Used Cars' Analysis and Prediction

DS mini-project

Group No: 5

Group Members:

Name	Roll No	GR No.	Batch
1. Atharva Adhav	321001	21810367	A1
2. Abhishek Ghule	321015	21810518	A1
3. Suyash Mudiya	321050	21810419	A2
4. Shloka Walekar	321060	21810507	A3

Problem Statement

Visualizing the Car's Data using different Data-visualization techniques & Predicting the price of car using different models and checking the accuracy.

Objectives

1. Data-Interpretation & Data-Preprocessing for better model accuracy.
2. Visualization of the dataset by using various plots and graphs.
3. Car Price Prediction using different attributes of data.
4. Using the Correlation and Confusion Matrix to interpret data better.
5. Checking accuracy of Linear Regression, Decision Tree and Naive Bayes Models.

THEORY

Data Visualization : Seaborn gives us the capability to create amplified data visuals. This helps us understand the data by displaying it in a visual context to unearth any hidden correlations between variables or trends that might not be obvious initially. Seaborn has a high-level interface as compared to the low level of Matplotlib.

Data cleaning : Data cleaning is the process of preparing data for analysis by removing or modifying data that is incorrect, incomplete, irrelevant, duplicated, or improperly formatted.This data is usually not necessary or helpful when it comes to analyzing data because it may hinder the process or provide inaccurate results.

Data integration : Data integration is the process of combining data from different sources into a single, unified view. Integration begins with the ingestion process, and includes steps such as cleansing, ETL mapping, and transformation.

Data transformation : Data transformation is the process of converting data from one format to another, typically from the format of a source system into the required format of a destination system. Data transformation is a component of most data integration and data management tasks, such as data wrangling and data warehousing.

Linear Regression

In statistics, linear regression is a linear approach to modelling the relationship between a scalar response (or dependent variable) and one or more explanatory variables (or independent variables).

In linear regression, the relationships are modeled using linear predictor functions whose unknown model parameters are estimated from the data.

Logistic Regression

Logistic regression is a statistical model that in its basic form uses a logistic function to model a binary dependent variable, although many more complex extensions exist. In regression analysis, logistic regression is estimating the parameters of a logistic model (a form of binary regression).

The logistic regression model itself simply models probability of output in terms of input and does not perform statistical classification (it is not a classifier), though it can be used to make a classifier, for instance by choosing a cutoff value and classifying inputs with probability greater than the cutoff as one class, below the cutoff as the other; this is a common way to make a binary classifier.

Decision Tree

A decision tree is a flowchart-like structure in which each internal node represents a "test" on an attribute (e.g. whether a coin flip comes up heads or tails), each branch represents the outcome of the test, and each leaf node represents a class label (decision taken after computing all attributes). The paths from root to leaf represent classification rules.

Naive Bayes Classifier

Naive Bayes classifiers are a collection of classification algorithms based on Bayes 'Theorem. It is not a single algorithm but a family of algorithms where all of them share a common principle

The fundamental Naive Bayes assumption is that each feature makes an:

1. Independent 2. Equal

Scatter plot:

A scatter chart shows the relationship between two different variables and it can reveal the distribution trends. It should be used when there are many different data points, and you want to highlight similarities in the data set. This is useful when looking for outliers and for understanding the distribution of your data.

Histogram:

The histogram represents the frequency of occurrence of specific phenomena which lie within a specific range of values and arranged in consecutive and fixed intervals.

Bar plot:

A bar plot or bar chart is a graph that represents the category of data with rectangular bars with lengths and heights that is proportional to the values which they represent. The bar plots can be plotted horizontally or vertically. A bar chart describes the comparisons between the discrete categories. One of the axis of the plot represents the specific categories being compared, while the other axis represents the measured values corresponding to those categories.

Box plot chart:

A box plot is a graphical representation of statistical data based on the minimum, first quartile, median, third quartile, and maximum. The term “box plot” comes from the fact that the graph looks like a rectangle with lines extending from the top and bottom. Because of the extending lines, this type of graph is sometimes called a box-and-whisker plot.

DATASET USED

USED-CARS-CATALOG:

The dataset is collected from various web resources in order to explore the used cars market and try to build a model that effectively predicts the price of the car based on its parameters (both numerical and categorical).

Source: Kaggle

METHODOLOGY /ALGORITHM

We used Linear Regression, Decision Tree and Naive Bayes Models. In linear regression, the relationships are modeled using linear predictor functions whose unknown model parameters are estimated from the data. Also, Decision tree and Naïve Bayes for predictive analysis.

CODE & OUTPUT:

Data Import & Data Cleaning

Dataset Used:

https://www.kaggle.com/lepchenkov/usedcarscatalog

[1]:
```
import numpy as np
import pandas as pd
df = pd.read_csv("cars.csv")
df
```

[1]:

	manufacturer_name	model_name	transmission	color	odometer_value	year_produced	engine_fuel	engine_has_gas	engine_type	engine_capacity	...	feature_1
0	Subaru	Outback	automatic	silver	190000	2010	gasoline	False	gasoline	2.5	...	True
1	Subaru	Outback	automatic	blue	290000	2002	gasoline	False	gasoline	3.0	...	True
2	Subaru	Forester	automatic	red	402000	2001	gasoline	False	gasoline	2.5	...	True
3	Subaru	Impreza	mechanical	blue	10000	1999	gasoline	False	gasoline	3.0	...	False
4	Subaru	Legacy	automatic	black	280000	2001	gasoline	False	gasoline	2.5	...	True
...	...	...	...	...	...	...	...	...	...	...	...	...
38526	Chrysler	300	automatic	silver	290000	2000	gasoline	False	gasoline	3.5	...	True
38527	Chrysler	PT Cruiser	mechanical	blue	321000	2004	diesel	False	diesel	2.2	...	True
38528	Chrysler	300	automatic	blue	777957	2000	gasoline	False	gasoline	3.5	...	True
38529	Chrysler	PT Cruiser	mechanical	black	20000	2001	gasoline	False	gasoline	2.0	...	True
38530	Chrysler	Voyager	automatic	silver	297729	2000	gasoline	False	gasoline	2.4	...	False

