7. CODING & SOLUTIONING

a. **Feature 1 -** Collect the dataset and preprocess the data.

Machine Learning has become a tool used in almost every task that requires estimation. So we need to build a model to estimate the price of used cars. The model should take car-related parameters and output a selling price. On sprint-1 the selling price of a used car depends on certain features datasets are collected from different open sources like kaggle.com, data.gov, UCI machine learning repository, the dataset which contains a set of features through which the resale price of the car can be identified is to be collected as

- price
- vehicle Type
- year Of Registration
- gearbox
- model
- kilo meter
- month Of Registration
- fuel Type
- brand
- not Repaired Damage

ML is a data hunger technology, it depends heavily on data, without data, it is impossible. It is the most crucial aspect that makes algorithm training possible. Collects Data, Import necessary packages, Pre-process images, and passes on to Network Model and Saves Model Weights. The libraries can be imported,

Pre-Process The Data:

Pre-processing the dataset that includes:

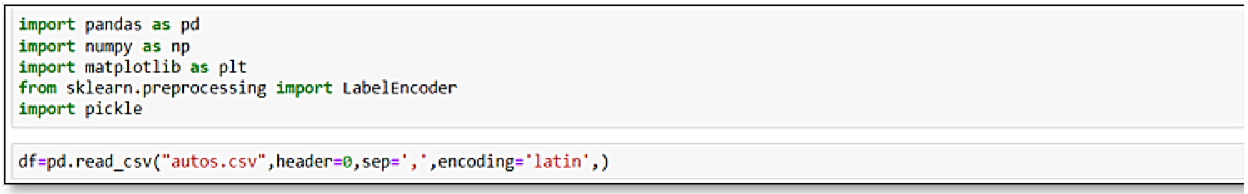
1. Handling the null values.

2. Handling the categorical values if any.

3. Normalize the data if required.

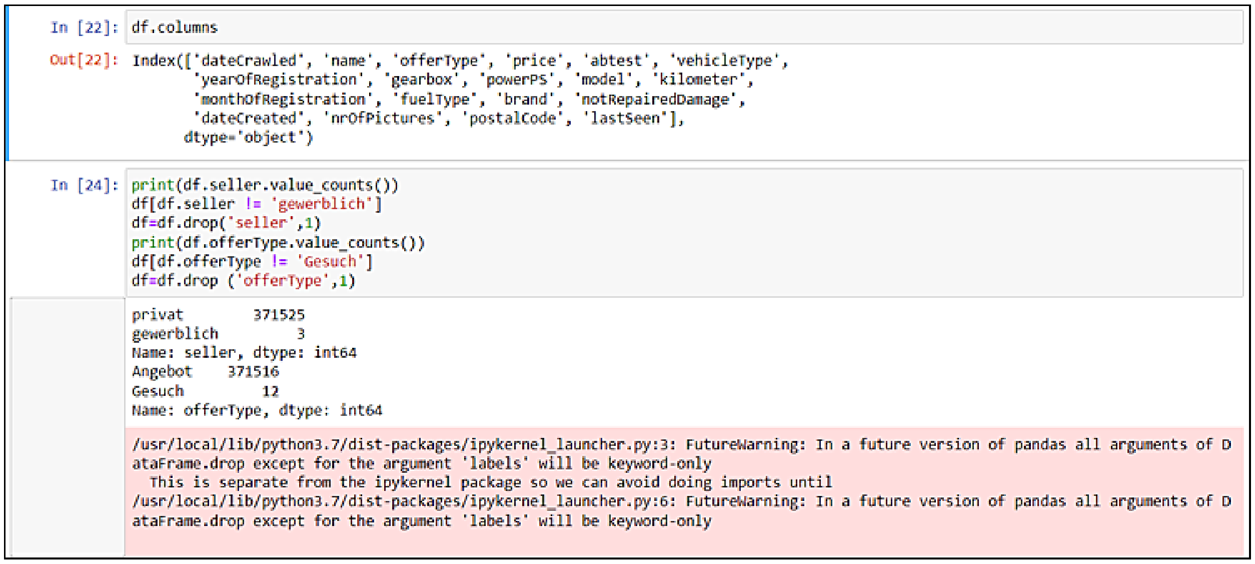
4. Identify the dependent and independent variables.

