Project Name: Car Price Prediction

Author: Laxmikant walzade

- **Problem Definition.**
- 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.
- This dataset include following columns:-
- 'Price', 'Name', 'Kilometers_Driven', 'Last_Service', 'Registration', 'Registered_in', 'Fuel_Type', 'Transmission', 'Insurance', 'Airbags', 'Seat_Upholstery', 'Integrated_Music', 'Rear_View_Mirrors', 'Engine_start_stop', 'Central_Locking', 'Sunroof_Moonroof', 'Rear_AC', 'Power_Windows', 'Headlamps', 'Fuel_type', 'Engine_type', 'Drivetrain', 'Mileage', 'Steering_type', 'Transmission_type', 'Max_power', 'Fuel_tank_capacity', 'Seating_capacity', 'Alternate_fuel_type', 'History'

- **Data Analysis.**
  DATASET

	Price	Name	Kilometers_Driven	Last_Service	Registration	Registered_in	Fuel_Type	Transmission	Insurance	Airbags	...
0	Fixed Price\n₹3,37,599	2019 Maruti Alto 800 LXI MANUAL	79,205 km	79,205km (28 Aug 2022)	GJ-17-x-xxxx	May-19	Petrol + CNG	MANUAL	Valid upto May 55660\n3rd Party	NaN	...
1	Fixed Price\n₹10,56,899	2020 Maruti S Cross ZETA AT 1.5 SHVS	11,707 km	11,707km (27 Jul 2022)	WB-06-x-xxxx	Dec-20	Petrol	NaN	Valid upto May 55660\n3rd Party	-	...
2	Fixed Price\n₹15,89,699	2019 Honda Civic VX CVT i-VTEC	13,878 km	13,878km (18 Jul 2022)	MH-02-x-xxxx	NaN	Petrol	NaN	Valid upto May 55660\n3rd Party	4 Airbags (Driver, Front Passenger, Driver Sid...	...
3	Fixed Price\n₹18,79,099	2020 MG HECTOR PLUS SHARP DCT	11,086 km	11,086km (29 Aug 2022)	MH-14-x-xxxx	Aug-20	Petrol	NaN	Valid upto May 55660\n3rd Party	6 Airbags (Driver, Front Passenger, 2 Curtain,...	...
4	Fixed Price\n₹7,08,299	2018 Maruti Swift ZXI AMT AUTOMATIC	41,249 km	41,249km (08 Sep 2022)	TN-14-x-xxxx	Aug-18	Petrol	AUTOMATIC	Valid upto May 55660\n3rd Party	2 Airbags (Driver, Front Passenger)	...

- We have total 30 columns including the label i.e Price column.

- **Pre-Processing Steps**

1. Identifying sources of the data
2. Analysing the information
3. Cleaning and handling the information
4. Selecting the most significant elements
5. Writing down findings and observations
6. Using various models to train the data
7. Selecting the best-fitted model for predictions
8. Predicting results for test information

- **Pre-Processing Pipeline.**

- First let's check the data type of the dataset.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4409 entries, 0 to 4408
Data columns (total 30 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Price                4409 non-null   object
 1   Name                 4409 non-null   object
 2   Kilometers_Driven    4409 non-null   object
 3   Last_Service         4409 non-null   object
 4   Registration         4409 non-null   object
 5   Registered_in        3844 non-null   object
 6   Fuel_Type            4409 non-null   object
 7   Transmission         4279 non-null   object
 8   Insurance            4409 non-null   object
 9   Airbags              2290 non-null   object
 10  Seat_Upholstery      900 non-null    object
 11  Integrated_Music     3580 non-null   object
 12  Rear_View_Mirrors    362 non-null    object
 13  Engine_start_stop    1502 non-null   object
 14  Central_Locking      3925 non-null   object
 15  Sunroof_Moonroof     681 non-null    object
 16  Rear_AC              1328 non-null   object
 17  Power_Windows        4257 non-null   object
 18  Headlamps            4125 non-null   object
 19  Fuel_type            4409 non-null   object
 20  Engine_type          4190 non-null   object
 21  Drivetrain           4300 non-null   object
 22  Mileage              4349 non-null   object
 23  Steering_type        4170 non-null   object
 24  Transmission_type    4409 non-null   object
 25  Max_power            4409 non-null   object
 26  Fuel_tank_capacity   4397 non-null   object
 27  Seating_capacity     4409 non-null   object
 28  Alternate_fuel_type  368 non-null    object
 29  History              4409 non-null   object
dtypes: object(30)
memory usage: 1.0+ MB
```

All columns are Object type data which needs to change to Integer, since it is important for model building as model does not consider the string value.

- There are no Null Values present in the dataset so we can move further.

