# CAR PRICE PREDICTION PROJECT

**Submitted by:**
**SHUBHAM SHUKLA**

# ACKNOWLEDGMENT

I have taken efforts in this project by referring from below websites,

1. www.google.com
2. https://stackoverflow.com/
3. https://www.geeksforgeeks.org/

# INTRODUCTION

## BUSINESS PROBLEM FRAMING

With the covid-19 impact in the market, we have seen lot of changes in the car market. Now some cars are in demand hence making them costly and some are not in demand hence cheaper. One of our clients works with small traders, who sell used cars. With the change in market due to covid-19 impact, our client is facing problems with their previous car price valuation machine learning models. So, they are looking for new machine learning models from new data. We have to make car price valuation model.

## REVIEW OF LITERATURE

This project is more about exploration, feature engineering and classification that can be done on this data. Since we scrape huge amount of data that includes more car related features, we can do better data exploration and derive some interesting features using the available columns.

The goal of this project is to build an application which can predict the car prices with the help of other features. In the long term, this would allow people to better explain and reviewing their purchase with each other in this increasingly digital world.

## ANALYTICAL PROBLEM FRAMING

## MATHEMATICAL/ANALYTICAL MODELLING OF THE PROBLEM

In our scrapped dataset, our target variable "**selling_price**" is a continuous variable. Therefore, we will be handling this modelling problem as classification.

This project is done in two parts:

➔ Data Collection phase

➔ Model building phase

**Data Collection phase:**

You have to scrape at least 5000 used cars data. You can scrape more data as well, it's up to you. More the data better the model

In this section you need to scrape the data of used cars from websites (Olx, cardekho, Cars24 etc.) You need web scraping for this. You have to fetch data for different locations. The number of columns for data doesn't have limit, it's up to you and your creativity. Generally, these columns are Brand, model, variant, manufacturing year, driven kilometers, fuel, number of owners, location and at last target variable Price of the car. This data is to give you a hint about important variables in used car model. You can make changes to it, you can add or you can remove some columns, it completely depends on the website from which you are fetching the data.

Try to include all types of cars in your data for example- SUV, Sedans, Coupe, minivan, Hatchback. Note – The data which you are collecting is important to us. Kindly don't share it on any public platforms.

**Model building phase:**

➔After collecting the data, you need to build a machine learning model. Before model building do all data pre-processing steps. Try different models with different hyper parameters and select the best model.

➔Follow the complete life cycle of data science. Include all the steps like:

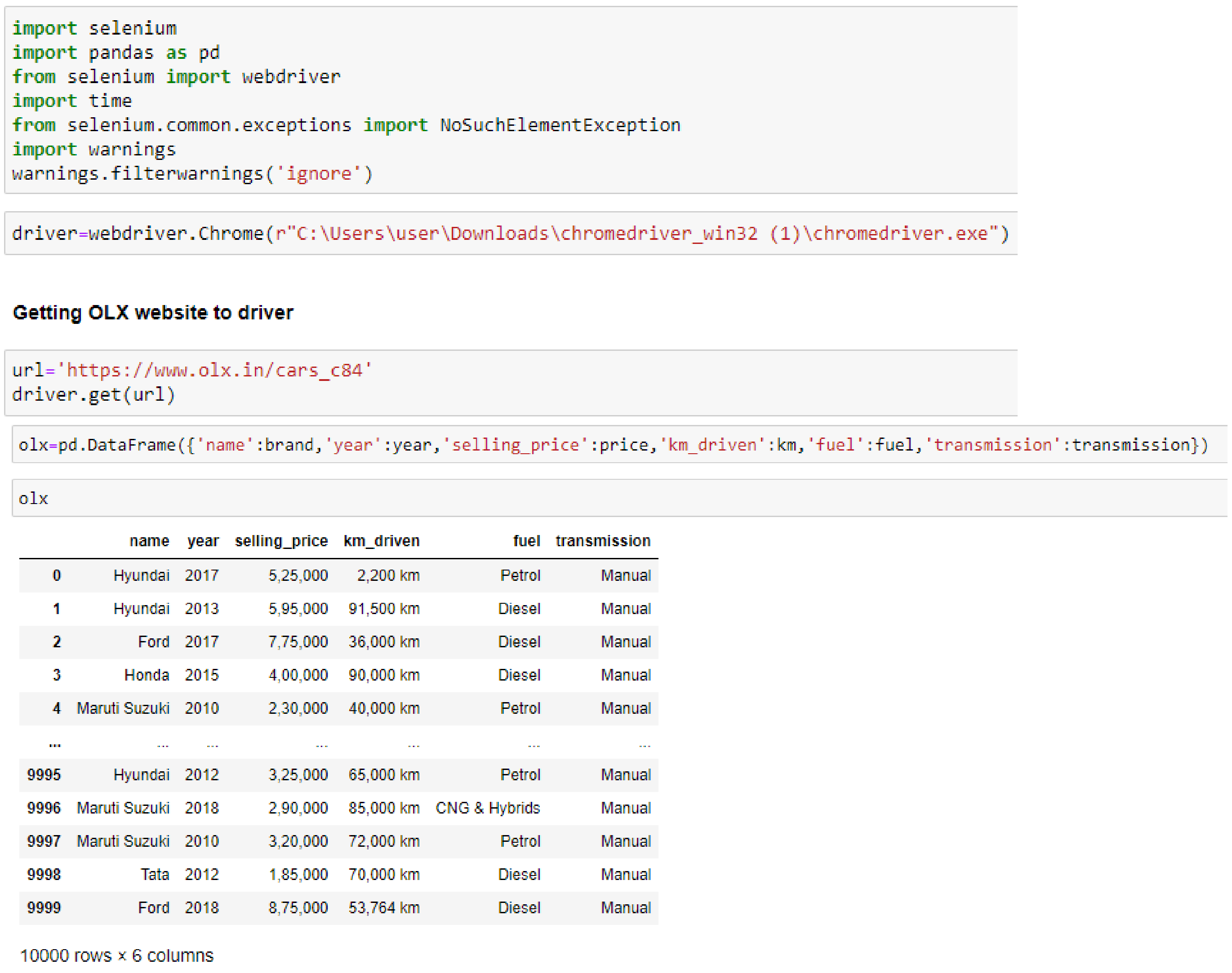
1. Data Cleaning
2. Exploratory Data Analysis
3. Data Pre-processing
4. Model Building
5. Model Evaluation
6. Selecting the best model

## DATA SOURCES AND THEIR FORMATS

➔We collected the data from difference websites like olx and cars24. The data is scrapped using Web scraping technique and the framework used is Selenium.

➔We scrapped nearly 10000 of the data and saved each data in a separate data frame

➔In the end, we combined all the data frames into a single data frame and it looks like as follows:



```
import selenium
import pandas as pd
from selenium import webdriver
import time
from selenium.common.exceptions import NoSuchElementException
import warnings
warnings.filterwarnings('ignore')
```

```
driver=webdriver.Chrome(r"C:\Users\user\Downloads\chromedriver_win32 (1)\chromedriver.exe")
```

**Getting OLX website to driver**

```
url='https://www.olx.in/cars_c84'
driver.get(url)
```

```
olx=pd.DataFrame({'name':brand,'year':year,'selling_price':price,'km_driven':km,'fuel':fuel,'transmission':transmission})
```

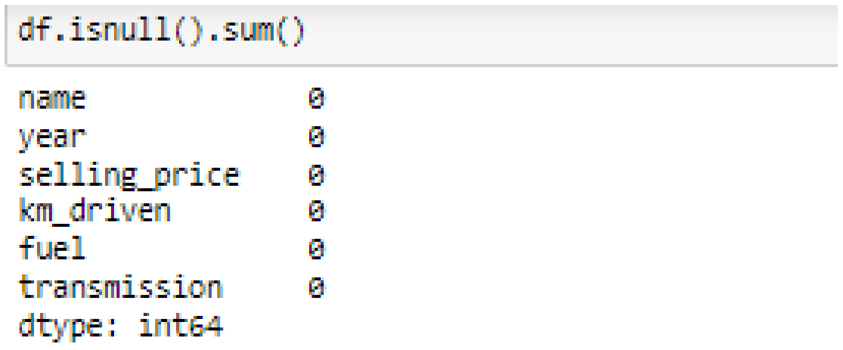
```
olx
```

| | name | year | selling_price | km_driven | fuel | transmission |
|---|---|---|---|---|---|---|
| **0** | Hyundai | 2017 | 5,25,000 | 2,200 km | Petrol | Manual |
| **1** | Hyundai | 2013 | 5,95,000 | 91,500 km | Diesel | Manual |
| **2** | Ford | 2017 | 7,75,000 | 36,000 km | Diesel | Manual |
| **3** | Honda | 2015 | 4,00,000 | 90,000 km | Diesel | Manual |
| **4** | Maruti Suzuki | 2010 | 2,30,000 | 40,000 km | Petrol | Manual |
| **...** | ... | ... | ... | ... | ... | ... |
| **9995** | Hyundai | 2012 | 3,25,000 | 65,000 km | Petrol | Manual |
| **9996** | Maruti Suzuki | 2018 | 2,90,000 | 85,000 km | CNG & Hybrids | Manual |
| **9997** | Maruti Suzuki | 2010 | 3,20,000 | 72,000 km | Petrol | Manual |
| **9998** | Tata | 2012 | 1,85,000 | 70,000 km | Diesel | Manual |
| **9999** | Ford | 2018 | 8,75,000 | 53,764 km | Diesel | Manual |

10000 rows × 6 columns

## DATA PRE-PROCESSING

## CHECKING NULL VALUES



```
df.isnull().sum()
```

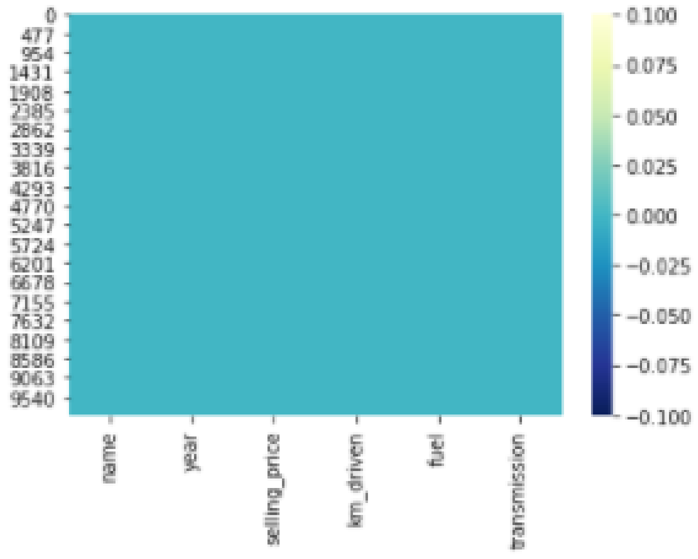
```
name             0
year             0
selling_price    0
km_driven        0
fuel             0
transmission     0
dtype: int64
```

There are no null values in the given dataset.

