Supervised Learning: XG-Boosting

Section 1. Main objective of the analysis that specifies whether your model will be focused on prediction or interpretation.

Main objective of the analysis is prediction of Used Car Price and finding the most important feature on determining the Used Car Price.

Section 2. Brief description of the data set you chose and a summary of its attributes.

Used Car Price Dataset

https://www.kaggle.com/datasets/ljanjughazyan/cars1

Input Variables:

- Make
- Model
- Type
- Origin
- Drivetrain
- Invoice
- Engine Size
- Cylinders
- Horsepower
- MPG_City
- MPG_Highway
- Weight
- Wheelbas
- Length

Output Variable:

- MSRP(Price)

Section 3. Data exploration and actions taken for data cleaning and feature engineering.

```python
import numpy as np # Multi-dimensional array object
import pandas as pd # Data Manipulation
import seaborn as sns # Data Visualization
import matplotlib.pyplot as plt # Data Visualization
import plotly.express as px # Interactive Data Visualization
from jupyterthemes  import jtplot # Jupyter Notebook Theme
jtplot.style(theme = 'monokai', context = 'notebook', ticks = True, grid = False)
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot # Offline version of the Plotly modules.
```

✓ 5.8s

```python
# Read the CSV file
car_df = pd.read_csv('cars_data.csv')
```

✓ 0.1s

```python
# Load the top 10 instances
car_df.head(10)
```

✓ 0.7s

	Make	Model	Type	Origin	DriveTrain	MSRP	Invoice	EngineSize	Cylinders	Horsepower	MPG_City	MPG_Highway	Weight	Wheelbase	Length
0	Acura	MDX	SUV	Asia	All	$36,945	$33,337	3.5	6.0	265	17	23	4451	106	189
1	Acura	RSX Type S 2dr	Sedan	Asia	Front	$23,820	$21,761	2.0	4.0	200	24	31	2778	101	172
2	Acura	TSX 4dr	Sedan	Asia	Front	$26,990	$24,647	2.4	4.0	200	22	29	3230	105	183
3	Acura	TL 4dr	Sedan	Asia	Front	$33,195	$30,299	3.2	6.0	270	20	28	3575	108	186
4	Acura	3.5 RL 4dr	Sedan	Asia	Front	$43,755	$39,014	3.5	6.0	225	18	24	3880	115	197
5	Acura	3.5 RL w/Navigation 4dr	Sedan	Asia	Front	$46,100	$41,100	3.5	6.0	225	18	24	3893	115	197
6	Acura	NSX coupe 2dr manual S	Sports	Asia	Rear	$89,765	$79,978	3.2	6.0	290	17	24	3153	100	174
7	Audi	A4 1.8T 4dr	Sedan	Europe	Front	$25,940	$23,508	1.8	4.0	170	22	31	3252	104	179
8	Audi	A41.8T convertible 2dr	Sedan	Europe	Front	$35,940	$32,506	1.8	4.0	170	23	30	3638	105	180
9	Audi	A4 3.0 4dr	Sedan	Europe	Front	$31,840	$28,846	3.0	6.0	220	20	28	3462	104	179

Check the shape of the dataframe

```python
# Check the shape of the dataframe
car_df.shape
```

✓ 0.4s

```
(428, 15)
```

Check the missing values

```python
# Check if any missing values are present in the dataframe
car_df.isnull().sum()
```

✓ 0.4s

```
Make           0
Model          0
Type           0
Origin         0
DriveTrain     0
MSRP           0
Invoice        0
EngineSize     0
Cylinders      2
Horsepower     0
MPG_City       0
MPG_Highway    0
Weight         0
Wheelbase      0
Length         0
dtype: int64
```

Drop the rows that contain the missing values

```
car_df = car_df.dropna()
✓ 0.3s
```

Check the duplicated column

```
car_df.duplicated().sum()
✓ 0.7s

0
```

 # In order to address the issue of missing values in the dataset, I implemented a strategy of removing rows that contained such values.

Summarize the dataset

```
# Obtain the summary of the dataframe
car_df.info()
✓ 0.5s

<class 'pandas.core.frame.DataFrame'>
Int64Index: 426 entries, 0 to 427
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Make         426 non-null    object 
 1   Model        426 non-null    object 
 2   Type         426 non-null    object 
 3   Origin       426 non-null    object 
 4   DriveTrain   426 non-null    object 
 5   MSRP         426 non-null    object 
 6   Invoice      426 non-null    object 
 7   EngineSize   426 non-null    float64
 8   Cylinders    426 non-null    float64
 9   Horsepower   426 non-null    int64  
 10  MPG_City     426 non-null    int64  
 11  MPG_Highway  426 non-null    int64  
 12  Weight       426 non-null    int64  
 13  Wheelbase    426 non-null    int64  
 14  Length       426 non-null    int64  
dtypes: float64(2), int64(6), object(7)
memory usage: 53.2+ KB
```

