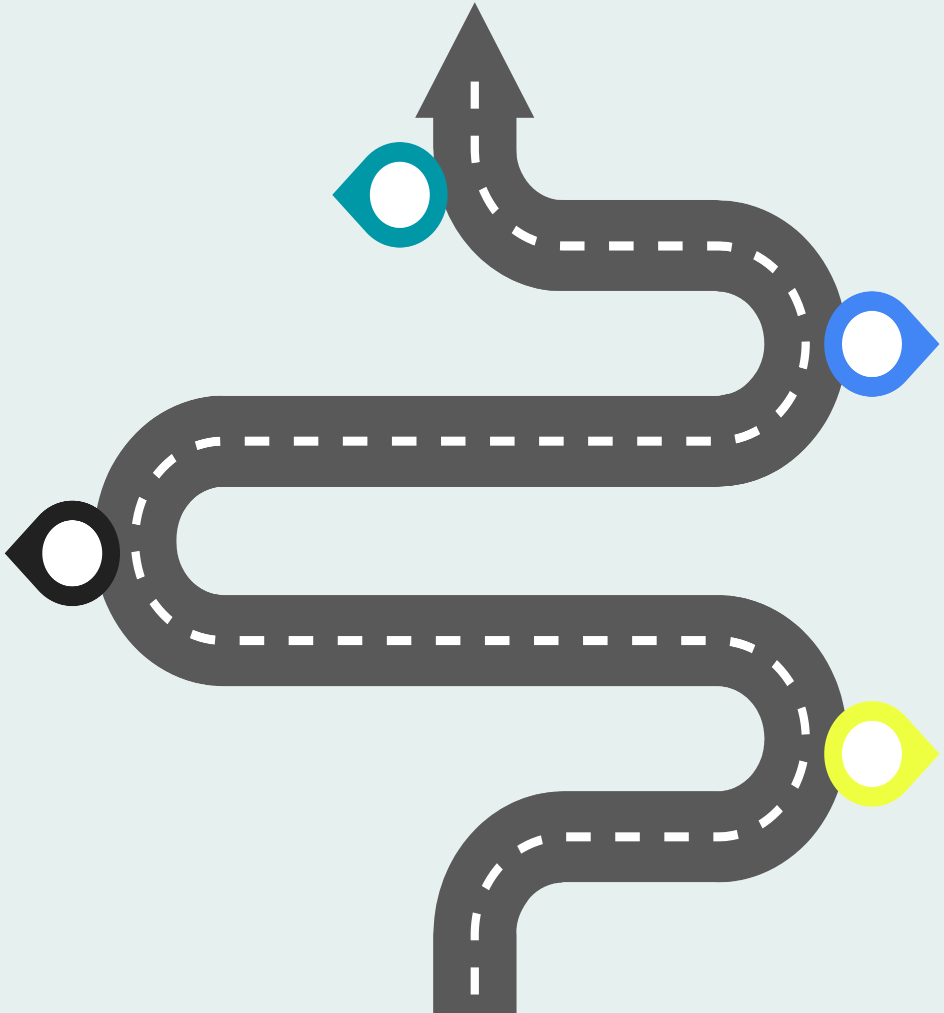
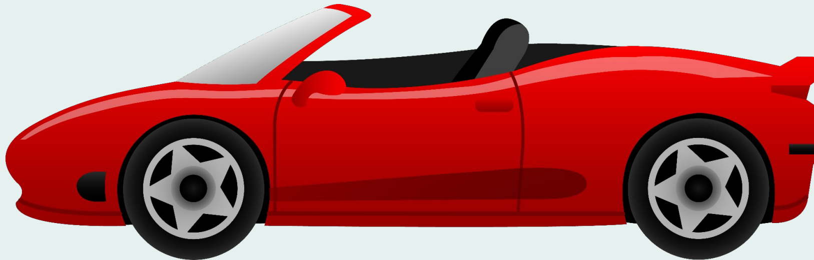
Final Project Presentation

Data Science Batch 33 Flip Learning



Used Car Price Prediction



Project Introduction

Data source: https://www.kaggle.com/datasets/nehalbirla/vehicle-dataset-from-cardekho/data

BACKGROUND:
Due to the increasing digitization and online market, the used car market receives significant growth. Some companies build applications to make buying and selling used cars easier and build used automotive marketplace so that buyers and sellers easily in making transactions.

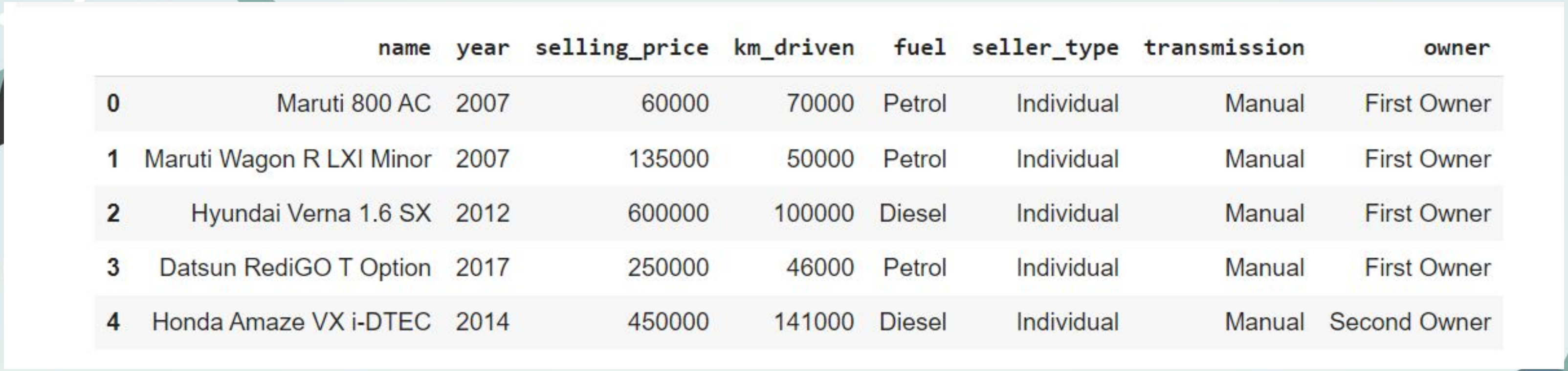
Project Introduction

GOAL:
Increase the growth of used car sales by predicting used car price to customers.

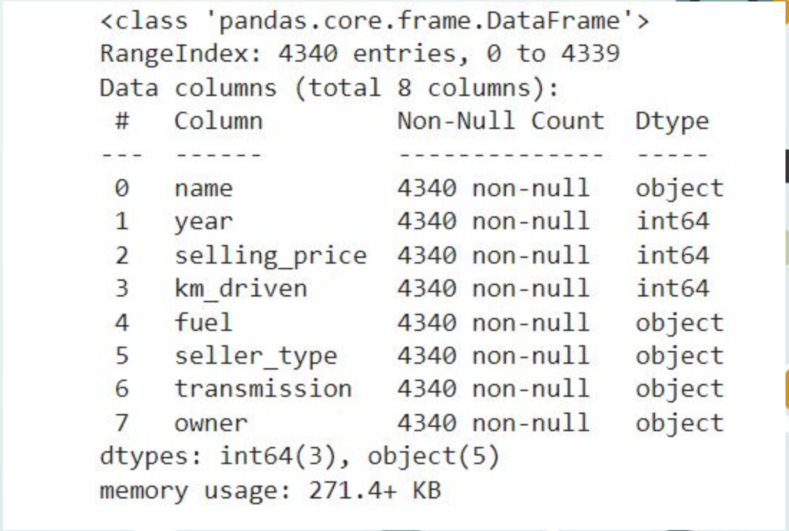
OBJECTIVE:
Develop machine learning model that can predict used car price based on related features.

EVALUATION METRICS:
- RMSE
- R2

Dataset Overview



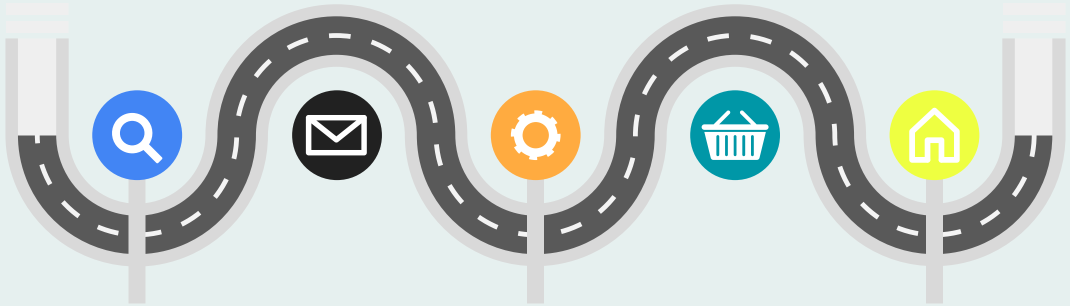
	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner
0	Maruti 800 AC	2007	60000	70000	Petrol	Individual	Manual	First Owner
1	Maruti Wagon R LXI Minor	2007	135000	50000	Petrol	Individual	Manual	First Owner
2	Hyundai Verna 1.6 SX	2012	600000	100000	Diesel	Individual	Manual	First Owner
3	Datsun RediGO T Option	2017	250000	46000	Petrol	Individual	Manual	First Owner
4	Honda Amaze VX i-DTEC	2014	450000	141000	Diesel	Individual	Manual	Second Owner