38531 rows × 29 columns

Step 1:Basic Data Quality Checks

[2]:
```
df.describe()
```

[2]:

	odometer_value	year_produced	engine_capacity	price_usd	number_of_photos	up_counter	duration_listed
count	38531.000000	38531.000000	38521.000000	38531.000000	38531.000000	38531.000000	38531.000000
mean	248864.638447	2002.943734	2.055161	6639.971021	9.649062	16.306091	80.577249
std	136072.376530	8.065731	0.671178	6428.152018	6.093217	43.286933	112.826569
min	0.000000	1942.000000	0.200000	1.000000	1.000000	1.000000	0.000000
25%	158000.000000	1998.000000	1.600000	2100.000000	5.000000	2.000000	23.000000
50%	250000.000000	2003.000000	2.000000	4800.000000	8.000000	5.000000	59.000000
75%	325000.000000	2009.000000	2.300000	8990.000000	12.000000	16.000000	91.000000
max	1000000.000000	2019.000000	8.000000	50000.000000	86.000000	1861.000000	2232.000000

[3]:
```
df.dtypes
```

[3]:
```
manufacturer_name     object
model_name            object
transmission          object
color                 object
odometer_value         int64
year_produced          int64
engine_fuel           object
engine_has_gas          bool
engine_type           object
engine_capacity      float64
body_type             object
has_warranty            bool
state                 object
drivetrain            object
price_usd            float64
is_exchangeable         bool
number_of_photos       int64
up_counter             int64
feature_0               bool
feature_1               bool
feature_2               bool
feature_3               bool
feature_4               bool
feature_5               bool
feature_6               bool
feature_7               bool
feature_8               bool
feature_9               bool
duration_listed        int64
dtype: object
```

[4]:
```
print(df.shape)
```

```
(38531, 29)
```

We dont know what the features (0 to 9) are for. Hence dropping these features.

[5]:
```
df = df.drop(columns=['feature_0','feature_1','feature_2','feature_3','feature_4','feature_5','feature_6','feature_7','feature_8','feature_9'
df.head()
```

[5]:

	manufacturer_name	model_name	transmission	color	odometer_value	year_produced	engine_fuel	engine_has_gas	engine_type	engine_capacity	body_type	has_war
0	Subaru	Outback	automatic	silver	190000	2010	gasoline	False	gasoline	2.5	universal	
1	Subaru	Outback	automatic	blue	290000	2002	gasoline	False	gasoline	3.0	universal	
2	Subaru	Forester	automatic	red	402000	2001	gasoline	False	gasoline	2.5	suv	
3	Subaru	Impreza	mechanical	blue	10000	1999	gasoline	False	gasoline	3.0	sedan	
4	Subaru	Legacy	automatic	black	280000	2001	gasoline	False	gasoline	2.5	universal	

[6]:
```
df.isnull().sum()
```

[6]:
```
manufacturer_name     0
model_name            0
transmission          0
color                 0
odometer_value        0
year_produced         0
engine_fuel           0
engine_has_gas        0
engine_type           0
engine_capacity      10
body_type             0
has_warranty          0
state                 0
drivetrain            0
price_usd             0
is_exchangeable       0
number_of_photos      0
up_counter            0
duration_listed       0
dtype: int64
```

Also Dropping Rows having 1 or more null values

[7]:
```
df = df.dropna()
```

[8]:
```
print(df.shape)
```

```
(38521, 19)
```

Performing some Data Transformation

[9]:
```
df["engine_has_gas"]=df["engine_has_gas"].astype(int)
df["has_warranty"]=df["has_warranty"].astype(int)
df["is_exchangeable"]=df["is_exchangeable"].astype(int)
```

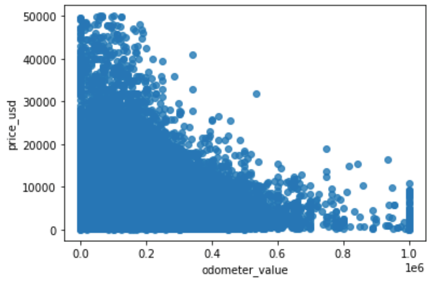
[10]:
```
# df.to_csv(r'C:\Users\abhis\Desktop\DS Mini-Project\cars_data.csv',index = False)
```

[11]:
```
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
```

Scatter plot

[12]:
```
sns.regplot(x="odometer_value",y="price_usd",data=df,fit_reg=False)
```

[12]: <matplotlib.axes._subplots.AxesSubplot at 0x7fd6b79f7ee0>

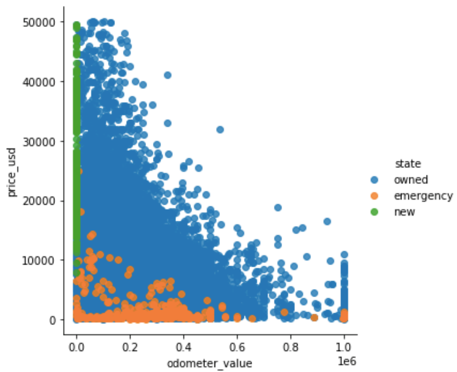


Scatter plot

Emergency means the car has been damaged, sometimes severely.

