Created by Kelvin Tang

Machine learning - Linear regression and random forest regression

-Mercedes Benz car price prediction

Introduction

In this article, it will be an extension of my previous data visualization blog. The main objective is to eventually train a model by using the same Mercedes dataset, that will help people to predict their Mercedes Benz car prices.

The features of dataset include: Model, year, transmission, mileage, fuel-type, tax, miles per gallon and engine-size. Based on these features, model will predict our target variable, price. The size of original dataset is 13119 rows x 9 columns, there is no null-value in it. However there is still some data that have to be cleaned, and it brings us to the next section.

Data cleaning

```
df.describe()
```

	year	price	mileage	tax	mpg	engineSize
count	13119.000000	13119.000000	13119.000000	13119.000000	13119.000000	13119.000000
mean	2017.296288	24698.596920	21949.559037	129.972178	55.155843	2.071530
std	2.224709	11842.675542	21176.512267	65.260286	15.220082	0.572426
min	1970.000000	650.000000	1.000000	0.000000	1.100000	0.000000
25%	2016.000000	17450.000000	6097.500000	125.000000	45.600000	1.800000
50%	2018.000000	22480.000000	15189.000000	145.000000	56.500000	2.000000
75%	2019.000000	28980.000000	31779.500000	145.000000	64.200000	2.100000
max	2020.000000	159999.000000	259000.000000	580.000000	217.300000	6.200000

Firstly, numerical data will be cleaned first. From left to right, in df.describe, year, price and mileage looks alright.

For tax column, there are some 0 values. I have considered dropping, but found out that there could be exemption such as disabled driver. So I keep them for now.

For mpg column, there are four ‘1’ and one ’11’ representing miles per gallon. However the mean is 55.1mpg, the gap is too big and it doesn’t make much sense for a car that can go only 1 mile or 11 miles per gallon of fuel. So i decided to impute those 5 values to its mean value.

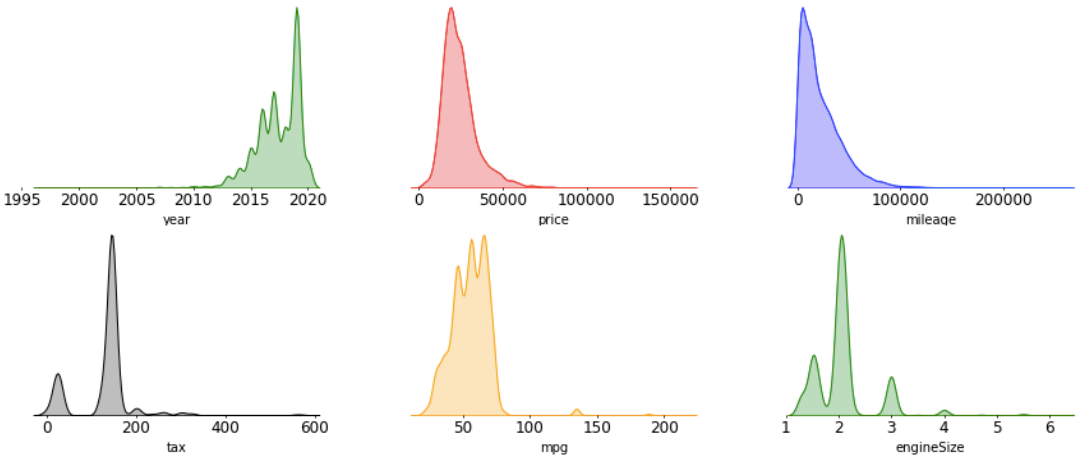
For engineSize column, rows that have 0 engine-size are also dropped.

For categorial column (model, transmission, fuelType), among the three columns, there are 10 rows of ‘other’. I dropped all of them since the ‘other' is not significant in the dataset.