```
for i in df.columns:
    a = df[i].isna().sum()
    if a > 0:
        print(i,'column has',a,'NaN values')
```

```
Registered_in column has 565 NaN values
Transmission column has 130 NaN values
Airbags column has 2119 NaN values
Seat_Upholstery column has 3509 NaN values
Integrated_Music column has 829 NaN values
Rear_View_Mirrors column has 4047 NaN values
Engine_start_stop column has 2907 NaN values
Central_Locking column has 484 NaN values
Sunroof_Moonroof column has 3728 NaN values
Rear_AC column has 3081 NaN values
Power_Windows column has 152 NaN values
Headlamps column has 284 NaN values
Engine_type column has 219 NaN values
Drivetrain column has 109 NaN values
Mileage column has 60 NaN values
Steering_type column has 239 NaN values
Fuel_tank_capacity column has 12 NaN values
Alternate_fuel_type column has 4041 NaN values
```

- Sometimes some unwanted things can be found in a dataset which are equivalent to Null values. It is important to take care of such cases.

- Need to clean and drop some columns
    - dropped 'Registered_in', 'Airbags', 'Seat_Upholstery', 'Integrated_Music', 'Rear_View_Mirrors', 'Engine_start_stop', 'Rear_AC' and 'Alternate_fuel_type columnn as it has a lot of NaNs as it may affect the result of the model

```
]:  df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4409 entries, 0 to 4408
Data columns (total 22 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Price               4409 non-null   object
 1   Name                4409 non-null   object
 2   Kilometers_Driven   4409 non-null   object
 3   Last_Service        4409 non-null   object
 4   Registration        4409 non-null   object
 5   Fuel_Type           4409 non-null   object
 6   Transmission        4279 non-null   object
 7   Insurance           4409 non-null   object
 8   Central_Locking     3569 non-null   object
 9   Sunroof_Moonroof    4053 non-null   object
 10  Power_Windows       3900 non-null   object
 11  Headlamps           3768 non-null   object
 12  Fuel_type           4409 non-null   object
 13  Engine_type         3836 non-null   object
 14  Drivetrain          3946 non-null   object
 15  Mileage             3987 non-null   object
 16  Steering_type       3816 non-null   object
 17  Transmission_type   4055 non-null   object
 18  Max_power           4055 non-null   object
 19  Fuel_tank_capacity  4043 non-null   object
 20  Seating_capacity    4055 non-null   object
 21  History             4408 non-null   object
dtypes: object(22)
memory usage: 757.9+ KB
```

- Filling the nan values

```
df['Transmission'] = df['Transmission'].fillna('MANUAL')
df['Sunroof_Moonroof'] = df['Sunroof_Moonroof'].fillna('No')
df['Power_Windows'] = df['Power_Windows'].fillna('Front & Rear')
df['Drivetrain'] = df['Drivetrain'].fillna('FWD')
df['Mileage'] = df['Mileage'].fillna(df['Mileage'].mean())
df['Transmission_type'] = df['Transmission_type'].fillna('Manual')
df['Max_power'] = df['Max_power'].fillna(df['Max_power'].mean())
df['Fuel_tank_capacity'] = df['Fuel_tank_capacity'].fillna(df['Fuel_tank_capacity'].mean())
df['Seating_capacity'] = df['Seating_capacity'].fillna(df['Seating_capacity'].mean())
df['Steering_type'] = df['Steering_type'].fillna('Power assisted (Electric)')
```

Filled NaN with Mode in case of categorical data and Mean in case of continuous data

We have only 18 columns left after EDA

- Data looks fine to encode into Int

```
In [83]:  df[['Price1','Price']] = df['Price'].str.split('\n', expand=True)

In [85]:  df = df.drop('Price1',axis=1)

In [86]:  df1 = df['Name'].str.split(' ', expand=True)

In [89]:  df['Launch_Year'] = df1[0]

In [91]:  df[['Kilometers_Driven','test']] = df['Kilometers_Driven'].str.split('km', expand=True)

