. The defualt value is 2

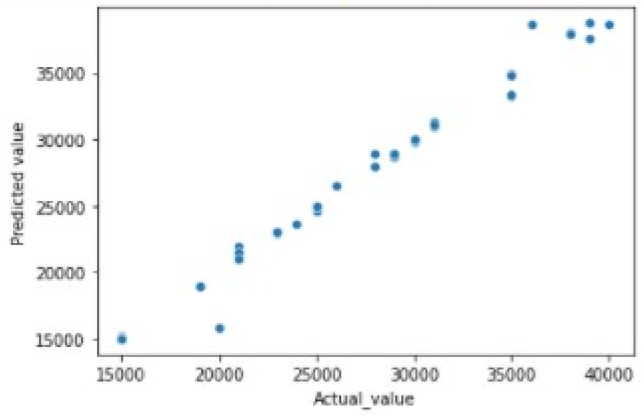
min_samples_leaf: The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least min_samples_leaf training samples in each of the left and right branches. The defualt value is 1

Code and Results:



```
model_2=RandomForestRegressor()
model_2.fit(X_train,y_train)
model_2_train_pred=model_2.predict(X_train)  #Predictng for train values
model_2_test_pred=model_2.predict(X_test)  #predicting for test values

#calculating Mean absolute percentage error for both train and test

mape1=mean_absolute_percentage_error(model_2_train_pred,y_train)
mape2=mean_absolute_percentage_error(model_2_test_pred,y_test)
print("The train_MAPE for RF  model",mape1)
print("The test_MAPE for  RF model",mape2)

#calculating R2_Score value for both train and test

r2_train=r2_score (y_train,model_2_train_pred)
r2_test=r2_score(y_test,model_2_test_pred)
print("The R2_score  for train values" ,r2_train)
print("The R2_score for test values",r2_test)

#creating dataframe for actual and predicted values
RF_table=pd.DataFrame({'Predicted value': model_2_test_pred, 'Actual_value': y_test})
RF_table.head()

sns.scatterplot(data=RF_table, x="Actual_value", y="Predicted value")
```

```
The train_MAPE for RF  model 0.0016469454839011914
The test_MAPE for  RF model 0.0041058333510692
The R2_score  for train values 0.9993784375767258
The R2_score for test values 0.9951714805541069
<matplotlib.axes._subplots.AxesSubplot at 0x7fcda054e590>
```



```
RF_table.head(8)
```

	Predicted value	Actual_value
231	23000.0	23000
688	23000.0	23000
27	28000.0	28000
366	23000.0	23000
715	37860.0	38000
482	23000.0	23000

From the above the result seems very good as MAPE value is 0.004 and r2_score is 0.99 but let's not conclude here we will check at other models as well.

Let's move on to next model

XGBRegressor

Theory: This model is ensemble boosting technique. Boosting is an ensemble technique where new models are added to correct the errors made by existing models. Models are added sequentially until no further improvements can be made. Gradient boosting is an approach where new models are created that predict the residuals or errors of prior models and then added together to make the final prediction. It is called gradient boosting because it uses a gradient descent algorithm to minimize the loss when adding new models. XGBoost is very fast when compared to implementation of other model techniques.

Parameters:

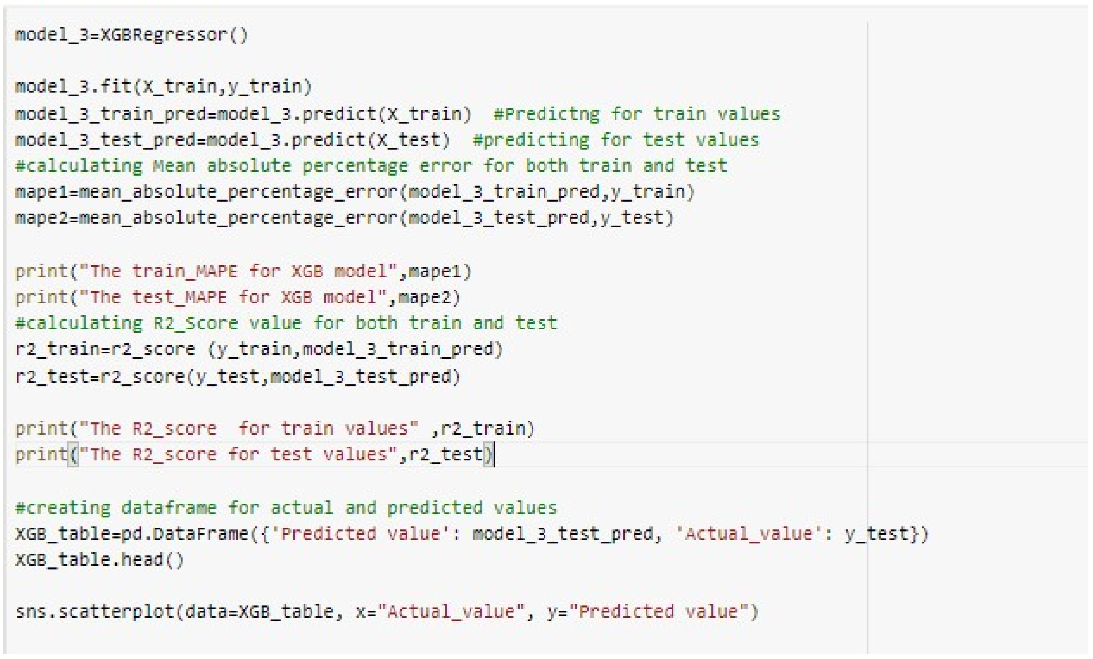
Default parameters:

Max_depth: determines how deeply each tree is allowed to grow during any boosting round. The default values is 3.