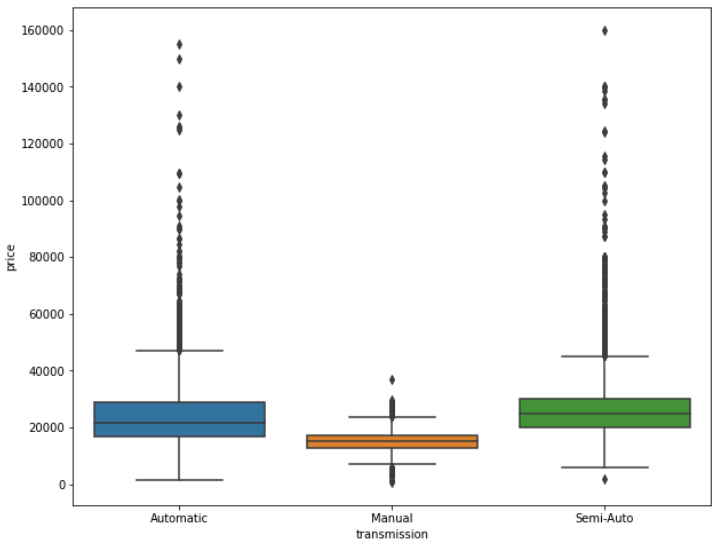
EDA

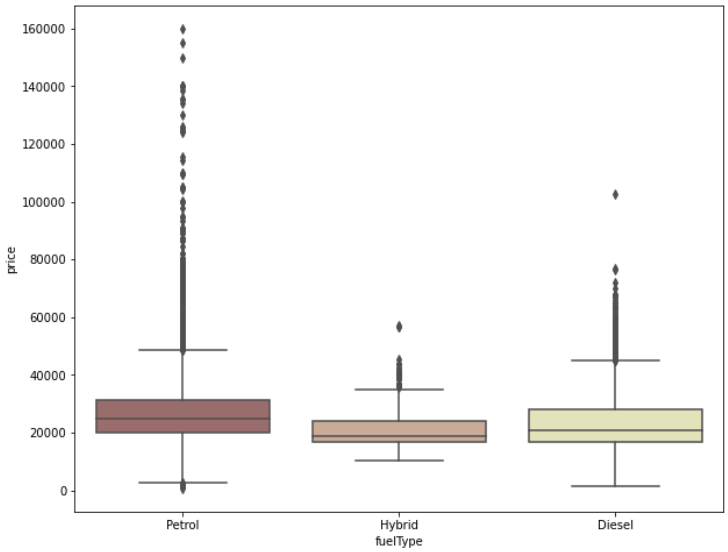
This EDA will mainly focus on the relationship between price and the other features, as price is the target variable. Seaborn and matplotlib are the libraries used for visualizing the data.

First graph will show us a general idea of the distribution of the numerical columns.

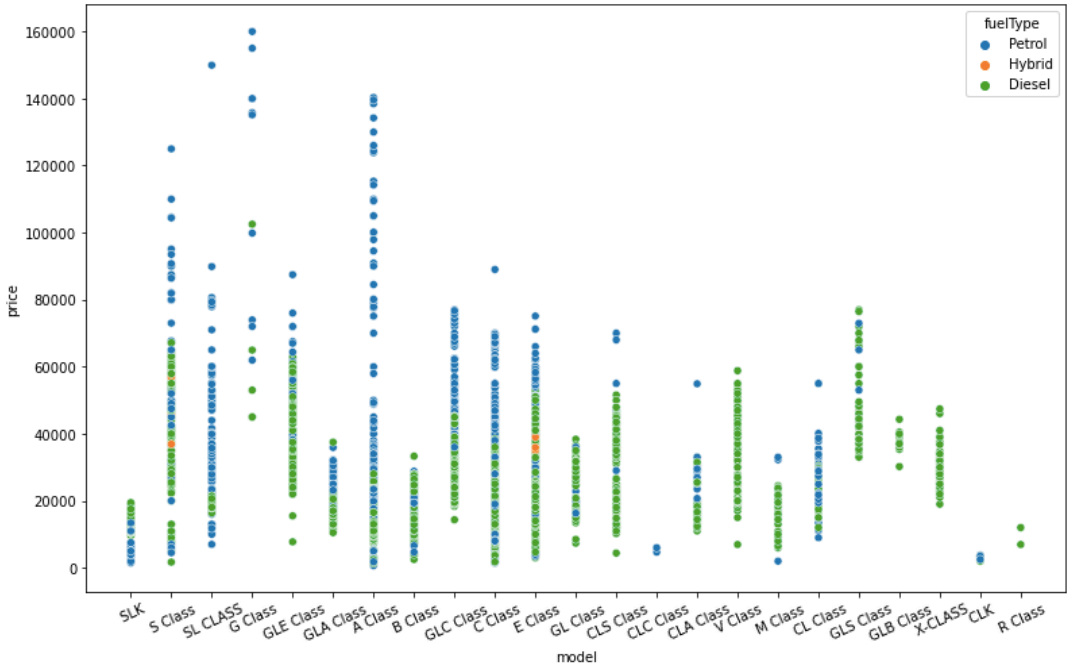


Below left boxplot shows that manual cars have a lower price than automatic and semi-auto.
Right boxplot shows that petrol cars are more expensive than hybrid and diesel.