If we look at the MSRP and Invoice columns, they contain prices but the type of values is 'object.' Therefore, I changed the type of the values to 'integer.'

```
# Convert MSRP and Invoice datatype to integer so we need to remove $ sign and comma (,) from these 2 columns

car_df["MSRP"] = car_df["MSRP"].str.replace("$", "")
car_df["MSRP"] = car_df["MSRP"].str.replace(",", "")
car_df["MSRP"] = car_df["MSRP"].astype(int)

car_df["Invoice"] = car_df["Invoice"].str.replace("$", "")
car_df["Invoice"] = car_df["Invoice"].str.replace(",", "")
car_df["Invoice"] = car_df["Invoice"].astype(int)
```

✓ 0.7s

Summarize the dataset

```
# Display the updated summary of the dataframe
car_df.info()
```

✓ 0.4s

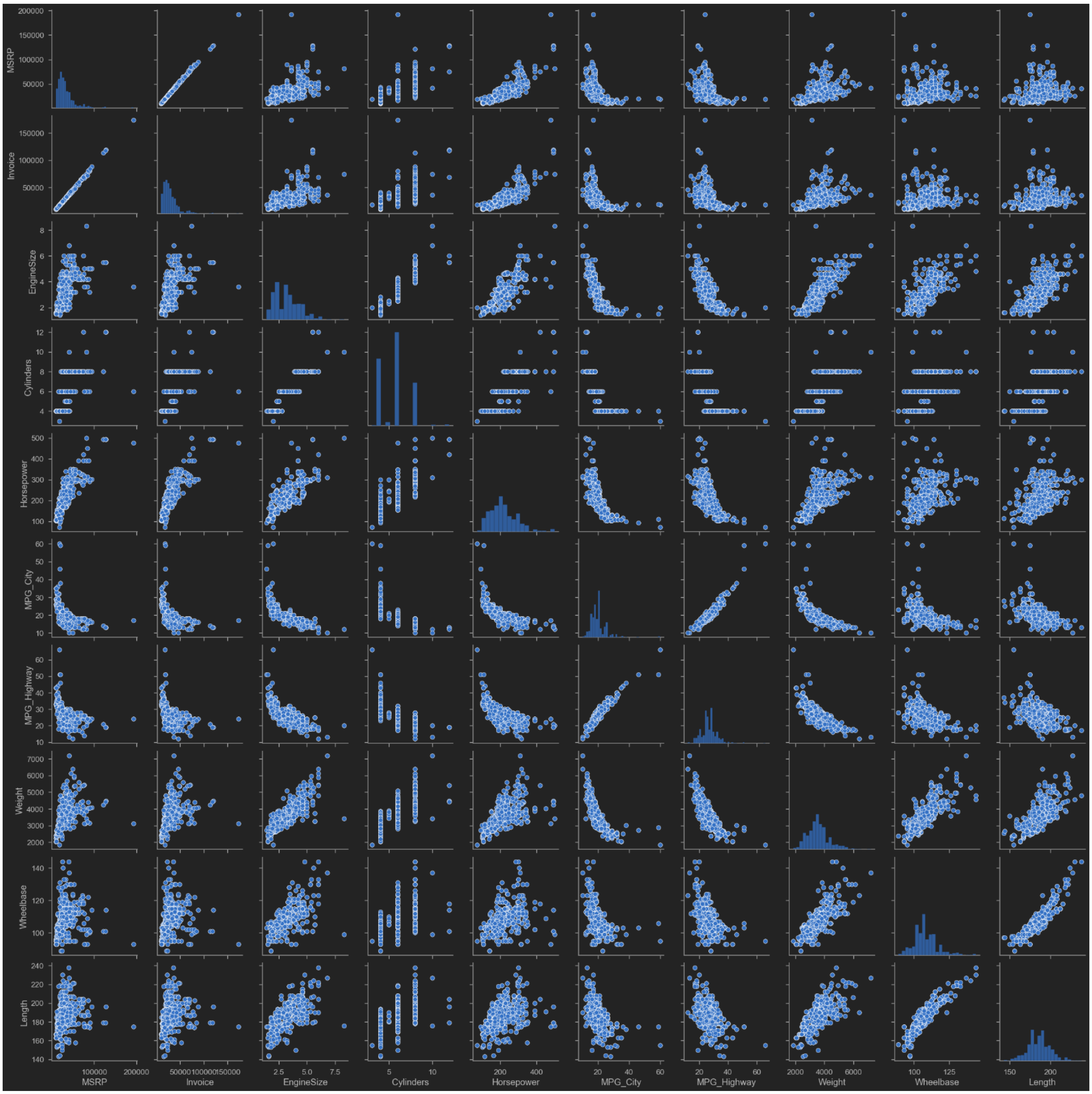
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 426 entries, 0 to 427
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Make         426 non-null    object
 1   Model        426 non-null    object
 2   Type         426 non-null    object
 3   Origin       426 non-null    object
 4   DriveTrain   426 non-null    object
 5   MSRP         426 non-null    int32
 6   Invoice      426 non-null    int32
 7   EngineSize   426 non-null    float64
 8   Cylinders    426 non-null    float64
 9   Horsepower   426 non-null    int64
 10  MPG_City     426 non-null    int64
 11  MPG_Highway  426 non-null    int64
 12  Weight       426 non-null    int64
 13  Wheelbase    426 non-null    int64
 14  Length       426 non-null    int64
dtypes: float64(2), int32(2), int64(6), object(5)
memory usage: 49.9+ KB
```

After that we will perform data visualization by

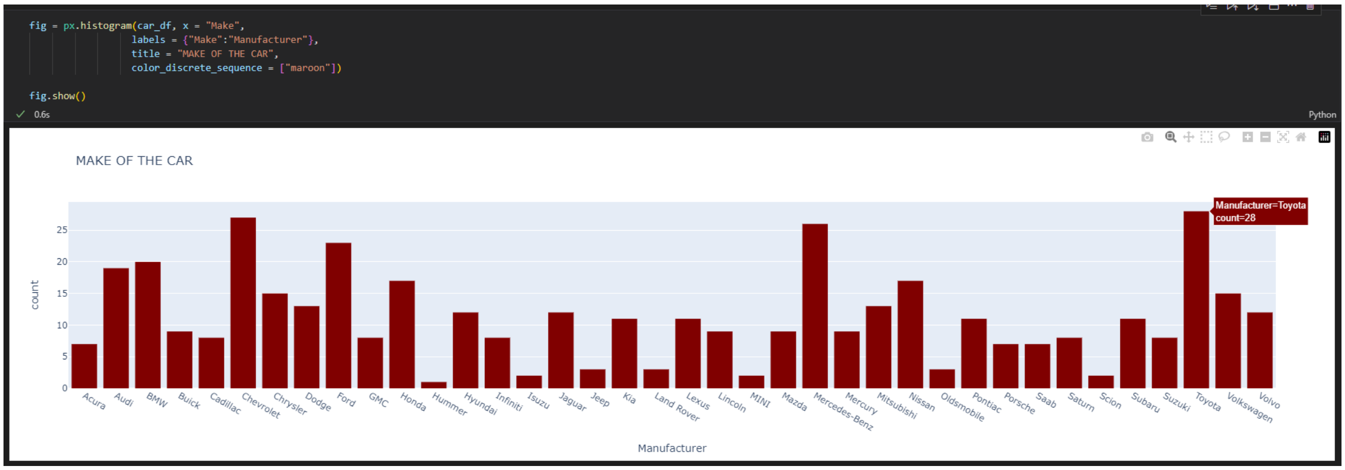
- Scatterplots for joint relationship
- Histogram for univariable distributions
-