In [93]:  df = df.drop('test',axis=1)

In [94]:  df[['Registration','Registration_code','test','test']] = df['Registration'].str.split('-', expand=True)

In [96]:  df['Registration'] = df['Registration'] + '-' + df['Registration_code']

In [98]:  df = df.drop(['Registration_code','test'],axis=1)

In [100]: df[['Last_Service','test']] = df['Last_Service'].str.split('km', expand=True)

In [102]: df = df.drop('test',axis=1)

In [104]: df.to_csv('CarData1_Final.csv')

In [109]: df = pd.read_csv('CarData1_Final.csv')
          df.head()
```

```
for i in df.columns:
    if df[i].dtypes=='object':
        df[i]=enc.fit_transform(df[i].values.reshape(-1,1))
```

Endcoded data into Int

- # Feature Engg

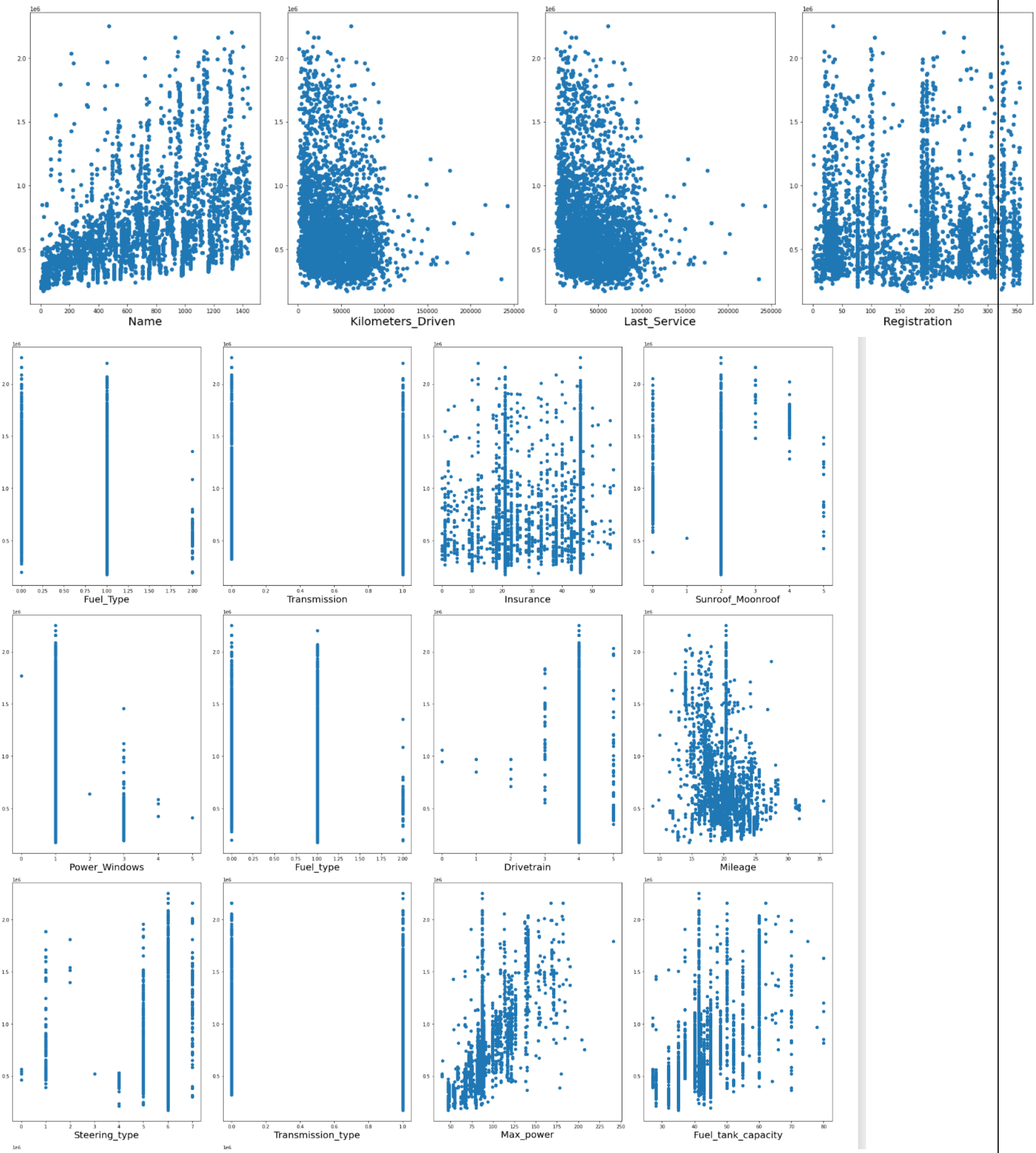
```python
x = df.drop(['Price'],axis=1)
y = df['Price']
```