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4340 entries, 0 to 4339
Data columns (total 8 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   name           4340 non-null   object
 1   year           4340 non-null   int64 
 2   selling_price  4340 non-null   int64 
 3   km_driven      4340 non-null   int64 
 4   fuel           4340 non-null   object
 5   seller_type    4340 non-null   object
 6   transmission   4340 non-null   object
 7   owner          4340 non-null   object
dtypes: int64(3), object(5)
memory usage: 271.4+ KB
```

Dataset has 4340 rows and 8 columns

Data Preprocessing Overview



763 entries were duplicated. Since the dataset was collected from internet source and each data does not have an identifier, we can consider to drop the same value (duplicated value). We still keep the original data in df and copy the removed duplicated data in df2

Handling Duplicates

Handling Missing Values

no missing values for each variables

Handling Outliers

The numerical data has some outlier

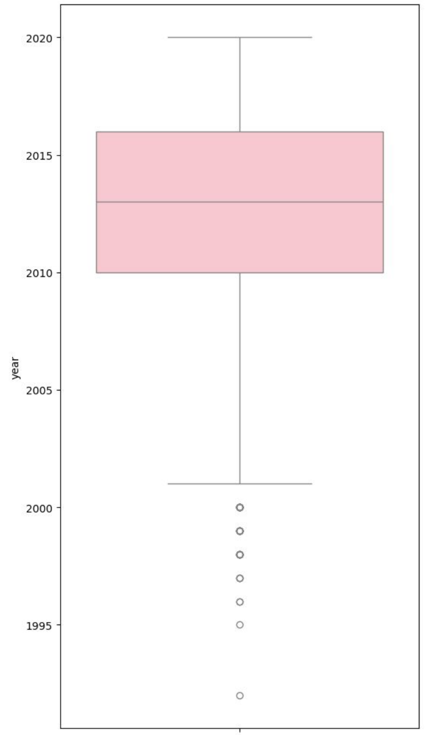
Descriptive Statistics

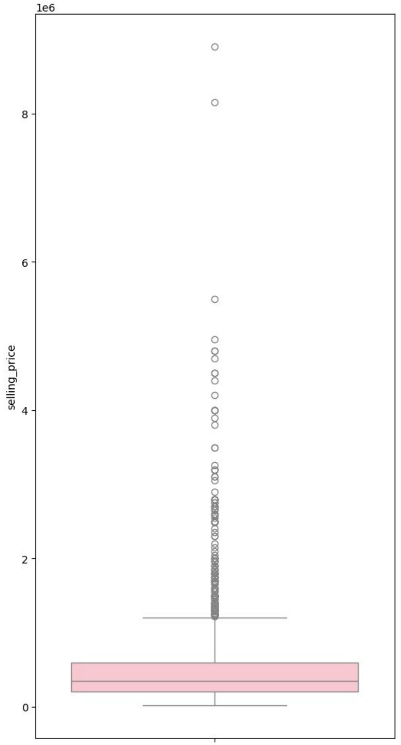
➔ Numerical Variables

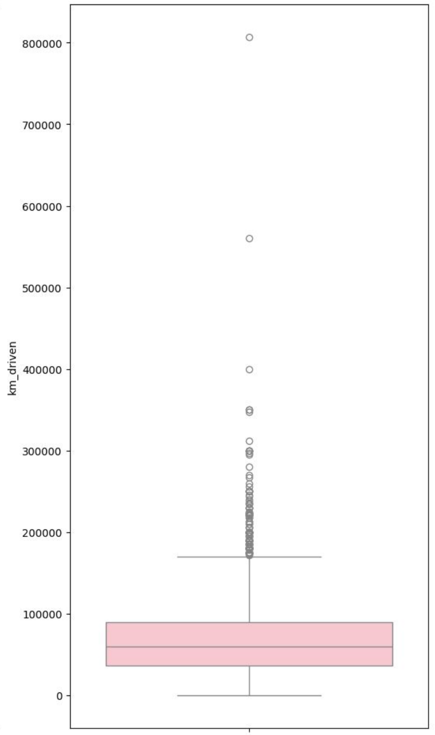
The distribution of each numerical variables have skewed distribution.

- Year: negative skewed
- selling price : positive skewed
- km_driven :positive skewed

based on interquartile range, each numerical variables have some outliers. Later we have to handle this before do modeling.

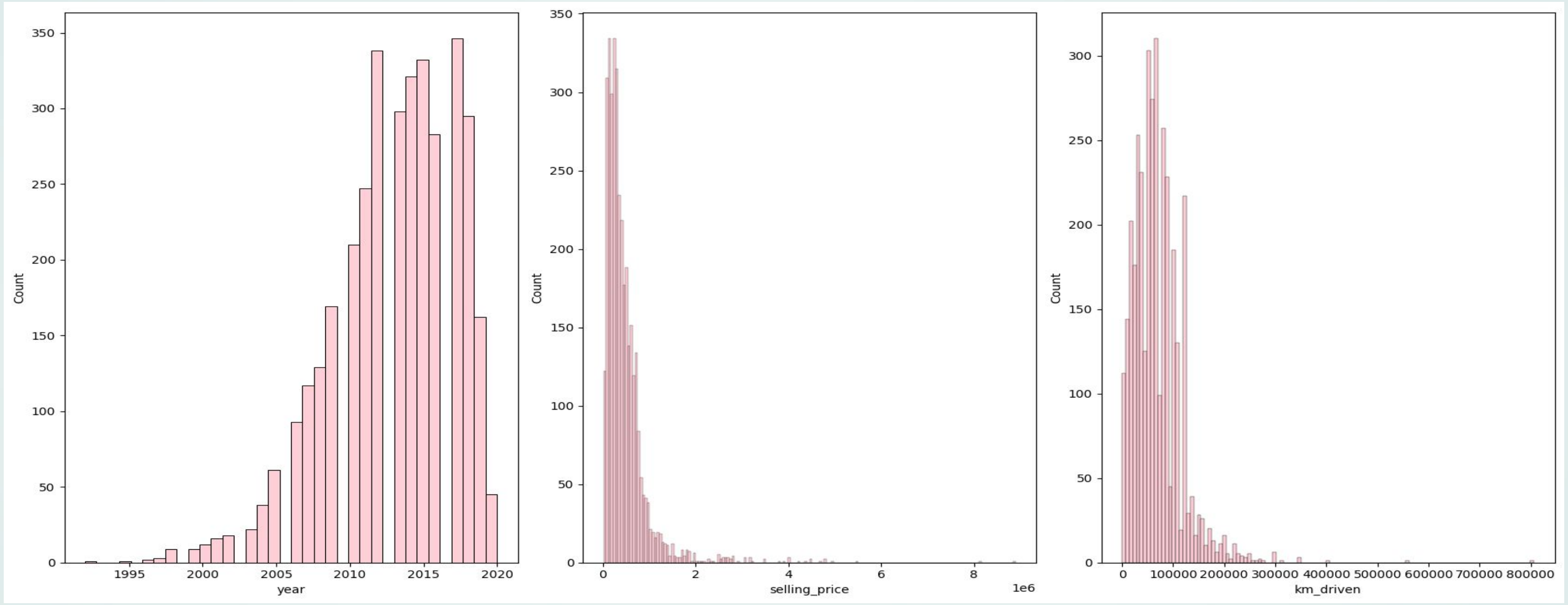






Descriptive Statistics

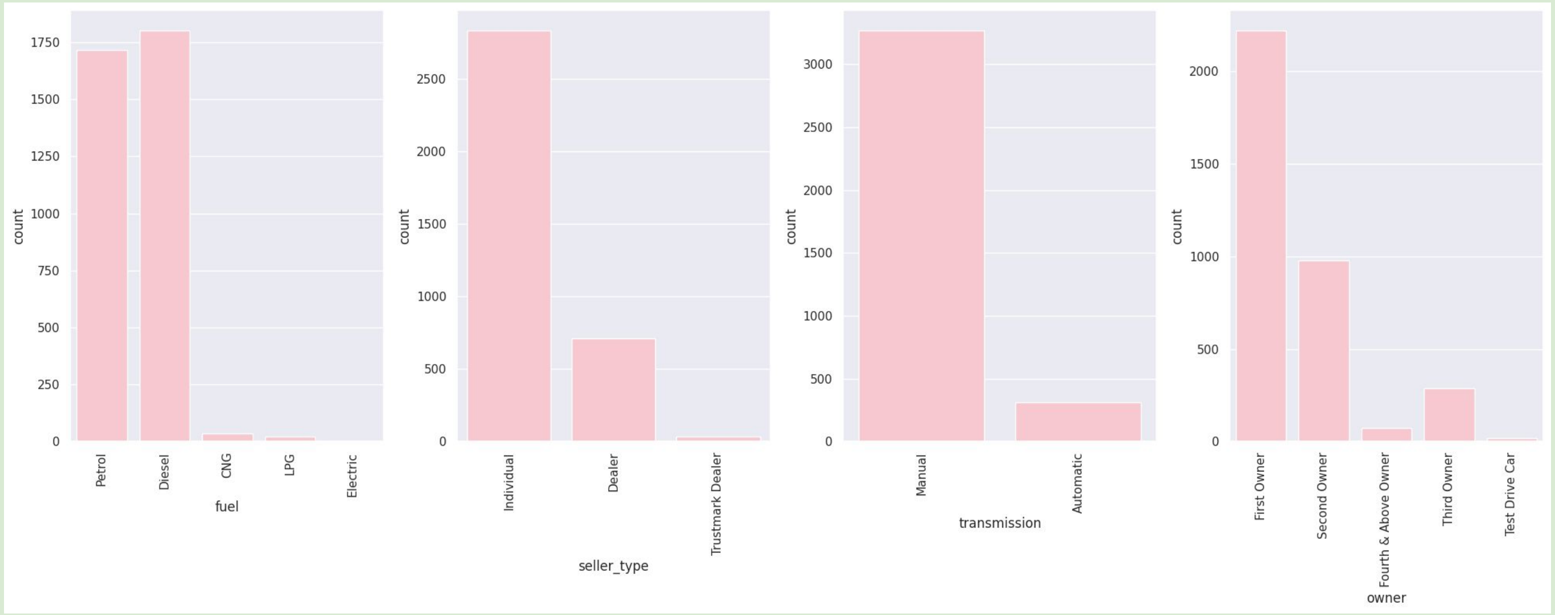
→ Numerical Variables



- Based on Year distribution, used car data are mostly from year 2010 - 2016.
- Based on Selling Price distribution, used car data are mostly below 1.000.000
- Based on km_driven distribution, used car data are mostly below 100.000 km

Descriptive Statistics

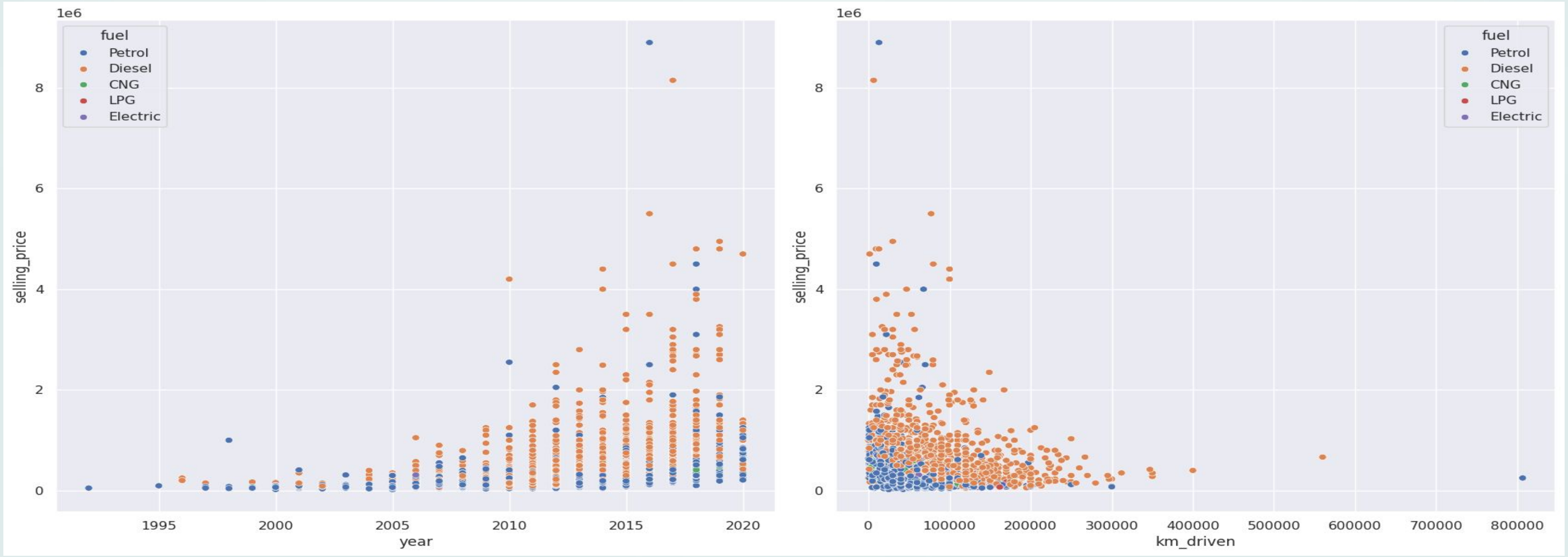
➔ Categorical Variables



- Used cars in dataset are mostly using 'Diesel' and 'Petrol' as fuel.
- Used cars in dataset are mostly 'Individual' seller type
- Used cars in dataset are mostly using 'manual' transmission
- Used cars in dataset are mostly First Owner.

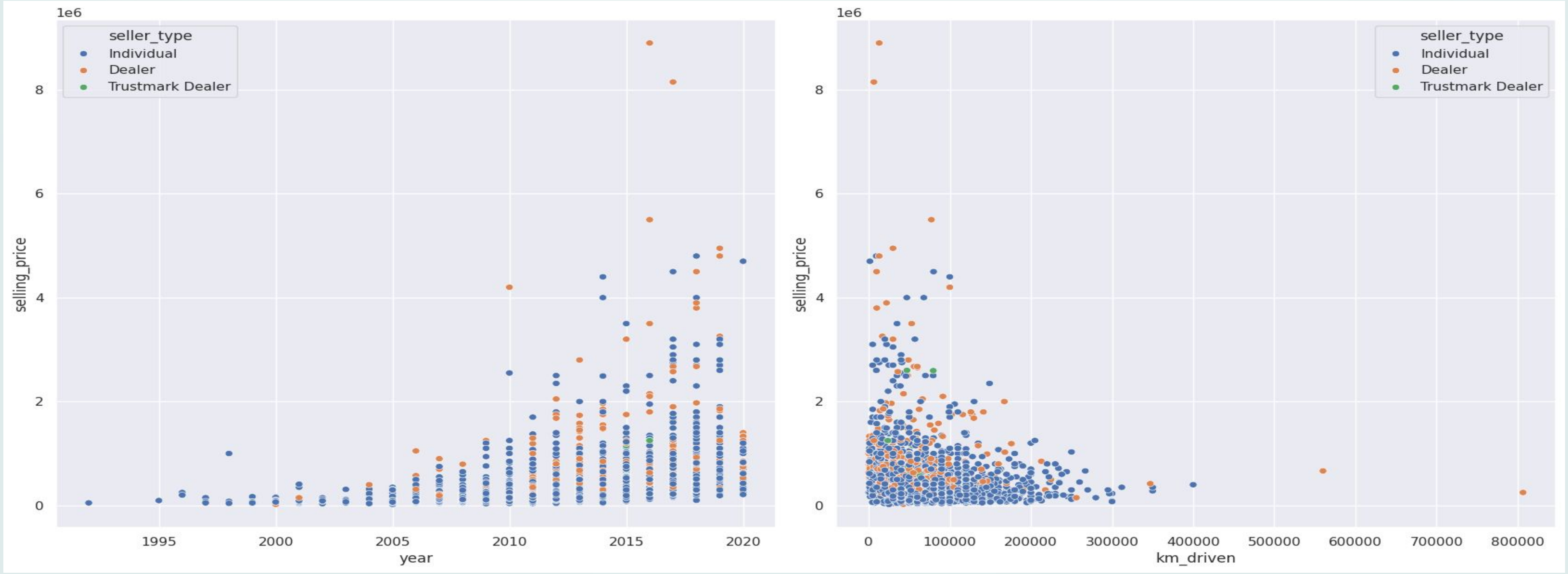
Univariate Analysis
→ Numerical Variables

- - Based on ' Selling_price' tren year to year, the newer of year production shows higher price than the older one. <br>
- - The lower the km_driven, the higher the selling price.<br>
- - Used cars fueled by petrol and diesel are evenly distributed every used car released year.
- - Diesel cars are widely sold starting from 2010 released year.