```python
# scatterplots for joint relationships and histograms for univariate distributions
sns.pairplot(data= car_df)
```

✓ 16.2s

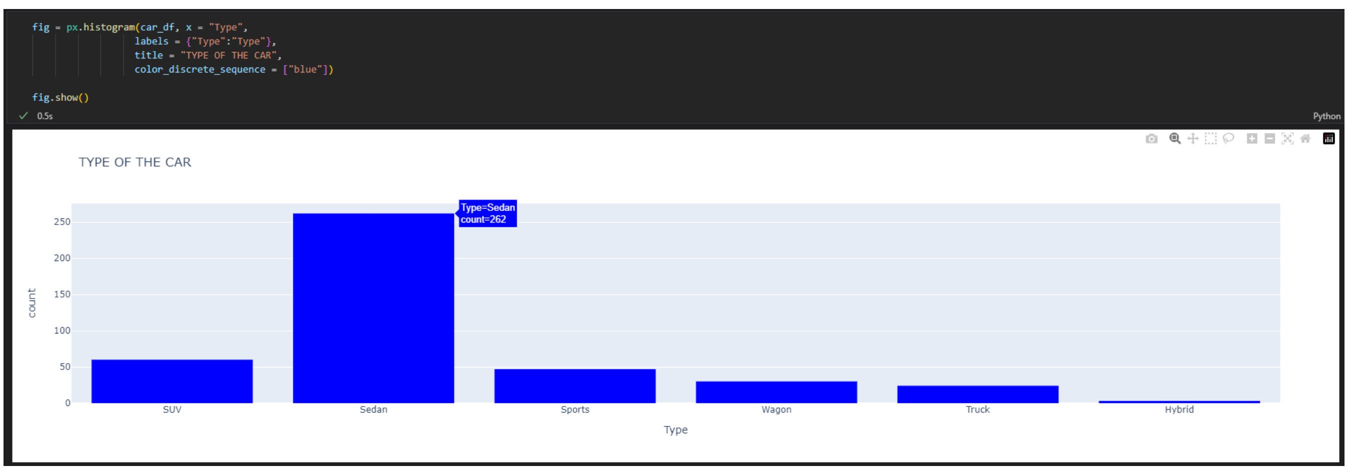


We observe the distribution of car makes by manufacturer through a histogram



From the histogram, we can see that the manufacturer with the highest number of cars is Toyota, with 28 cars

Next, we will observe the distribution of car types through a histogram



From the histogram, we can see that the highest number of car type is Sedan, with 262 cars

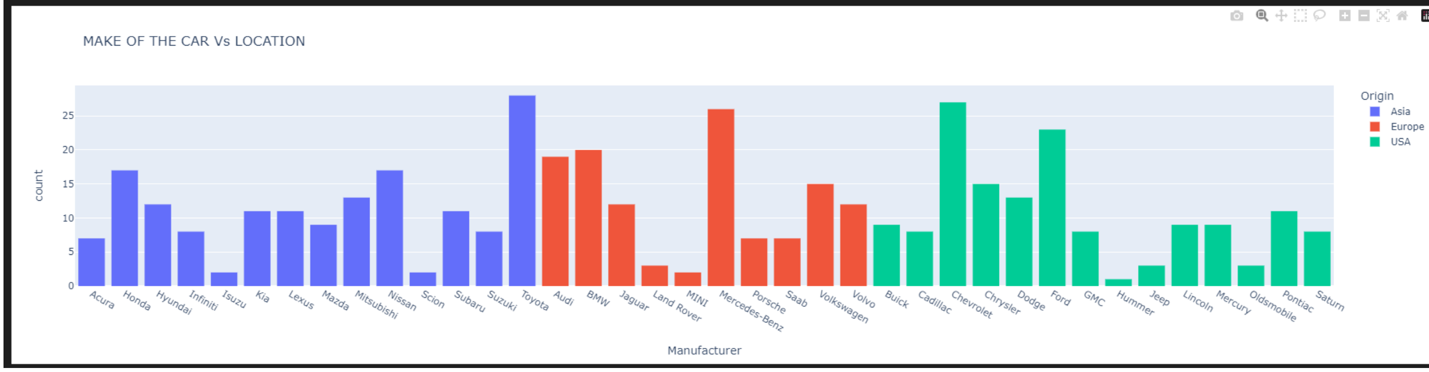
Next, we will observe the distribution of location of the car sales through a histogram

From the histogram, we can see that the highest number of locations of car sales is Asia, with 156 cars

We also observe the distribution of car makes with location through a histogram