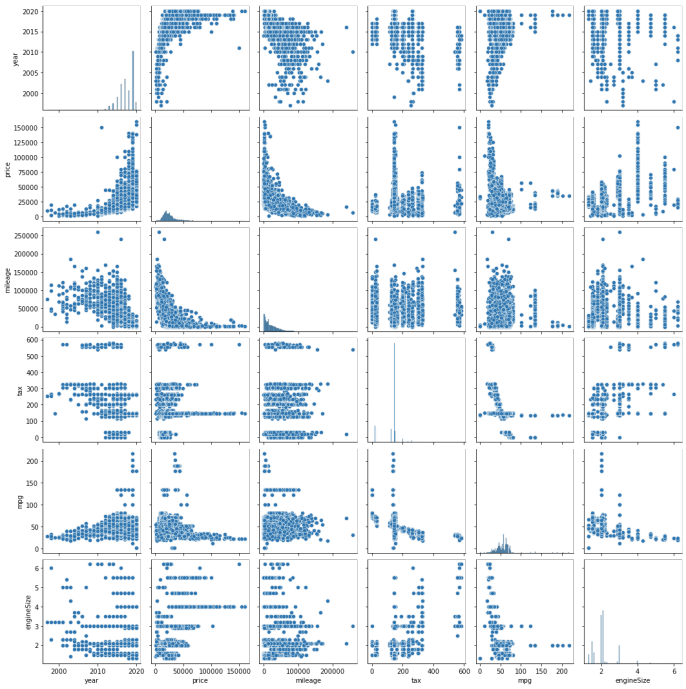


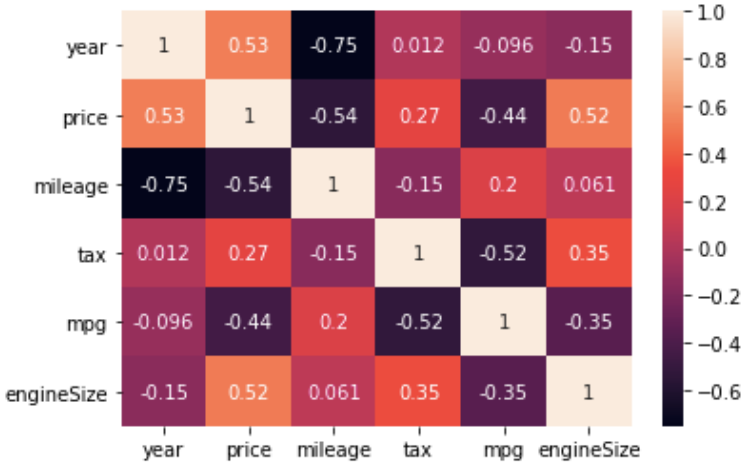


Below scatterplot has the model on x-axis and price on y-axis, also each color of the dot represents a fuel type. The most expensive model is G Class, cheapest model is CLK class in this dataset.



Pair plot(left) and heatmap(right) below. From the heatmap, there is a mild positive correlation in orange color, 0.53 and 0.52 between price-year and price-engineSize. Which means when price increase, year/ engineSize will increase as well. The pair plot on the left proves it. On the other hand, there is a negative 0.54 correlation between price-mileage. Which makes sense as when car has high mileage, the price will decrease.





Preparing the data for machine learning models

-Encoding
As machine learning model can only read numbers, categories in categorial column has to be changed to numerical values.

For transmission, semi-auto, automatic and manual are encoded to 0,1,2.

For fuelType, diesel, petrol and hybrid are encoded to 0,1,2.