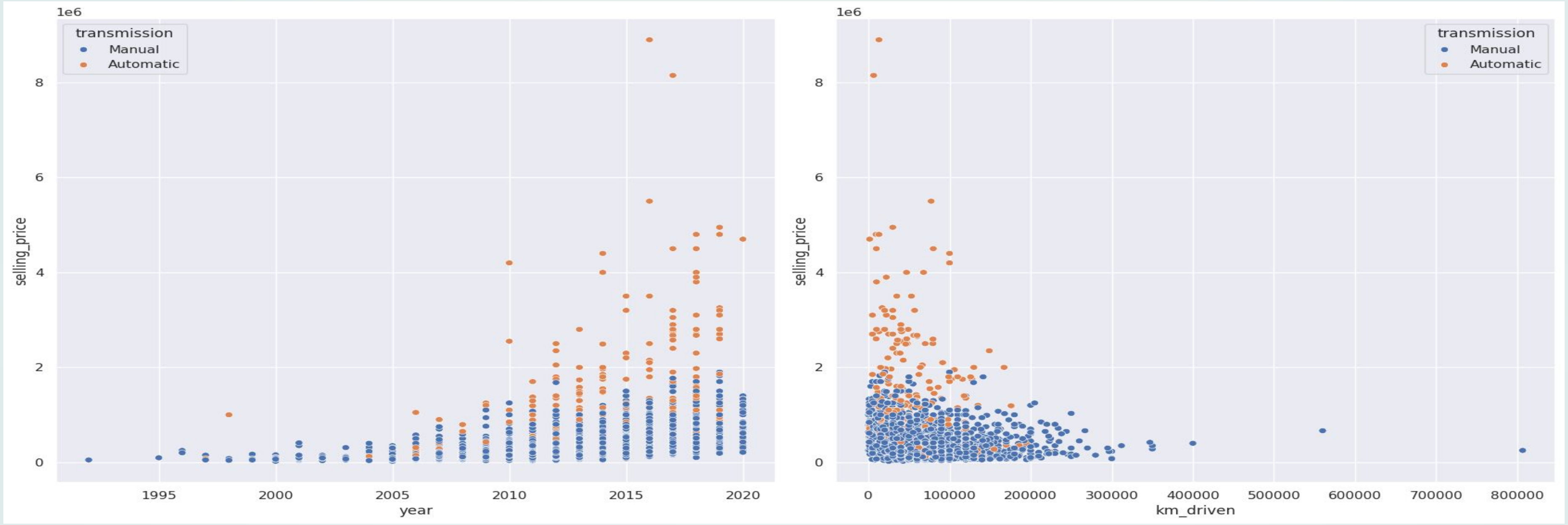


Univariate Analysis
→ Numerical Variables

- Used cars with seller type Individual and dealer are evenly distributed every year.
- Seller type dealer typically sold the used car that released on year above 2000.



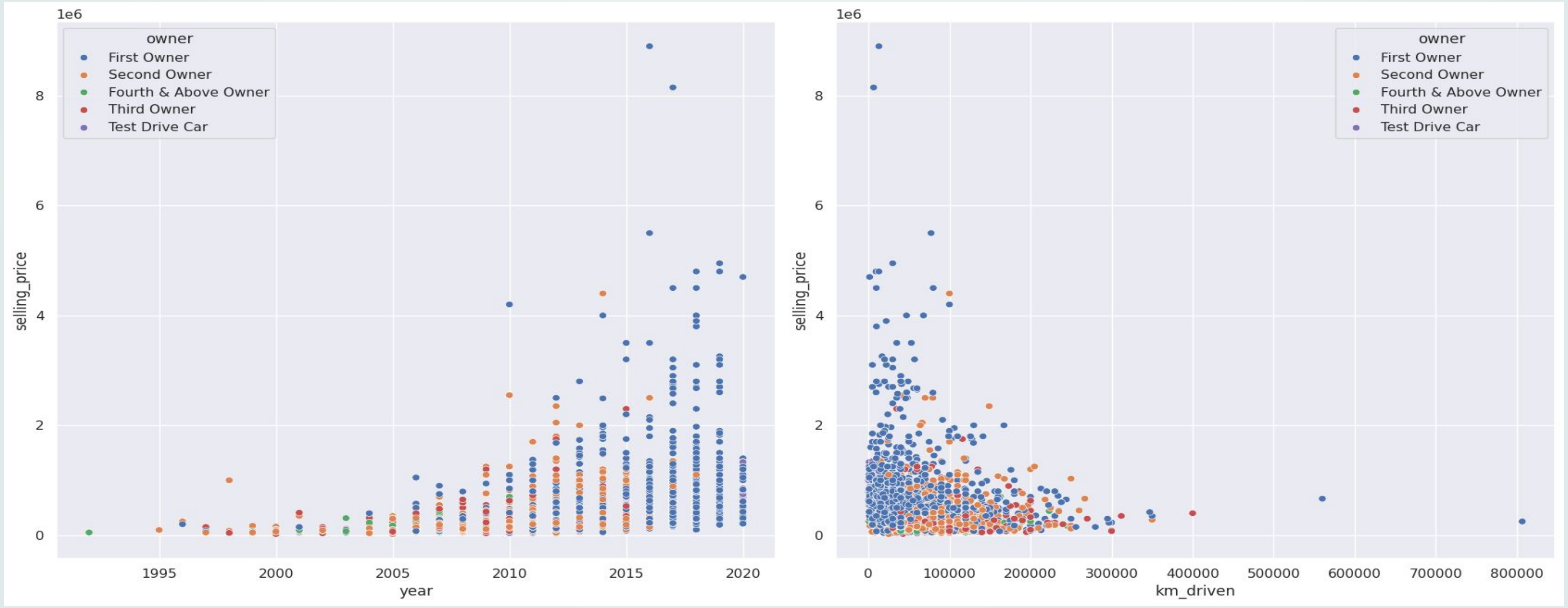
Univariate Analysis
→ Numerical Variables



- Used automatic cars which release before 2005 were rarely found
- The price of a used car with an automatic transmission is more expensive than the price of a manual car

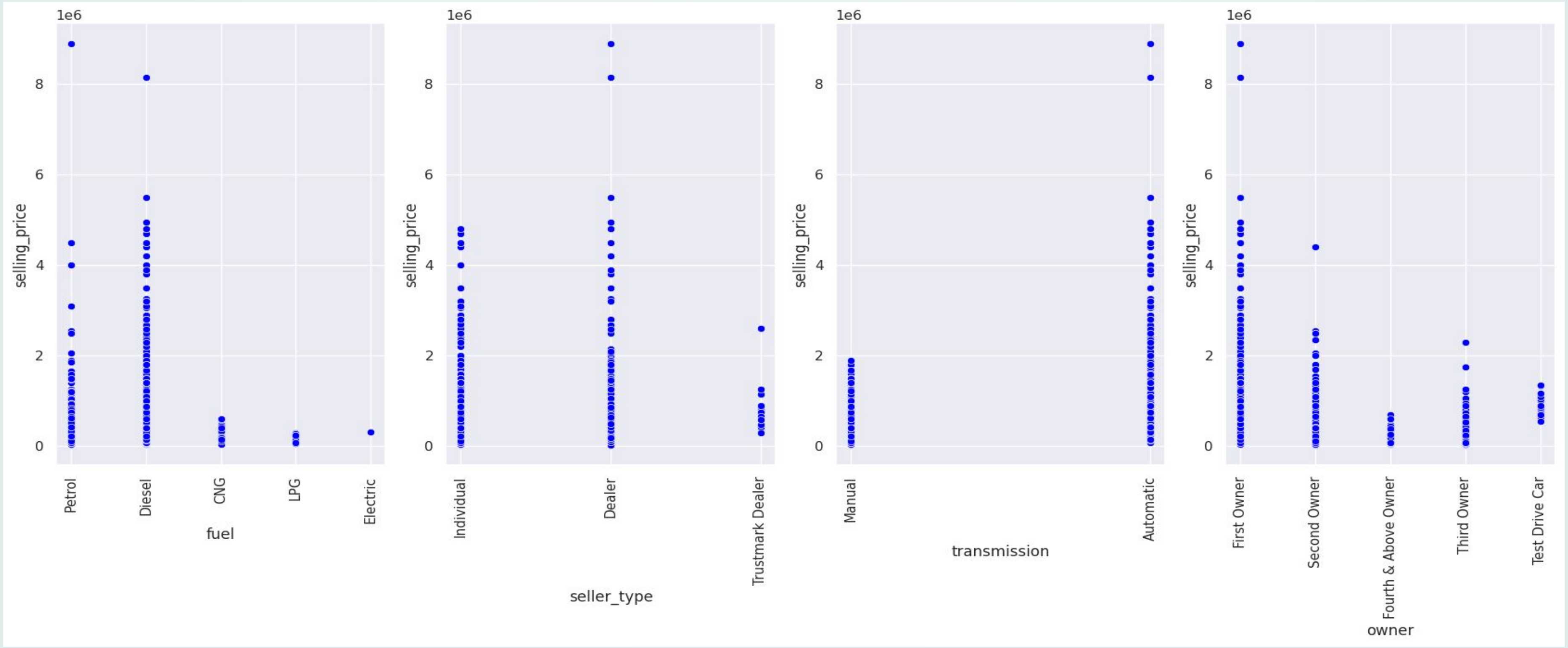
Univariate Analysis
➔ Numerical Variables

- Used cars whose individual owner are mostly car that released from year 2015 until 2020
- Used cars whose second owner are mostly car that released before 2015.
- First owner used car has higher price than others.
- The lower km_driven and from the first owner, the higher the selling price.



Univariate Analysis
→ Categorical Variables

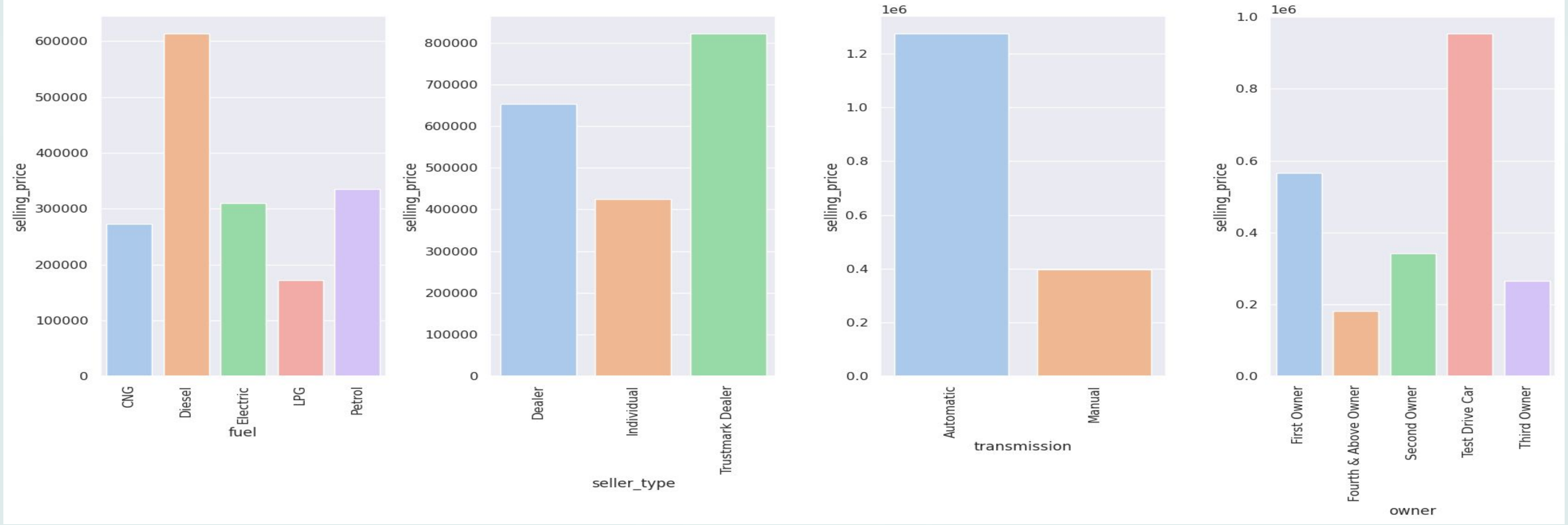
- Used car with diesel and petrol fuel have wider range of selling price (from low to high).
- Used car from individual seller and dealer have wider range of selling price (from low to high).
- Automatic used car commonly has higher price than manual.
- Used car from first owner have wider range of selling price (low to high)



Univariate Analysis

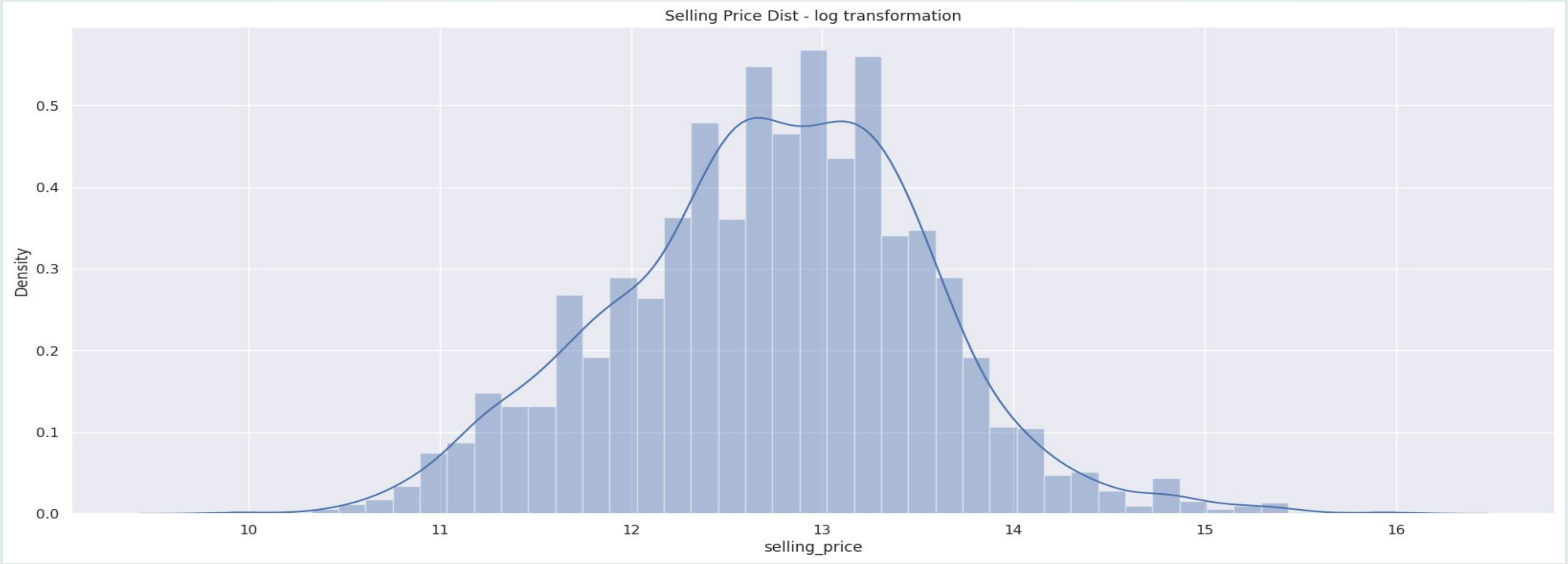
→ Categorical Variables

- Used car with diesel fuel has highest average selling price than others
- Used car from Trustmark Dealer has highest average selling price than others.
- Automatic used car has highest average selling price than others.
- Used car from Test Drive Car has highest average selling price than others. Surprisingly it has higher average selling price than first owner.

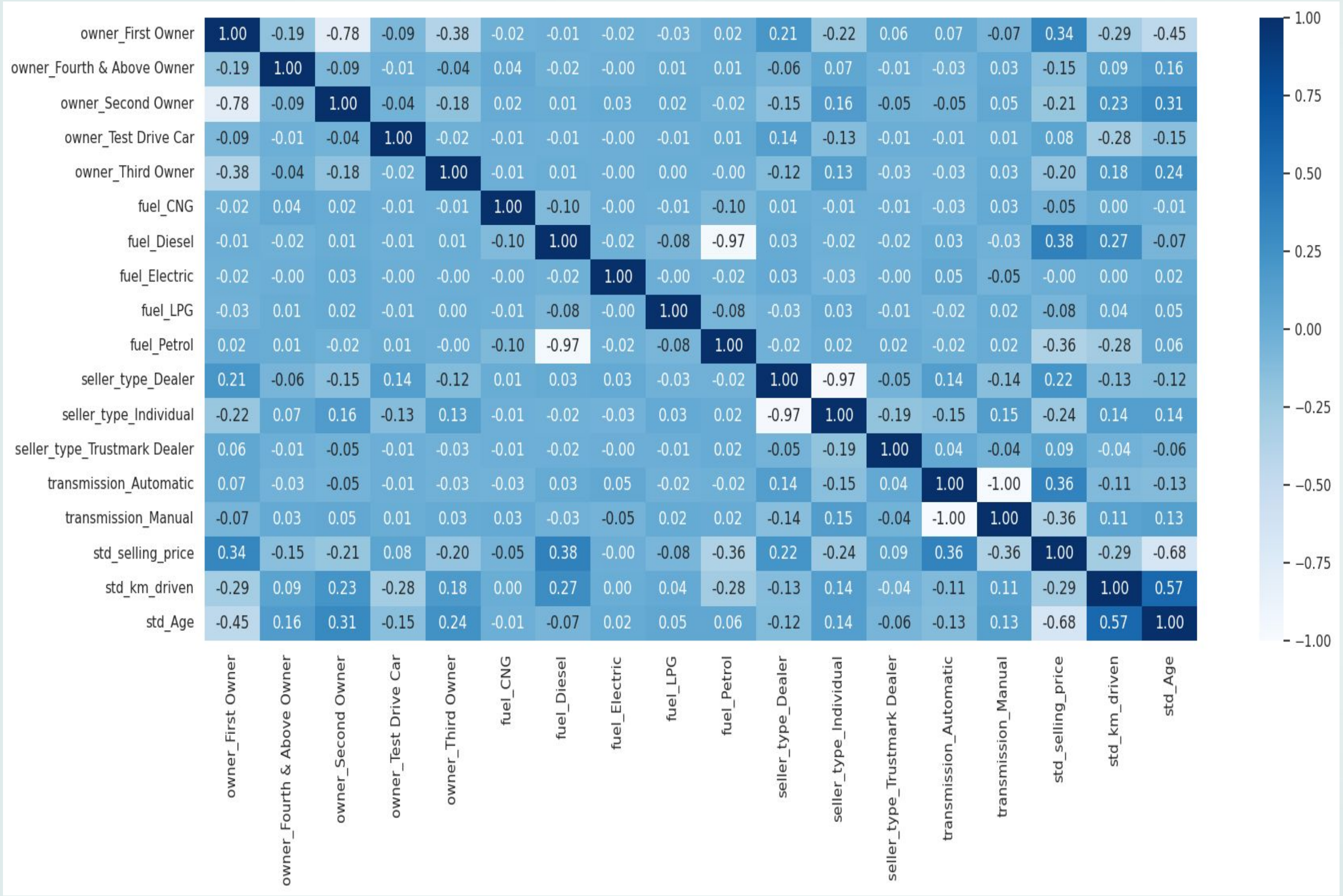


Target Distribution

As mentioned before, our numerical variables have some outlier. We consider not to remove the outliers data due to our small dataset. To handle this we use logarithmic transformation for numerical data. The distribution of logarithmic selling price is rather similar to normal distribution. Formerly, the selling price has positive skewed distribution.



Multivariate Analysis



	owner_First Owner	owner_Fourth & Above Owner	owner_Second Owner	owner_Test Drive Car	owner_Third Owner	fuel_CNG	fuel_Diesel	fuel_Electric	fuel_LPG	fuel_Petrol	seller_type_Dealer	seller_type_Individual	seller_type_Trustmark Dealer	transmission_Automatic	transmission_Manual	std_selling_price	std_km_driven	std_Age
owner_First Owner	1.00	-0.19	-0.78	-0.09	-0.38	-0.02	-0.01	-0.02	-0.03	0.02	0.21	-0.22	0.06	0.07	-0.07	0.34	-0.29	-0.45