[13]:
```
sns.lmplot(x="odometer_value",y="price_usd",data=df,hue="state",fit_reg=False)
```

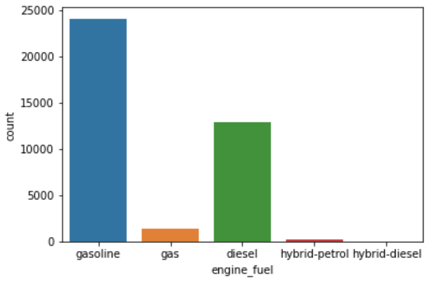
[13]: <seaborn.axisgrid.FacetGrid at 0x7fd6b4ddd640>



Barplot for Engine_fuel Types (Categorical)

[14]:
```
sns.countplot(data=df,x="engine_fuel")
```

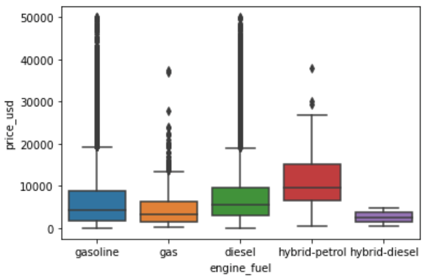
[14]: <matplotlib.axes._subplots.AxesSubplot at 0x7fd6b83b4790>



Boxplot to Check Which type of Engine falls in which Price range

[15]:
```
sns.boxplot(x="engine_fuel",y="price_usd",data=df)
```

[15]: <matplotlib.axes._subplots.AxesSubplot at 0x7fd6b55da130>



Calculate the age of the car

[16]:
```
df['age'] = 2020 - df['year_produced']
df.head()
```

[16]:

	manufacturer_name	model_name	transmission	color	odometer_value	year_produced	engine_fuel	engine_has_gas	engine_type	engine_capacity	body_type	has_war
0	Subaru	Outback	automatic	silver	190000	2010	gasoline	0	gasoline	2.5	universal	
1	Subaru	Outback	automatic	blue	290000	2002	gasoline	0	gasoline	3.0	universal	
2	Subaru	Forester	automatic	red	402000	2001	gasoline	0	gasoline	2.5	suv	
3	Subaru	Impreza	mechanical	blue	10000	1999	gasoline	0	gasoline	3.0	sedan	
4	Subaru	Legacy	automatic	black	280000	2001	gasoline	0	gasoline	2.5	universal	

All numeric (float and int) variables in the dataset

[17]:
```
cars_numeric = df.select_dtypes(include=['float64', 'int64'])
cars_numeric.head()
```

[17]:

	odometer_value	year_produced	engine_has_gas	engine_capacity	has_warranty	price_usd	is_exchangeable	number_of_photos	up_counter	duration_listed	age
0	190000	2010	0	2.5	0	10900.00	0	9	13	16	10
1	290000	2002	0	3.0	0	5000.00	1	12	54	83	18
2	402000	2001	0	2.5	0	2800.00	1	4	72	151	19
3	10000	1999	0	3.0	0	9999.00	1	9	42	86	21
4	280000	2001	0	2.5	0	2134.11	1	14	7	7	19

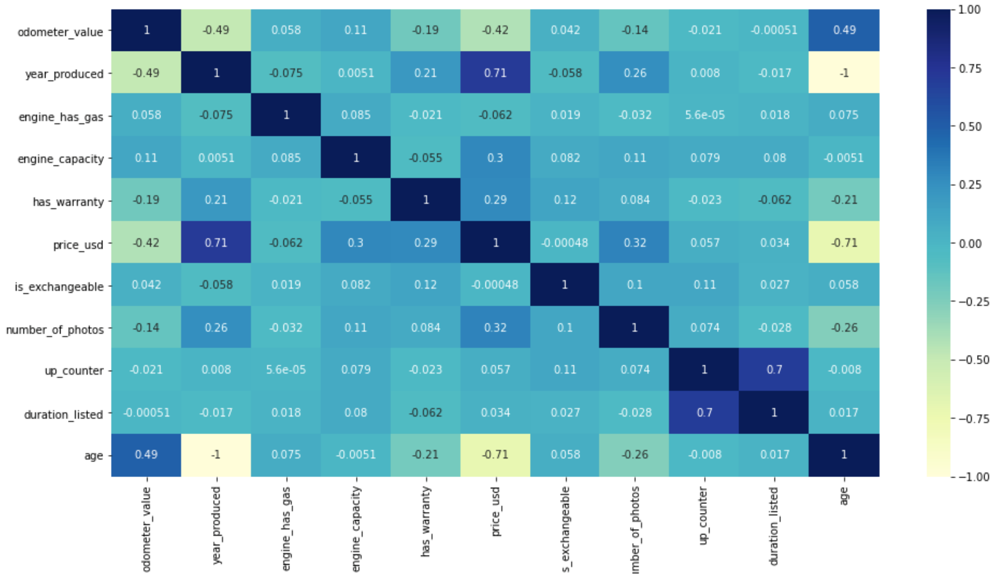
Correlation matrix

[18]:
```
mat = cars_numeric.corr()
mat
```

[18]:

	odometer_value	year_produced	engine_has_gas	engine_capacity	has_warranty	price_usd	is_exchangeable	number_of_photos	up_counter	duration_
odometer_value	1.000000	-0.488448	0.057736	0.105704	-0.189577	-0.420965	0.042370	-0.143564	-0.020976	-0.0(
year_produced	-0.488448	1.000000	-0.074637	0.005059	0.209322	0.705439	-0.057967	0.258064	0.007963	-0.0
engine_has_gas	0.057736	-0.074637	1.000000	0.084579	-0.020672	-0.062482	0.018654	-0.032076	0.000056	0.0
engine_capacity	0.105704	0.005059	0.084579	1.000000	-0.054583	0.296597	0.081636	0.106691	0.079152	0.0
has_warranty	-0.189577	0.209322	-0.020672	-0.054583	1.000000	0.285749	0.117795	0.084079	-0.023089	-0.0
price_usd	-0.420965	0.705439	-0.062482	0.296597	0.285749	1.000000	-0.000479	0.316879	0.057470	0.0:
is_exchangeable	0.042370	-0.057967	0.018654	0.081636	0.117795	-0.000479	1.000000	0.103671	0.106233	0.0:
number_of_photos	-0.143564	0.258064	-0.032076	0.106691	0.084079	0.316879	0.103671	1.000000	0.073880	-0.0
up_counter	-0.020976	0.007963	0.000056	0.079152	-0.023089	0.057470	0.106233	0.073880	1.000000	0.6
duration_listed	-0.000508	-0.016916	0.018252	0.080081	-0.061807	0.033662	0.026929	-0.028181	0.698128	1.0(
age	0.488448	-1.000000	0.074637	-0.005059	-0.209322	-0.705439	0.057967	-0.258064	-0.007963	0.0

