Four Wheeler Insurance Customer LifeTime Value Prediction **BY DHEERAJ KUMAR**

PROJECT GOAL

The **Requirement** of a Business Problem **is to develop a predictive model to Analyse and to Predit the LifeTime Value of customer** in a Four wheeler insurance company using **Regresson Analysis** with **Python**.

FEATURES DETAILS

FEATURE NAMES	EXPLANATION
CustomerID	ID of the Customer
Customer Lifetime Value	Life Time Profit to the owner by Customer
Coverage	Insurance Coverge to vehicle**[Categorical Variable]**
Education	Customer Education**[Categorical Variable]**
Employment Status	Employment Status of Customer**[Categorical Variable]**
Gender	Gender of Customer **[Categorical Variable]**
Income	Income of Customer
Location Geo	Latitude and Longitude of Customer **[Categorical Variable]**
Location Code	Location Code of Customer **[Categorical Variable]**
Marital Status	Marital Status of Customer **[Categorical Variable]**
Monthly Premium Auto	Monthly Premium Amount of Customers
Months Since Last Claim	Last Insurance pay of Customer
Months Since Policy Inception	Policy Inception of a Customer
Number of Open Complaints	Open Complaints of a Customer
Number of Policies	Number of Policies of Customer
Policy Type	Policy Type of Customer **[Categorical Variable]**
Policy	Policy taken by Customer **[Categorical Variable]**
Renew Offer Type	Renew Offers type for Customer **[Categorical Variable]**
Sales Channel	Sales Channel of Customer**[Categorical Variable]**
Total Claim Amount	Total Claim Amount of Customers
Vehicle Class	Vehicle Class of Customers **[Categorical Variable]**
Vehicle Size	Vehicle Size of Customers

Import Libraries

In [197]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
```

Loader Function to Load Dataframe

In [198]:
```
path ="D:\Data Science Python Programs\Terra Final Deployment\CLV_Train.csv"
class DataFrame_Loader():
    data = path

    def __init__(self):

        print("Loadind DataFrame")

    def read_csv(self,data):
        self.df = pd.read_csv(data)

    def load_csv(self):
        return self.df
```

In [199]:
```
data = DataFrame_Loader()
```

```
Loadind DataFrame
```

In [200]:
```
data.read_csv(path)
```

In [201]:
```
df=data.load_csv()
```

Exploratory Data Analysis

In [202]:
```
class Attribute_Information():

    def __init__(self):

        print("Attribute Information object created")

    def Column_information(self,df):

        data_info = pd.DataFrame(
                                columns=['No of observation',
                                        'No of Variables',
                                        'No of Numerical Variables',
                                        'No of Factor Variables',
                                        'No of Categorical Variables',
                                        'No of Logical Variables',
                                        'No of Date Variables',
                                        'No of zero variance variables'])


        data_info.loc[0,'No of observation'] = df.shape[0]
        data_info.loc[0,'No of Variables'] = df.shape[1]
        data_info.loc[0,'No of Numerical Variables'] = df._get_numeric_data().shape[1]
        data_info.loc[0,'No of Factor Variables'] = df.select_dtypes(include='category').shape[1]
        data_info.loc[0,'No of Logical Variables'] = df.select_dtypes(include='bool').shape[1]
        data_info.loc[0,'No of Categorical Variables'] = df.select_dtypes(include='object').shape[1]
        data_info.loc[0,'No of Date Variables'] = df.select_dtypes(include='datetime64').shape[1]
        data_info.loc[0,'No of zero variance variables'] = df.loc[:,df.apply(pd.Series.nunique)==1].shape[1]

        data_info =data_info.transpose()
        data_info.columns=['value']
        data_info['value'] = data_info['value'].astype(int)


        return data_info

    def _get_missing_values(self,data):

        #Getting sum of missing values for each feature
        missing_values = data.isnull().sum()
        #Feature missing values are sorted from few to many
        missing_values.sort_values(ascending=False, inplace=True)