-Get dummies
For model, pd.get_dummies is used. It is used for data manipulation and converts categorial data into dummy or indicator values. A new data frame dum2 is created, and the columns are the different model types. Below the columns, there will only be 0 and 1, 0 means false and 1 means true for that model. Next, df will concat with dum2, drop model from df and get this.

```
dum=pd.get_dummies(df["model"],drop_first=True)
dum2=pd.DataFrame(dum,columns=dum.columns)
df=pd.concat([df,dum2],axis=1)
df.drop("model",axis=1,inplace=True)
```

	year	price	transmission	mileage	fuelType	tax	mpg	engineSize	B Class	C Class	...	GLC Class	GLE Class	GLS Class	M Class	R Class	S Class	SL CLASS	SLK	V Class	X CLAS
0	2005	5200	1	63000	1	325	32.1	1.8	0	0	...	0	0	0	0	0	0	0	1	0	
1	2017	34948	1	27000	2	20	61.4	2.1	0	0	...	0	0	0	0	0	1	0	0	0	
2	2016	49948	1	6200	1	555	28.0	5.5	0	0	...	0	0	0	0	0	0	1	0	0	
3	2016	61948	1	16000	1	325	30.4	4.0	0	0	...	0	0	0	0	0	0	0	0	0	
4	2016	73948	1	4000	1	325	30.1	4.0	0	0	...	0	0	0	0	0	0	0	0	0	

5 rows × 30 columns

Model building 1 - Linear Regression

First model will be a linear regression. Linear regression is a supervised Machine learning model in which the model finds the best fit linear line between independent variable and dependent variable. In this dataset, independent variable (y) will be our price and dependent variable (X) will be the columns except price.

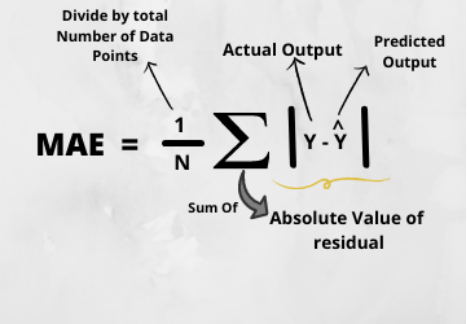
```
X=df.drop("price",axis=1)
y=df["price"]
```

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=0)
3501

from sklearn.linear_model import LinearRegression
model=LinearRegression()
model.fit(X_train,y_train)

