Used Car Price Prediction

1) Problem statement.

- This dataset comprises used cars sold on cardehko.com in India as well as important features of these cars.
- If user can predict the price of the car based on input features.
- Prediction results can be used to give new seller the price suggestion based on market condition.

2) Data Collection.

- The Dataset is collected from scrapping from cardheko webiste
- The data consists of 13 column and 15411 rows.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import warnings

warnings.filterwarnings("ignore")

%matplotlib inline
```

In [2]:
```
df = pd.read_csv(r"cardekho_imputated.csv", index_col=[0])
```

In [3]:
```
df.head()
```

Out[3]:

	car_name	brand	model	vehicle_age	km_driven	seller_type	fuel_type	transmission_type	mileage	engine	max_power	s
0	Maruti Alto	Maruti	Alto	9	120000	Individual	Petrol	Manual	19.70	796	46.30	
1	Hyundai Grand	Hyundai	Grand	5	20000	Individual	Petrol	Manual	18.90	1197	82.00	
2	Hyundai i20	Hyundai	i20	11	60000	Individual	Petrol	Manual	17.00	1197	80.00	
3	Maruti Alto	Maruti	Alto	9	37000	Individual	Petrol	Manual	20.92	998	67.10	
4	Ford Ecosport	Ford	Ecosport	6	30000	Dealer	Diesel	Manual	22.77	1498	98.59	

Data Cleaning

Handling Missing values

- Handling Missing values
- Handling Duplicates
- Check data type
- Understand the dataset

In [4]:
```
## Check Null Values
##Check features with nan value
df.isnull().sum()
```

Out[4]:
```
car_name             0
brand                0
model                0
vehicle_age          0
km_driven            0
seller_type          0
fuel_type            0
transmission_type    0
mileage              0
engine               0
max_power            0
seats                0
selling_price        0
dtype: int64
```

In [5]:
```
## Remove Unnecessary Columns
df.drop('car_name', axis=1, inplace=True)
df.drop('brand', axis=1, inplace=True)
```

In [6]:
```
df.head()
```

Out[6]:

	model	vehicle_age	km_driven	seller_type	fuel_type	transmission_type	mileage	engine	max_power	seats	selling_price
0	Alto	9	120000	Individual	Petrol	Manual	19.70	796	46.30	5	120000
1	Grand	5	20000	Individual	Petrol	Manual	18.90	1197	82.00	5	550000
2	i20	11	60000	Individual	Petrol	Manual	17.00	1197	80.00	5	215000
3	Alto	9	37000	Individual	Petrol	Manual	20.92	998	67.10	5	226000
4	Ecosport	6	30000	Dealer	Diesel	Manual	22.77	1498	98.59	5	570000

In [7]:
```
df['model'].unique()
```

Out[7]:
```
array(['Alto', 'Grand', 'i20', 'Ecosport', 'Wagon R', 'i10', 'Venue',
       'Swift', 'Verna', 'Duster', 'Cooper', 'Ciaz', 'C-Class', 'Innova',
       'Baleno', 'Swift Dzire', 'Vento', 'Creta', 'City', 'Bolero',
       'Fortuner', 'KWID', 'Amaze', 'Santro', 'XUV500', 'KUV100', 'Ignis',
       'RediGO', 'Scorpio', 'Marazzo', 'Aspire', 'Figo', 'Vitara',
       'Tiago', 'Polo', 'Seltos', 'Celerio', 'GO', '5', 'CR-V',
       'Endeavour', 'KUV', 'Jazz', '3', 'A4', 'Tigor', 'Ertiga', 'Safari',
       'Thar', 'Hexa', 'Rover', 'Eeco', 'A6', 'E-Class', 'Q7', 'Z4', '6',
       'XF', 'X5', 'Hector', 'Civic', 'D-Max', 'Cayenne', 'X1', 'Rapid',
       'Freestyle', 'Superb', 'Nexon', 'XUV300', 'Dzire VXI', 'S90',
       'WR-V', 'XL6', 'Triber', 'ES', 'Wrangler', 'Camry', 'Elantra',
       'Yaris', 'GL-Class', '7', 'S-Presso', 'Dzire LXI', 'Aura', 'XC',
       'Ghibli', 'Continental', 'CR', 'Kicks', 'S-Class', 'Tucson',
       'Harrier', 'X3', 'Octavia', 'Compass', 'CLS', 'redi-GO', 'Glanza',
       'Macan', 'X4', 'Dzire ZXI', 'XC90', 'F-PACE', 'A8', 'MUX',
       'GTC4Lusso', 'GLS', 'X-Trail', 'XE', 'XC60', 'Panamera', 'Alturas',
       'Altroz', 'NX', 'Carnival', 'C', 'RX', 'Ghost', 'Quattroporte',
       'Gurkha'], dtype=object)
```

In [8]:
```
## Getting All Different Types OF Features
num_features = [feature for feature in df.columns if df[feature].dtype != 'O']
print('Num of Numerical Features :', len(num_features))
cat_features = [feature for feature in df.columns if df[feature].dtype == 'O']
print('Num of Categorical Features :', len(cat_features))
discrete_features=[feature for feature in num_features if len(df[feature].unique())<=25]
print('Num of Discrete Features :',len(discrete_features))
continuous_features=[feature for feature in num_features if feature not in discrete_features]
print('Num of Continuous Features :',len(continuous_features))
```