owner_Fourth & Above Owner	-0.19	1.00	-0.09	-0.01	-0.04	0.04	-0.02	-0.00	0.01	0.01	-0.06	0.07	-0.01	-0.03	0.03	-0.15	0.09	0.16
owner_Second Owner	-0.78	-0.09	1.00	-0.04	-0.18	0.02	0.01	0.03	0.02	-0.02	-0.15	0.16	-0.05	-0.05	0.05	-0.21	0.23	0.31
owner_Test Drive Car	-0.09	-0.01	-0.04	1.00	-0.02	-0.01	-0.01	-0.00	-0.01	0.01	0.14	-0.13	-0.01	-0.01	0.01	0.08	-0.28	-0.15
owner_Third Owner	-0.38	-0.04	-0.18	-0.02	1.00	-0.01	0.01	-0.00	0.00	-0.00	-0.12	0.13	-0.03	-0.03	0.03	-0.20	0.18	0.24
fuel_CNG	-0.02	0.04	0.02	-0.01	-0.01	1.00	-0.10	-0.00	-0.01	-0.10	0.01	-0.01	-0.01	-0.03	0.03	-0.05	0.00	-0.01
fuel_Diesel	-0.01	-0.02	0.01	-0.01	0.01	-0.10	1.00	-0.02	-0.08	-0.97	0.03	-0.02	-0.02	0.03	-0.03	0.38	0.27	-0.07
fuel_Electric	-0.02	-0.00	0.03	-0.00	-0.00	-0.00	-0.02	1.00	-0.00	-0.02	0.03	-0.03	-0.00	0.05	-0.05	-0.00	0.00	0.02
fuel_LPG	-0.03	0.01	0.02	-0.01	0.00	-0.01	-0.08	-0.00	1.00	-0.08	-0.03	0.03	-0.01	-0.02	0.02	-0.08	0.04	0.05
fuel_Petrol	0.02	0.01	-0.02	0.01	-0.00	-0.10	-0.97	-0.02	-0.08	1.00	-0.02	0.02	0.02	-0.02	0.02	-0.36	-0.28	0.06
seller_type_Dealer	0.21	-0.06	-0.15	0.14	-0.12	0.01	0.03	0.03	-0.03	-0.02	1.00	-0.97	-0.05	0.14	-0.14	0.22	-0.13	-0.12
seller_type_Individual	-0.22	0.07	0.16	-0.13	0.13	-0.01	-0.02	-0.03	0.03	0.02	-0.97	1.00	-0.19	-0.15	0.15	-0.24	0.14	0.14
seller_type_Trustmark Dealer	0.06	-0.01	-0.05	-0.01	-0.03	-0.01	-0.02	-0.00	-0.01	0.02	-0.05	-0.19	1.00	0.04	-0.04	0.09	-0.04	-0.06
transmission_Automatic	0.07	-0.03	-0.05	-0.01	-0.03	-0.03	0.03	0.05	-0.02	-0.02	0.14	-0.15	0.04	1.00	-1.00	0.36	-0.11	-0.13
transmission_Manual	-0.07	0.03	0.05	0.01	0.03	0.03	-0.03	-0.05	0.02	0.02	-0.14	0.15	-0.04	-1.00	1.00	-0.36	0.11	0.13
std_selling_price	0.34	-0.15	-0.21	0.08	-0.20	-0.05	0.38	-0.00	-0.08	-0.36	0.22	-0.24	0.09	0.36	-0.36	1.00	-0.29	-0.68
std_km_driven	-0.29	0.09	0.23	-0.28	0.18	0.00	0.27	0.00	0.04	-0.28	-0.13	0.14	-0.04	-0.11	0.11	-0.29	1.00	0.57
std_Age	-0.45	0.16	0.31	-0.15	0.24	-0.01	-0.07	0.02	0.05	0.06	-0.12	0.14	-0.06	-0.13	0.13	-0.68	0.57	1.00

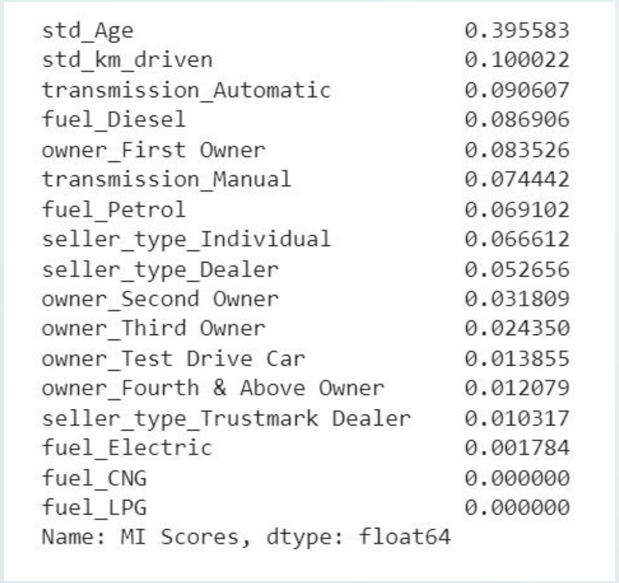
LINEAR CORRELATION

- 'Age' (std_age) has high correlation to selling price. The correlation score is -0.68 that show negative correlation between age and selling price.
- 'Fuel', 'first owner', transmission have correlation to selling price.
- 'Age' and 'km_driven' has correlation, but we consider to keep both since the correlation score is below 0.6.
- 'Age' and 'first owner' has correlation, but we consider to keep both since the correlation score is below 0.6.

Feature Engineering

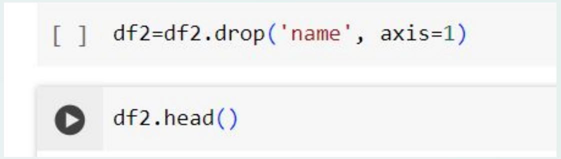
Feature Selection

Mutual Information Score