```python
plt.figure(figsize=(25,40), facecolor='white')

plotno = 1

for column in x:
    if plotno <= 18:
        ax = plt.subplot(5,4,plotno)
        plt.scatter(x[column],y)
        plt.xlabel(column,fontsize=20)

    plotno+=1
plt.tight_layout()
```



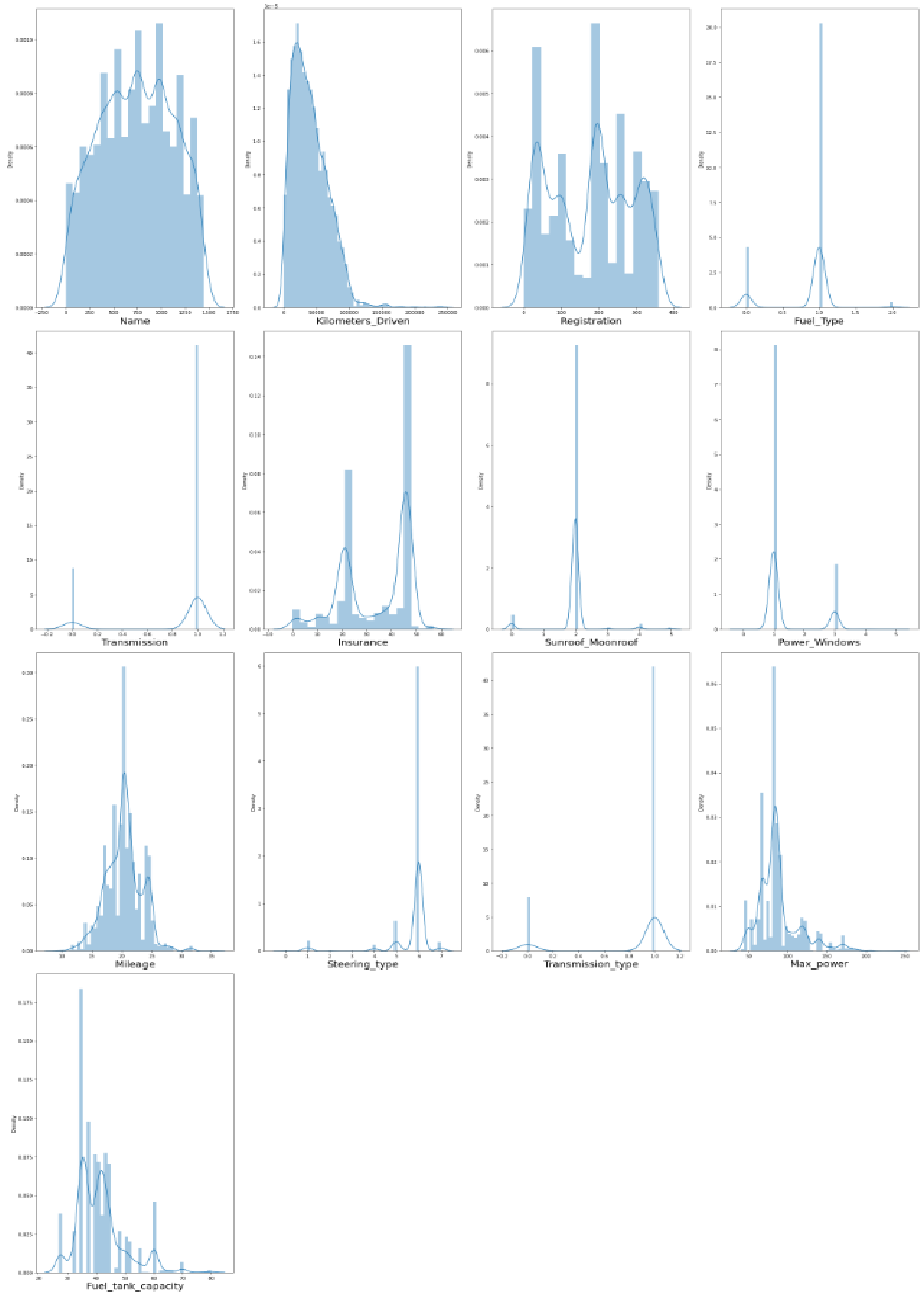
- Checking for skew data