```python
# Plot the make of the car and its location
fig = px.histogram(car_df, x = "Make",
                   color = "Origin",
                   labels = {"Make":"Manufacturer"},
                   title = "MAKE OF THE CAR Vs LOCATION")

fig.show()
```
✓ 0.7s Python



From the histogram, we can see that most manufacturers make cars in Asia

Next, we will see the relationship of each feature by correlation matrix

```python
# Obtain the correlation matrix
car_df.corr()
```
✓ 0.1s

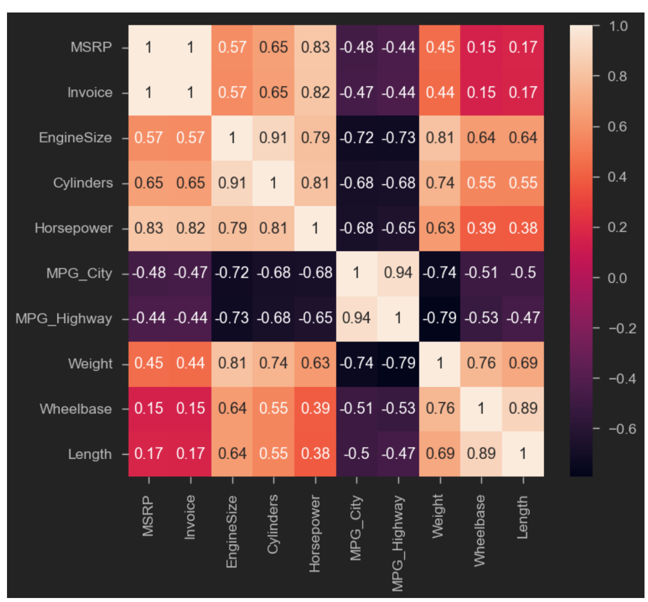
```
C:\Users\z004mbjn\AppData\Local\Temp\ipykernel_12180\2691162371.py:2: FutureWarning:

The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only valid columns or specify the value of numeric_only to silence this warning.
```

	MSRP	Invoice	EngineSize	Cylinders	Horsepower	MPG_City	MPG_Highway	Weight	Wheelbase	Length
MSRP	1.000000	0.999132	0.573238	0.649742	0.827296	-0.475916	-0.440523	0.447987	0.151665	0.171060
Invoice	0.999132	1.000000	0.566019	0.645226	0.824079	-0.471304	-0.435452	0.441916	0.148003	0.165645
EngineSize	0.573238	0.566019	1.000000	0.908002	0.793250	-0.717860	-0.725901	0.808707	0.638947	0.636015
Cylinders	0.649742	0.645226	0.908002	1.000000	0.810341	-0.684402	-0.676100	0.742209	0.546730	0.547783
Horsepower	0.827296	0.824079	0.793250	0.810341	1.000000	-0.677034	-0.647425	0.631758	0.387561	0.382386
MPG_City	-0.475916	-0.471304	-0.717860	-0.684402	-0.677034	1.000000	0.940993	-0.740418	-0.508029	-0.504184
MPG_Highway	-0.440523	-0.435452	-0.725901	-0.676100	-0.647425	0.940993	1.000000	-0.793615	-0.525457	-0.468756
Weight	0.447987	0.441916	0.808707	0.742209	0.631758	-0.740418	-0.793615	1.000000	0.760857	0.689168
Wheelbase	0.151665	0.148003	0.638947	0.546730	0.387561	-0.508029	-0.525457	0.760857	1.000000	0.889838
Length	0.171060	0.165645	0.636015	0.547783	0.382386	-0.504184	-0.468756	0.689168	0.889838	1.000000

We also visualize in heatmap

```python
sns.heatmap(data=car_df.corr(), annot = True)
```
✓ 0.4s



Next, we will prepare the data before model training

If we look at the dataset again, we can see that there are 5 columns of object type. To handle these columns, we need to perform One-Hot Encoding to convert them into binary variables

```
car_df.head()
```

✓ 0.1s

	Make	Model	Type	Origin	DriveTrain	MSRP	Invoice	EngineSize	Cylinders	Horsepower	MPG_City	MPG_Highway	Weight	Wheelbase	Length
0	Acura	MDX	SUV	Asia	All	36945	33337	3.5	6.0	265	17	23	4451	106	189
1	Acura	RSX Type S 2dr	Sedan	Asia	Front	23820	21761	2.0	4.0	200	24	31	2778	101	172
2	Acura	TSX 4dr	Sedan	Asia	Front	26990	24647	2.4	4.0	200	22	29	3230	105	183
3	Acura	TL 4dr	Sedan	Asia	Front	33195	30299	3.2	6.0	270	20	28	3575	108	186
4	Acura	3.5 RL 4dr	Sedan	Asia	Front	43755	39014	3.5	6.0	225	18	24	3880	115	197

```
# Perform One-Hot Encoding for "Make", "Model", "Type", "Origin", and "DriveTrain"
df_dummy = pd.get_dummies(car_df, columns=['Make', 'Model', 'Type', 'Origin', 'DriveTrain'])
```

The dataset that is encoded will contain in df_dummy

```
df_dummy
```

✓ 0.4s Python

	MSRP	Invoice	EngineSize	Cylinders	Horsepower	MPG_City	MPG_Highway	Weight	Wheelbase	Length	...	Type_Sedan	Type_Sports	Type_Truck	Type_Wagon	Origin_Asia	Origin_Europe	Origin_USA	DriveTrain_All	DriveTrain_Front	DriveTr
0	36945	33337	3.5	6.0	265	17	23	4451	106	189	...	0	0	0	0	1	0	0	1	0	
1	23820	21761	2.0	4.0	200	24	31	2778	101	172	...	1	0	0	0	1	0	0	0	1	
2	26990	24647	2.4	4.0	200	22	29	3230	105	183	...	1	0	0	0	1	0	0	0	1	
3	33195	30299	3.2	6.0	270	20	28	3575	108	186	...	1	0	0	0	1	0	0	0	1	
4	43755	39014	3.5	6.0	225	18	24	3880	115	197	...	1	0	0	0	1	0	0	0	1	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
423	40565	38203	2.4	5.0	197	21	28	3450	105	186	...	1	0	0	0	0	1	0	0	1	
424	42565	40083	2.3	5.0	242	20	26	3450	105	186	...	1	0	0	0	0	1	0	0	1	
425	45210	42573	2.9	6.0	268	19	26	3653	110	190	...	1	0	0	0	0	1	0	0	1	
426	26135	24641	1.9	4.0	170	22	29	2822	101	180	...	0	0	0	1	0	1	0	0	1	
427	35145	33112	2.5	5.0	208	20	27	3823	109	186	...	0	0	0	1	0	1	0	1	0	

426 rows × 483 columns

From the correlation values, we can see that the correlation value of Invoice with MSRP is 1 so we will reject that feature due to avoid redundant information to the model.