```
std_Age                          0.395583
std_km_driven                    0.100022
transmission_Automatic           0.090607
fuel_Diesel                      0.086906
owner_First Owner                0.083526
transmission_Manual              0.074442
fuel_Petrol                      0.069102
seller_type_Individual           0.066612
seller_type_Dealer               0.052656
owner_Second Owner               0.031809
owner_Third Owner                0.024350
owner_Test Drive Car             0.013855
owner_Fourth & Above Owner       0.012079
seller_type_Trustmark Dealer     0.010317
fuel_Electric                    0.001784
fuel_CNG                         0.000000
fuel_LPG                         0.000000
Name: MI Scores, dtype: float64
```

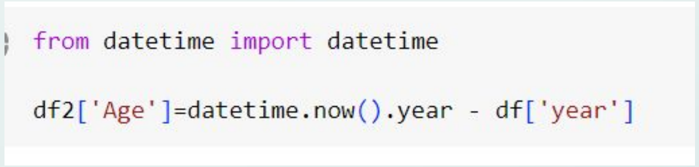
Drop 'name' features



```
[ ]  df2=df2.drop('name', axis=1)

     df2.head()
```

Based on mutual information score, we will keep the features. We also not utilize 'name' features for modeling since it has too many unique

Feature Extraction

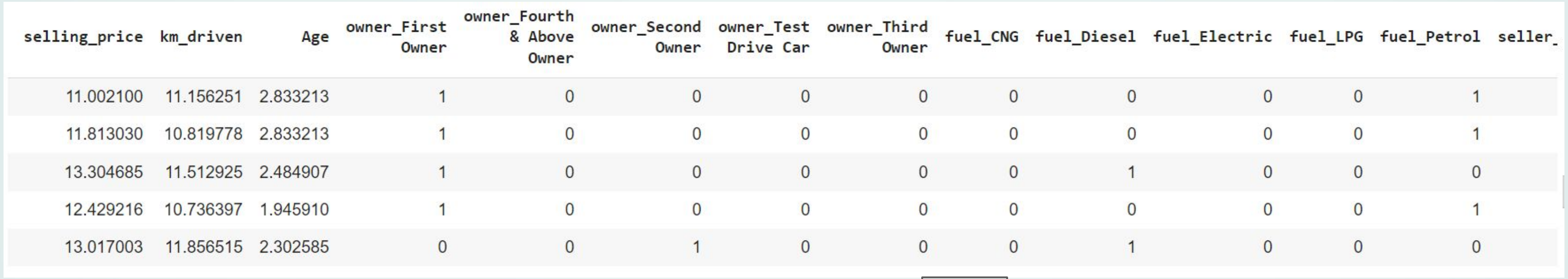