Learning Rate: The learning rate shrinks the conntribution of each tree by learning rate and the default will be 0.1

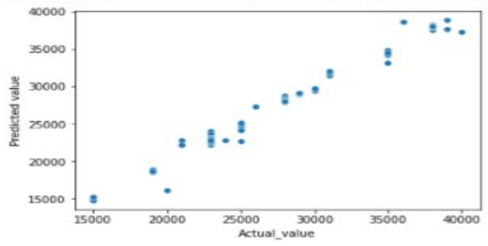
n_estimators: This is the number of bossting stages that model has to perform. As the number increases the model will have better performance and the default parameter is 100.

These are the main parameter as we still have several parameters. These parameters can be used for hyper parameter tuning.

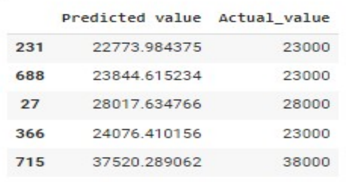


```
model_3=XGBRegressor()

model_3.fit(X_train,y_train)
model_3_train_pred=model_3.predict(X_train)  #Predictng for train values
model_3_test_pred=model_3.predict(X_test)  #predicting for test values
#calculating Mean absolute percentage error for both train and test
mape1=mean_absolute_percentage_error(model_3_train_pred,y_train)
mape2=mean_absolute_percentage_error(model_3_test_pred,y_test)

print("The train_MAPE for XGB model",mape1)
print("The test_MAPE for XGB model",mape2)
#calculating R2_Score value for both train and test
r2_train=r2_score (y_train,model_3_train_pred)
r2_test=r2_score(y_test,model_3_test_pred)

print("The R2_score  for train values" ,r2_train)
print("The R2_score for test values",r2_test)

#creating dataframe for actual and predicted values
XGB_table=pd.DataFrame({'Predicted value': model_3_test_pred, 'Actual_value': y_test})
XGB_table.head()

sns.scatterplot(data=XGB_table, x="Actual_value", y="Predicted value")
```

```
[09:29:40] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
The train_MAPE for LR model 0.012108936814987377
The test_MAPE for LRmodel 0.014500334050526484
The R2_score  for train values 0.9942167758050079
The R2_score for test values 0.9896762617564292
<matplotlib.axes._subplots.AxesSubplot at 0x7fcda060a710>
```



```
XGB_table.head()
```



	Predicted value	Actual_value
231	22773.984375	23000
688	23844.615234	23000
27	28017.634766	28000
366	24076.410156	23000
715	37520.289062	38000

The results are okay because this is better than the Linear Regression model but as Random Forest model is giving better result than this so this is not right model for this problem.

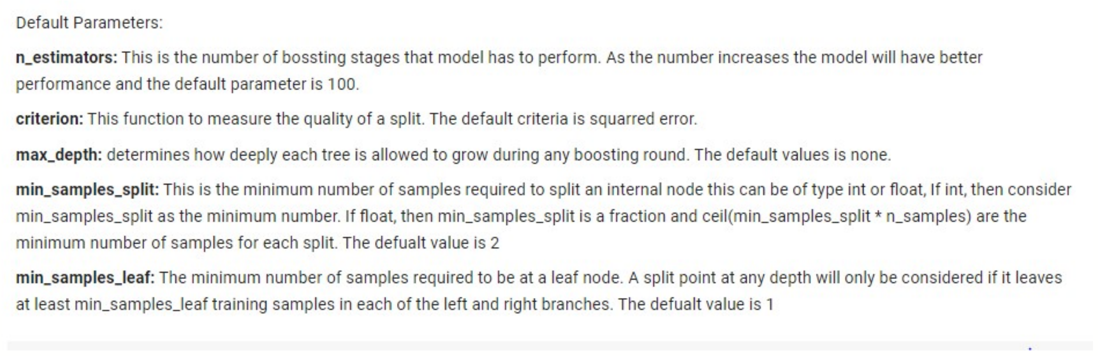
Let’s now move on to next model.

ExtraTreeRegressor:

Theory:

This class implements a Meta estimator that fits a number of randomized decision trees on various sub-samples of the dataset and use averaging to improve the predictive accuracy which will lead to minimize the error and also control over-fitting. The parameter here is same as the decision tree but one parameter which is added here is n_estimator as it asks for number of trees.