Data cleaning and wrangling methods are applied on the *used cars* data file. Before making data cleaning, some explorations and data visualizations were applied on data set. This gave some idea and guide about how to deal with missing values and extreme values. After data cleaning, data exploration was applied again in order to understand cleaned version of the data.



```
import pandas as pd
import numpy as np
import matplotlib as plt
from sklearn.preprocessing import LabelEncoder
import pickle
```

```
df=pd.read_csv("autos.csv",header=0,sep=',',encoding='latin',)
```

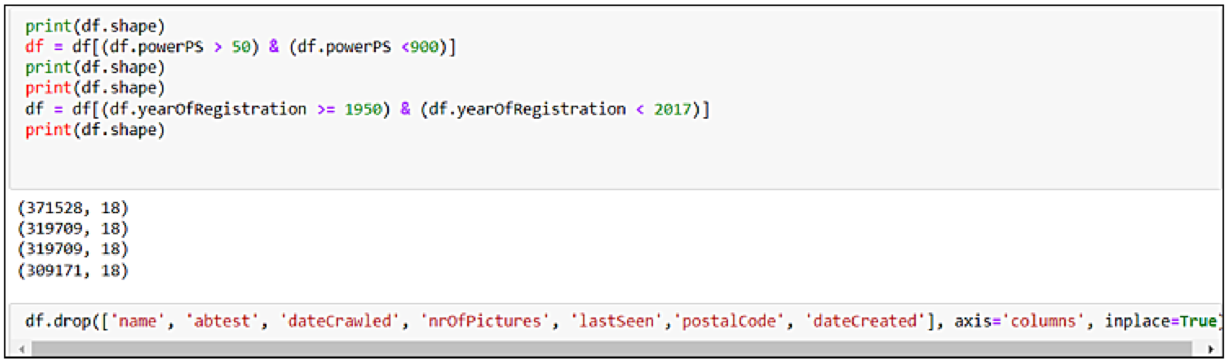


```
In [22]: df.columns

Out[22]: Index(['dateCrawled', 'name', 'offerType', 'price', 'abtest', 'vehicleType',
                'yearOfRegistration', 'gearbox', 'powerPS', 'model', 'kilometer',
                'monthOfRegistration', 'fuelType', 'brand', 'notRepairedDamage',
                'dateCreated', 'nrOfPictures', 'postalCode', 'lastSeen'],
               dtype='object')
```

```
In [24]: print(df.seller.value_counts())
         df[df.seller != 'gewerblich']
         df=df.drop('seller',1)
         print(df.offerType.value_counts())
         df[df.offerType != 'Gesuch']
         df=df.drop ('offerType',1)
```

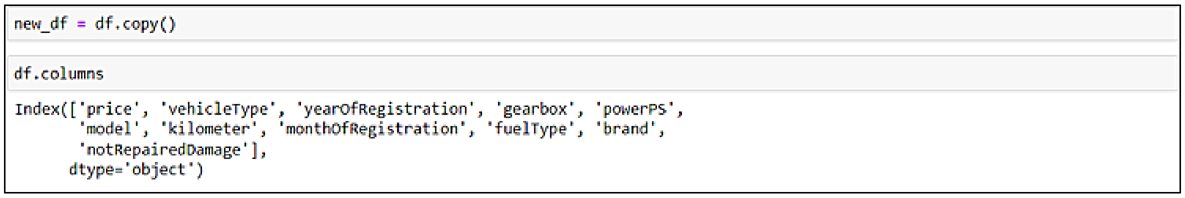
```
privat        371525
gewerblich         3
Name: seller, dtype: int64
Angebot    371516
Gesuch         12
Name: offerType, dtype: int64
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: FutureWarning: In a future version of pandas all arguments of D
ataFrame.drop except for the argument 'labels' will be keyword-only
  This is separate from the ipykernel package so we can avoid doing imports until
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:6: FutureWarning: In a future version of pandas all arguments of D
ataFrame.drop except for the argument 'labels' will be keyword-only
```



```
print(df.shape)
df = df[(df.powerPS > 50) & (df.powerPS <900)]
print(df.shape)
print(df.shape)
df = df[(df.yearOfRegistration >= 1950) & (df.yearOfRegistration < 2017)]
print(df.shape)
```

```
(371528, 18)
(319709, 18)
(319709, 18)
(309171, 18)
```