        #Returning missing values
        return missing_values

    def Generate_Schema(self,data):

        feature_dtypes=data.dtypes
        self.missing_values=self._get_missing_values(data)

        print("=" * 110)

        print("{:16} {:16} {:20} {:16}".format("Feature Name".upper(),
                                            "Data Type".upper(),
                                            "# of Missing Values".upper(),
                                            "Samples".upper()))
        for feature_name, dtype, missing_value in zip(self.missing_values.index.values,
                                                    feature_dtypes[self.missing_values.index.values],
                                                    self.missing_values.values):
            print("{:18} {:19} {:19} ".format(feature_name, str(dtype), str(missing_value)), end="")
            for v in data[feature_name].values[:5]:
                print(v, end=",")
            print()

        print("="*110)

    def Agg_Tabulation(self,data):

        print("=" * 110)
        print("Aggregation of Table")
        print("=" * 110)
        table = pd.DataFrame(data.dtypes,columns=['dtypes'])
        table1 =pd.DataFrame(data.columns,columns=['Names'])
        table = table.reset_index()
        table= table.rename(columns={'index':'Name'})
        table['No of Missing'] = data.isnull().sum().values
        table['No of Uniques'] = data.nunique().values
        table['Percent of Missing'] = ((data.isnull().sum().values)/ (data.shape[0])) *100
        table['First Observation'] = data.loc[0].values
        table['Second Observation'] = data.loc[1].values
        table['Third Observation'] = data.loc[2].values
        for name in table['Name'].value_counts().index:
            table.loc[table['Name'] == name, 'Entropy'] = round(stats.entropy(data[name].value_counts(normalize=True), ba
se=2),2)
        return table

        print("=" * 110)

    def _iqr(self,x):
        return x.quantile(q=0.75) - x.quantile(q=0.25)

    def _outlier_count(self,x):
        upper_out = x.quantile(q=0.75) + 1.5 * self._iqr(x)
        lower_out = x.quantile(q=0.25) - 1.5 * self._iqr(x)
        return len(x[x > upper_out]) + len(x[x < lower_out])

    def num_count_summary(self,df):
        df_num = df._get_numeric_data()
        data_info_num = pd.DataFrame()
        i=0
        for c in  df_num.columns:
            data_info_num.loc[c,'Negative values count']= df_num[df_num[c]<0].shape[0]
            data_info_num.loc[c,'Positive values count']= df_num[df_num[c]>0].shape[0]
            data_info_num.loc[c,'Zero count']= df_num[df_num[c]==0].shape[0]
            data_info_num.loc[c,'Unique count']= len(df_num[c].unique())
            data_info_num.loc[c,'Negative Infinity count']= df_num[df_num[c]== -np.inf].shape[0]
            data_info_num.loc[c,'Positive Infinity count']= df_num[df_num[c]== np.inf].shape[0]
            data_info_num.loc[c,'Missing Percentage']= df_num[df_num[c].isnull()].shape[0]/ df_num.shape[0]
            data_info_num.loc[c,'Count of outliers']= self._outlier_count(df_num[c])
            i = i+1
        return data_info_num
```

In [203]:
```
Info = Attribute_Information()
```

```
Attribute Information object created
```

In [204]:
```
Info.Column_information(df)
```

Out[204]:

	value
No of observation	9806
No of Variables	22
No of Numerical Variables	9
No of Factor Variables	0
No of Categorical Variables	13
No of Logical Variables	0
No of Date Variables	0
No of zero variance variables	0

In [205]:
```
Info._get_missing_values(df)
```

Out[205]:
```
Coverage                         925
Policy.Type                      891
Number.of.Open.Complaints        818
Monthly.Premium.Auto             794
Education                        129
Gender                           129
Marital.Status                   129
Sales.Channel                    128
Renew.Offer.Type                 128
Vehicle.Class                    126
Vehicle.Size                     126
Number.of.Policies               121
Policy                           121
Location.Code                    119
EmploymentStatus                 118
Months.Since.Last.Claim            0
Months.Since.Policy.Inception      0
Location.Geo                       0
Income                             0
Total.Claim.Amount                 0
Customer.Lifetime.Value            0
CustomerID                         0
dtype: int64
```

In [206]:
```
Info.Generate_Schema(df)
```