```
plt.figure(figsize=(25,40), facecolor='white')

plotno = 1

for column in x:
    if plotno <= 16:
        ax = plt.subplot(4,4,plotno)
        sns.distplot(x[column])
        plt.xlabel(column,fontsize=20)

    plotno+=1
plt.tight_layout()
```



```python
for i in df.columns:
    a = df[i].skew()
    print(i,'=',a)
```

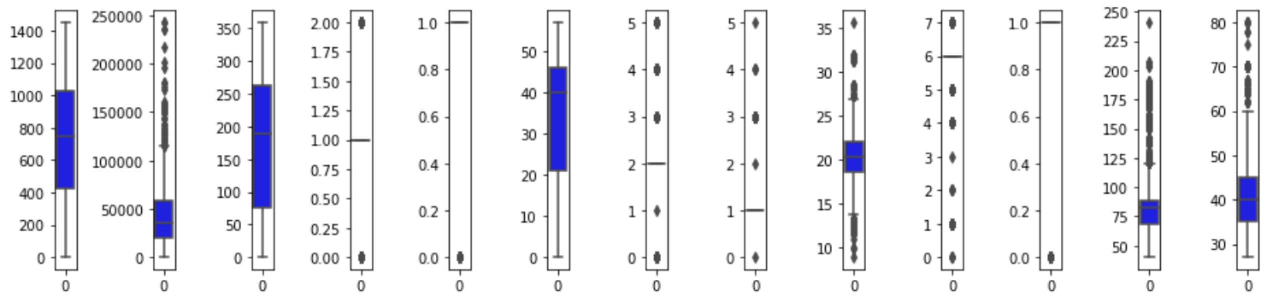
```
Price = 1.5283538565208503
Name = -0.04499556646526514
Kilometers_Driven = 1.2745889548113183
Registration = -0.018343671915760 4
Fuel_Type = -1.1372243790287218
Transmission = -1.6898684909473625
Insurance = -0.5436849312158353
Sunroof_Moonroof = -0.3684005541026811
Power_Windows = 1.6248449443874342
Mileage = 0.08432065037061806
Steering_type = -3.9960390541366597
Transmission_type = -1.8634271510542266
Max_power = 1.454121727378636
Fuel_tank_capacity = 1.1907958191669374
```

Not considering skewness of categorical data columns

- Kilometers_Driven, Max_power and Fuel_tank_capacity has skewness

Database is ready to remove outliers if exist

```python
a = x.columns.values
col = 35
row = 30
plt.figure(figsize = (col,3*row))
for i in range(0, len(a)):
    plt.subplot(row,col,i+1)
    sns.boxplot(data = x[a[i]],color='blue',orient='v')
    plt.tight_layout()
```



LotFrontage, LotArea, BsmtFinSF1, BsmtUnfSF, TotalBsmtSF, 1stFlrSF, LowQualFinSF, GrLivArea, GarageArea, WoodDeckSF, OpenPorchSF, EnclosedPorch, ScreenPorch and PoolArea Column has outliers

- **Skewness removed by quantile method**

- **Scaling the data**