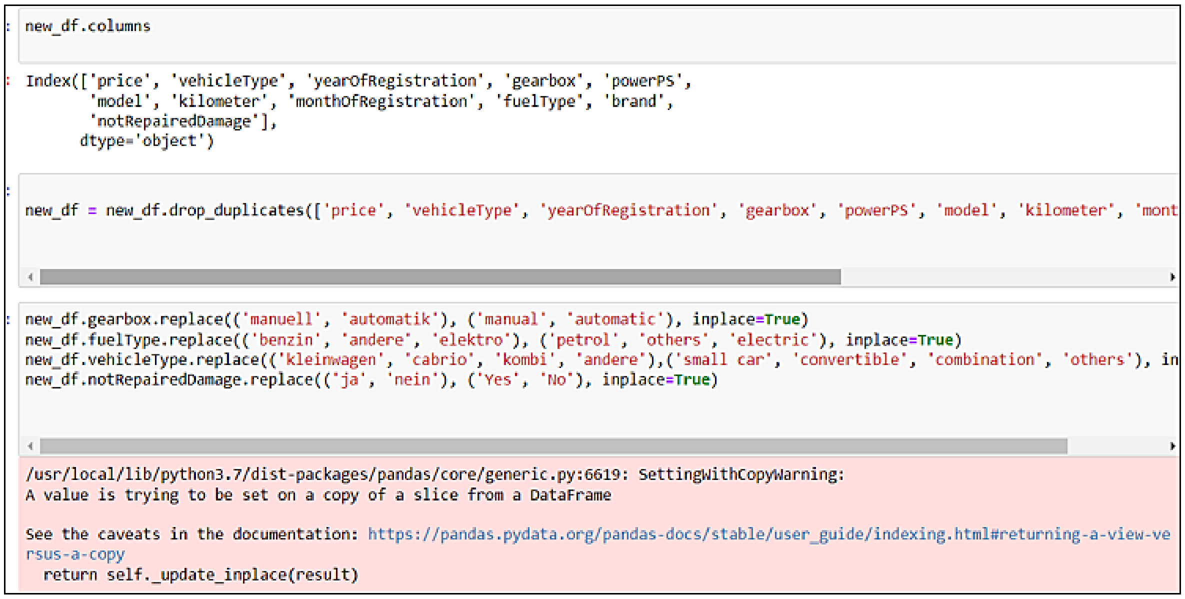
```
df.drop(['name', 'abtest', 'dateCrawled', 'nrOfPictures', 'lastSeen','postalCode', 'dateCreated'], axis='columns', inplace=True
```



```
new_df = df.copy()
```

```
df.columns
```

```
Index(['price', 'vehicleType', 'yearOfRegistration', 'gearbox', 'powerPS',
       'model', 'kilometer', 'monthOfRegistration', 'fuelType', 'brand',
       'notRepairedDamage'],
      dtype='object')
```



```
new_df.columns
```

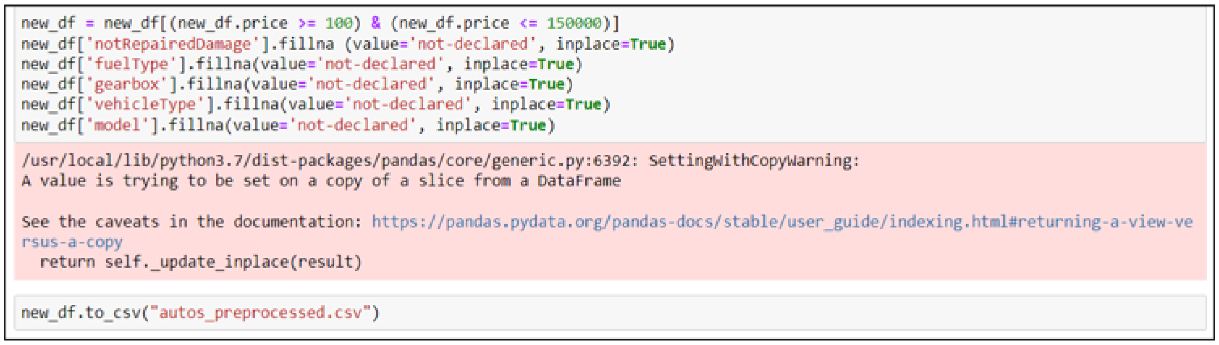
```
Index(['price', 'vehicleType', 'yearOfRegistration', 'gearbox', 'powerPS',
       'model', 'kilometer', 'monthOfRegistration', 'fuelType', 'brand',
       'notRepairedDamage'],
      dtype='object')
```

```
new_df = new_df.drop_duplicates(['price', 'vehicleType', 'yearOfRegistration', 'gearbox', 'powerPS', 'model', 'kilometer', 'mont
```

```
new_df.gearbox.replace(('manuell', 'automatik'), ('manual', 'automatic'), inplace=True)
new_df.fuelType.replace(('benzin', 'andere', 'elektro'), ('petrol', 'others', 'electric'), inplace=True)
new_df.vehicleType.replace(('kleinwagen', 'cabrio', 'kombi', 'andere'),('small car', 'convertible', 'combination', 'others'), in
new_df.notRepairedDamage.replace(('ja', 'nein'), ('Yes', 'No'), inplace=True)
```

```
/usr/local/lib/python3.7/dist-packages/pandas/core/generic.py:6619: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ve
rsus-a-copy
  return self._update_inplace(result)
```