```
==============================================================================================================
FEATURE NAME     DATA TYPE        # OF MISSING VALUES  SAMPLES
Coverage           object              925                 Basic,Basic,Basic,Basic,Basic,
Policy.Type        object              891                 Personal Auto,Personal Auto,Personal Auto,Corporate Auto,Per
sonal Auto,
Number.of.Open.Complaints float64             818                 nan,0.0,0.0,nan,0.0,
Monthly.Premium.Auto float64             794                 67.0,101.0,108.0,116.0,72.0,
Education          object              129                 Bachelor,College,High School or Below,College,Bachelor,
Gender             object              129                 F,M,F,M,F,
Marital.Status     object              129                 Married,Married,Married,Married,Married,
Sales.Channel      object              128                 Branch,Agent,Branch,Branch,Web,
Renew.Offer.Type   object              128                 Offer2,Offer2,Offer2,Offer1,Offer2,
Vehicle.Class      object              126                 Four-Door Car,SUV,SUV,SUV,Two-Door Car,
Vehicle.Size       float64             126                 2.0,2.0,1.0,3.0,3.0,
Number.of.Policies float64             121                 2.0,5.0,3.0,3.0,5.0,
Policy             object              121                 Personal L2,Personal L2,Personal L1,Corporate L3,Personal L
1,
Location.Code      object              119                 Urban,Suburban,Urban,Suburban,Suburban,
EmploymentStatus   object              118                 Unemployed,Employed,Employed,Employed,Retired,
Months.Since.Last.Claim int64               0                   2,26,3,2,3,
Months.Since.Policy.Inception int64               0                   33,42,44,15,68,
Location.Geo       object              0                   17.7,77.7,28.8,76.6,21.6,88.4,19,72.5,19.1,74.7,
Income             object              0                   0,63357,64125,67544,19651,
Total.Claim.Amount float64             0                   267.214383,565.508572,369.818708,556.8,345.6,
Customer.Lifetime.Value float64             0                   7824.372789,8005.964669,8646.504109,9294.088719,5595.97
1364999999,
CustomerID         int64               0                   5917,2057,4119,1801,9618,
==============================================================================================================
```

In [207]:
```
Info.Agg_Tabulation(df)
```

```
==============================================================================================================
Aggregation of Table
==============================================================================================================
```

Out[207]:

	Name	dtypes	No of Missing	No of Uniques	Percent of Missing	First Observation	Second Observation	Third Observation	Entropy
0	CustomerID	int64	0	9806	0.000000	5917	2057	4119	13.26
1	Customer.Lifetime.Value	float64	0	6477	0.000000	7824.37	8005.96	8646.5	12.40
2	Coverage	object	925	3	9.433000	Basic	Basic	Basic	1.28
3	Education	object	129	5	1.315521	Bachelor	College	High School or Below	2.02
4	EmploymentStatus	object	118	5	1.203345	Unemployed	Employed	Employed	1.50
5	Gender	object	129	2	1.315521	F	M	F	1.00
6	Income	object	0	4622	0.000000	0	63357	64125	9.67
7	Location.Geo	object	0	2840	0.000000	17.7,77.7	28.8,76.6	21.6,88.4	10.91
8	Location.Code	object	119	3	1.213543	Urban	Suburban	Urban	1.30
9	Marital.Status	object	129	3	1.315521	Married	Married	Married	1.37
10	Monthly.Premium.Auto	float64	794	191	8.097083	67	101	108	6.26
11	Months.Since.Last.Claim	int64	0	36	0.000000	2	26	3	5.12
12	Months.Since.Policy.Inception	int64	0	100	0.000000	33	42	44	6.62
13	Number.of.Open.Complaints	float64	818	6	8.341832	NaN	0	0	1.10
14	Number.of.Policies	float64	121	9	1.233938	2	5	3	2.60
15	Policy.Type	object	891	3	9.086274	Personal Auto	Personal Auto	Personal Auto	0.99
16	Policy	object	121	9	1.233938	Personal L2	Personal L2	Personal L1	2.46
17	Renew.Offer.Type	object	128	4	1.305323	Offer2	Offer2	Offer2	1.83
18	Sales.Channel	object	128	4	1.305323	Branch	Agent	Branch	1.91
19	Total.Claim.Amount	float64	0	4125	0.000000	267.214	565.509	369.819	10.66
20	Vehicle.Class	object	126	6	1.284928	Four-Door Car	SUV	SUV	1.88
21	Vehicle.Size	float64	126	3	1.284928	2	2	1	1.16

In [140]:
```
Info._iqr(df)
```

Out[140]:
```
CustomerID                       5799.500000
Customer.Lifetime.Value          4946.331174
Monthly.Premium.Auto               40.250000
Months.Since.Last.Claim            17.000000
Months.Since.Policy.Inception      47.750000
Number.of.Open.Complaints           0.000000
Number.of.Policies                  3.000000
Total.Claim.Amount                273.188206
Vehicle.Size                        0.000000
dtype: float64
```

In [141]:
```
Info._outlier_count(df)
```

Out[141]:
```
19612
```

In [208]:
```
Info.num_count_summary(df)
```

Out[208]:

	Negative values count	Positive values count	Zero count	Unique count	Negative Infinity count	Positive Infinity count	Missing Percentage	Count of outliers
CustomerID	0.0	9806.0	0.0	9806.0	0.0	0.0	0.000000	0.0
Customer.Lifetime.Value	0.0	9806.0	0.0	6477.0	0.0	0.0	0.000000	883.0
Monthly.Premium.Auto	0.0	9012.0	0.0	192.0	0.0	0.0	0.080971	432.0
Months.Since.Last.Claim	0.0	9510.0	296.0	36.0	0.0	0.0	0.000000	0.0
Months.Since.Policy.Inception	0.0	9712.0	94.0	100.0	0.0	0.0	0.000000	0.0
Number.of.Open.Complaints	0.0	1858.0	7130.0	7.0	0.0	0.0	0.083418	1858.0
Number.of.Policies	0.0	9685.0	0.0	10.0	0.0	0.0	0.012339	446.0
Total.Claim.Amount	0.0	9806.0	0.0	4125.0	0.0	0.0	0.000000	495.0
Vehicle.Size	0.0	9680.0	0.0	4.0	0.0	0.0	0.012849	2885.0

Haversine distance Computaion

In [143]:
```
import math