```
Num of Numerical Features : 7
Num of Categorical Features : 4
Num of Discrete Features : 2
Num of Continuous Features : 5
```

In [9]:
```
## Indpendent and dependent features
from sklearn.model_selection import train_test_split
X = df.drop(['selling_price'], axis=1)
y = df['selling_price']
```

In [10]:
```
X.head()
```

Out[10]:

	model	vehicle_age	km_driven	seller_type	fuel_type	transmission_type	mileage	engine	max_power	seats
0	Alto	9	120000	Individual	Petrol	Manual	19.70	796	46.30	5
1	Grand	5	20000	Individual	Petrol	Manual	18.90	1197	82.00	5
2	i20	11	60000	Individual	Petrol	Manual	17.00	1197	80.00	5
3	Alto	9	37000	Individual	Petrol	Manual	20.92	998	67.10	5
4	Ecosport	6	30000	Dealer	Diesel	Manual	22.77	1498	98.59	5

Feature Encoding and Scaling

One Hot Encoding for Columns which had lesser unique values and not ordinal

- One hot encoding is a process by which categorical variables are converted into a form that could be provided to ML algorithms to do a better job in prediction.

In [13]:
```
len(df['model'].unique())
```

Out[13]: 120

In [14]:
```
df['model'].value_counts()
```

Out[14]:
```
i20              906
Swift Dzire      890
Swift            781
Alto             778
City             757
                ...
Quattroporte       1
GTC4Lusso          1
C                  1
Gurkha             1
Altroz             1
Name: model, Length: 120, dtype: int64
```

In [15]:
```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
X['model']=le.fit_transform(X['model'])
```

In [16]:
```
X.head()
```

Out[16]:

	model	vehicle_age	km_driven	seller_type	fuel_type	transmission_type	mileage	engine	max_power	seats
0	7	9	120000	Individual	Petrol	Manual	19.70	796	46.30	5
1	54	5	20000	Individual	Petrol	Manual	18.90	1197	82.00	5
2	118	11	60000	Individual	Petrol	Manual	17.00	1197	80.00	5
3	7	9	37000	Individual	Petrol	Manual	20.92	998	67.10	5
4	38	6	30000	Dealer	Diesel	Manual	22.77	1498	98.59	5

In [19]:
```
len(df['seller_type'].unique()),len(df['fuel_type'].unique()),len(df['transmission_type'].unique())
```

Out[19]: (3, 5, 2)

In [20]:
```
# Create Column Transformer with 3 types of transformers
num_features = X.select_dtypes(exclude="object").columns
onehot_columns = ['seller_type','fuel_type','transmission_type']

from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer

numeric_transformer = StandardScaler()
oh_transformer = OneHotEncoder(drop='first')

preprocessor = ColumnTransformer(
    [
        ("OneHotEncoder", oh_transformer, onehot_columns),
        ("StandardScaler", numeric_transformer, num_features)

    ],remainder='passthrough'