y_pred=model.predict(X_test)
```

Evaluation metrics for Linear Regression

Evaluation metrics give us information how well the model is doing, including Mean Absolute Error (MAE), Mean Squared error (MSE), Root Mean Squared error (RMSE) and R2 score (R2).

-MAE
MAE for linear regression is 3502 which shows the absolute difference between actual output and predicted output. A low MAE value is desired. However, according to the MAE formula on the right, if actual output is bigger than predicted output, it returns a negative number which decreases MAE. It can easily give an illusion that the model is doing well. Which leads us to the next evaluation metric MSE.



-MSE
MSE for LR is 33,686,559 and it seems like a big number. Because we take the square of the difference between actual and predicted value, a negative difference can turn into a positive number after squaring them, hence it is a better error rate indicator of the model. Same as MAE, a low number is desired because it is a loss.

$$MSE = \frac{1}{n}\Sigma\underbrace{\left(y - \widehat{y}\right)^2}_{\text{The square of the difference between actual and predicted}}$$

-RMSE
RMSE is just the square root of MSE, so it has the same unit as the required output variable. The price RMSE is 5805 for linear regression, given that our mean value for pice is 25,000.

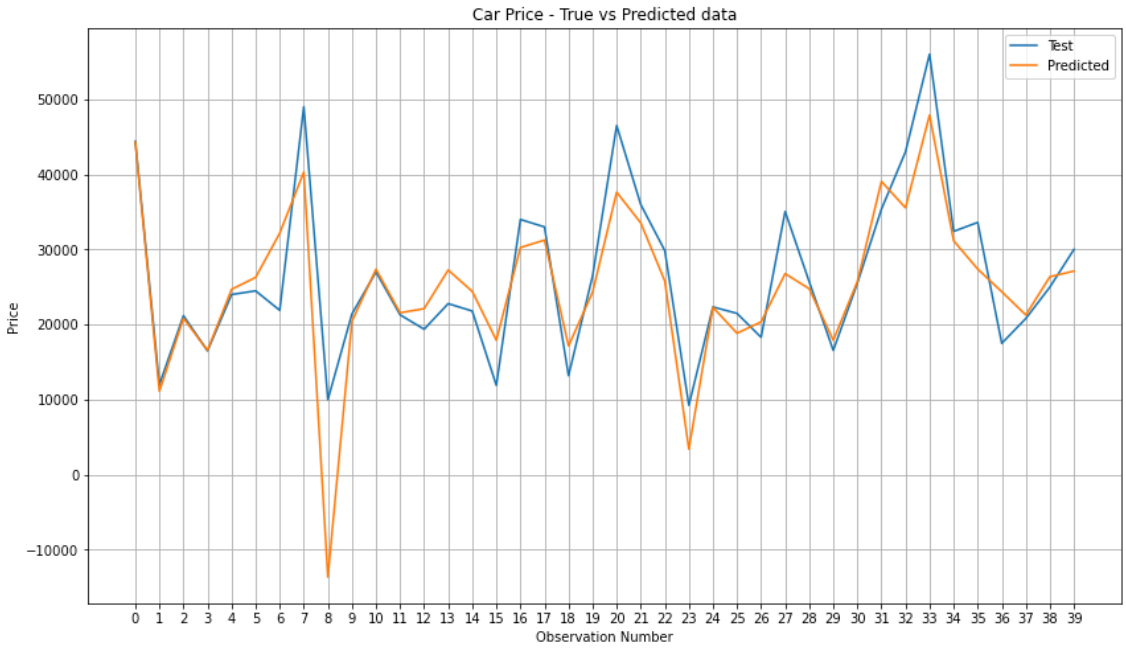
-R2 score
R2 score aka R-squared is a statistical measure of how close the data are to the fitted regression line. It is always a value between 0 and 1, a number closer to 1 the better the model. We get 0.755 r2 score which means 75.5% of the data can be fitted to the regression line.

$$R^2 = 1 - \frac{SS_{RES}}{SS_{TOT}} = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \overline{y})^2}$$

Metric evaluation summary table for linear regression: (I didn't include MSE as RMSE can represent it)

	Metric	Score
0	mean_absolute_error_lr	3502.010679
1	root_mean_squared_error_lr	5805.285173
2	r2_score_lr	0.755352

From the graph below, we can see the variance of the first 40 observations between test and predicted price. Where blue line is the test data price and orange line is the predicted price from our linear regression model.



Model building 2 - Random Forest Regressor

The second model will be random forest regressor, throughout the code, n_estimators, which is the number of tress to be used in the forest, has to be entered. The number of n_estimators can directly impact the performance of the model, for initial process, I will put n_estimators =10 first and tune it later on.

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor

# Divide data into training and validation subsets
X_train_rf,X_test_rf,y_train_rf,y_test_rf=train_test_split(X,y,test_size=0.2,random_state=0)

model = RandomForestRegressor(n_estimators=10, random_state=0)
model.fit(X_train_rf, y_train_rf)
y_pred_rf=model.predict(X_test_rf)
```

Evaluation metrics for Random Forest Regressor

When n_estimators = 10,

-MAE is 1647

-MSE is 7272080

-RMSE 2697

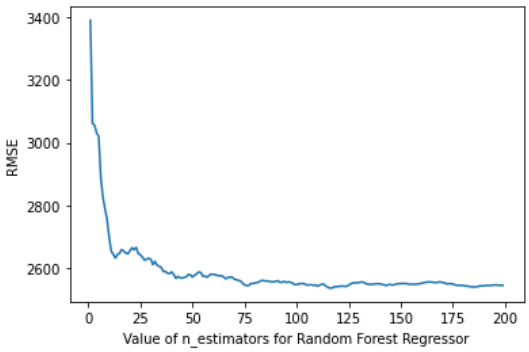
-R2 score 0.947

	Metric	Score
0	mean_absolute_error_rf	1647.107647
1	root_mean_squared_error_rf	2696.679378
2	r2_score_rf	0.947210