class Compute_Haversine_Distance():

    def __init__(self):
        print("Distance object created")

    def Split_Location_geo(self,Location_Geo):
        df['Lati'], df['Longi'] = df['Location.Geo'].str.split(',', 1).str

    def harvasine_distance(self,lati,longi):
        df['Lati']= pd.to_numeric(df['Lati'],errors='coerce')
        df['Longi']= pd.to_numeric(df['Longi'],errors='coerce')
        df['LAT_rad'],df['LON_rad'] = np.radians(df['Lati']), np.radians(df['Longi'])
        df['dLON'] = df['LON_rad'] - math.radians(-56.7213600)
        df['dLAT'] = df['LAT_rad'] - math.radians(37.2175900)
        df['distance'] = 6367 * 2 * np.arcsin(np.sqrt(np.sin(df['dLAT']/2)**2 + math.cos(math.radians(37.2175900)) * np.c
os(df['LAT_rad']) * np.sin(df['dLON']/2)**2))
```

In [144]:
```
distance = Compute_Haversine_Distance()
```

```
Distance object created
```

In [145]:
```
distance.Split_Location_geo(df)
```

```
C:\Users\DELL\Anaconda3\envs\Tensorflow env\lib\site-packages\ipykernel_launcher.py:9: FutureWarning: Columnar iteratio
n over characters will be deprecated in future releases.
  if __name__ == '__main__':
```

In [146]:
```
distance.harvasine_distance(df['Lati'],df['Longi'])
```

Data Preprocessing

In [147]:
```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
class Imputer(Attribute_Information):

    def __init__(self):
        print("Imputation object created")

    def fit(self, data):

        self.fill = pd.Series([data[column].value_counts().index[0]
            if data[column].dtype == np.dtype('O') else data[column].mean() for column in data],
            index=data.columns)

        return self

    def transform(self, data):
        return data.fillna(self.fill)
```

In [148]:
```
impute = Imputer()
```