)
```

In [21]:
```
X=preprocessor.fit_transform(X)
```

In [23]:
```
pd.DataFrame(X)
```

Out[23]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	1.0	0.0	0.0	0.0	0.0	1.0	1.0	-1.519714	0.983562	1.247335	-0.000276	-1.324259	-1.263352	-0.403022
1	1.0	0.0	0.0	0.0	0.0	1.0	1.0	-0.225693	-0.343933	-0.690016	-0.192071	-0.554718	-0.432571	-0.403022
2	1.0	0.0	0.0	0.0	0.0	1.0	1.0	1.536377	1.647309	0.084924	-0.647583	-0.554718	-0.479113	-0.403022
3	1.0	0.0	0.0	0.0	0.0	1.0	1.0	-1.519714	0.983562	-0.360667	0.292211	-0.936610	-0.779312	-0.403022
4	0.0	0.0	1.0	0.0	0.0	0.0	1.0	-0.666211	-0.012060	-0.496281	0.735736	0.022918	-0.046502	-0.403022
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
15406	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.508844	0.983562	-0.869744	0.026096	-0.767733	-0.757204	-0.403022
15407	0.0	0.0	0.0	0.0	0.0	1.0	1.0	-0.556082	-1.339555	-0.728763	-0.527711	-0.216964	-0.220803	2.073444
15408	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.407551	-0.012060	0.220539	0.344954	0.022918	0.068225	-0.403022
15409	0.0	0.0	1.0	0.0	0.0	0.0	1.0	1.426247	-0.343933	72.541850	-0.887326	1.329794	0.917158	2.073444
15410	0.0	0.0	0.0	0.0	0.0	1.0	0.0	-1.024131	-1.339555	-0.825631	-0.407839	0.020999	0.395884	-0.403022

15411 rows × 14 columns

In [24]:
```
# separate dataset into train and test
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state=42)
X_train.shape, X_test.shape
```

Out[24]:
```
((12328, 14), (3083, 14))
```

In [25]:
```
X_train
```

Out[25]:
```
array([[ 0.        ,  0.        ,  1.        , ...,  1.75390551,
         2.66249771, -0.40302241],
       [ 1.        ,  0.        ,  0.        , ..., -0.55087963,
        -0.38602844, -0.40302241],
       [ 0.        ,  0.        ,  1.        , ...,  0.89033072,
         3.27453006, -0.40302241],
       ...,
       [ 1.        ,  0.        ,  0.        , ..., -0.9366097 ,
        -0.78070786, -0.40302241],
       [ 0.        ,  0.        ,  0.        , ..., -0.55471774,
        -0.43582879, -0.40302241],
       [ 1.        ,  0.        ,  0.        , ..., -0.04616815,
         0.06194201, -0.40302241]])
```

Model Training And Model Selection

In [26]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression, Ridge,Lasso
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
```

In [27]:
```
##Create a Function to Evaluate Model
def evaluate_model(true, predicted):
    mae = mean_absolute_error(true, predicted)
    mse = mean_squared_error(true, predicted)
    rmse = np.sqrt(mean_squared_error(true, predicted))
    r2_square = r2_score(true, predicted)
    return mae, rmse, r2_square
```

In [28]:
```
## Beginning Model Training
models = {
    "Linear Regression": LinearRegression(),
    "Lasso": Lasso(),
    "Ridge": Ridge(),
    "K-Neighbors Regressor": KNeighborsRegressor(),
    "Decision Tree": DecisionTreeRegressor(),
    "Random Forest Regressor": RandomForestRegressor(),

}

for i in range(len(list(models))):
    model = list(models.values())[i]
    model.fit(X_train, y_train) # Train model

    # Make predictions
    y_train_pred = model.predict(X_train)
    y_test_pred = model.predict(X_test)
```

```python
    # Evaluate Train and Test dataset
    model_train_mae , model_train_rmse, model_train_r2 = evaluate_model(y_train, y_train_pred)

    model_test_mae , model_test_rmse, model_test_r2 = evaluate_model(y_test, y_test_pred)


    print(list(models.keys())[i])

    print('Model performance for Training set')
    print("- Root Mean Squared Error: {:.4f}".format(model_train_rmse))
    print("- Mean Absolute Error: {:.4f}".format(model_train_mae))
    print("- R2 Score: {:.4f}".format(model_train_r2))

    print('----------------------------------')

    print('Model performance for Test set')
    print("- Root Mean Squared Error: {:.4f}".format(model_test_rmse))
    print("- Mean Absolute Error: {:.4f}".format(model_test_mae))
    print("- R2 Score: {:.4f}".format(model_test_r2))

    print('='*35)
    print('\n')
```

```
Linear Regression
Model performance for Training set
- Root Mean Squared Error: 553855.6665
- Mean Absolute Error: 268101.6071
- R2 Score: 0.6218
----------------------------------
Model performance for Test set
- Root Mean Squared Error: 502543.5930
- Mean Absolute Error: 279618.5794
- R2 Score: 0.6645
===================================


Lasso
Model performance for Training set
- Root Mean Squared Error: 553855.6710
- Mean Absolute Error: 268099.2226
- R2 Score: 0.6218
----------------------------------
Model performance for Test set
- Root Mean Squared Error: 502542.6696
- Mean Absolute Error: 279614.7461
- R2 Score: 0.6645
===================================


Ridge
Model performance for Training set
- Root Mean Squared Error: 553856.3160
- Mean Absolute Error: 268059.8015
- R2 Score: 0.6218
----------------------------------
Model performance for Test set
- Root Mean Squared Error: 502533.8230
- Mean Absolute Error: 279557.2169
- R2 Score: 0.6645
===================================


K-Neighbors Regressor
Model performance for Training set
- Root Mean Squared Error: 325886.8736
- Mean Absolute Error: 91467.6671
- R2 Score: 0.8691
----------------------------------
Model performance for Test set
- Root Mean Squared Error: 253118.4156
- Mean Absolute Error: 112704.3545
- R2 Score: 0.9149
===================================


Decision Tree
Model performance for Training set
- Root Mean Squared Error: 20797.2352
- Mean Absolute Error: 5164.8199
- R2 Score: 0.9995
----------------------------------
Model performance for Test set
- Root Mean Squared Error: 309775.5497
- Mean Absolute Error: 125501.4245
- R2 Score: 0.8725
===================================


Random Forest Regressor
Model performance for Training set
- Root Mean Squared Error: 139138.1663
- Mean Absolute Error: 39895.9839
- R2 Score: 0.9761
----------------------------------
Model performance for Test set
- Root Mean Squared Error: 221936.2665
- Mean Absolute Error: 100966.5873
- R2 Score: 0.9346
===================================
```