As I just randomly put 10 as n_estimators before, next step will be finding the best estimator value which returns the least number of RMSE. It is tested from 1 to 200, and it shows that the curve starts to flatten at 75, so 75 is picked as the n_estimators and the model will be evaluated again.

```
scores =[]

for k in range(1,200):
    model_1 = RandomForestRegressor(n_estimators=k, random_state=0)
    X_train_rf1,X_test_rf1,y_train_rf1,y_test_rf1=train_test_split(X,y,test_size=0.2,random_state=0)
    model_1.fit(X_train_rf1, y_train_rf1)
    y_pred_rf1=model_1.predict(X_test_rf1)
    scores.append(np.sqrt(mean_squared_error(y_test_rf1,y_pred_rf1)))


# plot the relationship between K and testing accuracy
# plt.plot(x_axis, y_axis)
plt.plot(range(1,200), scores)
plt.xlabel('Value of n_estimators for Random Forest Regressor')
plt.ylabel('RMSE')
```

```
Text(0, 0.5, 'RMSE')
```




```
#when n_estimators = 75, RMSE starts to stablize
model = RandomForestRegressor(n_estimators=75, random_state=0)
model.fit(X_train_rf, y_train_rf)
y_pred_rf=model.predict(X_test_rf)
```

When n_estimators = 75,

-MAE = 1587

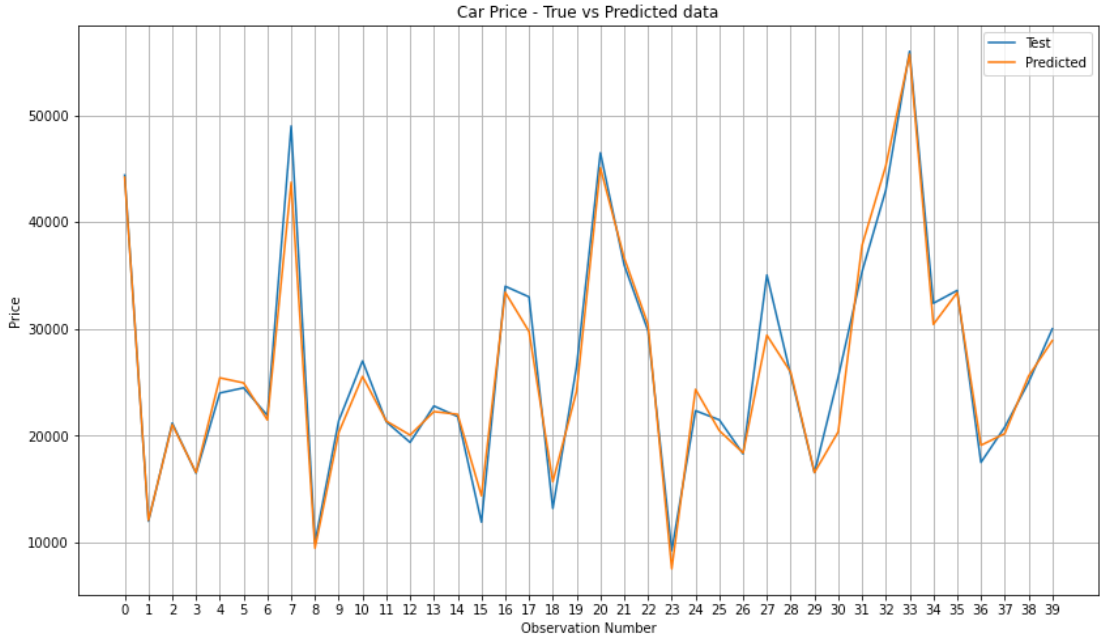
-MSE = 6,486,746

-RMSE = 2547

-R2 score = 0.953

	Metric	Score
0	mean_absolute_error_rf	1586.897965
1	root_mean_squared_error_rf	2546.909089
2	r2_score_rf	0.952911

n_estimators = 75 shows a better performance than n_estimators =10 after fine tuning the number. The same graph as linear regression test vs predicted graph will be plotted and the orange line clearly is moving much closer to the blue line.



Summary

Random Forest regressor definitely out-performs linear regression for this case, it gives a lower MSE, RMSE, MAE but also a higher R2 score.

Linear regression evaluation metrics:

	Metric	Score
0	mean_absolute_error_lr	3502.010679
1	root_mean_squared_error_lr	5805.285173
2	r2_score_lr	0.755352

Random Forest Regressor evaluation metric

	Metric	Score
0	mean_absolute_error_rf	1586.897965
1	root_mean_squared_error_rf	2546.909089
2	r2_score_rf	0.952911