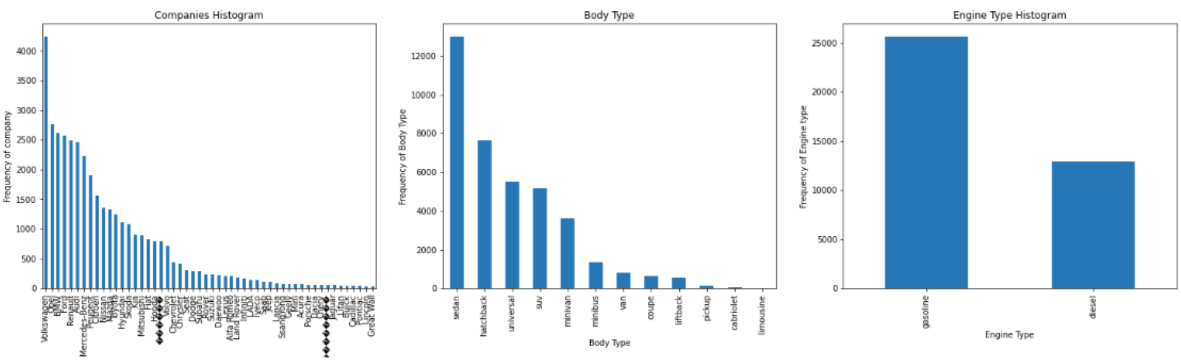
```
[19]: # Figure size
plt.figure(figsize=(16,8))

# Heatmap
sns.heatmap(mat, cmap="YlGnBu", annot=True)
plt.show()
```



Here we can see that "Year Produced" or "age" has highest correlation with the Price of the car.

Also it is correlated with Odometer Value, Number of photos uploaded by user, and engine capacity.

```
[20]: plt.figure(figsize=(25, 6))

plt.subplot(1,3,1)
plt1 = df.manufacturer_name.value_counts().plot(kind='bar')
plt.title('Companies Histogram')
plt1.set(xlabel = 'Car company', ylabel='Frequency of company')

plt.subplot(1,3,2)
plt1 = df.body_type.value_counts().plot(kind='bar')
plt.title('Body Type')
plt1.set(xlabel = 'Body Type', ylabel='Frequency of Body Type')

plt.subplot(1,3,3)
plt1 = df.engine_type.value_counts().plot(kind='bar')
plt.title('Engine Type Histogram')
plt1.set(xlabel = 'Engine Type', ylabel='Frequency of Engine type')

plt.show()
```



Inference:

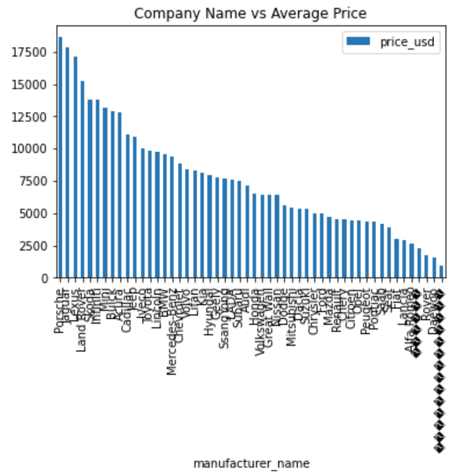
1. Volkswagen is preffered than other cars.

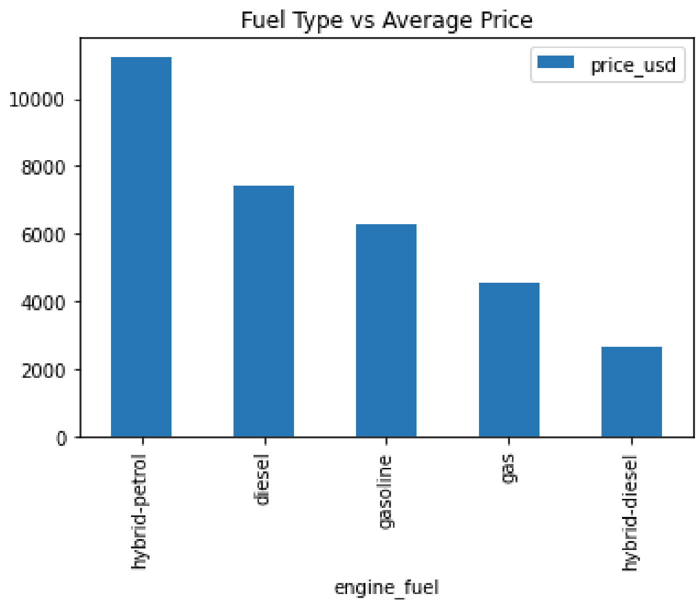
2. Sedan seems to be the popular type.