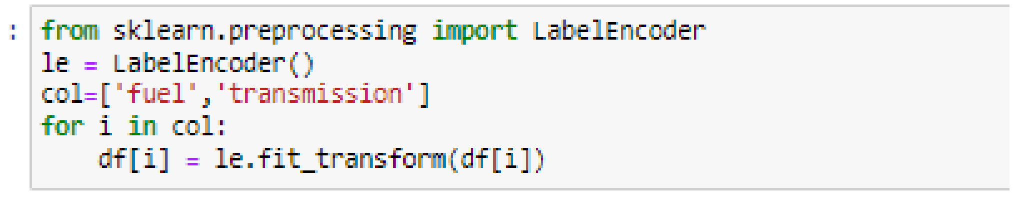
PLOTTING HEATMAP FOR  MISSING DATA

```
: sns.heatmap(df.isnull(), cmap='YlGnBu_r')
```

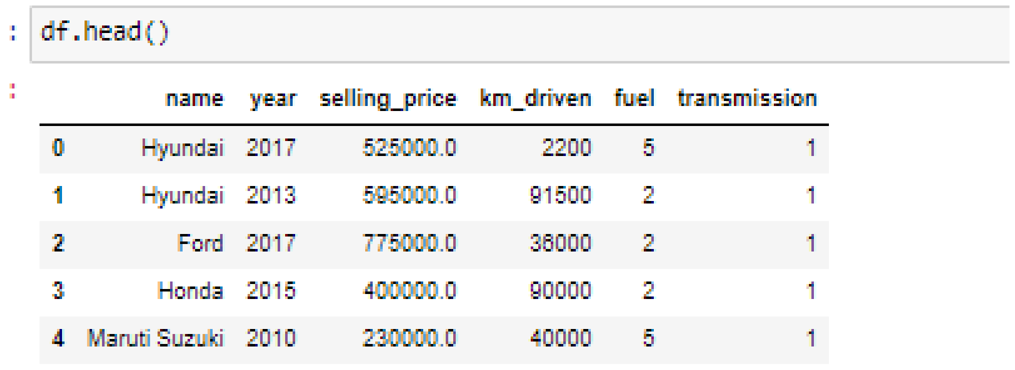
```
: <AxesSubplot:>
```



## Converting categorical data into numeric data



```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
col=['fuel','transmission']
for i in col:
    df[i] = le.fit_transform(df[i])
```

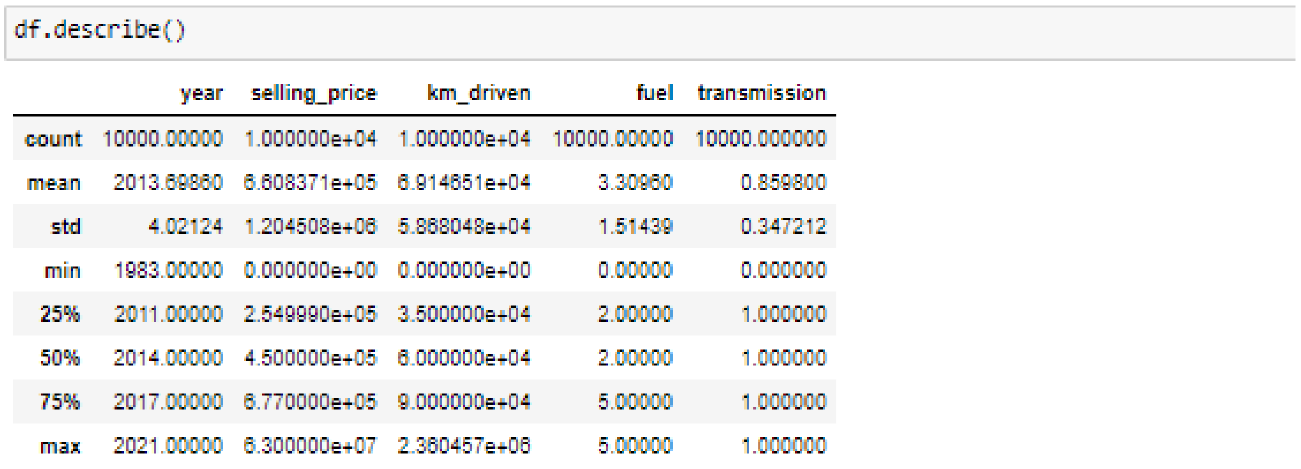


```
df.head()
```

| | name | year | selling_price | km_driven | fuel | transmission |
|---|---|---|---|---|---|---|
| 0 | Hyundai | 2017 | 525000.0 | 2200 | 5 | 1 |
| 1 | Hyundai | 2013 | 595000.0 | 91500 | 2 | 1 |
| 2 | Ford | 2017 | 775000.0 | 36000 | 2 | 1 |
| 3 | Honda | 2015 | 400000.0 | 90000 | 2 | 1 |
| 4 | Maruti Suzuki | 2010 | 230000.0 | 40000 | 5 | 1 |

By using label encoder, we had converted object data into numerical data

## Statistical summary of the dataset



```
df.describe()
```

| | year | selling_price | km_driven | fuel | transmission |
|---|---|---|---|---|---|
| count | 10000.00000 | 1.000000e+04 | 1.000000e+04 | 10000.00000 | 10000.000000 |
| mean | 2013.69860 | 6.608371e+05 | 6.914651e+04 | 3.30960 | 0.859800 |
| std | 4.02124 | 1.204508e+06 | 5.868048e+04 | 1.51439 | 0.347212 |
| min | 1983.00000 | 0.000000e+00 | 0.000000e+00 | 0.00000 | 0.000000 |
| 25% | 2011.00000 | 2.549990e+05 | 3.500000e+04 | 2.00000 | 1.000000 |
| 50% | 2014.00000 | 4.500000e+05 | 6.000000e+04 | 2.00000 | 1.000000 |
| 75% | 2017.00000 | 6.770000e+05 | 9.000000e+04 | 5.00000 | 1.000000 |
| max | 2021.00000 | 6.300000e+07 | 2.360457e+06 | 5.00000 | 1.000000 |

Observations:

1.There is not much difference between the mean and the median

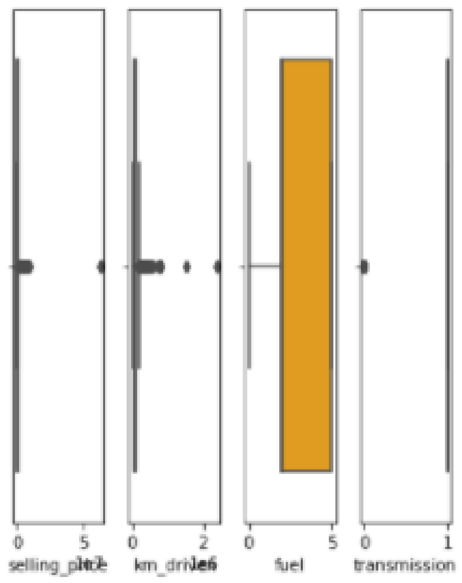
2.The minimum value is 0 for all other columns.

3.The 75th percentile and max value difference for km_driven column is high and it is due to the presence of outliers.

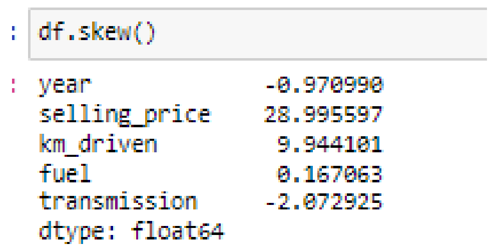
## Checking outliers

```
collist=df.columns.values
ncol=20
nrows=20
```

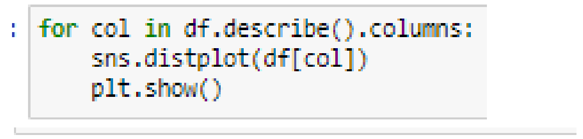
```
plt.figure(figsize=(ncol,5*ncol))
for i in range (1,len(collist)):
    plt.subplot(nrows,ncol,i+1)
    sns.boxplot(df[collist[i]],color='orange',orient='v')
    plt.tight_layout()
```



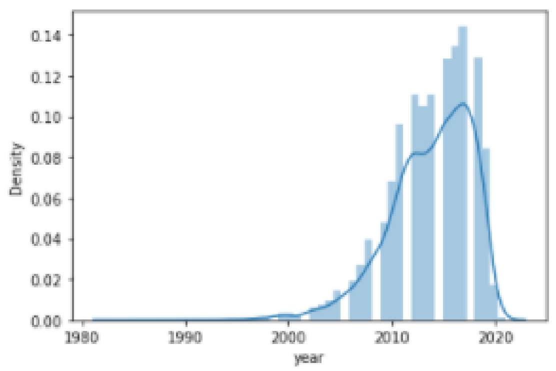
## Checking skewness and data distribution

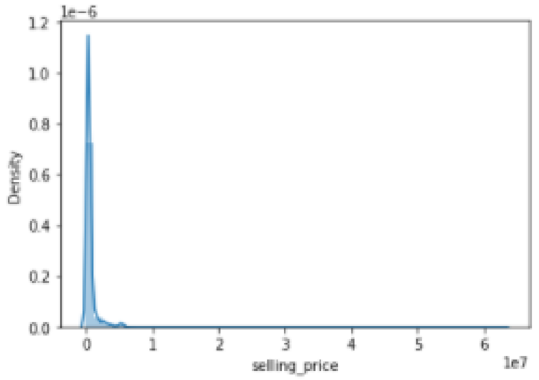


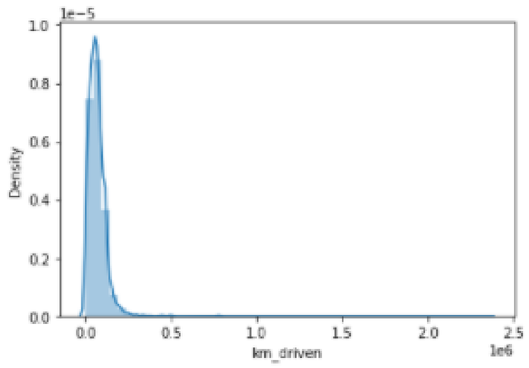
```
: df.skew()

: year             -0.970990
  selling_price    28.995597
  km_driven         9.944101
  fuel              0.167063
  transmission     -2.072925
  dtype: float64
```