Parameter:



Default Parameters:

n_estimators: This is the number of bossting stages that model has to perform. As the number increases the model will have better performance and the default parameter is 100.

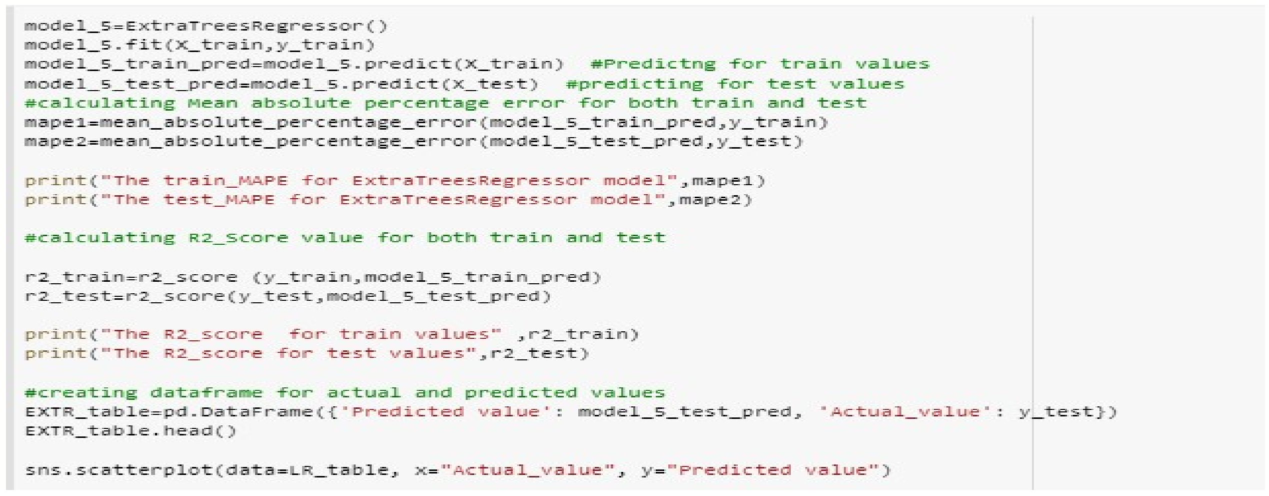
criterion: This function to measure the quality of a split. The default criteria is squarred error.

max_depth: determines how deeply each tree is allowed to grow during any boosting round. The default values is none.

min_samples_split: This is the minimum number of samples required to split an internal node this can be of type int or float, If int, then consider min_samples_split as the minimum number. If float, then min_samples_split is a fraction and ceil(min_samples_split * n_samples) are the minimum number of samples for each split. The defualt value is 2

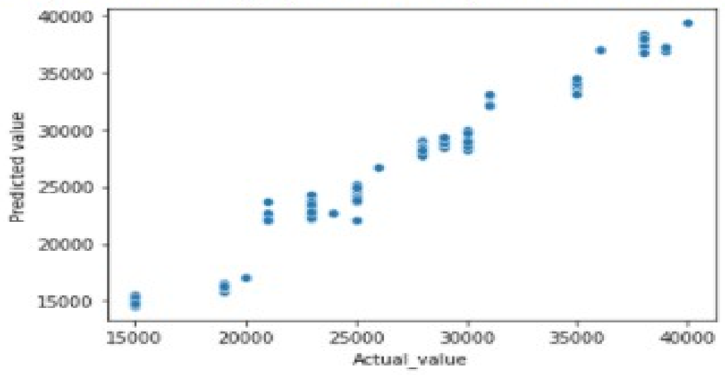
min_samples_leaf: The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least min_samples_leaf training samples in each of the left and right branches. The defualt value is 1

Code and Result:



```
model_5=ExtraTreesRegressor()
model_5.fit(X_train,y_train)
model_5_train_pred=model_5.predict(X_train)  #Predictng for train values
model_5_test_pred=model_5.predict(X_test)  #predicting for test values
#calculating Mean absolute percentage error for both train and test
mape1=mean_absolute_percentage_error(model_5_train_pred,y_train)
mape2=mean_absolute_percentage_error(model_5_test_pred,y_test)

print("The train_MAPE for ExtraTreesRegressor model",mape1)
print("The test_MAPE for ExtraTreesRegressor model",mape2)

#calculating R2_Score value for both train and test

r2_train=r2_score (y_train,model_5_train_pred)
r2_test=r2_score(y_test,model_5_test_pred)

print("The R2_score  for train values" ,r2_train)
print("The R2_score for test values",r2_test)

#creating dataframe for actual and predicted values
EXTR_table=pd.DataFrame({'Predicted value': model_5_test_pred, 'Actual_value': y_test})
EXTR_table.head()

sns.scatterplot(data=LR_table, x="Actual_value", y="Predicted value")
```

```
The train_MAPE for ExtraTreesRegressor model 0.0
The test_MAPE for ExtraTreesRegressor model 0.005571121759022764
The R2_score  for train values 1.0
The R2_score for test values 0.9936336576532947
<matplotlib.axes._subplots.AxesSubplot at 0x7fcd9ba41ad0>
```



```
EXTR_table.head()
```