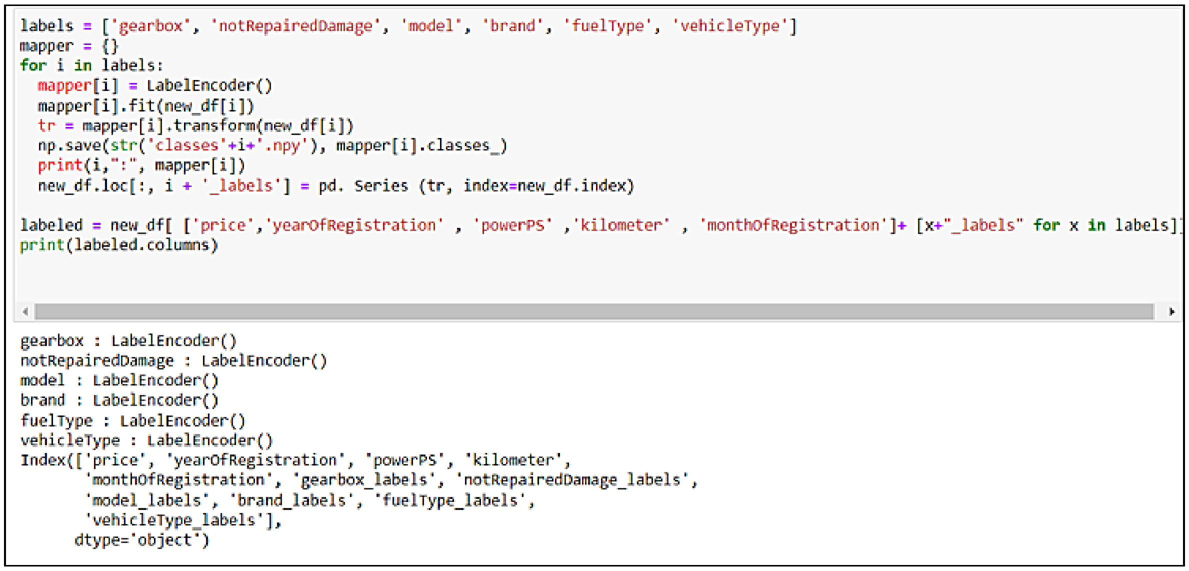


```
new_df = new_df[(new_df.price >= 100) & (new_df.price <= 150000)]
new_df['notRepairedDamage'].fillna (value='not-declared', inplace=True)
new_df['fuelType'].fillna(value='not-declared', inplace=True)
new_df['gearbox'].fillna(value='not-declared', inplace=True)
new_df['vehicleType'].fillna(value='not-declared', inplace=True)
new_df['model'].fillna(value='not-declared', inplace=True)
```

```
/usr/local/lib/python3.7/dist-packages/pandas/core/generic.py:6392: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ve
rsus-a-copy
  return self._update_inplace(result)
```

```
new_df.to_csv("autos_preprocessed.csv")
```