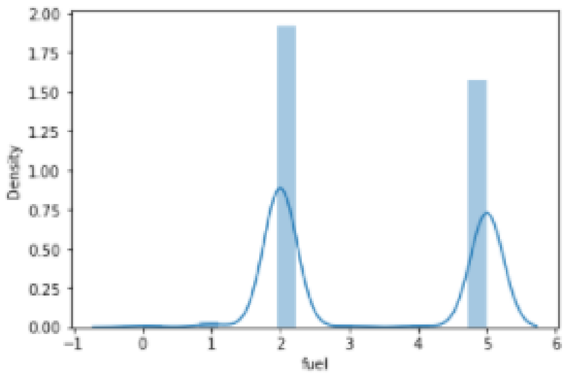


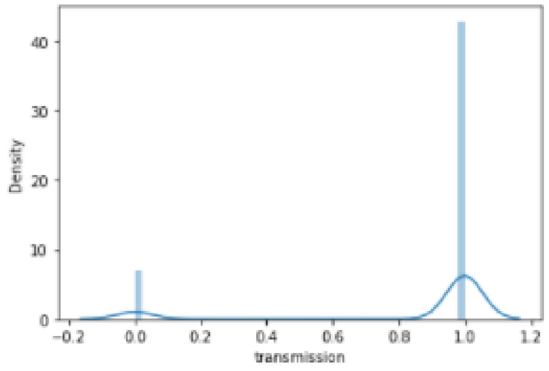
```
: for col in df.describe().columns:
      sns.distplot(df[col])
      plt.show()
```



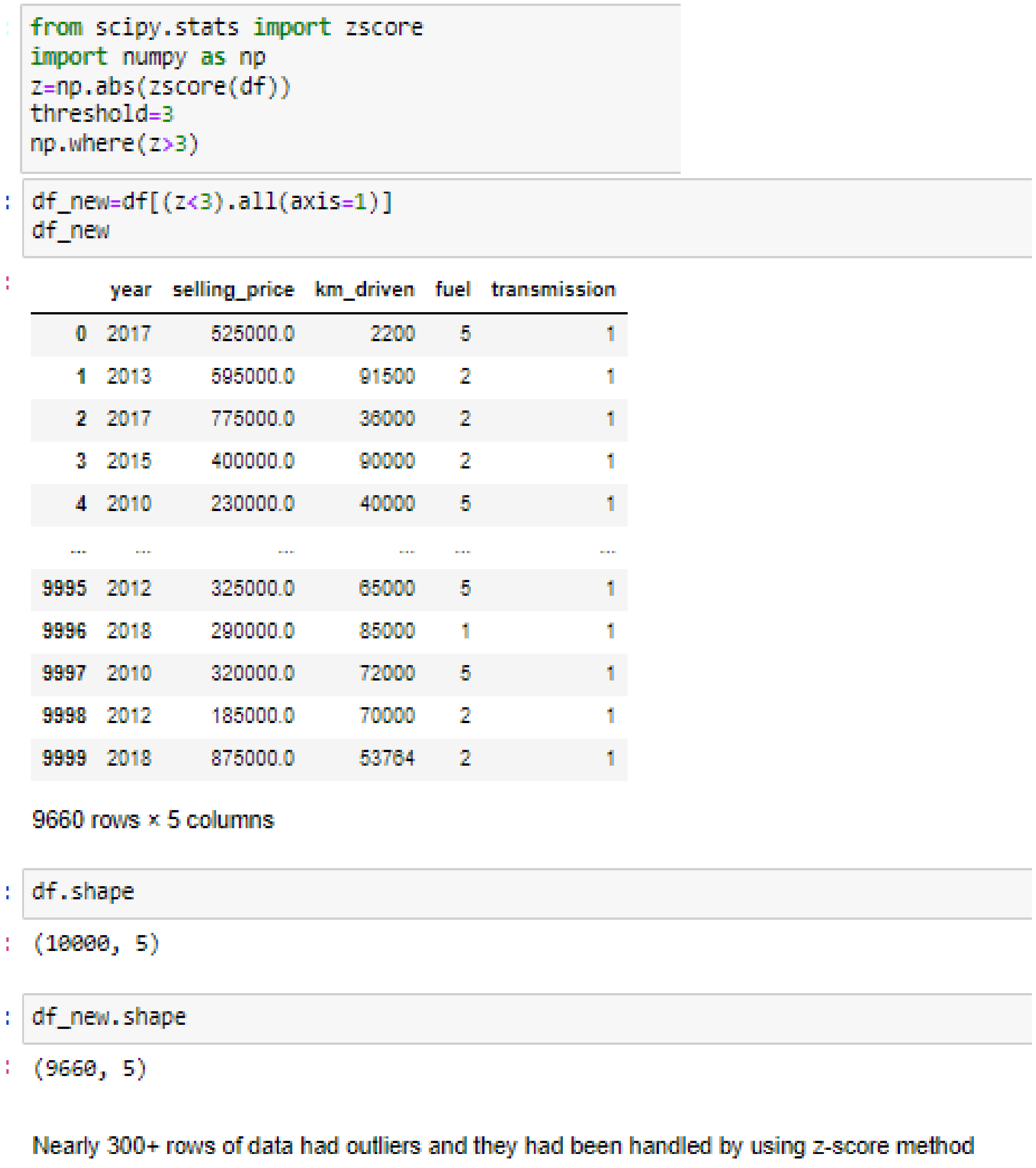








## Handling outliers using z-score method



```
from scipy.stats import zscore
import numpy as np
z=np.abs(zscore(df))
threshold=3
np.where(z>3)
```

```
df_new=df[(z<3).all(axis=1)]
df_new
```

| | year | selling_price | km_driven | fuel | transmission |
|---|---|---|---|---|---|
| **0** | 2017 | 525000.0 | 2200 | 5 | 1 |
| **1** | 2013 | 595000.0 | 91500 | 2 | 1 |
| **2** | 2017 | 775000.0 | 36000 | 2 | 1 |
| **3** | 2015 | 400000.0 | 90000 | 2 | 1 |
| **4** | 2010 | 230000.0 | 40000 | 5 | 1 |
| ... | ... | ... | ... | ... | ... |
| **9995** | 2012 | 325000.0 | 65000 | 5 | 1 |
| **9996** | 2018 | 290000.0 | 85000 | 1 | 1 |
| **9997** | 2010 | 320000.0 | 72000 | 5 | 1 |
| **9998** | 2012 | 185000.0 | 70000 | 2 | 1 |
| **9999** | 2018 | 875000.0 | 53764 | 2 | 1 |

9660 rows × 5 columns

```
df.shape
```

```
(10000, 5)
```

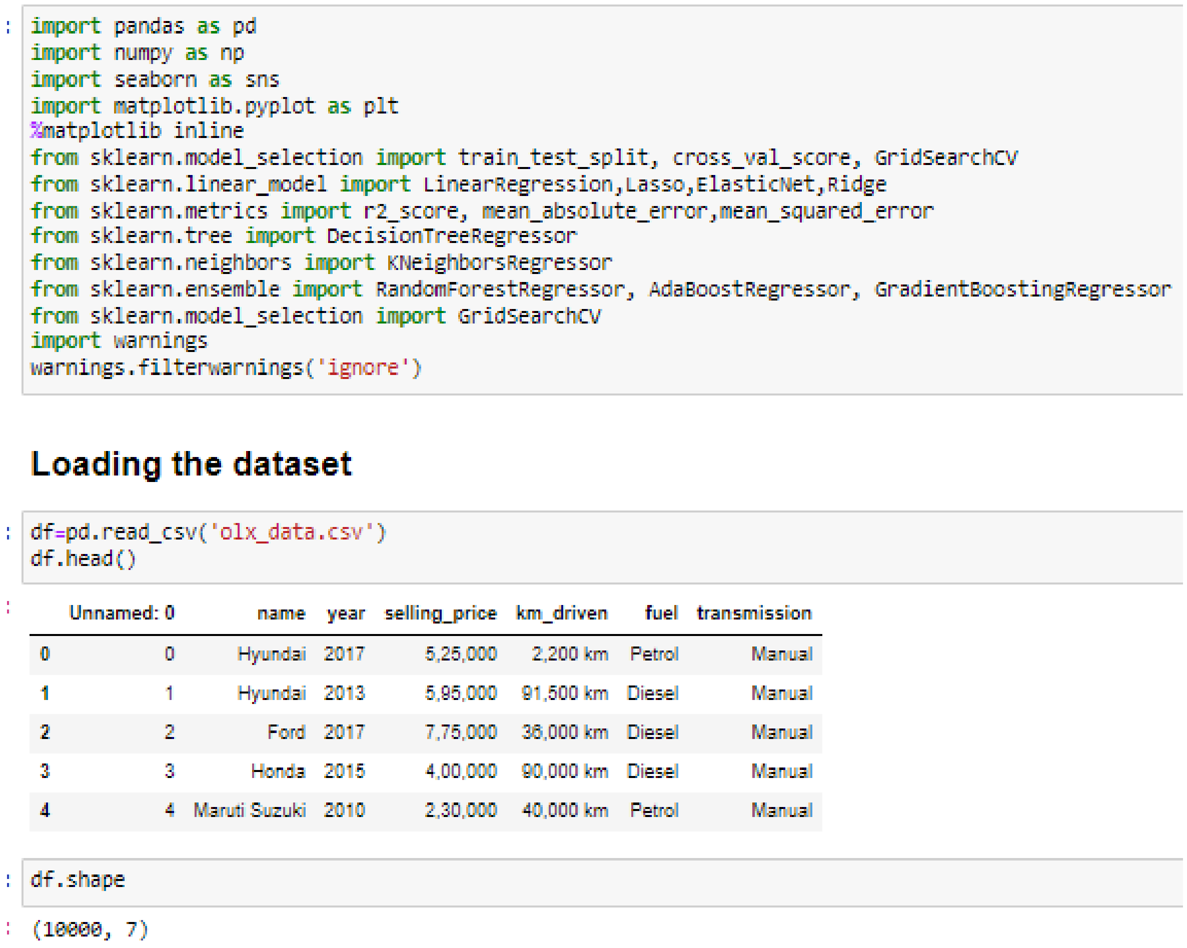
```
df_new.shape
```

```
(9660, 5)
```

Nearly 300+ rows of data had outliers and they had been handled by using z-score method