	Predicted value	Actual_value
231	23000.0	23000
688	23100.0	23000
27	28000.0	28000
366	23150.0	23000
715	38000.0	38000

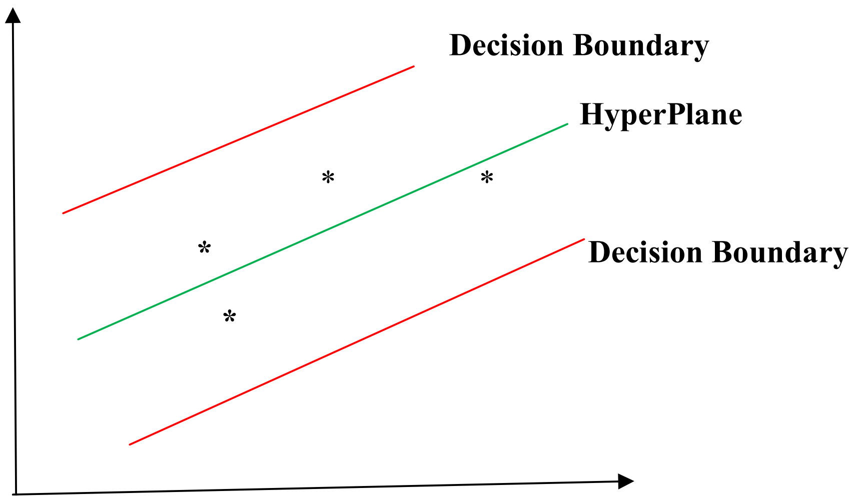
The results look perfect let’s just look at other model results. This model can be perfect suitable for this Problem.

Let’s now move on to next model.

Support Vector Regression (SVR)

Theory:

The idea behind SVR is to find a function that approximates mapping from an input domain to real numbers on the basis of a training sample. Let’s understand it by a example



Consider these two red lines as the decision boundary and the green line as the hyperplane. Our objective, when we are moving on with SVR, is to basically consider the points that are within the decision boundary line. Our best fit line is the hyperplane that has a maximum number of points.

The first thing that we'll understand is what is the decision boundary i.e. the danger red line as shown above. Consider these lines as being at any distance, say 'a', from the hyperplane. So, these are the lines that we draw at distance '+a' and '-a' from the hyperplane. This 'a' in the text is basically referred to as epsilon.

Assuming that the equation of the hyperplane is as follows:

$Y= wx+b$

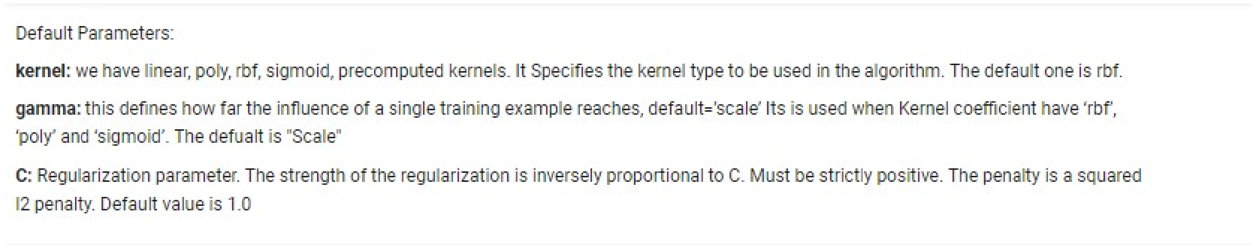
Then the equation of decision boundary becomes:

$wx+b=+a$ and $wx+b=-a$

Thus any hyperplane that satisfies our SVR should satisfy the equation $-a< Y-wx+b < +a$.

Our main aim here is to decide a decision boundary at 'a' distance from the original hyperplane such that data points closest to the hyperplane or the support vectors are within that boundary line. Hence, we are going to take only those points that are within the decision boundary and have the least error rate, or are within the Margin of Tolerance. This gives us a better fitting model.

Parameters:



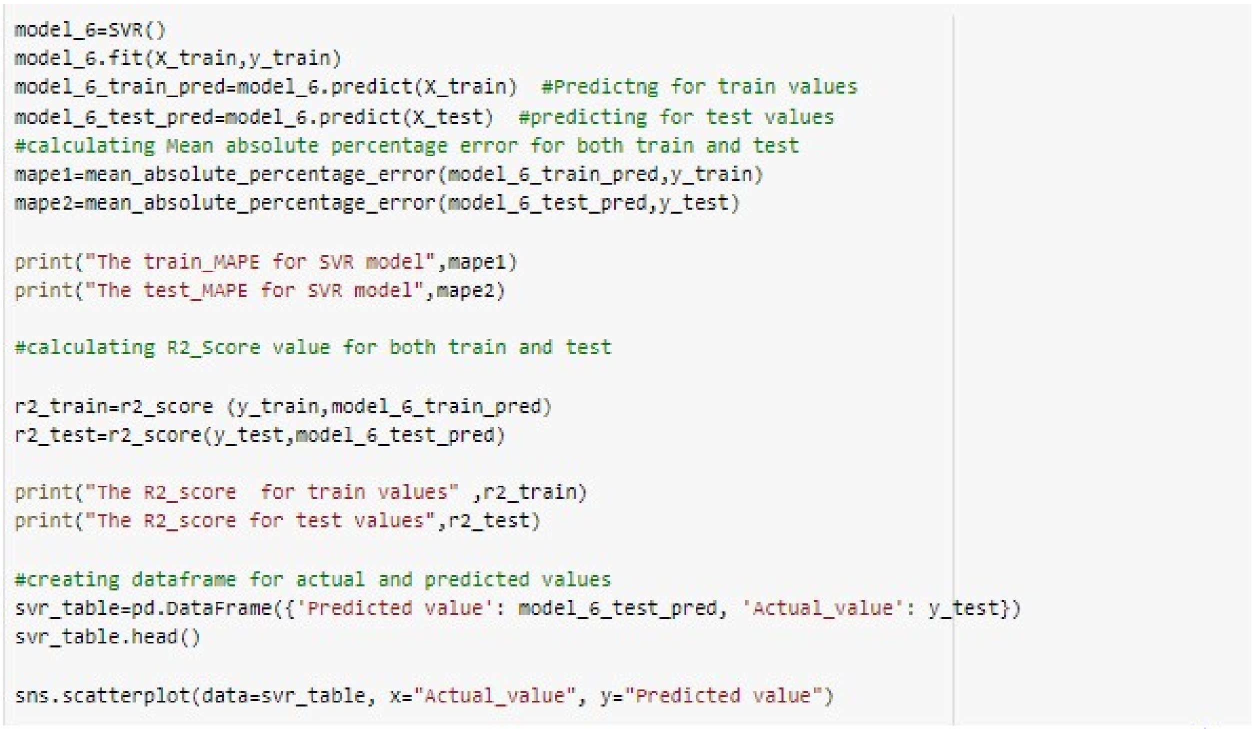
Default Parameters:

kernel: we have linear, poly, rbf, sigmoid, precomputed kernels. It Specifies the kernel type to be used in the algorithm. The default one is rbf.

gamma: this defines how far the influence of a single training example reaches, default='scale' Its is used when Kernel coefficient have 'rbf', 'poly' and 'sigmoid'. The defualt is "Scale"

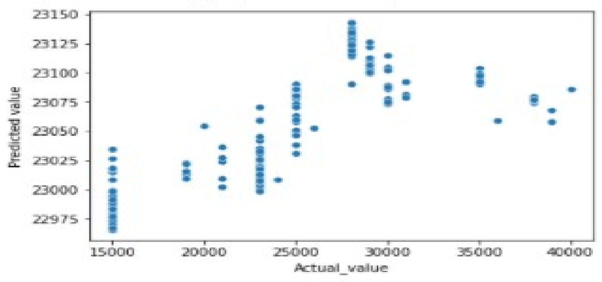
C: Regularization parameter. The strength of the regularization is inversely proportional to C. Must be strictly positive. The penalty is a squared l2 penalty. Default value is 1.0

Code and Results:



```
model_6=SVR()
model_6.fit(X_train,y_train)
model_6_train_pred=model_6.predict(X_train)  #Predictng for train values
model_6_test_pred=model_6.predict(X_test)  #predicting for test values
#calculating Mean absolute percentage error for both train and test
mape1=mean_absolute_percentage_error(model_6_train_pred,y_train)
mape2=mean_absolute_percentage_error(model_6_test_pred,y_test)

print("The train_MAPE for SVR model",mape1)
print("The test_MAPE for SVR model",mape2)

#calculating R2_Score value for both train and test

r2_train=r2_score (y_train,model_6_train_pred)
r2_test=r2_score(y_test,model_6_test_pred)

print("The R2_score  for train values" ,r2_train)
print("The R2_score for test values",r2_test)

#creating dataframe for actual and predicted values
svr_table=pd.DataFrame({'Predicted value': model_6_test_pred, 'Actual_value': y_test})
svr_table.head()

sns.scatterplot(data=svr_table, x="Actual_value", y="Predicted value")
```

```
The train_MAPE for SVR model 0.21269900123227095
The test_MAPE for SVR model 0.20790316470834988
The R2_score  for train values -0.02386717273477701
The R2_score for test values -0.06025964932088090 4
<matplotlib.axes._subplots.AxesSubplot at 0x7fcd9b901290>
```



```
svr_table.head()
```

	Predicted value	Actual_value
231	23004.560420	23000
688	23041.908484	23000
27	23125.493842	28000
366	23070.661074	23000
715	23077.262787	38000

The result is very bad as this model is giving r2_Score in negatives. This model is not suitable for this problem.