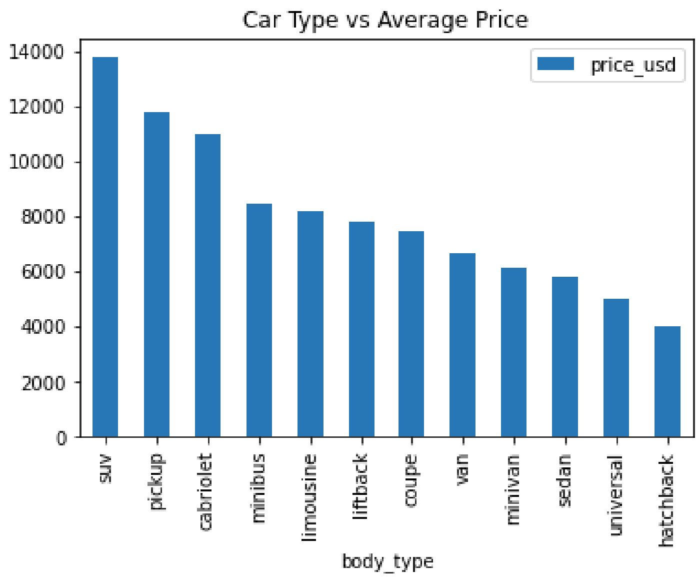
3. Vehicles with gasoline are preffered.

[21]:
```python
plt.figure(figsize=(30, 10))

df2 = pd.DataFrame(df.groupby(['manufacturer_name'])['price_usd'].mean().sort_values(ascending = False))
df2.plot.bar()
plt.title('Company Name vs Average Price')
plt.show()

df2 = pd.DataFrame(df.groupby(['engine_fuel'])['price_usd'].mean().sort_values(ascending = False))
df2.plot.bar()
plt.title('Fuel Type vs Average Price')
plt.show()

df2 = pd.DataFrame(df.groupby(['body_type'])['price_usd'].mean().sort_values(ascending = False))
df2.plot.bar()
plt.title('Car Type vs Average Price')
plt.show()
```

```
<Figure size 2160x720 with 0 Axes>
```







Inference:

- 1. Porsche and Jaguar seems to have highest average price.
- 2. Hybrid vehicles have high average price than both Diesel and Gasoline vehicles.
- 3. SUV has the highest average price.

Binning Companies based on Average Price

> We have around 45 different Car Manufacturing Companies with Different Model Names. If we create dummy variables for all these names, it will result in large number of coulmns which is not feasible for model building. Hence, we will try and create different groups based on Average Price of the cars.

[22]:
```
df['price_usd'] = df['price_usd'].astype('float64')
temp = df.copy()

table = temp.groupby(['manufacturer_name'])['price_usd'].mean()
temp = temp.merge(table.reset_index(), how='left', on='manufacturer_name')
bins = [0,10000,25000,50000]
cars_bins = ['Budget','Medium', 'Highend']
```

[23]:
```
temp.head()
```

[23]:

	manufacturer_name	model_name	transmission	color	odometer_value	year_produced	engine_fuel	engine_has_gas	engine_type	engine_capacity	...	has_warranty	
0	Subaru	Outback	automatic	silver	190000	2010	gasoline	0	gasoline	2.5	...	0	c
1	Subaru	Outback	automatic	blue	290000	2002	gasoline	0	gasoline	3.0	...	0	c
2	Subaru	Forester	automatic	red	402000	2001	gasoline	0	gasoline	2.5	...	0	c
3	Subaru	Impreza	mechanical	blue	10000	1999	gasoline	0	gasoline	3.0	...	0	c
4	Subaru	Legacy	automatic	black	280000	2001	gasoline	0	gasoline	2.5	...	0	c

5 rows × 21 columns

[24]:
```
df['CarRange'] = pd.cut(temp['price_usd_y'], bins,  right=False, labels=cars_bins)
df.head()
```

[24]:

	manufacturer_name	model_name	transmission	color	odometer_value	year_produced	engine_fuel	engine_has_gas	engine_type	engine_capacity	...	has_warranty	
0	Subaru	Outback	automatic	silver	190000	2010	gasoline	0	gasoline	2.5	...	0	c
1	Subaru	Outback	automatic	blue	290000	2002	gasoline	0	gasoline	3.0	...	0	c
2	Subaru	Forester	automatic	red	402000	2001	gasoline	0	gasoline	2.5	...	0	c
3	Subaru	Impreza	mechanical	blue	10000	1999	gasoline	0	gasoline	3.0	...	0	c
4	Subaru	Legacy	automatic	black	280000	2001	gasoline	0	gasoline	2.5	...	0	c

5 rows × 21 columns

We will leave out variables like "manufacturer_name","model_name".

We will be using CarsRange variable instead of these as discussed above.

[25]:
```
cars_new = df[['transmission','color','odometer_value','engine_fuel','engine_has_gas','engine_type','engine_capacity','body_type'
                  , 'has_warranty','state','drivetrain','is_exchangeable','number_of_photos', 'up_counter','duration_listed', 'age','CarRange',
cars_new.head()
```

[25]:

	transmission	color	odometer_value	engine_fuel	engine_has_gas	engine_type	engine_capacity	body_type	has_warranty	state	drivetrain	is_exchangeable	number_
0	automatic	silver	190000	gasoline	0	gasoline	2.5	universal	0	owned	all	0	
1	automatic	blue	290000	gasoline	0	gasoline	3.0	universal	0	owned	all	1	
2	automatic	red	402000	gasoline	0	gasoline	2.5	suv	0	owned	all	1	
3	mechanical	blue	10000	gasoline	0	gasoline	3.0	sedan	0	owned	all	1	
4	automatic	black	280000	gasoline	0	gasoline	2.5	universal	0	owned	all	1	

Define a function to generate dummy variables and merging it with data frame

[26]:
```python
def dummies(x,df1):
    temp = pd.get_dummies(df[[x]], drop_first=True)
    df1 = pd.concat([df1,temp], axis=1)
    df1.drop([x], axis=1, inplace=True)
    return df1

# Apply function to the cars_new df
cars_new = dummies('transmission', cars_new)
cars_new = dummies('color', cars_new)
cars_new = dummies('engine_fuel', cars_new)
cars_new = dummies('engine_has_gas', cars_new)
cars_new = dummies('engine_type', cars_new)
cars_new = dummies('body_type', cars_new)
cars_new = dummies('has_warranty', cars_new)
cars_new = dummies('state', cars_new)
cars_new = dummies('drivetrain', cars_new)
cars_new = dummies('is_exchangeable', cars_new)
cars_new = dummies('CarRange', cars_new)
```