## HARDWARE AND SOFTWARE REQUIREMENTS AND TOOLS USED

For doing this project, the hardware used is a laptop with high end specification and a stable internet connection. While coming to software part, I had used anaconda navigator and in that I have used **Jupyter notebook** to do my python programming and analysis.
For using a CSV file, Microsoft excel is needed. In Jupyter notebook, I had used lots of python libraries to carry out this project and I have mentioned below with proper justification:



```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn.linear_model import LinearRegression,Lasso,ElasticNet,Ridge
from sklearn.metrics import r2_score, mean_absolute_error,mean_squared_error
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor, GradientBoostingRegressor
from sklearn.model_selection import GridSearchCV
import warnings
warnings.filterwarnings('ignore')
```

### Loading the dataset

```
df=pd.read_csv('olx_data.csv')
df.head()
```

| | Unnamed: 0 | name | year | selling_price | km_driven | fuel | transmission |
|---|---|---|---|---|---|---|---|
| 0 | 0 | Hyundai | 2017 | 5,25,000 | 2,200 km | Petrol | Manual |
| 1 | 1 | Hyundai | 2013 | 5,95,000 | 91,500 km | Diesel | Manual |
| 2 | 2 | Ford | 2017 | 7,75,000 | 36,000 km | Diesel | Manual |
| 3 | 3 | Honda | 2015 | 4,00,000 | 90,000 km | Diesel | Manual |
| 4 | 4 | Maruti Suzuki | 2010 | 2,30,000 | 40,000 km | Petrol | Manual |

```
df.shape
```

```
(10000, 7)
```

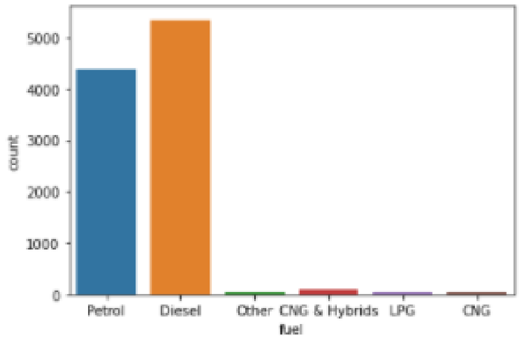
# DATA VISUALIZATION

## Univariate and Bivariate Analysis

```
: print(df['fuel'].value_counts())
  sns.countplot(df['fuel'])
```

```
Diesel            5345
Petrol            4386
CNG & Hybrids      108
CNG                 57
Other               56
LPG                 48
Name: fuel, dtype: int64
```

```
: <AxesSubplot:xlabel='fuel', ylabel='count'>
```

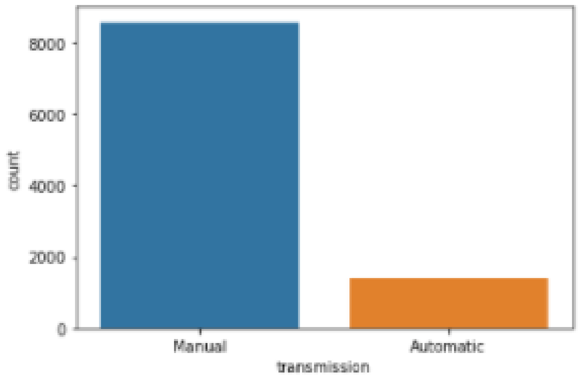


We can see that Diesel and Petrol are the maximum fuels used by cars, whereas CNG and LPG are the least used.

```
print(df['transmission'].value_counts())
sns.countplot(df['transmission'])
```

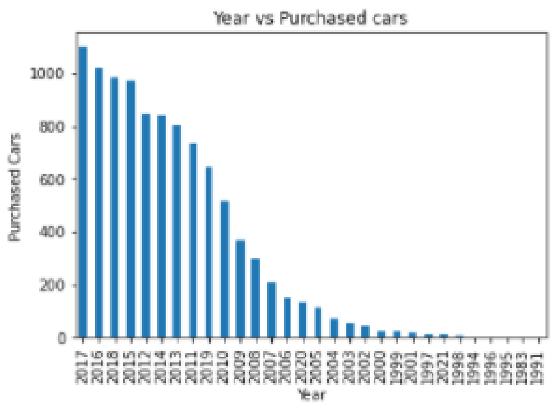
```
Manual       8598
Automatic    1402
Name: transmission, dtype: int64
```

```
<AxesSubplot:xlabel='transmission', ylabel='count'>
```



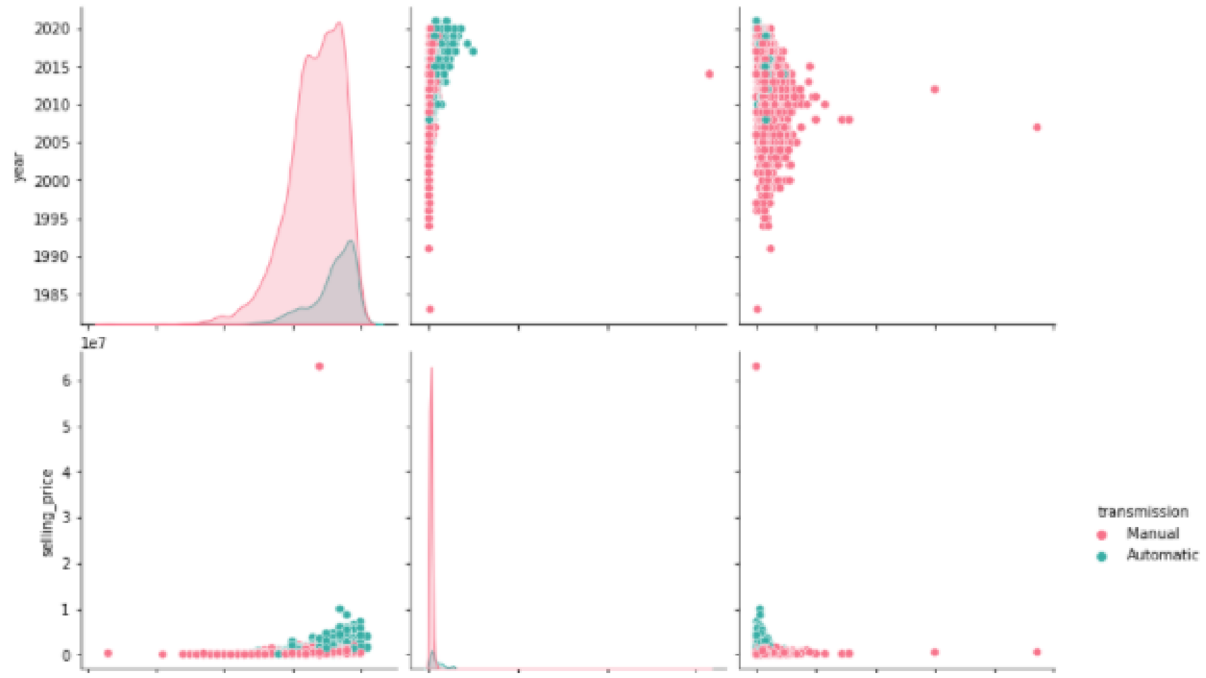
Most of the cars have manual transmission as the highest weightage

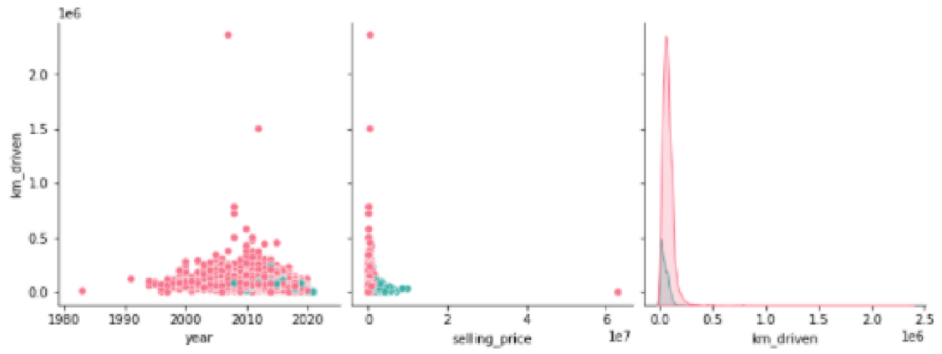
```
print(df['year'].value_counts())
purchased_car_per_year = df['year'].value_counts()
purchased_car_per_year.plot(kind='bar')
plt.xlabel("Year")
plt.ylabel("Purchased Cars")
plt.title("Year vs Purchased cars")
plt.show()
```



Maximum number of cars are bought in the year 2017 whereas minimum number of cars were brought in 1991 and 1983.

```
sns.pairplot(df,hue = 'transmission',diag_kind = "kde",kind = "scatter",palette = "husl",height=3.5)
plt.show()
```