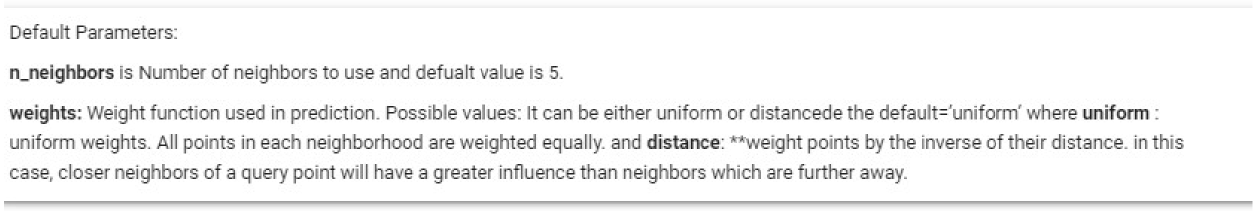
Let's now move on to next model.

KNeighborsRegressor

Theory:

KNN regression is a non-parametric method that, in an intuitive manner, approximates the association between independent variables and the continuous outcome by averaging the observations in the same neighborhood. The principle behind nearest neighbor methods is to find a predefined number of training samples closest in distance to the new point, and predict the label from these. The number of samples can be a user-defined constant (k-nearest neighbor learning), or vary based on the local density of points (radius-based neighbor learning). The distance can, in general, be any metric measure: standard Euclidean distance is the most common choice.

Parameters:

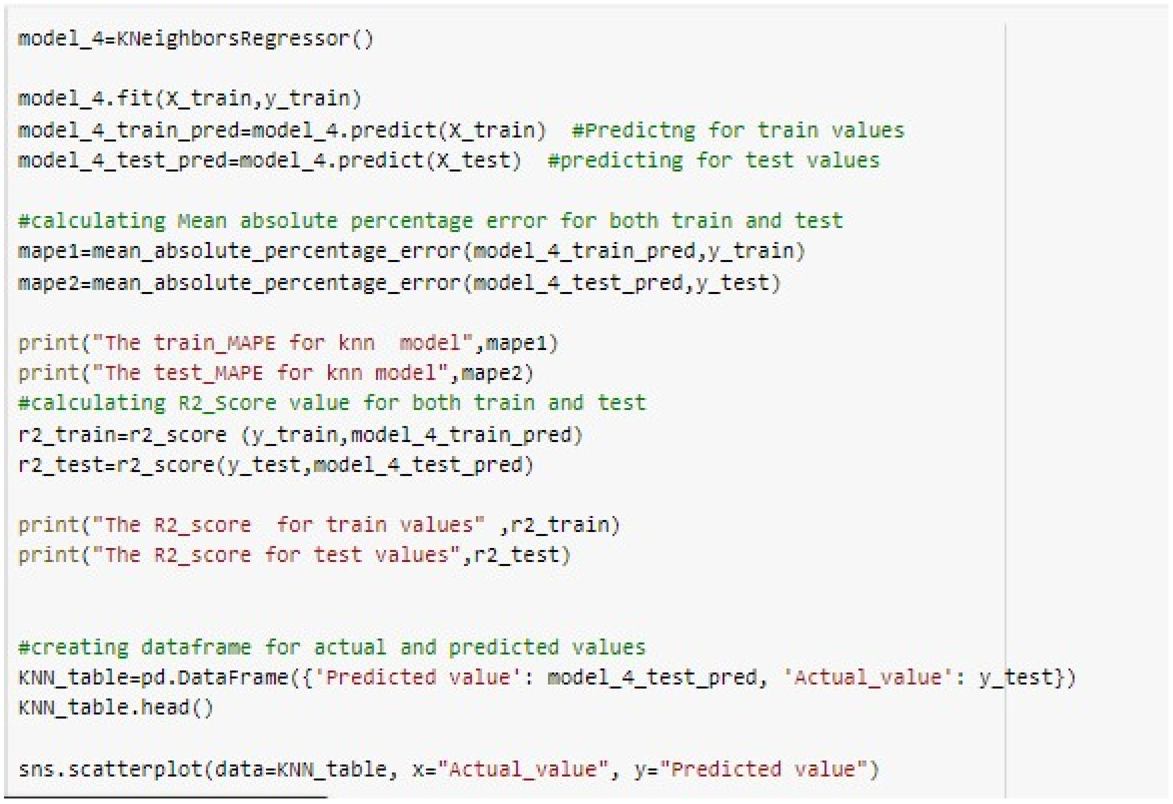


Default Parameters:

n_neighbors is Number of neighbors to use and defualt value is 5.

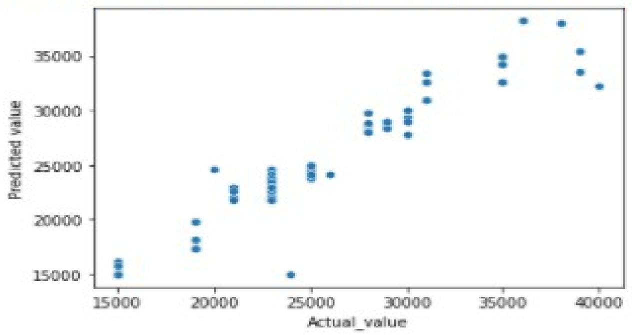
weights: Weight function used in prediction. Possible values: It can be either uniform or distancede the default='uniform' where **uniform** : uniform weights. All points in each neighborhood are weighted equally. and **distance**: **weight points by the inverse of their distance. in this case, closer neighbors of a query point will have a greater influence than neighbors which are further away.

Code and Results:



```
model_4=KNeighborsRegressor()

model_4.fit(X_train,y_train)
model_4_train_pred=model_4.predict(X_train)  #Predictng for train values
model_4_test_pred=model_4.predict(X_test)  #predicting for test values

#calculating Mean absolute percentage error for both train and test
mape1=mean_absolute_percentage_error(model_4_train_pred,y_train)
mape2=mean_absolute_percentage_error(model_4_test_pred,y_test)

print("The train_MAPE for knn  model",mape1)
print("The test_MAPE for knn model",mape2)
#calculating R2_Score value for both train and test
r2_train=r2_score (y_train,model_4_train_pred)
r2_test=r2_score(y_test,model_4_test_pred)

print("The R2_score  for train values" ,r2_train)
print("The R2_score for test values",r2_test)


#creating dataframe for actual and predicted values
KNN_table=pd.DataFrame({'Predicted value': model_4_test_pred, 'Actual_value': y_test})
KNN_table.head()

sns.scatterplot(data=KNN_table, x="Actual_value", y="Predicted value")
```

```
The train_MAPE for knn  model 0.013922200542679218
The test_MAPE for knn model 0.020356444334411267
The R2_score  for train values 0.9792695741754284
The R2_score for test values 0.9615919661958638
<matplotlib.axes._subplots.AxesSubplot at 0x7fcd9b9c9210>
```



```
KNN_table.head()
```

	Predicted value	Actual_value
231	23000.0	23000
688	23000.0	23000
27	28600.0	28000
366	24600.0	23000
715	38000.0	38000

Results are good but not better than the ExtraTreeRegressor. As we have better model then this , so this is not right model for this problem.

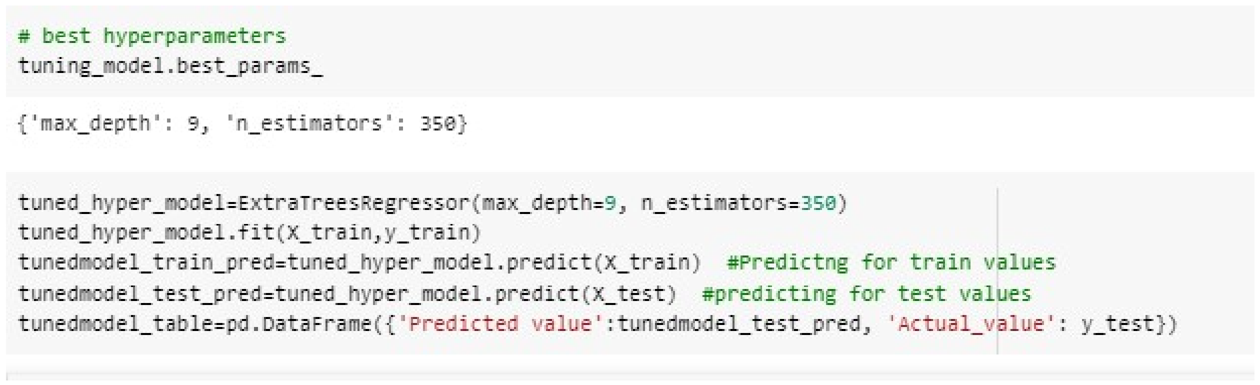
Now we have ExtraTreeRegressor which is very much suitable for this dataset let’s try to do hyper Parameter tuning to that and check if we can improve the model.

Hyper Parameter Tuning for ExtraTreeRegressor Model

Code and Results:

```
#Innitialising the Parameters
parameters={ "max_depth" : [3,5,7,9],
            "n_estimators": [300,350,400,450]
             }
```