[27]:
```python
cars_new.head()
```

[27]:

	odometer_value	engine_capacity	number_of_photos	up_counter	duration_listed	age	price_usd	transmission_mechanical	color_blue	color_brown	...	body_type_sed
0	190000	2.5	9	13	16	10	10900.00	0	0	0	...	
1	290000	3.0	12	54	83	18	5000.00	0	1	0	...	
2	402000	2.5	4	72	151	19	2800.00	0	0	0	...	
3	10000	3.0	9	42	86	21	9999.00	1	1	0	...	
4	280000	2.5	14	7	7	19	2134.11	0	0	0	...	

5 rows × 41 columns

[28]:
```python
cars_new.shape
```

[28]: (38521, 41)

Step 3 : Train Test Split

[29]:
```python
from sklearn.model_selection import train_test_split

df_train, df_test = train_test_split(cars_new, train_size=0.7, random_state=42)
```

[30]:
```python
df_train.tail()
```

[30]:

	odometer_value	engine_capacity	number_of_photos	up_counter	duration_listed	age	price_usd	transmission_mechanical	color_blue	color_brown	...	body_type
6265	300000	1.8	6	4	4	22	820.0	1	0	0	...	
11286	380000	1.9	5	1	34	22	3200.0	1	0	0	...	
38168	459186	3.3	7	9	14	15	4900.0	0	0	0	...	
860	77921	2.0	15	24	41	6	14900.0	0	0	0	...	
15797	296550	1.6	16	1	1	5	6700.0	1	0	0	...	

5 rows × 41 columns

[31]:
```python
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
num_vars= ['odometer_value', 'engine_capacity', 'number_of_photos','up_counter','duration_listed', 'age','price_usd']
df_train[num_vars] = scaler.fit_transform(df_train[num_vars])
```

```
<ipython-input-31-dea064584dfa>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_train[num_vars] = scaler.fit_transform(df_train[num_vars])
/Users/kalichai/opt/anaconda3/lib/python3.8/site-packages/pandas/core/indexing.py:966: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self.obj[item] = s
```

[32]:
```python
from sklearn.linear_model import LinearRegression
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

[33]:
```python
df_test.tail()
```

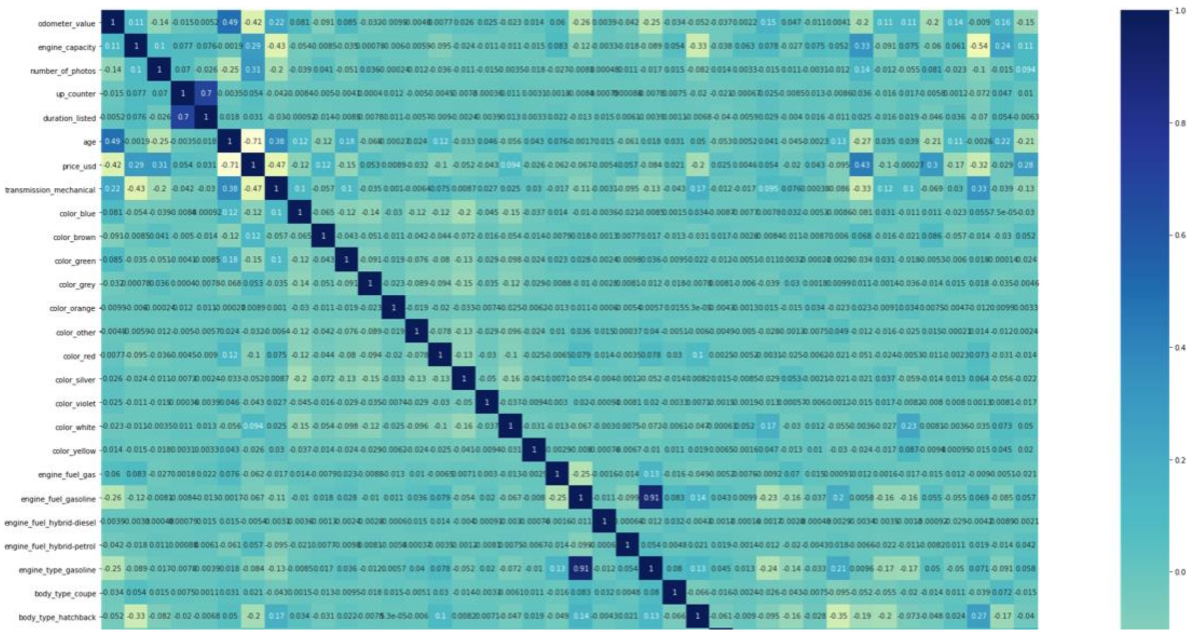
[33]:

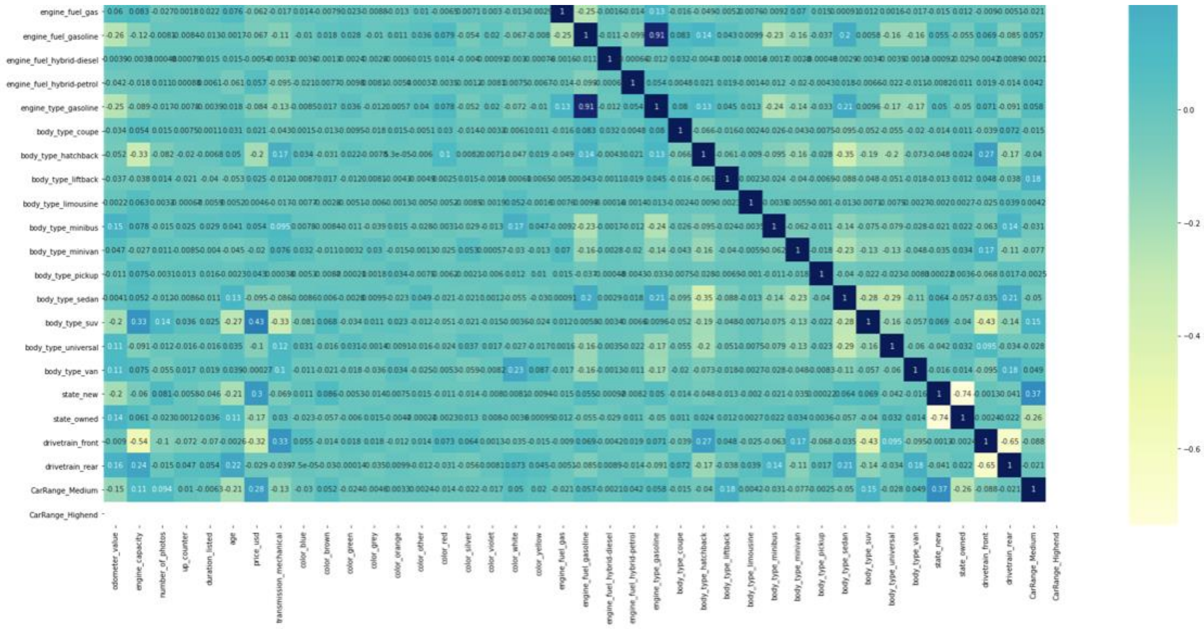
	odometer_value	engine_capacity	number_of_photos	up_counter	duration_listed	age	price_usd	transmission_mechanical	color_blue	color_brown	...	body_type
28622	333333	2.8	7	16	84	34	850.0	1	0	0	...	
22181	321869	4.7	10	44	220	16	14000.0	0	0	0	...	
7603	120000	1.6	1	128	516	6	8000.0	0	0	0	...	
22551	75000	2.0	9	6	17	12	15000.0	0	0	0	...	
6075	320000	2.2	2	5	14	17	6200.0	1	0	0	...	

5 rows × 41 columns

[34]:
```python
plt.figure(figsize = (30, 25))
sns.heatmap(df_train.corr(), annot = True, cmap="YlGnBu")
plt.show()
```


[34]:
```python
plt.figure(figsize = (30, 25))
sns.heatmap(df_train.corr(), annot = True, cmap="YlGnBu")
plt.show()
```



> **Age, Odometer Value, Engine Capacity are some of the high correaltion variables.**

Step 4 : Model Building

Model : Linear Regression

[35]:
```
from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