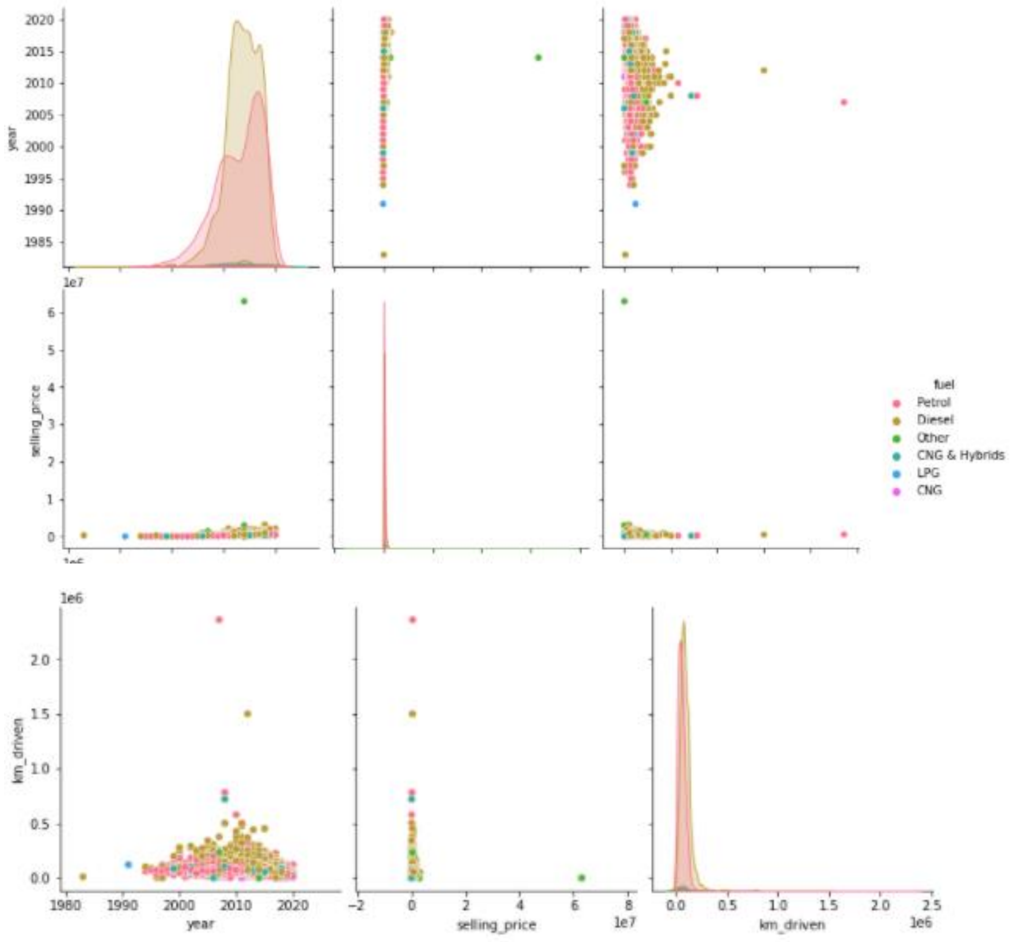




We can see that manual cars have higher price range than automatic type cars though the distribution were skewed to the right. We can also see the increase of cars between 2010 and 2020.

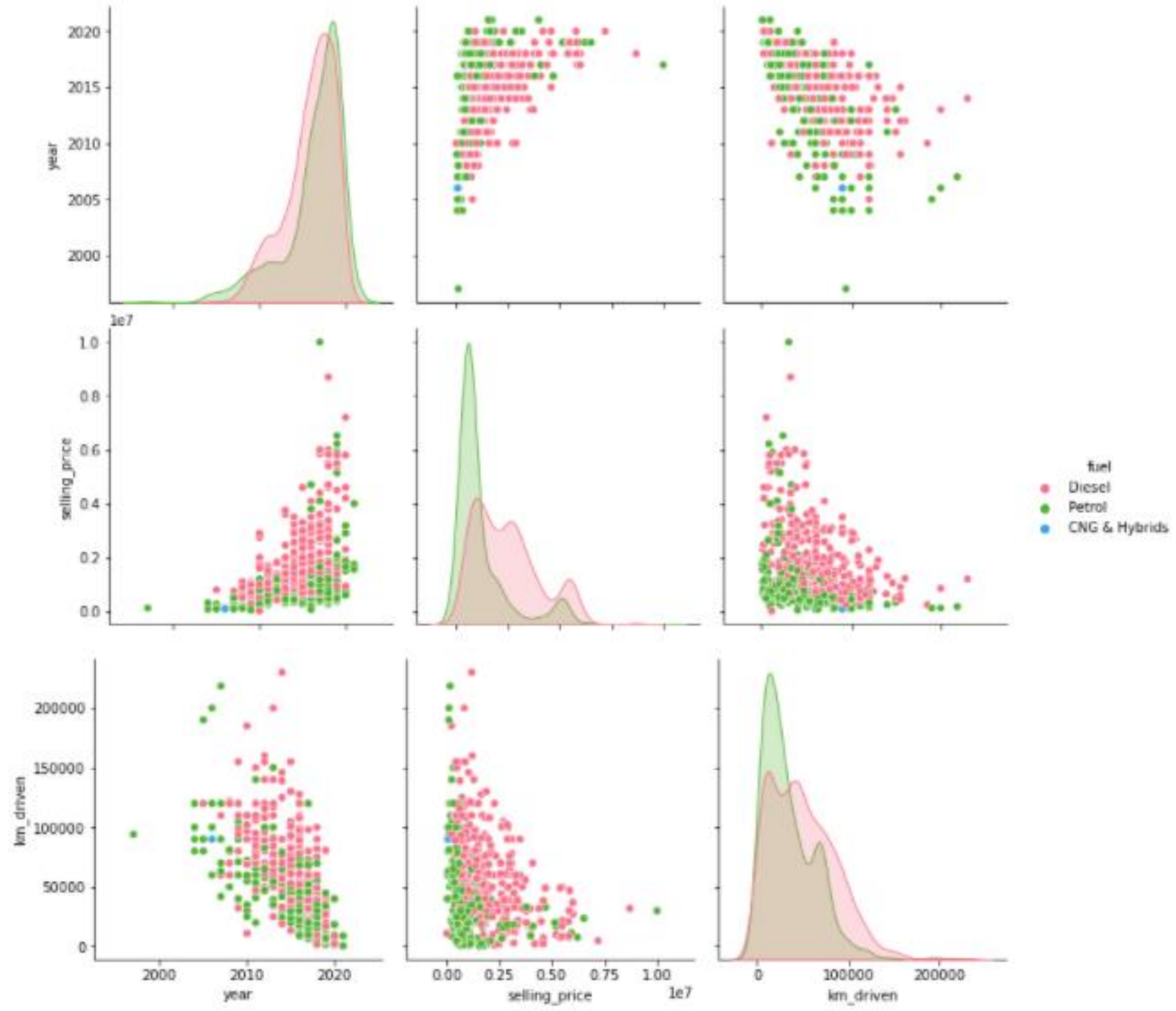
```
manual = df[df['transmission']=='Manual']
automatic = df[df['transmission']=='Automatic']
```

```
print('Manual type car')
sns.pairplot(manual,hue = 'fuel',diag_kind = "kde",kind = "scatter",palette = "husl",height=3.5)
plt.show()
```



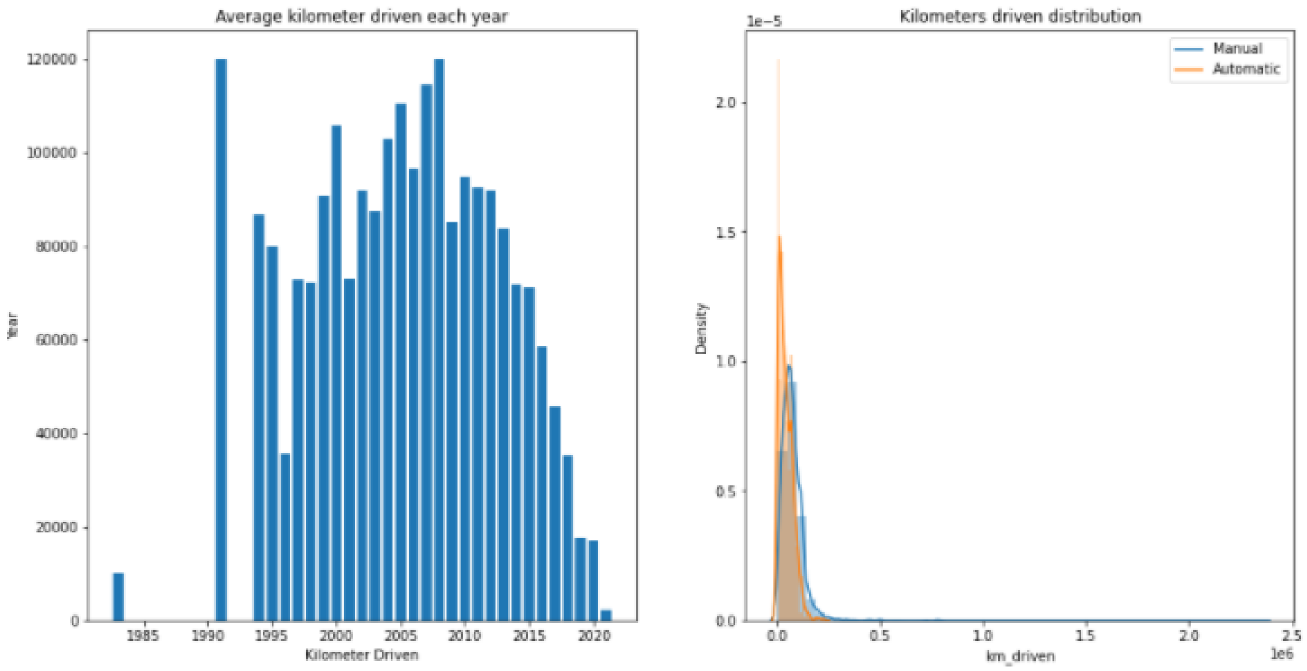
```
print('Automatic type car')
sns.pairplot(automatic,hue = 'fuel',diag_kind = "kde",kind = "scatter",palette = "husl",height=3.5)
plt.show()
```

```
Automatic type car
```



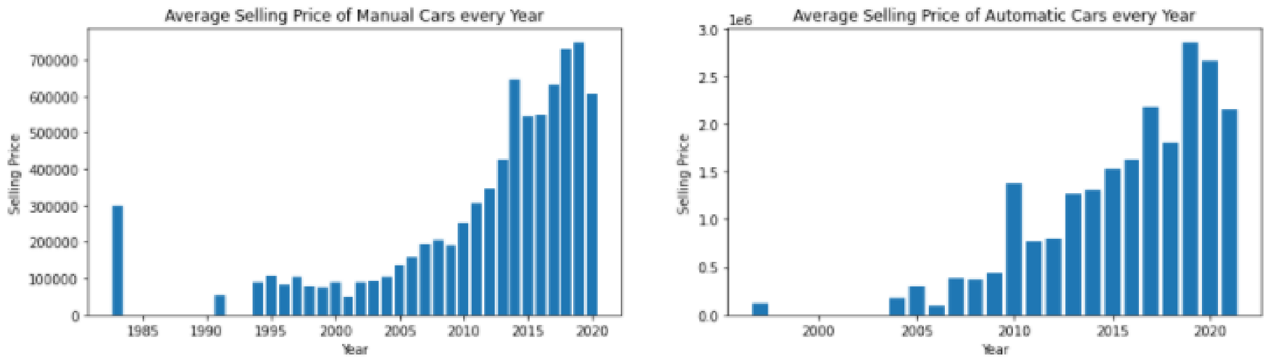
/Ve can see that the selling price of diesel type cars in both manual and automatic were more spread than diesel and other fuel hence getting higher average ınd range of selling price.