```
x = df.drop(['Price'],axis=1)
y = df['Price']
```

```
scaler = StandardScaler()
X_scale = scaler.fit_transform(x)
```

-

- **EDA Concluding Remark.**

1. The information was not organized and coordinated and subsequently cleaned the information utilizing different information cleaning and pre-handling techniques.
2. There are numerous anomalies present in the information consequently eliminating exceptions
3. There was a skewness in the information thus have eliminated the skewness from the information.
4. There was an irregularity in the information thus have utilized SMOTE strategy to balance the information.
5. Scaled the data utilizing Standard Scalar to make the information normalized to fabricate a model.

- **Hardware and Software Requirements and Tools Used**

1. Libraries and packages used

- import numpy as np - For Numpy work
- import pandas as pd - To work on DataFrame
- import seaborn as sns - Plotting Graphs
- import matplotlib.pyplot as plt - Plotting Graphs
- import pickle – To save the Model
- from sklearn.preprocessing import StandardScaler (To scale the train data), OrdinalEncoder(To encode object data to Integer), PowerTransformer (To remove skewness from dataset)
- from statsmodels.stats.outliers_influence import variance_inflation_factor
- enc = OrdinalEncoder() = Assigned OrdinalEncoder to variable
- from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score,(To split the data into train and test, Search the best parameters, to calculate cross validation score)
- from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, roc_curve, roc_auc_score - To calculate and analyse model metrics.
- from sklearn import metrics

Models which are used

- from sklearn.ensemble import RandomForestClassifier
- from sklearn.linear_model import LogisticRegression
- from sklearn.ensemble import GradientBoostingClassifier
- from sklearn.tree import DecisionTreeClassifier
- from sklearn.neighbors import KNeighborsClassifier
- from sklearn.svm import SVC

- import warnings
- warnings.filterwarnings('ignore') - To ignore unwanted Warnings

2. Hardware used – 11th Gen Intel(R) Core (TM) i3-1115G4 @ 3.00GHz 3.00 GHz with 8.00 GB RAM and Windows 11
3. Software used – Anaconda and Jupyter Notebook to build the model.

- **Building Machine Learning Models.**

I have built 6 machine learning models to predict the label. Below are the machine learning models which are been used.

1. LogisticsRegression
2. RandomForestClassifier
3. DecisionTreeClassifier
4. GradientBoostingClassifier
5. Support Vector Classifier
6. KNeighborsClassifier

1. LogisticsRegression:

Have used "For Loop" to find out the highest accuracy score with different random state ranging from 0-100 and using that random state to split the data into train and test data.

```
]:  reg = LinearRegression()
    for i in range(0,100):
        x_train,x_test,y_train,y_test = train_test_split(X_scale,y,test_size = 0.25,random_state = i)
        reg.fit(x_train, y_train)
        x_pred = reg.predict(x_train)
        y_pred = reg.predict(x_test)
        print("At Random state", (i), "the training accuracy is :-", (r2_score (y_train,x_pred)))
        print("At Random state", (i), "the testing accuracy is :-", (r2_score (y_test,y_pred)))
        print('\n')
```

```
At Random state 24 the testing accuracy is :- 0.6722909420571519


At Random state 25 the training accuracy is :- 0.6726468395656839
At Random state 25 the testing accuracy is :- 0.6746602758572529


At Random state 26 the training accuracy is :- 0.6634338773478726
At Random state 26 the testing accuracy is :- 0.704169495101693


At Random state 27 the training accuracy is :- 0.6695078428641499
At Random state 27 the testing accuracy is :- 0.6858194397585636


At Random state 28 the training accuracy is :- 0.6681201936149961
At Random state 28 the testing accuracy is :- 0.6908866202508435


At Random state 29 the training accuracy is :- 0.6671114526023241
```

Have used "Define Function" to define a machine learning model code that automatically provides the train and test accuracy code.

Formulas:

y_pred = clf.predict(x_train) = Predicting train data

accuracy_score(y_train, y_pred) = Calculating train accuracy score (comparing y_pred data with y_train data) pred = clf.predict(x_test) = Predicting test data

accuracy_score(y_test, pred) = Calculating test accuracy score (comparing pred data with y_test data)

```
]: x_train,x_test,y_train,y_test = train_test_split(X_scale,y,test_size = 0.25,random_state = 66)
```

```
]: def print_score(clf, x_train,x_test,y_train,y_test, train=True):
       if train:
           y_pred = clf.predict(x_train)

           print('\n===============Train Result===============')
           print(f'Accuracy Score: {r2_score (y_train,y_pred)*100:.2f}%')


       elif train==False:
           pred = clf.predict(x_test)

           print('\n===============Test Result===============')
           print(f'Accuracy Score: {r2_score (y_test,pred)*100:.2f}%')

           print ('\n mean_absolute_error',mean_absolute_error(y_test,pred))
           print ('\n mean_squared_error',mean_squared_error (y_test,pred))
```

Trained the data and run the "Def" function