[36]:
```
# dividing variables in to X and y
y_train = df_train.pop('price_usd')
X_train = df_train
```

[37]:
```
lm = LinearRegression()
lm.fit(X_train, y_train)
rfe = RFE(lm, 10)
rfe = rfe.fit(X_train, y_train)
```

```
/Users/kalichai/opt/anaconda3/lib/python3.8/site-packages/sklearn/utils/validation.py:68: FutureWarning: Pass n_features_to_select=10 as keyw
ord args. From version 0.25 passing these as positional arguments will result in an error
  warnings.warn("Pass {} as keyword args. From version 0.25 "
```

[38]:
```
list(zip(X_train.columns,rfe.support_,rfe.ranking_))
```

[38]:
```
[('odometer_value', True, 1),
 ('engine_capacity', True, 1),
 ('number_of_photos', True, 1),
 ('up_counter', False, 30),
 ('duration_listed', True, 1),
 ('age', True, 1),
 ('transmission_mechanical', False, 10),
 ('color_blue', False, 23),
 ('color_brown', False, 24),
 ('color_green', False, 20),
 ('color_grey', False, 29),
 ('color_orange', False, 27),
 ('color_other', False, 21),
 ('color_red', False, 28),
 ('color_silver', False, 19),
 ('color_violet', False, 22),
 ('color_white', False, 25),
 ('color_yellow', False, 26),
 ('engine_fuel_gas', False, 11),
 ('engine_fuel_gasoline', False, 14),
 ('engine_fuel_hybrid-diesel', False, 15),
 ('engine_fuel_hybrid-petrol', False, 17),
 ('engine_type_gasoline', False, 18),
 ('body_type_coupe', False, 7),
 ('body_type_hatchback', True, 1),
 ('body_type_liftback', False, 4),
 ('body_type_limousine', True, 1),
 ('body_type_minibus', False, 13),
 ('body_type_minivan', False, 3),
 ('body_type_pickup', False, 12),
 ('body_type_sedan', False, 2),
 ('body_type_suv', False, 8),
 ('body_type_universal', True, 1),
 ('body_type_van', False, 6),
 ('state_new', True, 1),
 ('state_owned', False, 9),
 ('drivetrain_front', True, 1),
 ('drivetrain_rear', False, 5),
 ('CarRange_Medium', False, 16),
 ('CarRange_Highend', False, 31)]
```

[39]:
```
X_train.columns[rfe.support_]
```

[39]:
```
Index(['odometer_value', 'engine_capacity', 'number_of_photos',
       'duration_listed', 'age', 'body_type_hatchback', 'body_type_limousine',
       'body_type_universal', 'state_new', 'drivetrain_front'],
      dtype='object')
```

Building model using statsmodel, for the detailed statistics

[40]:
```
X_train_rfe = X_train[X_train.columns[rfe.support_]]
X_train_rfe.head()
```