```
from datetime import datetime

df2['Age']=datetime.now().year - df['year']
```

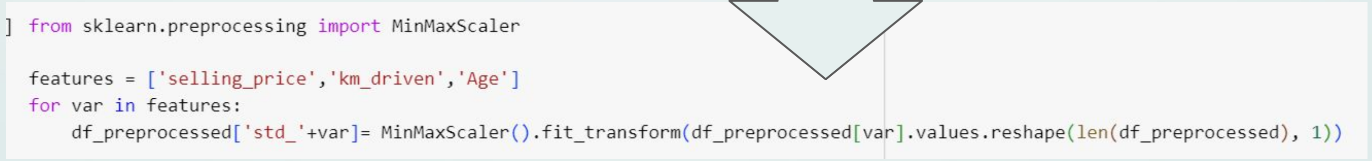
We consider to use 'Age' of car than 'year'. So we extract the age from year column

Normalization & One Hot Encoding

Since our categorical data are mostly nominal not an ordinal type, we consider to use one hot encoding than label encoding.
We applied logaritmic transformation then use minmaxscaler for our numerical data to set the minimum is 0 and the maximum number is 1.



selling_price	km_driven	Age	owner_First Owner	owner_Fourth & Above Owner	owner_Second Owner	owner_Test Drive Car	owner_Third Owner	fuel_CNG	fuel_Diesel	fuel_Electric	fuel_LPG	fuel_Petrol	seller_
11.002100	11.156251	2.833213	1	0	0	0	0	0	0	0	0	1	
11.813030	10.819778	2.833213	1	0	0	0	0	0	0	0	0	1	
13.304685	11.512925	2.484907	1	0	0	0	0	0	1	0	0	0	
12.429216	10.736397	1.945910	1	0	0	0	0	0	0	0	0	1	
13.017003	11.856515	2.302585	0	0	1	0	0	0	1	0	0	0	



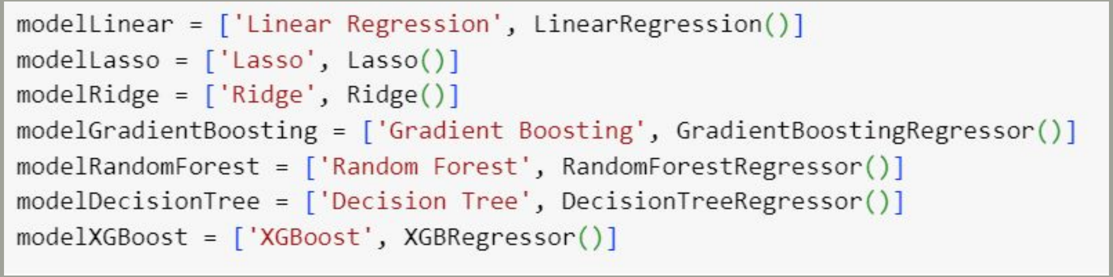
```
from sklearn.preprocessing import MinMaxScaler

features = ['selling_price','km_driven','Age']
for var in features:
    df_preprocessed['std_'+var]= MinMaxScaler().fit_transform(df_preprocessed[var].values.reshape(len(df_preprocessed), 1))
```

Training Model

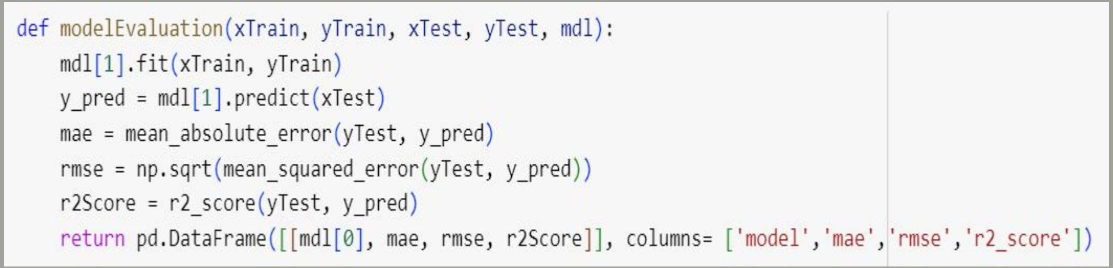
Train & Test Split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =0.2, random_state = 43)

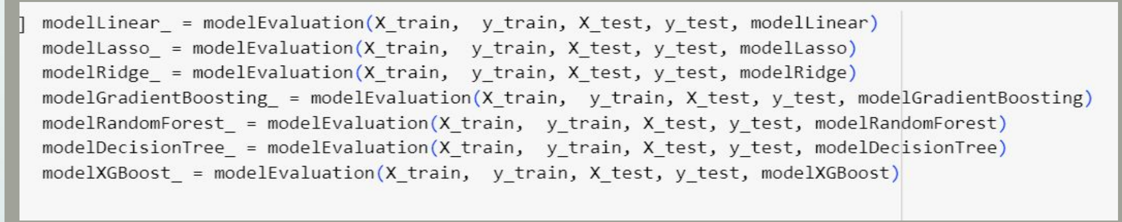
Train Models