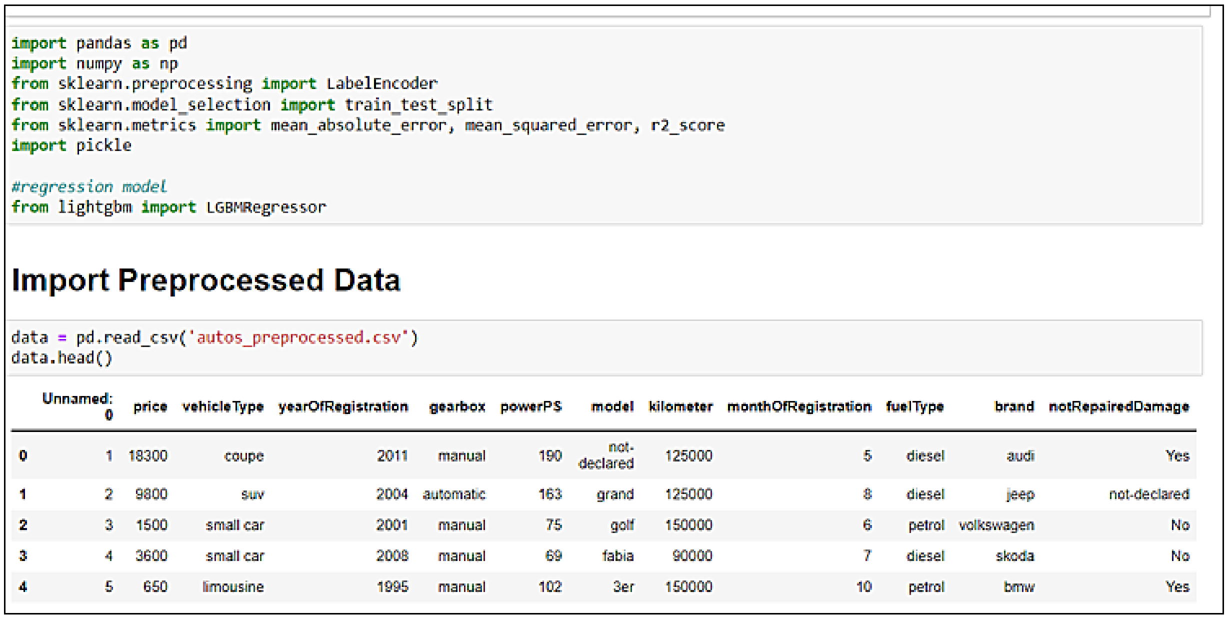
```
labels = ['gearbox', 'notRepairedDamage', 'model', 'brand', 'fuelType', 'vehicleType']
mapper = {}
for i in labels:
  mapper[i] = LabelEncoder()
  mapper[i].fit(new_df[i])
  tr = mapper[i].transform(new_df[i])
  np.save(str('classes'+i+'.npy'), mapper[i].classes_)
  print(i,":", mapper[i])
  new_df.loc[:, i + '_labels'] = pd. Series (tr, index=new_df.index)

labeled = new_df[ ['price','yearOfRegistration' , 'powerPS' ,'kilometer' , 'monthOfRegistration']+ [x+"_labels" for x in labels]
print(labeled.columns)
```

```
gearbox : LabelEncoder()
notRepairedDamage : LabelEncoder()
model : LabelEncoder()
brand : LabelEncoder()
fuelType : LabelEncoder()
vehicleType : LabelEncoder()
Index(['price', 'yearOfRegistration', 'powerPS', 'kilometer',
       'monthOfRegistration', 'gearbox_labels', 'notRepairedDamage_labels',
       'model_labels', 'brand_labels', 'fuelType_labels',
       'vehicleType_labels'],
      dtype='object')
```

b. Feature 2 - Training and testing and creating the application

A training model is a dataset that is used to train an algorithm. It consists of the sample output data and the corresponding sets of input data that have an influence on the output. The training model is used to run the input data through the algorithm to correlate the processed output against the sample output. The result from this correlation is used to modify the model. This iterative process is called “model fitting”. The accuracy of the training dataset or the validation dataset is critical for the precision of the model. Model training is the process of feeding an algorithm with data to help identify and learn good values for all