In [29]:

```
#Initialize few parameter for Hyperparamter tuning
knn_params = {"n_neighbors": [2, 3, 10, 20, 40, 50]}
rf_params = {"max_depth": [5, 8, 15, None, 10],
             "max_features": [5, 7, "auto", 8],
```

```python
              "min_samples_split": [2, 8, 15, 20],
              "n_estimators": [100, 200, 500, 1000]}
```

In [30]:
```python
# Models list for Hyperparameter tuning
randomcv_models = [('KNN', KNeighborsRegressor(), knn_params),
                   ("RF", RandomForestRegressor(), rf_params)

                   ]
```

In [31]:
```python
##Hyperparameter Tuning
from sklearn.model_selection import RandomizedSearchCV

model_param = {}
for name, model, params in randomcv_models:
    random = RandomizedSearchCV(estimator=model,
                                   param_distributions=params,
                                   n_iter=100,
                                   cv=3,
                                   verbose=2,
                                   n_jobs=-1)
    random.fit(X_train, y_train)
    model_param[name] = random.best_params_

for model_name in model_param:
    print(f"---------------- Best Params for {model_name} -------------------")
    print(model_param[model_name])
```

```
Fitting 3 folds for each of 6 candidates, totalling 18 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.
[Parallel(n_jobs=-1)]: Done   5 out of  18 | elapsed:    2.8s remaining:    7.3s
[Parallel(n_jobs=-1)]: Done  15 out of  18 | elapsed:    3.0s remaining:    0.5s
[Parallel(n_jobs=-1)]: Done  18 out of  18 | elapsed:    3.0s finished
Fitting 3 folds for each of 100 candidates, totalling 300 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.
[Parallel(n_jobs=-1)]: Done  98 tasks      | elapsed:   15.9s
[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed:   53.9s finished
---------------- Best Params for KNN -------------------
{'n_neighbors': 10}
---------------- Best Params for RF -------------------
{'n_estimators': 100, 'min_samples_split': 2, 'max_features': 'auto', 'max_depth': None}
```

In [33]:
```python
## Retraining the models with best parameters
models = {
    "Random Forest Regressor": RandomForestRegressor(n_estimators=100, min_samples_split=2, max_features='auto'
                                                     n_jobs=-1),
     "K-Neighbors Regressor": KNeighborsRegressor(n_neighbors=10, n_jobs=-1)

}
for i in range(len(list(models))):
    model = list(models.values())[i]
    model.fit(X_train, y_train) # Train model

    # Make predictions
    y_train_pred = model.predict(X_train)
    y_test_pred = model.predict(X_test)

    model_train_mae , model_train_rmse, model_train_r2 = evaluate_model(y_train, y_train_pred)

    model_test_mae , model_test_rmse, model_test_r2 = evaluate_model(y_test, y_test_pred)

    print(list(models.keys())[i])

    print('Model performance for Training set')
    print("- Root Mean Squared Error: {:.4f}".format(model_train_rmse))
    print("- Mean Absolute Error: {:.4f}".format(model_train_mae))
    print("- R2 Score: {:.4f}".format(model_train_r2))

    print('----------------------------------')

    print('Model performance for Test set')
    print("- Root Mean Squared Error: {:.4f}".format(model_test_rmse))
    print("- Mean Absolute Error: {:.4f}".format(model_test_mae))
    print("- R2 Score: {:.4f}".format(model_test_r2))

    print('='*35)
    print('\n')
```

```
Random Forest Regressor
Model performance for Training set
- Root Mean Squared Error: 129998.4877
- Mean Absolute Error: 39738.5156
- R2 Score: 0.9792
----------------------------------
Model performance for Test set
- Root Mean Squared Error: 228415.2018
- Mean Absolute Error: 102398.2134
- R2 Score: 0.9307
===================================


K-Neighbors Regressor
Model performance for Training set
- Root Mean Squared Error: 363464.0671
- Mean Absolute Error: 103451.3465
- R2 Score: 0.8371
----------------------------------
Model performance for Test set
- Root Mean Squared Error: 263872.0571
- Mean Absolute Error: 117483.0441
- R2 Score: 0.9075
===================================
```