```
# Invoice feature does not contribute to car price prediction
df_data = df_dummy.drop(['Invoice'], axis=1)
```

✓ 0.4s Python

```
df_data
```

✓ 0.5s Python

	MSRP	EngineSize	Cylinders	Horsepower	MPG_City	MPG_Highway	Weight	Wheelbase	Length	Make_Acura	...	Type_Sedan	Type_Sports	Type_Truck	Type_Wagon	Origin_Asia	Origin_Europe	Origin_USA	DriveTrain_All	DriveTrain_Front	Dr
0	36945	3.5	6.0	265	17	23	4451	106	189	1	...	0	0	0	0	1	0	0	1	0	
1	23820	2.0	4.0	200	24	31	2778	101	172	1	...	1	0	0	0	1	0	0	0	1	
2	26990	2.4	4.0	200	22	29	3230	105	183	1	...	1	0	0	0	1	0	0	0	1	
3	33195	3.2	6.0	270	20	28	3575	108	186	1	...	1	0	0	0	1	0	0	0	1	
4	43755	3.5	6.0	225	18	24	3880	115	197	1	...	1	0	0	0	1	0	0	0	1	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
423	40565	2.4	5.0	197	21	28	3450	105	186	0	...	1	0	0	0	0	1	0	0	1	
424	42565	2.3	5.0	242	20	26	3450	105	186	0	...	1	0	0	0	0	1	0	0	1	
425	45210	2.9	6.0	268	19	26	3653	110	190	0	...	1	0	0	0	0	1	0	0	1	
426	26135	1.9	4.0	170	22	29	2822	101	180	0	...	0	0	0	1	0	1	0	0	1	
427	35145	2.5	5.0	208	20	27	3823	109	186	0	...	0	0	0	1	0	1	0	1	0	

426 rows × 482 columns

Next is we will separate the input and output by X is for input features and Y is for output

```python
# Feeding input features to X and output (MSRP) to y
X = df_data.drop("MSRP", axis = 1)
y = df_data["MSRP"]
```

✓ 0.4s

Section 4. Train-Test Split and Regression

We will separate the data into train and test. For in train data, we will use 70% of dataset and test data, we will use 30% of dataset

```
from sklearn.model_selection import train_test_split
```
✓ 0.5s

```
X_train, X_test, y_train, y_test= train_test_split(X, y, test_size = 0.3)
```
✓ 0.5s

We will use 4 regressor which are Linear Regression, Decision Tree, Random Forest and XG-Boosting

First regressor is Linear Regression

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error, accuracy_score
from math import sqrt
```
✓ 0.3s

```
LinearRegression_model = LinearRegression()
LinearRegression_model.fit(X_train, y_train)
```
✓ 0.1s

```
▾ LinearRegression
LinearRegression()
```

```
accuracy_LinearRegression = LinearRegression_model.score(X_test, y_test)
accuracy_LinearRegression
```
✓ 0.4s

```
0.7842411519617806
```

We can see that the accuracy of the Linear Regression is 78.42%

Second regressor is Decision Tree

```
from sklearn.tree import DecisionTreeRegressor
DecisionTree_model = DecisionTreeRegressor()
DecisionTree_model.fit(X_train, y_train)
```
✓ 0.4s

```
▾ DecisionTreeRegressor
DecisionTreeRegressor()
```

```
accuracy_DecisionTree = DecisionTree_model.score(X_test, y_test)
accuracy_DecisionTree
```
✓ 0.4s

```
0.7873205242271537
```

We can see that the accuracy of the Decision Tree is 78.73%

Third regressor is Random Forest

```
from sklearn.ensemble import RandomForestRegressor
```
✓ 0.3s

```
RandomForest_model = RandomForestRegressor(n_estimators= 5, max_depth=5)
RandomForest_model.fit(X_train, y_train)
```
✓ 0.6s

```
▾ RandomForestRegressor
RandomForestRegressor(max_depth=5, n_estimators=5)
```

```
accuracy_RandomForest= RandomForest_model.score(X_test, y_test)
accuracy_RandomForest
```
✓ 0.3s

```
0.713104420311033
```