```
Imputation object created
```

In [149]:
```
impute.fit(df)
```

Out[149]:
```
<__main__.Imputer at 0x1dfd21ee908>
```

In [150]:
```
df=impute.transform(df)
```

Feature Engineering

In [151]:
```
from sklearn.preprocessing import LabelEncoder
from sklearn import preprocessing
col_list=['Renew.Offer.Type','Policy.Type']
nondummy=['CustomerID','Customer.Lifetime.Value','Income','Location.Geo','Months.Since.Policy.Inception','Number.of.Open.
Complaints','Number.of.Policies','Total.Claim.Amount','Vehicle.Size']
class Base_Feature_Engineering(Imputer):

    def __init__(self):
        print("Feature Engineering object created")

    def _Label_Encoding(self,data):
        category_col =[var for var in data.columns if data[var].dtypes =="object"]
        labelEncoder = preprocessing.LabelEncoder()
        mapping_dict={}
        for col in category_col:
            data[col] = labelEncoder.fit_transform(data[col])
            le_name_mapping = dict(zip(labelEncoder.classes_, labelEncoder.transform(labelEncoder.classes_)))
            mapping_dict[col]=le_name_mapping
            return mapping_dict

    def _get_dummies(self, data, prefered_columns=None):

        if prefered_columns is None:
            columns=data.columns.values
            non_dummies=None
        else:
            non_dummies=[col for col in data.columns.values if col not in prefered_columns ]

            columns=prefered_columns


        dummies_data=[pd.get_dummies(data[col],prefix=col) for col in columns]

        if non_dummies is not None:
            for non_dummy in non_dummies:
                dummies_data.append(data[non_dummy])

        return pd.concat(dummies_data, axis=1)
```

In [152]:
```
FE = Base_Feature_Engineering()
```

```
Feature Engineering object created
```

In [166]:
```
FE._Label_Encoding(df)
```

Dropper Function to drop unwanted variables

In [167]:
```
col_list=['Location.Geo','Lati', 'Longi', 'LAT_rad', 'LON_rad', 'dLON', 'dLAT']
class Column_Dopper():

    def __init__(self):
        print("Column Dopper object created")

    def dropper(self,x):
        data=[]
        for i in x.columns:
            if i not in col_list:
                data.append(i)
        return df[data]

    def remove_outliers(self,data):
        q1 =df['Customer.Lifetime.Value'].quantile(.25)
        q3 = df['Customer.Lifetime.Value'].quantile(.75)
        iqr = q3-q1
        df_out = df[~((df['Customer.Lifetime.Value'] < (q1 - 1.5 *iqr))  |  (df['Customer.Lifetime.Value'] > (q3+ 1.5 * i
qr)))]
        return df_out
```

In [168]:
```
drop = Column_Dopper()
```

```
Column Dopper object created
```

In [169]:
```
df=drop.dropper(df)
```

In [170]:
```
df=drop.remove_outliers(df)
```

Model Building

In [177]:
```
from sklearn import metrics
from sklearn.metrics import r2_score
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split

class Data_Modelling():

    def __init__(self,n_estimators=100,random_state=42,max_depth=10):
        print("Data Modelling object created")

    def Linear_Regression_Model(self,df):
        x = df.drop(['Customer.Lifetime.Value'],axis=1)
        y = df['Customer.Lifetime.Value']
        x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.30,random_state=42)
        regressor = LinearRegression()
        reg=regressor.fit(x_train,y_train)
        LR_pred=regressor.predict(x_test)
        LR_RMSE = np.sqrt(metrics.mean_squared_error(y_test,LR_pred))
        LR_r2_score = r2_score(y_test,LR_pred)
        return LR_RMSE,LR_r2_score

    def Decision_Tree_Model(self,df):
        x = df.drop(['Customer.Lifetime.Value'],axis=1)
        y = df['Customer.Lifetime.Value']
        x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.30,random_state=42)
        regressor = DecisionTreeRegressor(random_state=29)
        reg=regressor.fit(x_train,y_train)
        DT_pred=regressor.predict(x_test)
        DT_RMSE = np.sqrt(metrics.mean_squared_error(y_test,DT_pred))
        DT_r2_score = r2_score(y_test,DT_pred)
        return DT_RMSE,DT_r2_score

    def Random_Forest_Model(self,df):
        x = df.drop(['Customer.Lifetime.Value'],axis=1)
        y = df['Customer.Lifetime.Value']
        x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.30,random_state=42)
        regressor = RandomForestRegressor(n_estimators=100,random_state=29,max_depth=12)
        reg=regressor.fit(x_train,y_train)
        RF_pred=regressor.predict(x_test)
        RF_RMSE = np.sqrt(metrics.mean_squared_error(y_test,RF_pred))
        RF_r2_score = r2_score(y_test,RF_pred)
        return RF_RMSE,RF_r2_score