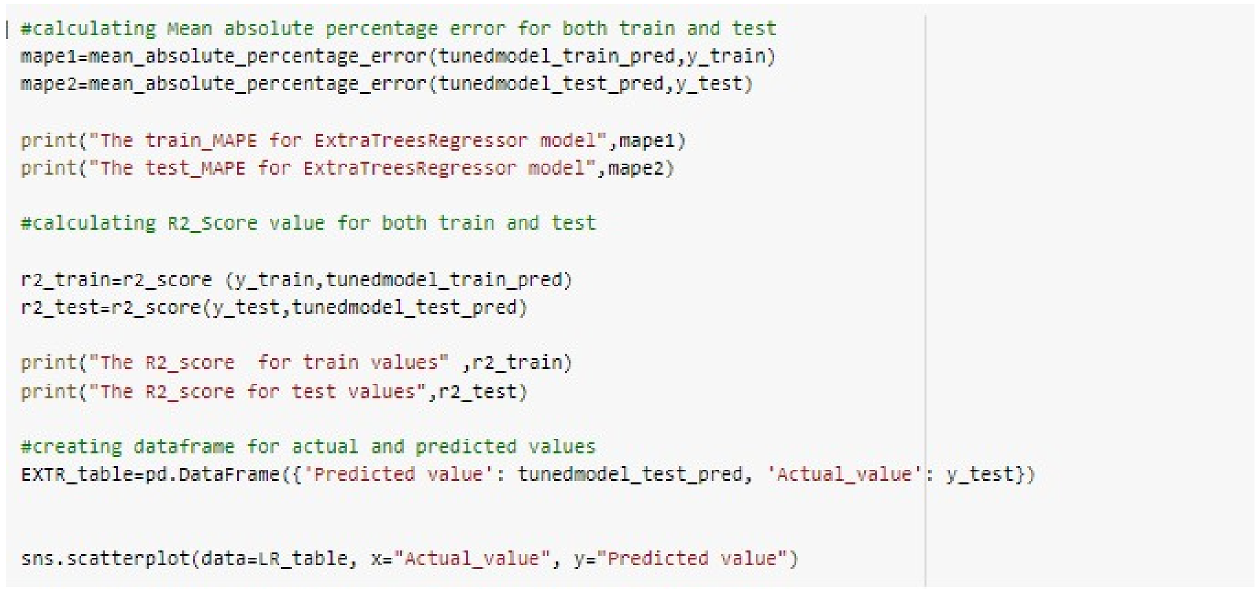
```
tuning_model=GridSearchCV(ExtraTreesRegressor(),param_grid=parameters,scoring='neg_mean_absolute_percentage_error',verbose=3)
tuning_model.fit(X_train,y_train)
```



```
# best hyperparameters
tuning_model.best_params_
```

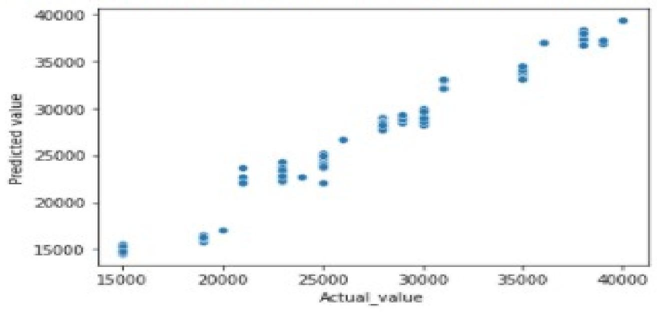
```
{'max_depth': 9, 'n_estimators': 350}
```

```
tuned_hyper_model=ExtraTreesRegressor(max_depth=9, n_estimators=350)
tuned_hyper_model.fit(X_train,y_train)
tunedmodel_train_pred=tuned_hyper_model.predict(X_train)  #Predictng for train values
tunedmodel_test_pred=tuned_hyper_model.predict(X_test)  #predicting for test values
tunedmodel_table=pd.DataFrame({'Predicted value':tunedmodel_test_pred, 'Actual_value': y_test})
```



```
#calculating Mean absolute percentage error for both train and test
mape1=mean_absolute_percentage_error(tunedmodel_train_pred,y_train)
mape2=mean_absolute_percentage_error(tunedmodel_test_pred,y_test)

print("The train_MAPE for ExtraTreesRegressor model",mape1)
print("The test_MAPE for ExtraTreesRegressor model",mape2)

#calculating R2_Score value for both train and test

r2_train=r2_score (y_train,tunedmodel_train_pred)
r2_test=r2_score(y_test,tunedmodel_test_pred)

print("The R2_score  for train values" ,r2_train)
print("The R2_score for test values",r2_test)

#creating dataframe for actual and predicted values
EXTR_table=pd.DataFrame({'Predicted value': tunedmodel_test_pred, 'Actual_value': y_test})


sns.scatterplot(data=LR_table, x="Actual_value", y="Predicted value")
sns.scatterplot(data=LR_table, x="Actual_value", y="Predicted value")
```

```
The train_MAPE for ExtraTreesRegressor model 0.0020699183306142847
The test_MAPE for ExtraTreesRegressor model 0.006566192829508732
The R2_score  for train values 0.9993873968937947
The R2_score for test values 0.9932288498259727
<matplotlib.axes._subplots.AxesSubplot at 0x7fcd9b6c4cd0>
```



```
EXTR_table.head(5)
```

	Predicted value	Actual_value
231	22990.332073	23000
688	23287.133151	23000
27	28005.415183	28000
366	23241.983137	23000
715	38000.000000	38000

From above output we can say that there is no much change in the results, but one observation is that the difference between train error and test error is reduced.

Future Works:

- The models can be trained such that it is 100% error free, if we get more number of data samples.
- Because this dataset has 986 number of samples which is not actually good enough.
- Looking at the target variable value counts there are few premium prices for which the number of samples are just 1 or 2 because of which the model is predicting near to those values.
- If we increase the number of samples for those premium prices model can be 100% error free.
- One more way is to add up other features like income of person, his regular check up schedule etc, can help to predict the premium price.