```
]: reg = LinearRegression()
   reg.fit(x_train,y_train)

   print_score(reg,x_train,x_test,y_train,y_test, train=True)
   print_score(reg,x_train,x_test,y_train,y_test, train=False)
```

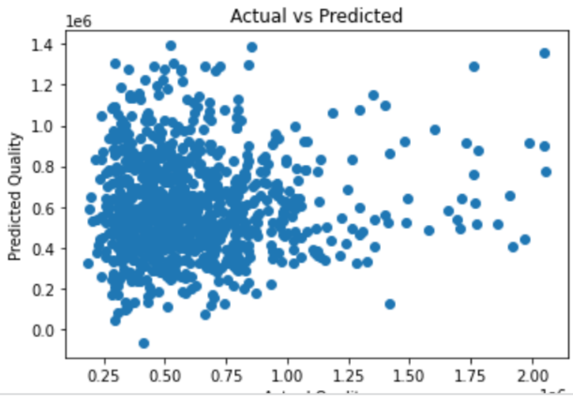
```
===============Train Result===============
Accuracy Score: 67.35%

===============Test Result===============
Accuracy Score: 67.33%

 mean_absolute_error 102376.51939422284

 mean_squared_error 28665347329.86425
```

```
]: plt.scatter(y_test, y_pred)
   plt.xlabel('Actual Quality')
   plt.ylabel('Predicted Quality')
   plt.title('Actual vs Predicted')
   plt.show()
```



2. RandomForestClassifier:

[62]:

```
rfr = RandomForestRegressor()
rfr.fit(x_train,y_train)

print_score(rfr,x_train,x_test,y_train,y_test, train=True)
print_score(rfr,x_train,x_test,y_train,y_test, train=False)
```

```
===============Train Result===============
Accuracy Score: 97.84%

===============Test Result===============
Accuracy Score: 86.79%

 mean_absolute_error 54218.23786166842

 mean_squared_error 11587753426.816917
```

3. DecisionTreeClassifier:

]:

```
dtr = DecisionTreeRegressor()
dtr.fit(x_train,y_train)

print_score(dtr,x_train,x_test,y_train,y_test, train=True)
print_score(dtr,x_train,x_test,y_train,y_test, train=False)
```

```
===============Train Result===============
Accuracy Score: 100.00%

===============Test Result===============
Accuracy Score: 76.61%

 mean_absolute_error 61022.305174234425

 mean_squared_error 20520848167.263992
```

4. GradientBoostingClassifier:

```
gbdt = GradientBoostingRegressor()
gbdt.fit(x_train,y_train)

print_score(gbdt,x_train,x_test,y_train,y_test, train=True)
print_score(gbdt,x_train,x_test,y_train,y_test, train=False)
```

```
===============Train Result===============
Accuracy Score: 85.11%

===============Test Result===============
Accuracy Score: 79.90%

 mean_absolute_error 75192.5135144641

 mean_squared_error 17635591320.572407
```

5. Support Vector Classifier:

```
svr = SVR()
svr.fit(x_train,y_train)

print_score(svr,x_train,x_test,y_train,y_test, train=True)
print_score(svr,x_train,x_test,y_train,y_test, train=False)
```

```
===============Train Result===============
Accuracy Score: -5.74%

===============Test Result===============
Accuracy Score: -4.99%

 mean_absolute_error 205857.07725353728

 mean_squared_error 92104971953.68492
```

6. KNeighborsClassifier:

```
knr = KNeighborsRegressor()
knr.fit(x_train,y_train)

print_score(knr,x_train,x_test,y_train,y_test, train=True)
print_score(knr,x_train,x_test,y_train,y_test, train=False)
```

```
===============Train Result===============
Accuracy Score: 82.39%

===============Test Result===============
Accuracy Score: 71.56%

 mean_absolute_error 89830.3626187962

 mean_squared_error 24947042897.9007
```

- **Findings**

- LinearRegression train accuracy score 67.35% and test accuracy score 67.33%
- Support Vector Regression train accuracy score -5.74% and test accuracy score -4.99%
- DecisionTreeRegressor train accuracy score 100.00% and test accuracy score 76.61%
- AdaBoostRegressor train accuracy score 57.60% and test accuracy score 55.07%
- GradientBoostingRegressor train accuracy score 85.11% and test accuracy score 79.90%
- RandomForestRegressor train accuracy score 97.84% and test accuracy score 86.79%
- KNeighborsRegressor train accuracy score 82.39% and test accuracy score 71.56%

- **Model Selection:**

  LinearRegression it has low variance between train and test result and has high 67.35% and 67.33% accuracy i.e., respectively.