[40]:

	odometer_value	engine_capacity	number_of_photos	duration_listed	age	body_type_hatchback	body_type_limousine	body_type_universal	state_new	drivet
21050	0.237000	0.410959	0.152941	0.009857	0.216667	0	0	1	0	
29578	0.440000	0.246575	0.058824	0.024194	0.366667	0	0	0	0	
22650	0.252000	0.164384	0.141176	0.019265	0.183333	1	0	0	0	
1035	0.196096	0.246575	0.035294	0.036290	0.233333	0	0	0	0	
9746	0.057300	0.109589	0.023529	0.072581	0.050000	0	0	0	0	

[41]:
```
# Building a model

def build_Lr_model(X,y):
    X = sm.add_constant(X) #add constant
    lm = sm.OLS(y,X).fit() #fit the model
    print(lm.summary())
    return X

def checkingVIF(X):
    vif = pd.DataFrame()
    vif['features'] = X.columns
    vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
    vif['VIF'] = round(vif['VIF'], 2)
    vif = vif.sort_values(by = "VIF", ascending = False)
    return(vif)
```

```
[42]: X_train_1 = build_Lr_model(X_train_rfe, y_train)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              price_usd   R-squared:                       0.671
Model:                            OLS   Adj. R-squared:                  0.671
Method:                 Least Squares   F-statistic:                     5494.
Date:                Sat, 05 Dec 2020   Prob (F-statistic):               0.00
Time:                        14:33:44   Log-Likelihood:                 32238.
No. Observations:               26964   AIC:                        -6.445e+04
Df Residuals:                   26953   BIC:                        -6.436e+04
Df Model:                          10
Covariance Type:            nonrobust
=========================================================================================
                          coef    std err          t      P>|t|      [0.025      0.975]
-----------------------------------------------------------------------------------------
const                   0.2718      0.003    105.881      0.000       0.267       0.277
odometer_value         -0.0989      0.004    -25.763      0.000      -0.106      -0.091
engine_capacity         0.2412      0.006     39.611      0.000       0.229       0.253
number_of_photos        0.1646      0.006     25.495      0.000       0.152       0.177
duration_listed         0.0617      0.009      6.848      0.000       0.044       0.079
age                    -0.5677      0.004   -143.935      0.000      -0.575      -0.560
body_type_hatchback    -0.0190      0.001    -15.259      0.000      -0.021      -0.017
body_type_limousine    -0.0592      0.024     -2.419      0.016      -0.107      -0.011
body_type_universal    -0.0141      0.001    -10.575      0.000      -0.017      -0.011
state_new               0.1876      0.004     43.032      0.000       0.179       0.196
drivetrain_front       -0.0551      0.001    -45.908      0.000      -0.057      -0.053
==============================================================================
Omnibus:                    15280.661   Durbin-Watson:                   2.003
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           227684.216
Skew:                           2.424   Prob(JB):                         0.00
Kurtosis:                      16.384   Cond. No.                         74.9
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Building model using statsmodel, for the detailed statistics

```
[43]: checkingVIF(X_train_1)
```

[43]:

	features	VIF
0	const	33.16
2	engine_capacity	1.56
10	drivetrain_front	1.46
5	age	1.41
1	odometer_value	1.37
6	body_type_hatchback	1.23
8	body_type_universal	1.10
3	number_of_photos	1.09
9	state_new	1.07
4	duration_listed	1.01
7	body_type_limousine	1.00

Decision Tree

```
[44]: from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn import metrics
import pandas as pd
import numpy as np
```

```
[45]: df.head()
```

[45]:

	manufacturer_name	model_name	transmission	color	odometer_value	year_produced	engine_fuel	engine_has_gas	engine_type	engine_capacity	...	has_warranty	
0	Subaru	Outback	automatic	silver	190000	2010	gasoline	0	gasoline	2.5	...	0	c
1	Subaru	Outback	automatic	blue	290000	2002	gasoline	0	gasoline	3.0	...	0	c
2	Subaru	Forester	automatic	red	402000	2001	gasoline	0	gasoline	2.5	...	0	c
3	Subaru	Impreza	mechanical	blue	10000	1999	gasoline	0	gasoline	3.0	...	0	c
4	Subaru	Legacy	automatic	black	280000	2001	gasoline	0	gasoline	2.5	...	0	c

5 rows × 21 columns

```
[46]: feature_cols=['odometer_value','engine_has_gas','engine_capacity', 'has_warranty','number_of_photos', 'up_counter','duration_listed', 'age','
X=df[feature_cols]
y=df.is_exchangeable
```

In [112]:
```python
# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1) # 70% training and 30% test
```


In [113]:
```python
X_train.dtypes
```

Out[113]:
```
odometer_value        int64
engine_has_gas        int32
engine_capacity     float64
has_warranty          int32
number_of_photos      int64
up_counter            int64
duration_listed       int64
age                   int64
price_usd           float64
dtype: object
```

In [114]:
```python
clf = DecisionTreeClassifier()

# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)
```

In [115]:
```python
# Model Accuracy, how often is the classifier correct?
print("Accuracy:", metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.5985117244959764
```

Decision Tree has achieved accuracy of 59.85%

Naive Bayes

In [118]:
```python
from sklearn.naive_bayes import GaussianNB
```

In [119]:
```python
feature_cols=['odometer_value','engine_has_gas','engine_capacity', 'has_warranty','number_of_photos', 'up_counter','duration
X1=df[feature_cols]
y1=df.is_exchangeable
```

In [120]:
```python
# Split dataset into training set and test set
X1_train, X1_test, y1_train, y1_test = train_test_split(X1, y1, test_size=0.3, random_state=1) # 70% training and 30% test
```

In [122]:
```python
nb = GaussianNB()
nb.fit(X1_train, y1_train)
accuracy_score(y1_test, nb.predict(X1_test))
```

Out[122]:
```
0.6394393008566237
```

Naive Bayes has achieved accuracy of 63.94%

REFERENCE

1. Dataset is taken from Kaggle
2. Link: click here.