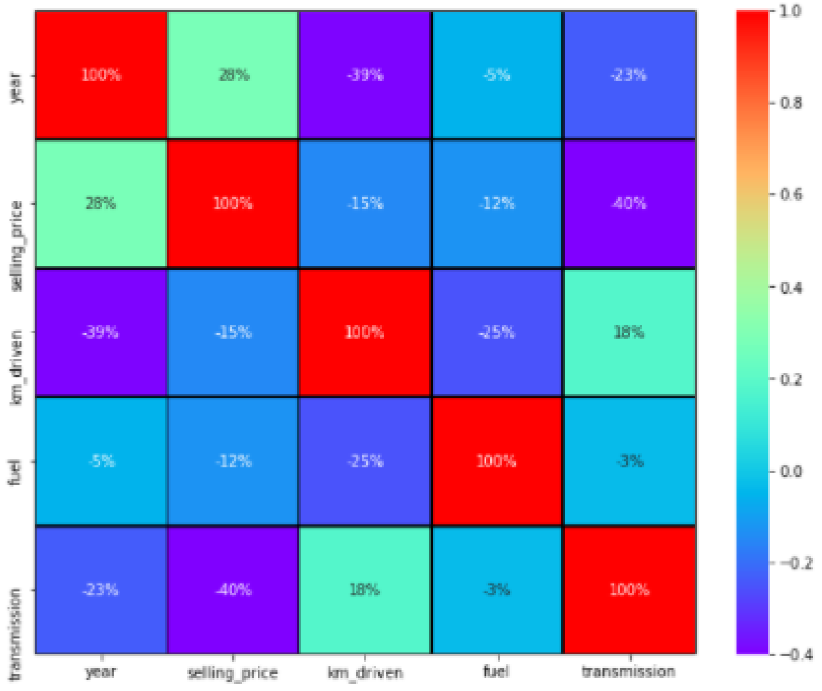
```
km_mean = df.groupby('year')['km_driven'].mean()

fig,ax = plt.subplots(nrows=1,ncols=2,figsize=(16,8))

ax[0].bar(km_mean.index,km_mean)
sns.distplot(manual['km_driven'],ax=ax[1])
sns.distplot(automatic['km_driven'],ax=ax[1])

ax[0].set_title('Average kilometer driven each year')
ax[0].set_xlabel('Kilometer Driven')
ax[0].set_ylabel('Year')

ax[1].set_title('Kilometers driven distribution')
ax[1].legend(['Manual','Automatic'])

plt.show()
```



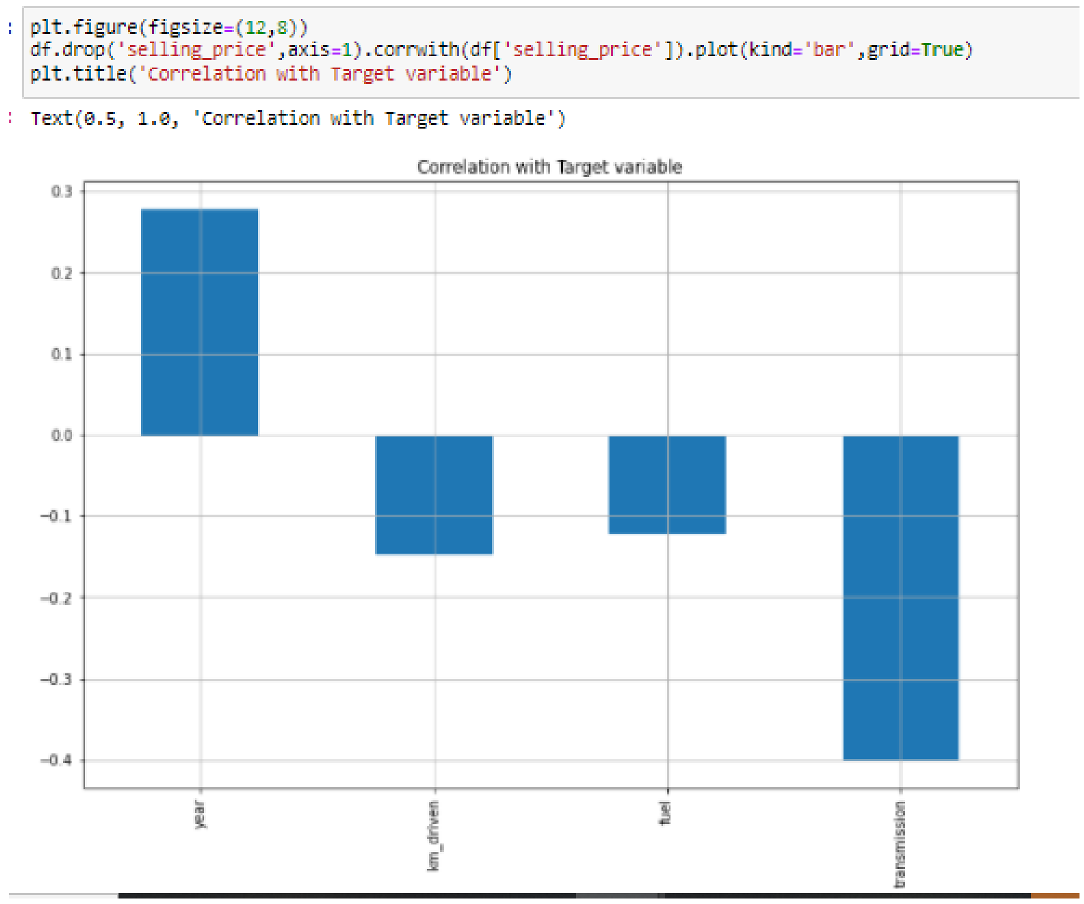
Ve can see that average kilometers driven rises up from 1995 until 2008 and linearly goes down until 2020. We can also see some outliers present in the listribution plot.

```
year_mean_manual = df[df['transmission']=='Manual'].groupby('year')['selling_price'].mean()
year_mean_automatic = df[df['transmission']=='Automatic'].groupby('year')['selling_price'].mean()

fig,ax = plt.subplots(nrows=1,ncols=2,figsize=(16,4))

ax[0].bar(year_mean_manual.index,year_mean_manual)
ax[1].bar(year_mean_automatic.index,year_mean_automatic)

ax[0].set_title('Average Selling Price of Manual Cars every Year')
ax[0].set_xlabel('Year')
ax[0].set_ylabel('Selling Price')

ax[1].set_title('Average Selling Price of Automatic Cars every Year')
ax[1].set_xlabel('Year')
ax[1].set_ylabel('Selling Price')

plt.show()
```



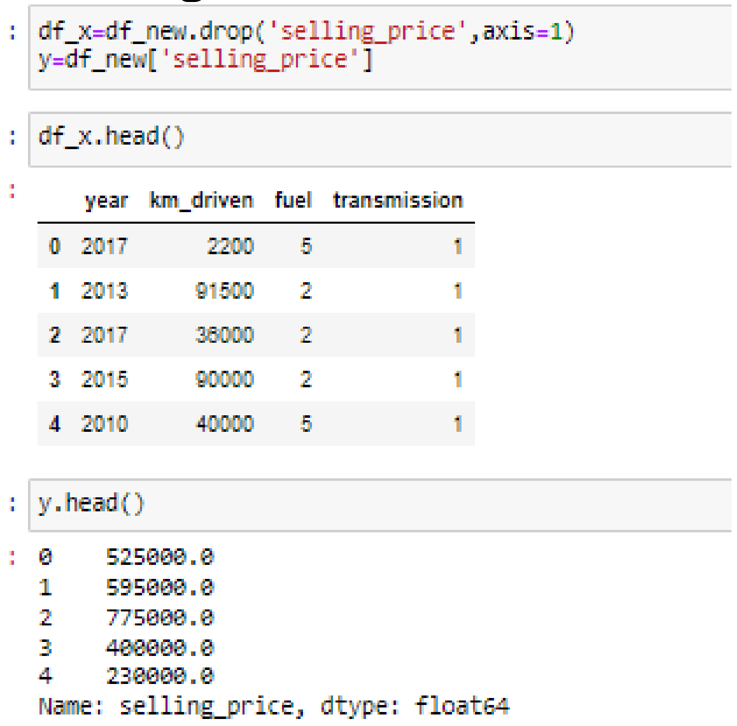


|  | year | selling_price | km_driven | fuel | transmission |
|---|---|---|---|---|---|
| year | 100% | 28% | -39% | -5% | -23% |
| selling_price | 28% | 100% | -15% | -12% | -40% |
| km_driven | -39% | -15% | 100% | -25% | 18% |
| fuel | -5% | -12% | -25% | 100% | -3% |
| transmission | -23% | -40% | 18% | -3% | 100% |


```
: plt.figure(figsize=(12,8))
  df.drop('selling_price',axis=1).corrwith(df['selling_price']).plot(kind='bar',grid=True)
  plt.title('Correlation with Target variable')
```

```
: Text(0.5, 1.0, 'Correlation with Target variable')
```

## Preparing dataset for model training
## Treating



```
df_x=df_new.drop('selling_price',axis=1)
y=df_new['selling_price']
```