```python
modelLinear = ['Linear Regression', LinearRegression()]
modelLasso = ['Lasso', Lasso()]
modelRidge = ['Ridge', Ridge()]
modelGradientBoosting = ['Gradient Boosting', GradientBoostingRegressor()]
modelRandomForest = ['Random Forest', RandomForestRegressor()]
modelDecisionTree = ['Decision Tree', DecisionTreeRegressor()]
modelXGBoost = ['XGBoost', XGBRegressor()]
```

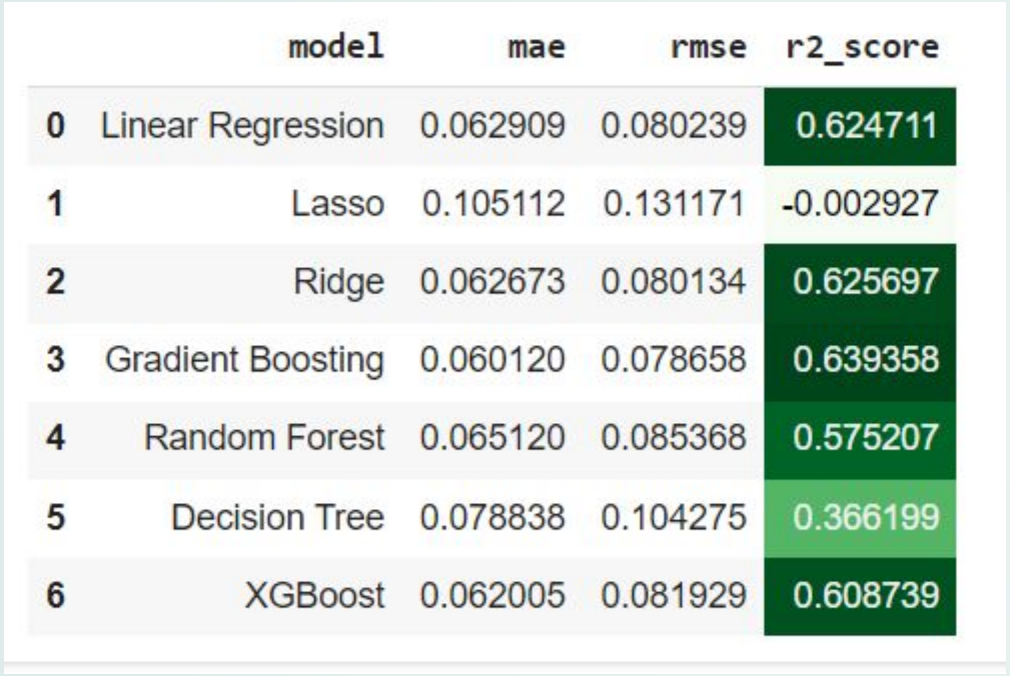


```python
def modelEvaluation(xTrain, yTrain, xTest, yTest, mdl):
    mdl[1].fit(xTrain, yTrain)
    y_pred = mdl[1].predict(xTest)
    mae = mean_absolute_error(yTest, y_pred)
    rmse = np.sqrt(mean_squared_error(yTest, y_pred))
    r2Score = r2_score(yTest, y_pred)
    return pd.DataFrame([[mdl[0], mae, rmse, r2Score]], columns= ['model','mae','rmse','r2_score'])
```



```python
modelLinear_ = modelEvaluation(X_train,  y_train, X_test, y_test, modelLinear)
modelLasso_ = modelEvaluation(X_train,  y_train, X_test, y_test, modelLasso)
modelRidge_ = modelEvaluation(X_train,  y_train, X_test, y_test, modelRidge)
modelGradientBoosting_ = modelEvaluation(X_train,  y_train, X_test, y_test, modelGradientBoosting)
modelRandomForest_ = modelEvaluation(X_train,  y_train, X_test, y_test, modelRandomForest)
modelDecisionTree_ = modelEvaluation(X_train,  y_train, X_test, y_test, modelDecisionTree)
modelXGBoost_ = modelEvaluation(X_train,  y_train, X_test, y_test, modelXGBoost)
```

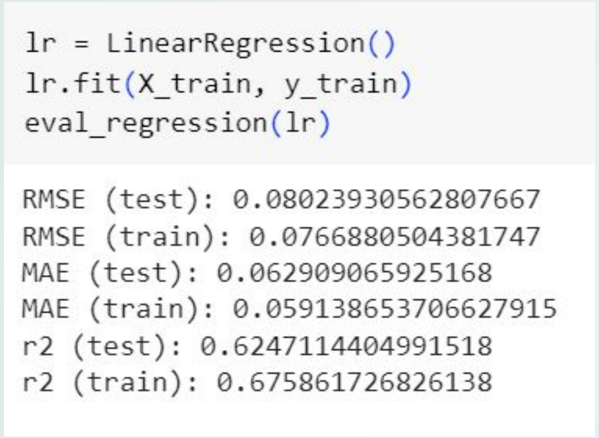
Training MODEL RESULT



	model	mae	rmse	r2_score
0	Linear Regression	0.062909	0.080239	0.624711
1	Lasso	0.105112	0.131171	-0.002927
2	Ridge	0.062673	0.080134	0.625697
3	Gradient Boosting	0.060120	0.078658	0.639358
4	Random Forest	0.065120	0.085368	0.575207
5	Decision Tree	0.078838	0.104275	0.366199
6	XGBoost	0.062005	0.081929	0.608739

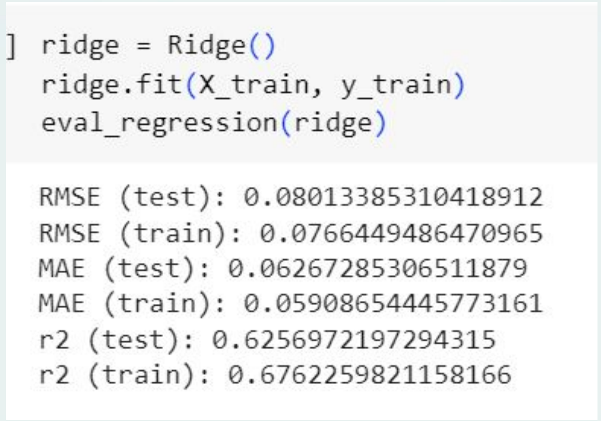
Based on R2 and RMSE score, we got three best algorithm: linear reg, Ridge, and Gradient Boosting. Lets make evaluation by comparing the performance on data train vs data test.

THREE BEST MODEL PERFORMANCE COMPARISON (ON DATA TRAIN & DATA TEST)



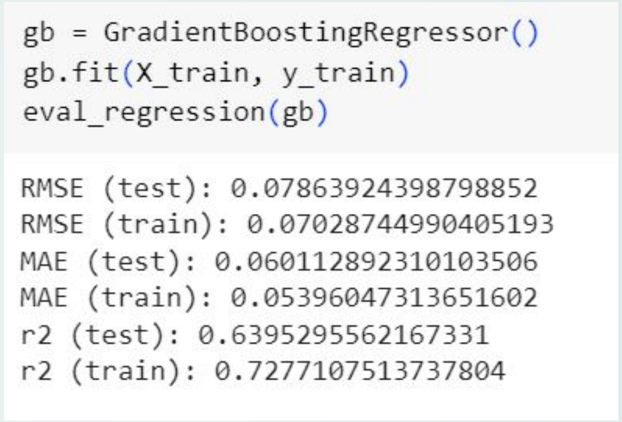
```
lr = LinearRegression()
lr.fit(X_train, y_train)
eval_regression(lr)
```

```
RMSE (test): 0.08023930562807667
RMSE (train): 0.0766880504381747
MAE (test): 0.062909065925168
MAE (train): 0.05913865370627915
r2 (test): 0.6247114404991518
r2 (train): 0.675861726826138
```



```
] ridge = Ridge()
  ridge.fit(X_train, y_train)
  eval_regression(ridge)
```

```
RMSE (test): 0.08013385310418912
RMSE (train): 0.0766449486470965
MAE (test): 0.06267285306511879
MAE (train): 0.05908654445773161
r2 (test): 0.6256972197294315
r2 (train): 0.6762259821158166
```



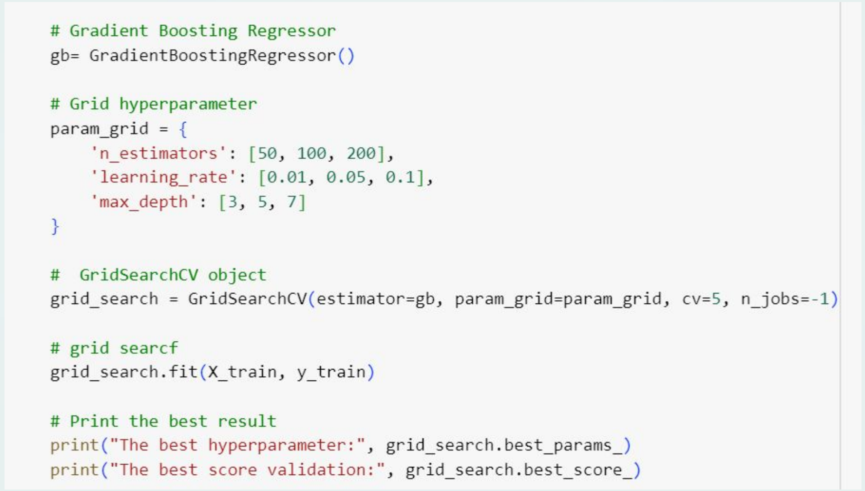
```
gb = GradientBoostingRegressor()
gb.fit(X_train, y_train)
eval_regression(gb)
```

```
RMSE (test): 0.07863924398798852
RMSE (train): 0.07028744990405193
MAE (test): 0.060112892310103506
MAE (train): 0.05396047313651602
r2 (test): 0.6395295562167331
r2 (train): 0.7277107513737804
```

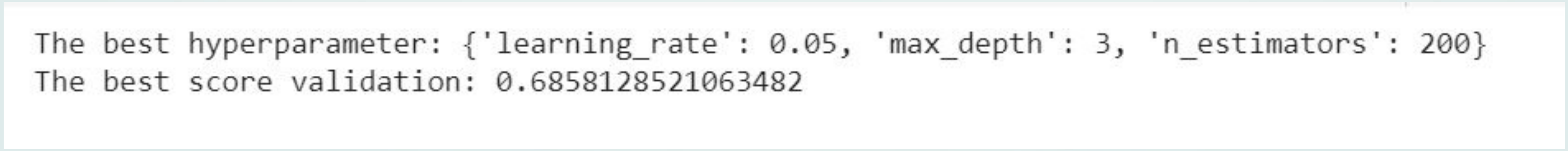
By comparing performance of three models on data train and data test, we consider to use Gradient Boosting as our algorithm models. The RMSE score from GB is the lowest and the R2 is the highest

Hyperparameter Tuning

To optimize the Gradient Boosting performance we do hyperparameter tuning to search the best score and the best parameters.

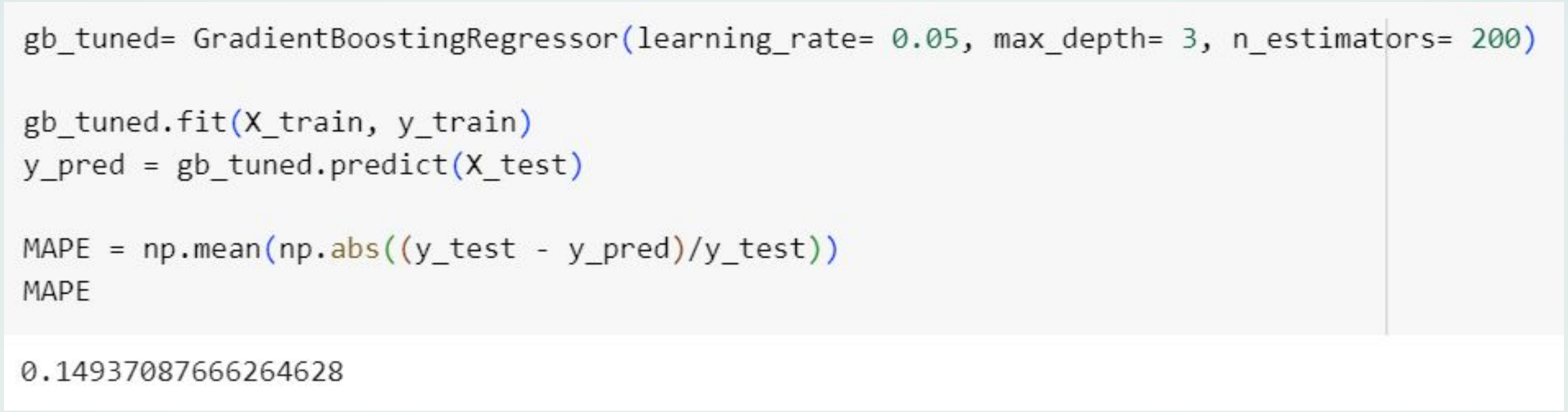