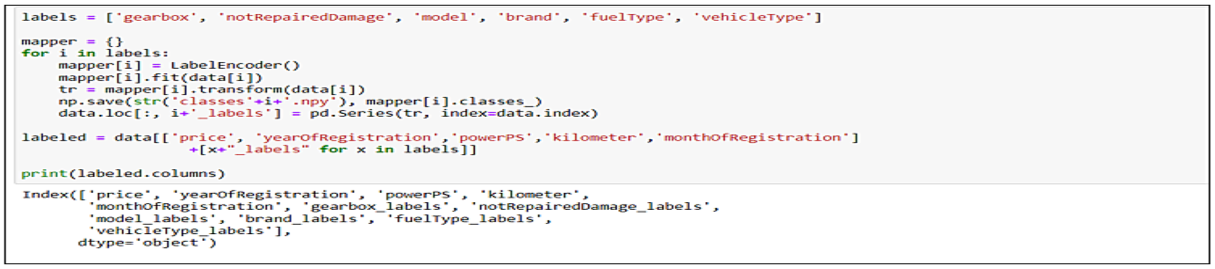
```python
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import pickle

#regression model
from lightgbm import LGBMRegressor
```

Import Preprocessed Data

```python
data = pd.read_csv('autos_preprocessed.csv')
data.head()
```

	Unnamed: 0	price	vehicleType	yearOfRegistration	gearbox	powerPS	model	kilometer	monthOfRegistration	fuelType	brand	notRepairedDamage
0	1	18300	coupe	2011	manual	190	not-declared	125000	5	diesel	audi	Yes
1	2	9800	suv	2004	automatic	163	grand	125000	8	diesel	jeep	not-declared
2	3	1500	small car	2001	manual	75	golf	150000	6	petrol	volkswagen	No
3	4	3600	small car	2008	manual	69	fabia	90000	7	diesel	skoda	No
4	5	650	limousine	1995	manual	102	3er	150000	10	petrol	bmw	Yes



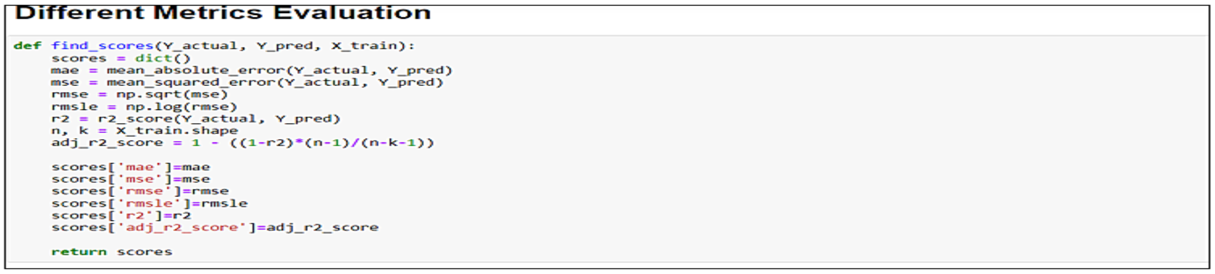
```python
labels = ['gearbox', 'notRepairedDamage', 'model', 'brand', 'fuelType', 'vehicleType']

mapper = {}
for i in labels:
    mapper[i] = LabelEncoder()
    mapper[i].fit(data[i])
    tr = mapper[i].transform(data[i])
    np.save(str('classes'+i+'.npy'), mapper[i].classes_)
    data.loc[:, i+'_labels'] = pd.Series(tr, index=data.index)

labeled = data[['price', 'yearOfRegistration','powerPS','kilometer','monthOfRegistration']
                    +[x+"_labels" for x in labels]]

print(labeled.columns)
```

```
Index(['price', 'yearOfRegistration', 'powerPS', 'kilometer',
       'monthOfRegistration', 'gearbox_labels', 'notRepairedDamage_labels',
       'model_labels', 'brand_labels', 'fuelType_labels',
       'vehicleType_labels'],
      dtype='object')
```



Different Metrics Evaluation

```python
def find_scores(Y_actual, Y_pred, X_train):
    scores = dict()
    mae = mean_absolute_error(Y_actual, Y_pred)
    mse = mean_squared_error(Y_actual, Y_pred)
    rmse = np.sqrt(mse)
    rmsle = np.log(rmse)
    r2 = r2_score(Y_actual, Y_pred)
    n, k = X_train.shape
    adj_r2_score = 1 - ((1-r2)*(n-1)/(n-k-1))

    scores['mae']=mae
    scores['mse']=mse
    scores['rmse']=rmse
    scores['rmsle']=rmsle
    scores['r2']=r2
    scores['adj_r2_score']=adj_r2_score

    return scores
```