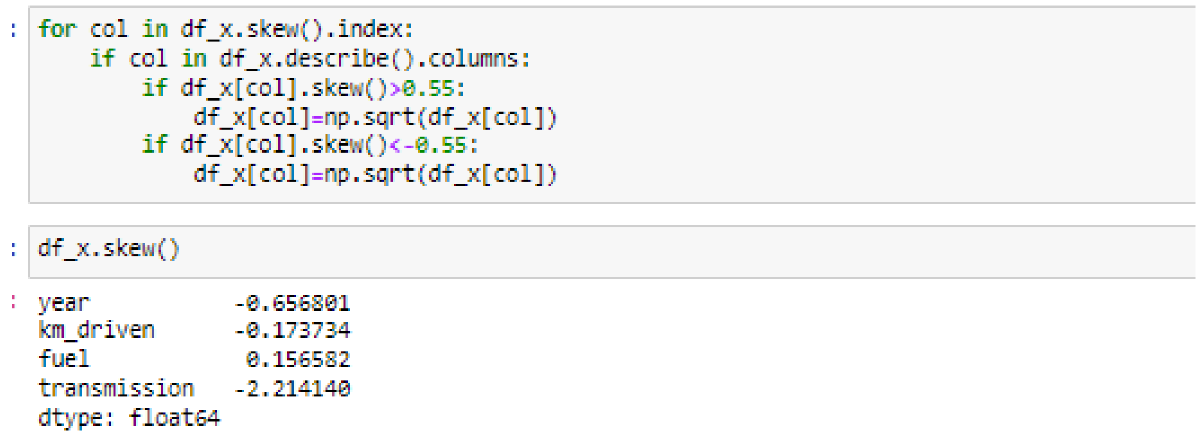
```
df_x.head()
```

|   | year | km_driven | fuel | transmission |
|---|---|---|---|---|
| 0 | 2017 | 2200 | 5 | 1 |
| 1 | 2013 | 91500 | 2 | 1 |
| 2 | 2017 | 36000 | 2 | 1 |
| 3 | 2015 | 90000 | 2 | 1 |
| 4 | 2010 | 40000 | 5 | 1 |

```
y.head()
```

```
0    525000.0
1    595000.0
2    775000.0
3    400000.0
4    230000.0
Name: selling_price, dtype: float64
```

## Treating Skewness



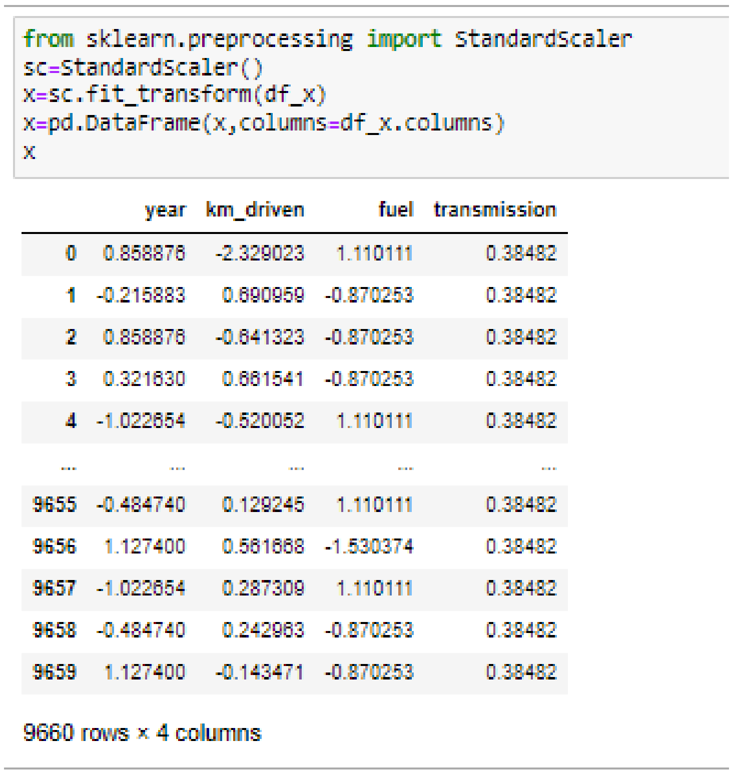
```
for col in df_x.skew().index:
    if col in df_x.describe().columns:
        if df_x[col].skew()>0.55:
            df_x[col]=np.sqrt(df_x[col])
        if df_x[col].skew()<-0.55:
            df_x[col]=np.sqrt(df_x[col])
```

```
df_x.skew()
```

```
year           -0.656801
km_driven      -0.173734
fuel            0.156582
transmission   -2.214140
dtype: float64
```

We can see that skewness has been treated and we can proceed further model building process

## Scaling the data

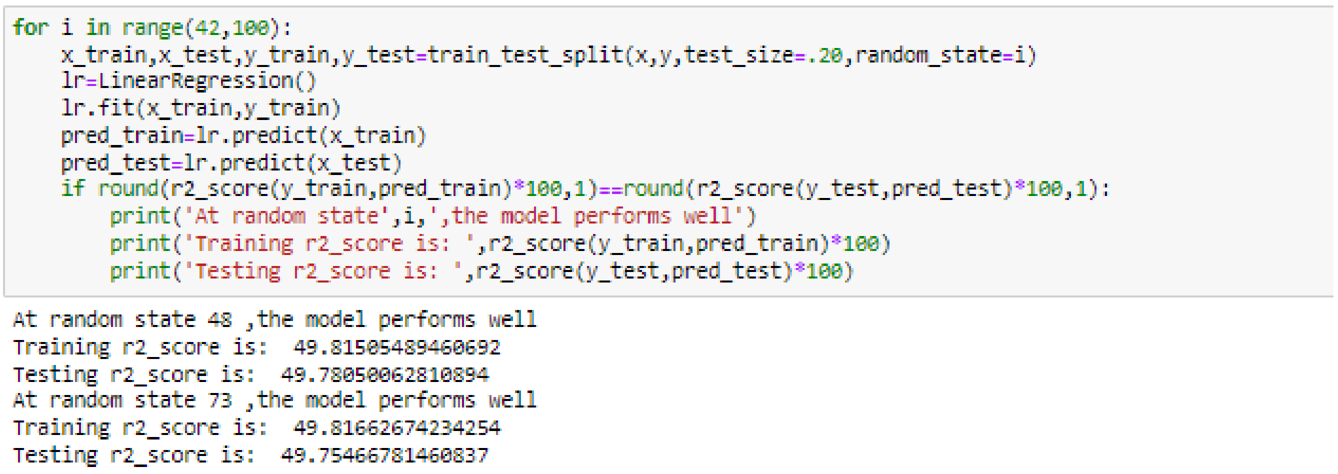


```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x=sc.fit_transform(df_x)
x=pd.DataFrame(x,columns=df_x.columns)
x
```

|  | year | km_driven | fuel | transmission |
|---|---|---|---|---|
| 0 | 0.858876 | -2.329023 | 1.110111 | 0.38482 |
| 1 | -0.215883 | 0.890959 | -0.870253 | 0.38482 |
| 2 | 0.858876 | -0.841323 | -0.870253 | 0.38482 |
| 3 | 0.321630 | 0.661541 | -0.870253 | 0.38482 |
| 4 | -1.022854 | -0.520052 | 1.110111 | 0.38482 |
| ... | ... | ... | ... | ... |
| 9655 | -0.484740 | 0.129245 | 1.110111 | 0.38482 |
| 9656 | 1.127400 | 0.561668 | -1.530374 | 0.38482 |
| 9657 | -1.022854 | 0.287309 | 1.110111 | 0.38482 |
| 9658 | -0.484740 | 0.242983 | -0.870253 | 0.38482 |
| 9659 | 1.127400 | -0.143471 | -0.870253 | 0.38482 |

9660 rows × 4 columns

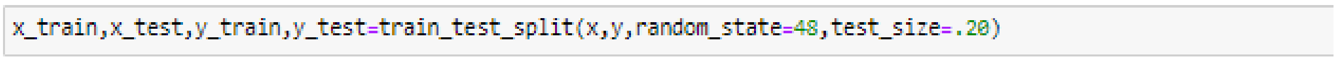
## Model Building



```
for i in range(42,100):
    x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=.20,random_state=i)
    lr=LinearRegression()
    lr.fit(x_train,y_train)
    pred_train=lr.predict(x_train)
    pred_test=lr.predict(x_test)
    if round(r2_score(y_train,pred_train)*100,1)==round(r2_score(y_test,pred_test)*100,1):
        print('At random state',i,',the model performs well')
        print('Training r2_score is: ',r2_score(y_train,pred_train)*100)
        print('Testing r2_score is: ',r2_score(y_test,pred_test)*100)
```

```
At random state 48 ,the model performs well
Training r2_score is:  49.81505489460692
Testing r2_score is:  49.78050062810894
At random state 73 ,the model performs well
Training r2_score is:  49.81662674234254
Testing r2_score is:  49.75466781460837
```

We can see that at random_state=48, the best r2_score is obtained so that we can create our train_test_split with this random state.



```
x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=48,test_size=.20)
```

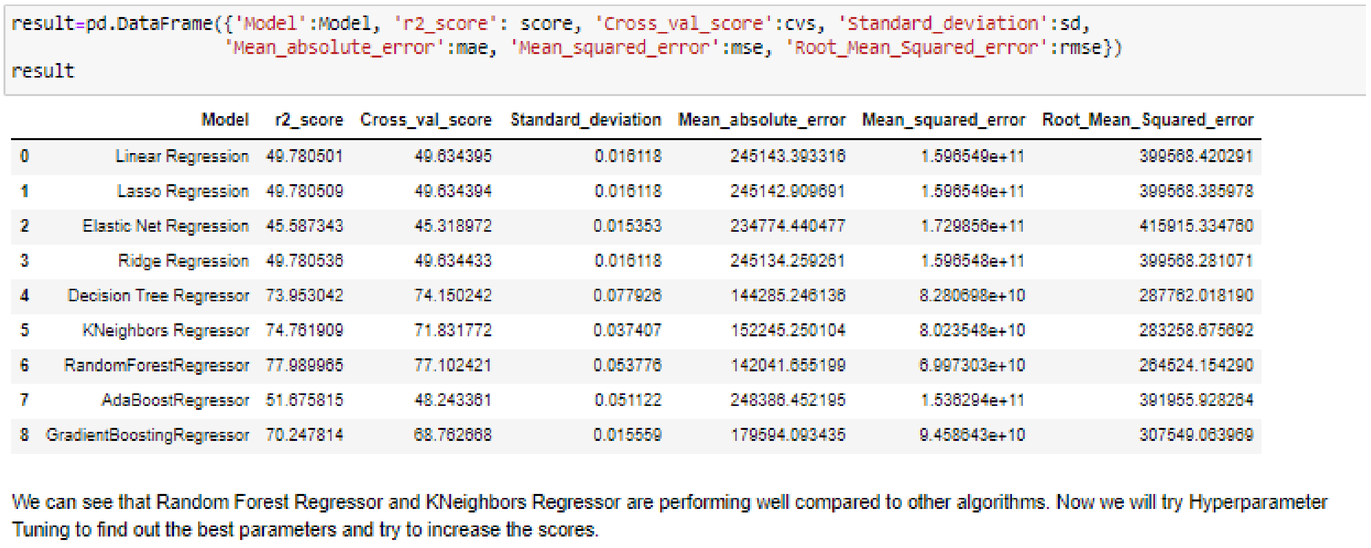
## Finding the best model

```
LR=LinearRegression()
l=Lasso()
en=ElasticNet()
rd=Ridge()
dtr=DecisionTreeRegressor()
knr=KNeighborsRegressor()
rf=RandomForestRegressor()
ab=AdaBoostRegressor()
gb=GradientBoostingRegressor()
```

```
models= []
models.append(('Linear Regression',LR))
models.append(('Lasso Regression',l))
models.append(('Elastic Net Regression',en))
models.append(('Ridge Regression',rd))
models.append(('Decision Tree Regressor',dtr))
models.append(('KNeighbors Regressor',knr))
models.append(('RandomForestRegressor',rf))
models.append(('AdaBoostRegressor',ab))
models.append(('GradientBoostingRegressor',gb))
```

```
Model=[]
score=[]
cvs=[]
sd=[]
mae=[]
mse=[]
rmse=[]
for name,model in models:
    print('*******************************',name,'****************************')
    print('\n')
    Model.append(name)
    model.fit(x_train,y_train)
    print(model)
    pre=model.predict(x_test)
    print('\n')
    AS=r2_score(y_test,pre)
    print('r2_score: ',AS)
    score.append(AS*100)
    print('\n')
    sc=cross_val_score(model,x,y,cv=5,scoring='r2').mean()
    print('cross_val_score: ',sc)
    cvs.append(sc*100)
    print('\n')
    std=cross_val_score(model,x,y,cv=5,scoring='r2').std()
    print('Standard Deviation: ',std)
    sd.append(std)
    print('\n')
    MAE=mean_absolute_error(y_test,pre)
    print('Mean Absolute Error: ',MAE)
    mae.append(MAE)
    print('\n')
    MSE=mean_squared_error(y_test,pre)
    print('Mean Squared Error: ',MSE)
    mse.append(MSE)
    print('\n')
    RMSE=np.sqrt(mean_squared_error(y_test,pre))
    print('Root Mean Squared Error: ',RMSE)
    rmse.append(RMSE)
    print('\n\n')
```