We can see that the accuracy of the Random Forest is 71.31%

The last regressor is XG-Boosting

```
from xgboost import XGBRegressor
```

✓ 0.3s

```
model = XGBRegressor()
model.fit(X_train,y_train)
```

✓ 0.1s

```
XGBRegressor

XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             n_estimators=100, n_jobs=None, num_parallel_tree=None,
             predictor=None, random_state=None, ...)
```

```
accuracy_XGBoost = model.score(X_test, y_test)
accuracy_XGBoost
```

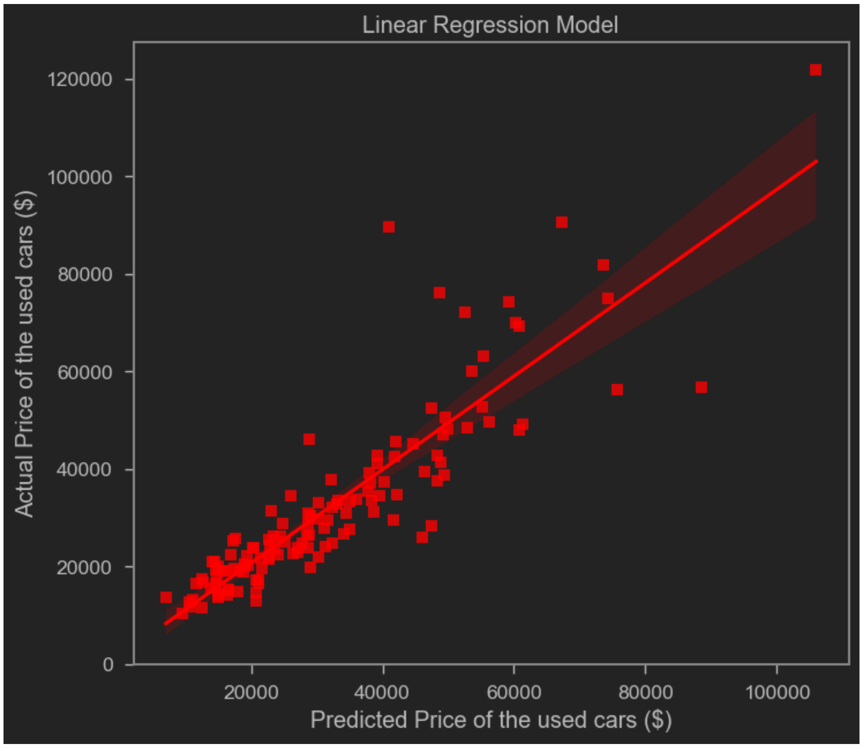
✓ 0.3s

```
0.8181797617682139
```

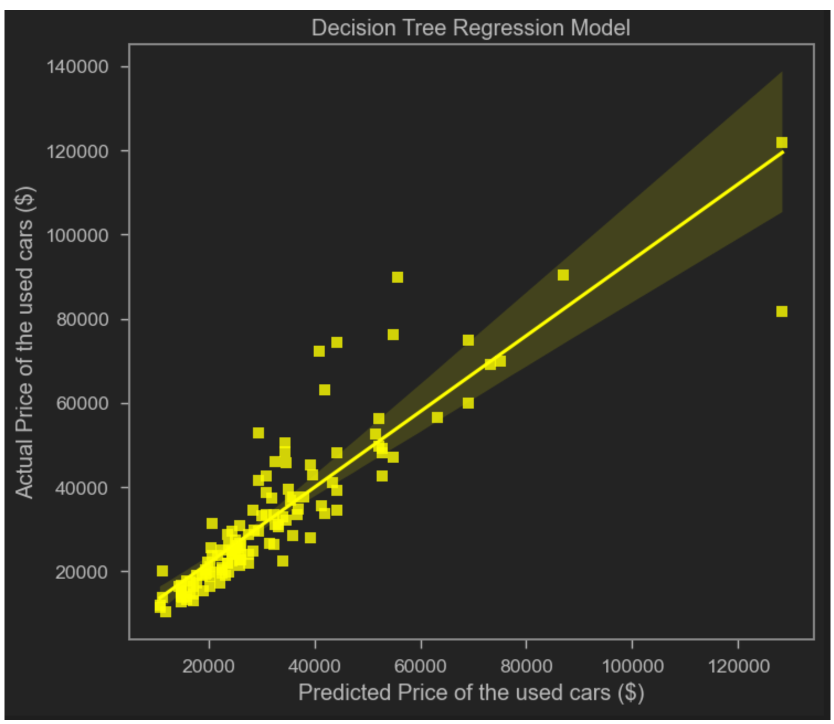
We can see that the accuracy of the XG-Boosting is 81.82%

Next, we will compare models and calculate regression KPIs

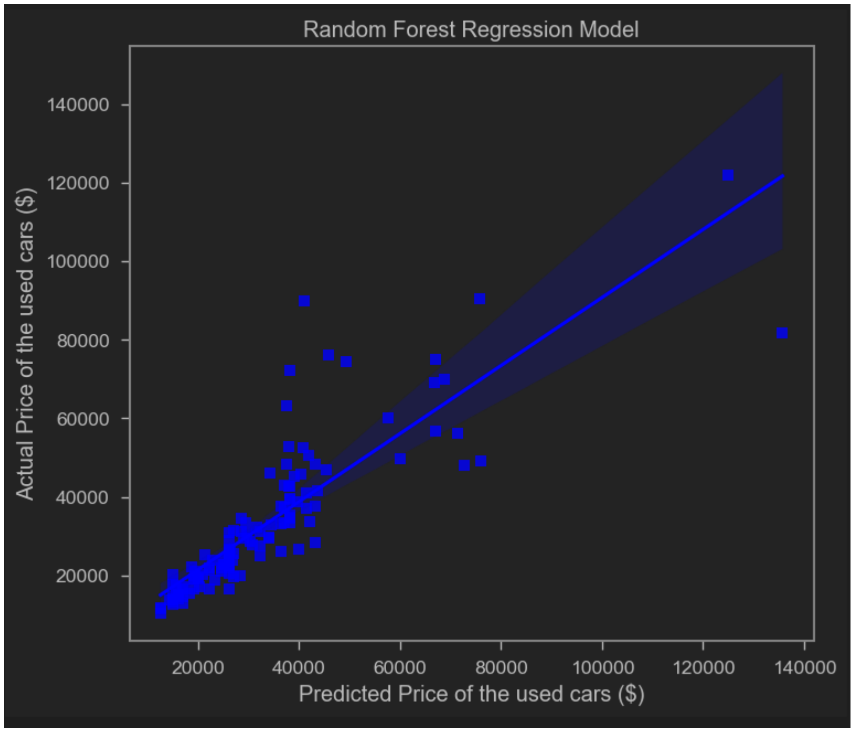
Linear Regression



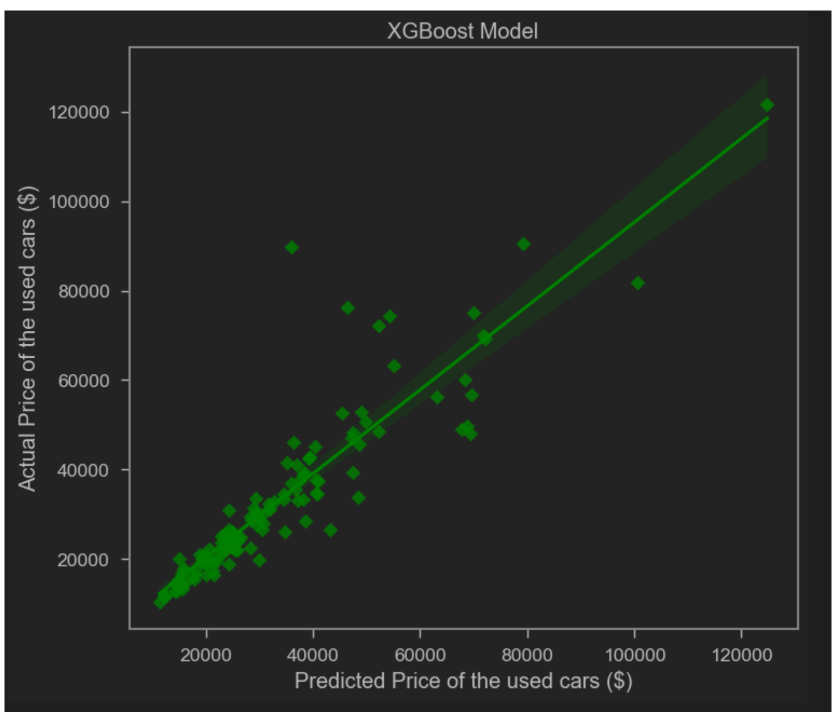
Decision Tree



Random Forest



XG-Boosting



KPIs for each Regressor

	Linear Regression	Decision Tree	Random Forest	XG-Boosting
RMSE	8773.27	8710.43	10116.70	8053.75
MSE	76970197.94	75871657.16	102347643.02	64862877.48
MAE	5575.07	5123.66	5511.31	4304.74
R2	0.78	0.79	0.71	0.82

Section 5. Final Model

According to the results Gradient Boosting Regressor is the best model for this dataset.

Section 6. Results

Considering correlations between features, Invoice is the key features to determine used car price. Accuracy is %82For better results, hyperparameter tuning to the models will increase accuracy.