```
# Gradient Boosting Regressor
gb= GradientBoostingRegressor()

# Grid hyperparameter
param_grid = {
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.01, 0.05, 0.1],
    'max_depth': [3, 5, 7]
}

#  GridSearchCV object
grid_search = GridSearchCV(estimator=gb, param_grid=param_grid, cv=5, n_jobs=-1)

# grid searcf
grid_search.fit(X_train, y_train)

# Print the best result
print("The best hyperparameter:", grid_search.best_params_)
print("The best score validation:", grid_search.best_score_)
```



```
The best hyperparameter: {'learning_rate': 0.05, 'max_depth': 3, 'n_estimators': 200}
The best score validation: 0.6858128521063482
```

MAPE (Mean Absolute Percentage Error)

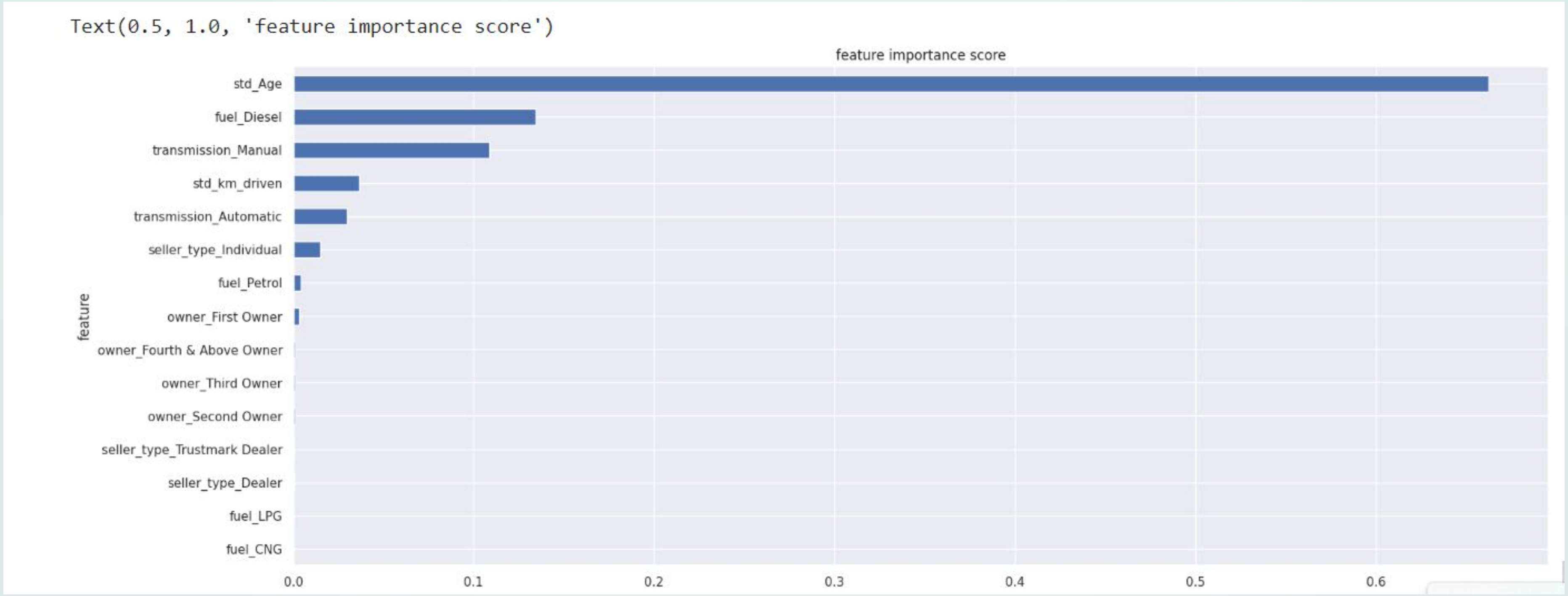


```
gb_tuned= GradientBoostingRegressor(learning_rate= 0.05, max_depth= 3, n_estimators= 200)

gb_tuned.fit(X_train, y_train)
y_pred = gb_tuned.predict(X_test)

MAPE = np.mean(np.abs((y_test - y_pred)/y_test))
MAPE
```

```
0.1493708766264628
```

Our model has MAPE score 0.149 (14.9%). MAPE is a percentage error metric where the value corresponds to the average amount of error that predictions have. Therefore, a lower MAPE is better, where the lower the value the more accurate the model is. Some source stated that MAPE value lower than 20% is considered a good value. This can indicate our model can be considered good.

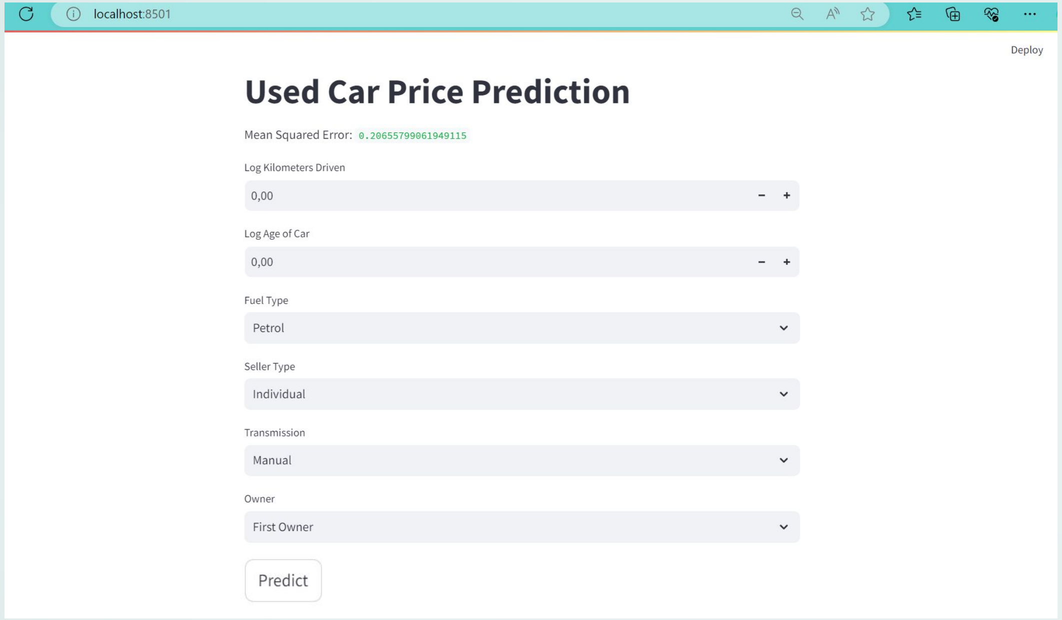
FEATURE IMPORTANCE



```
Text(0.5, 1.0, 'feature importance score')
```

The variables that has strong relationship with selling price is 'Age', fuel type, transmission, and km_driven

MODEL DEPLOYMENT



Used Car Price Prediction

Mean Squared Error: `0.20655799061949115`

Log Kilometers Driven: 0,00

Log Age of Car: 0,00

Fuel Type: Petrol

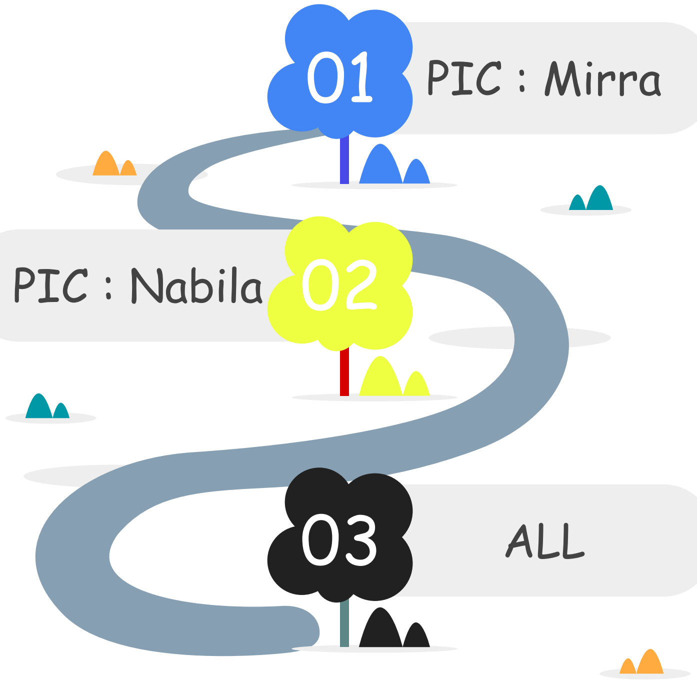
Seller Type: Individual

Transmission: Manual

Owner: First Owner

We deploy the model using Streamlit and run the app locally.

Pembagian Tugas



01	PIC : Mirra	Google Colab
02	PIC : Nabila	Web App
03	ALL	PPT

THANK YOU

“It is better to fail in originality than to succeed in imitation.” —Herman Melville