The final outputs of the models we used are stored in a dataframe and it is shown below:



```
result=pd.DataFrame({'Model':Model, 'r2_score': score, 'Cross_val_score':cvs, 'Standard_deviation':sd,
                    'Mean_absolute_error':mae, 'Mean_squared_error':mse, 'Root_Mean_Squared_error':rmse})
result
```

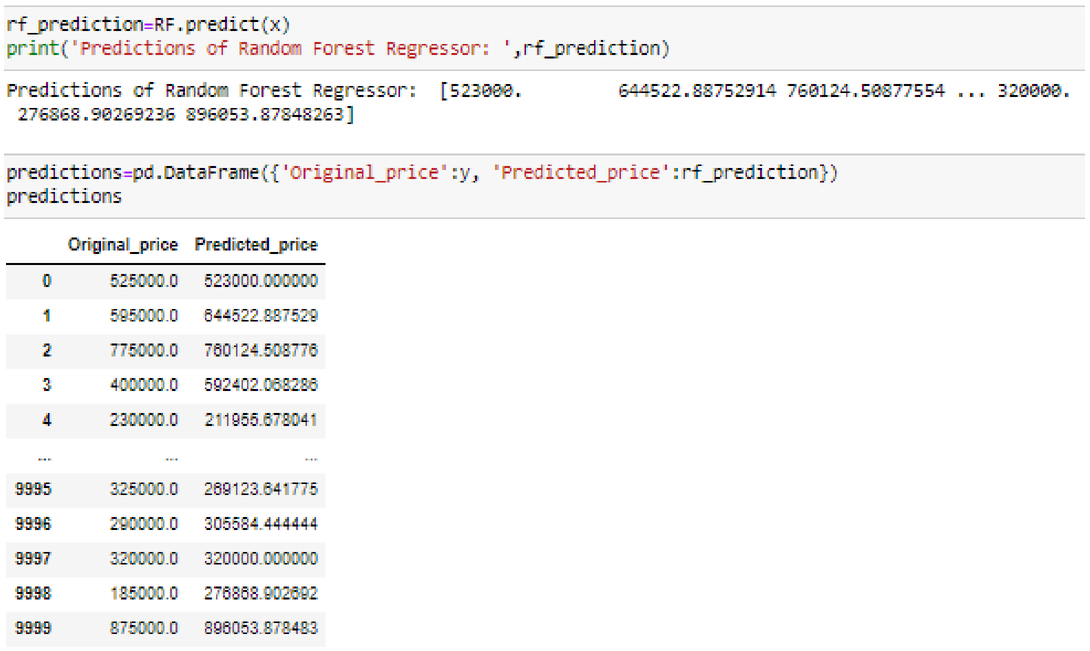
| | Model | r2_score | Cross_val_score | Standard_deviation | Mean_absolute_error | Mean_squared_error | Root_Mean_Squared_error |
|---|---|---|---|---|---|---|---|
| 0 | Linear Regression | 49.780501 | 49.634395 | 0.016118 | 245143.393316 | 1.596549e+11 | 399568.420291 |
| 1 | Lasso Regression | 49.780509 | 49.634394 | 0.016118 | 245142.909691 | 1.596549e+11 | 399568.385978 |
| 2 | Elastic Net Regression | 45.587343 | 45.318972 | 0.015353 | 234774.440477 | 1.729856e+11 | 415915.334760 |
| 3 | Ridge Regression | 49.780538 | 49.634433 | 0.016118 | 245134.259261 | 1.596548e+11 | 399568.281071 |
| 4 | Decision Tree Regressor | 73.953042 | 74.150242 | 0.077926 | 144285.246136 | 8.280698e+10 | 287762.018190 |
| 5 | KNeighbors Regressor | 74.761909 | 71.831772 | 0.037407 | 152245.250104 | 8.023548e+10 | 283258.675692 |
| 6 | RandomForestRegressor | 77.989965 | 77.102421 | 0.053776 | 142041.655199 | 6.997303e+10 | 264524.154290 |
| 7 | AdaBoostRegressor | 51.675815 | 48.243361 | 0.051122 | 248388.452195 | 1.536294e+11 | 391955.928264 |
| 8 | GradientBoostingRegressor | 70.247814 | 68.762668 | 0.015559 | 179594.093435 | 9.458643e+10 | 307549.063969 |

We can see that Random Forest Regressor and KNeighbors Regressor are performing well compared to other algorithms. Now we will try Hyperparameter Tuning to find out the best parameters and try to increase the scores.

Now we will try Hyperparameter Tuning to find out the best parameters and try to increase the scores.

After applying Hyperparameter Tuning, we can see that RandomForestRegressor is the best performing algorithm among all other algorithms. It has also the less amount of error values obtained. Lesser the RMSE score, the better the model. Now we will finalize the model.

# FINALISING THE MODEL

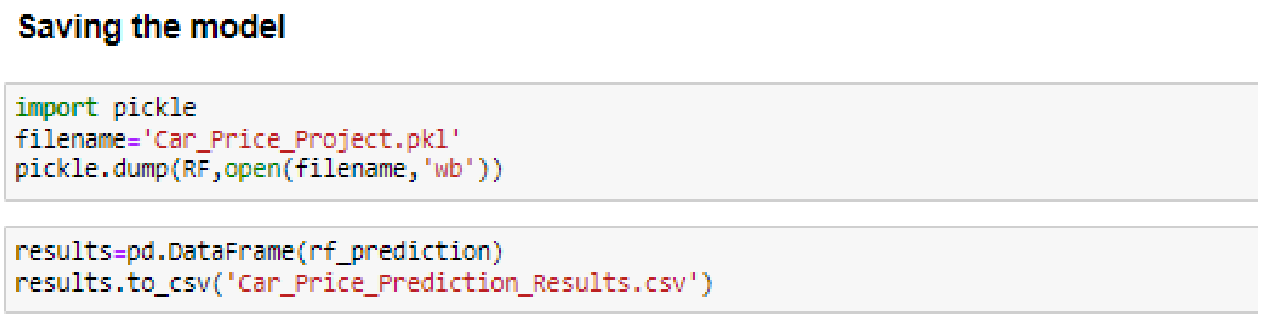


```
rf_prediction=RF.predict(x)
print('Predictions of Random Forest Regressor: ',rf_prediction)
```

```
Predictions of Random Forest Regressor:  [523000.         644522.88752914 760124.50877554 ... 320000.
 276868.90269236 896053.87848263]
```

```
predictions=pd.DataFrame({'Original_price':y, 'Predicted_price':rf_prediction})
predictions
```

| | Original_price | Predicted_price |
|---|---|---|
| 0 | 525000.0 | 523000.000000 |
| 1 | 595000.0 | 644522.887529 |
| 2 | 775000.0 | 760124.508776 |
| 3 | 400000.0 | 592402.068286 |
| 4 | 230000.0 | 211955.678041 |
| ... | ... | ... |
| 9995 | 325000.0 | 289123.641775 |
| 9996 | 290000.0 | 305584.444444 |
| 9997 | 320000.0 | 320000.000000 |
| 9998 | 185000.0 | 276868.902692 |
| 9999 | 875000.0 | 896053.878483 |



## Saving the model

```
import pickle
filename='Car_Price_Project.pkl'
pickle.dump(RF,open(filename,'wb'))
```

```
results=pd.DataFrame(rf_prediction)
results.to_csv('Car_Price_Prediction_Results.csv')
```

# CONCLUSION

**Key Findings and Conclusions of the Study**

-> After the completion of this project, we got an insight of how to collect data, pre-processing the data, analysing the data and building a model.

-> First, we collected the used cars data from different websites like olx, cardekho and it was done by using Web scraping. The framework used for web scraping was Selenium, which has an advantage of automating our process of collecting data.

-> We collected almost 10000 of data which contained the selling price and other related features.

-> Then, the scrapped data was combined in a single data frame and saved in a csv file so that we can open it and analyse the data.

-> We did data cleaning, data-preprocessing steps like finding and handling null values, removing words from numbers, converting object to int type, data visualization, handling outliers, etc.

-> After separating our train and test data, we started running different machine learning classification algorithms to find out the best performing model.

-> We found that RandomForest and KNeighbors Algorithms were performing well, according to their r2_score and cross val scores.

-> Then, we performed Hyperparameter Tuning techniques using GridSearchCV for getting the best parameters and improving the scores. In that, RandomForestRegressor performed well and we finalised that model.

-> We saved the model in pkl format and then saved the predicted values in a csv format.

THANKYOU