    def Extreme_Gradient_Boosting_Model(self,df):
        x = df.drop(['Customer.Lifetime.Value'],axis=1)
        y = df['Customer.Lifetime.Value']
        x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.30,random_state=42)
        regressor = XGBRegressor(n_estimators=100,random_state=29,max_depth=9,learning_rate=0.07)
        reg=regressor.fit(x_train,y_train)
        XGB_pred=regressor.predict(x_test)
        XGB_RMSE = np.sqrt(metrics.mean_squared_error(y_test,XGB_pred))
        XGB_r2_score = r2_score(y_test,XGB_pred)
        return XGB_RMSE,XGB_r2_score
```

In [178]:
```
model = Data_Modelling()
```

```
Data Modelling object created
```

In [179]:
```
model.Linear_Regression_Model(df)
```

Out[179]:
```
(2761.7760128710843, 0.28489962900593524)
```

In [180]:
```
model.Decision_Tree_Model(df)
```

Out[180]:
```
(1248.3766139170116, 0.8538895273997682)
```

In [181]:
```
model.Random_Forest_Model(df)
```

Out[181]:
```
(939.539369016892, 0.9172400799465625)
```

In [182]:
```
model.Extreme_Gradient_Boosting_Model(df)
```

Out[182]:
```
(930.416580304399, 0.9188394504760382)
```

In [196]:
```
from sklearn.model_selection import GridSearchCV
import warnings
warnings.filterwarnings("ignore")
class Regression_Cross_Validator():

    def __init__(self,n_estimators=100,random_state=42,max_depth=10):
        print("Cross Validation object created")


    def Cross_Validated_Random_Forest_Model(self,data):
        param_grid = [
        {'n_estimators': [10, 100], 'max_features': [5, 60],
         'max_depth': [10, 200, None], 'bootstrap': [True, False]}
        ]

        x = df.drop(['Customer.Lifetime.Value'],axis=1)
        y = df['Customer.Lifetime.Value']
        x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.30,random_state=42)
        regressor = RandomForestRegressor(n_estimators=100,random_state=29,max_depth=12)
        reg=regressor.fit(x_train,y_train)
        grid_search_forest = GridSearchCV(regressor, param_grid, cv=10, scoring='neg_mean_squared_error')
        grid_search_forest.fit(x_train, y_train)

        grid_best= grid_search_forest.best_estimator_.predict(x_test)
        grid_mse = metrics.mean_squared_error(y_test, grid_best)
        grid_rScore = r2_score(y_test, grid_best)
        grid_rmse = np.sqrt(grid_mse)
        return grid_rmse,grid_rScore

    def Cross_Validated_Extreme_Gradient_Boosting_Model(self,data):
        params={
         "learning_rate"    : [0.05, 0.1] ,
         "max_depth"        : [ 10,100],
         "n_estimators"     : [ 10, 100],
         "colsample_bytree" : [ 0.3, 0.4, 0.5 , 0.7 ]

        }

        x = df.drop(['Customer.Lifetime.Value'],axis=1)
        y = df['Customer.Lifetime.Value']
        x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.30,random_state=42)
        XGBregressor = XGBRegressor(n_estimators=100,random_state=29,max_depth=9,learning_rate=0.07)
        reg=XGBregressor.fit(x_train,y_train)
        grid_search_forest = GridSearchCV(XGBregressor, params, cv=10, scoring='neg_mean_squared_error')
        grid_search_forest.fit(x_train, y_train)

        grid_best= grid_search_forest.best_estimator_.predict(x_test)
        grid_mse = metrics.mean_squared_error(y_test, grid_best)
        grid_rScore = r2_score(y_test, grid_best)
        grid_rmse = np.sqrt(grid_mse)
        return grid_rmse,grid_rScore
```

In [194]:
```
cv = Regression_Cross_Validator()
```

```
Cross Validation object created
```

In [185]:
```
cv.Cross_Validated_Random_Forest_Model(df)
```

Out[185]:
```
(945.722881537151, 0.9161471379533397)
```

In [195]:
```
cv.Cross_Validated_Extreme_Gradient_Boosting_Model(df)
```

Out[195]:
```
(965.8521620284849, 0.9125396100381109)
```

In [ ]:

In [ ]: