2023aiml554-miniproject

July 21, 2024

FEATURE ENGINEERING End-to End PROJECT (30M)

AIML Certification Programme

0.1 Student Name and ID: Shreenath Omar and 2023AIML554

Mention your name and ID if done individually If done as a group,clearly mention the contribution from each group member qualitatively and as a precentage. 1.

2.

0.2 Business Understanding (1M)

Students are expected to identify a regression problem of your choice. You have to detail the Business Understanding part of your problem under this heading which basically addresses the following questions.

1. What is the business problem that you are trying to solve

   Answer:

   Business Problem: The business problem at hand is predicting car prices based on various attributes. Accurate car price predictions can help
   manufacturers, dealers, and customers make informed decisions. For instance, manufacturers can set competitive prices, dealers can provide better recommendations to customers, and customers can understand the value of a car based on its features.

2. What data do you need to answer the above problem?What are the different sources of data?

   Answer:

   Data Needed: To predict car prices, we need data on various car attributes such as make_model, model,body type, body color, km , Gearing Type, and extras. This data will help in understanding the factors that influence car prices and build a predictive model.

   Sources of Data: The dataset used for this analysis is sourced from Kaggle: https://www.kaggle.com/datasets/ersany/car-price-prediction

0.3 Data Requirements and Data Collection (3+1M)

In the initial data collection stage, data scientists identify and gather the available data resources. These can be in the form of structured, unstructured, and even semi-structured data relevant to the problem domain.

Identify the required data that fulfills the data requirements stage of the data science methodology Mention the source of the data.(Give the link if you have sourced it from any public data set) Briefly explain the data set identified .

Answer: Data Requirements and Data Collection

Required Data:

The required data includes structured data on car attributes. This data should have features that are likely to impact the car price such as make_model, model,body type, body color, km , Gearing Type, and extras.

Source:

The dataset is publicly available on Kaggle: Car Price Prediction Dataset (link:- https://www.kaggle.com/datasets/ersany/car-price-prediction)

dataset is uploaded in Github repository :- https://github.com/shreenath197/car_pricing_dataSet.git

raw content url of car pricing dataset csv file:- https://raw.githubusercontent.com/shreenath197/car_pricing_data

Dataset Overview:

The dataset contains various attributes of cars which will be used to predict their prices. We will import this dataset and perform an initial analysis.

The dataset includes the following features:- 1- make_model—> (Categorical Attributes)

2- body type—-> (Categorical Attributes)

3- body color—> (Categorical Attributes)

4- km———> (Numerical Attributes)

5- hp———> (Numerical Attributes)

6- Gearing Type-> (Categorical Attributes)

7- extras——-> (Categorical Attributes)

8- Price———> (Numerical Attributes) target variable

0.3.1 Import the above data and read it into a data frame

[157]:
```
# importing required package
import pandas as pd
import warnings as war
war.filterwarnings('ignore')
# defining dataSet github raw url for csv dataSet file
github_filePath="https://raw.githubusercontent.com/shreenath197/
↪car_pricing_dataSet/main/car_price.csv"
# loading dataSet
dataSet=pd.read_csv(github_filePath)
```

0.3.2 Confirm the data has been correctly by displaying the first 5 and last 5 records.

[158]:
```
# displaying first 5 records to confirming data loading
print("************************************Displaying below first 5␣
 ↪records*************************************")
dataSet.head()
```

```
************************************Displaying below first 5
records*************************************
```

[158]:

	make_model	body_type	Body Color	km	hp	Gearing Type	Extras	price
0	Audi A1	Sedans	Black	56013.0	85	Automatic	Converter	15770
1	Audi A1	Sedans	Red	80000.0	85	Automatic	Sport	14500
2	Audi A1	Sedans	Black	83450.0	85	Automatic	Others	14640
3	Audi A1	Sedans	Brown	73000.0	85	Automatic	Sport	14500
4	Audi A1	Sedans	Black	16200.0	85	Automatic	Sport	16790

[159]:
```
# displaying last 5 records to confirming data loading
print("**************************************Displaying below last 5␣
 ↪records***************************************")
dataSet.tail()
```

```
**************************************Displaying below last 5
records***************************************
```

[159]:

	make_model	body_type	Body Color	km	hp	Gearing Type	Extras	\
4795	Audi A3	Sedans	White	54.0	85	Manual	Sport	
4796	Audi A3	Sedans	White	50.0	85	Manual	Others	
4797	Audi A3	Station wagon	Silver	6666.0	85	Manual	Others	
4798	Audi A3	Sedans	Silver	10.0	85	Manual	Others	
4799	Audi A3	Sedans	Silver	10.0	85	Manual	Others	

	price
4795	25000
4796	24980
4797	24980
4798	24980
4799	24980

0.3.3 Get the dimensions of the dataframe.

[160]:
```
# displaying the dimensions of the DataFrame
print("Dimention of dataframe:- {}".format(dataSet.shape[0:2]))
print("Total no. of rows:-      {} ".format(dataSet.shape[0:2][0]))
print("Total no. of columns:-   {} ".format(dataSet.shape[0:2][1]))
```

```
Dimention of dataframe:- (4800, 8)
Total no. of rows:-      4800
```

```
Total no. of columns:-   8
```

0.3.4 Display the description and statistical summary of the data.

[161]:
```
# displaying the description and statistical summary of the data
dataSet.describe()
```

[161]:

	km	hp	price
count	4797.000000	4800.0	4800.000000
mean	31931.949135	85.0	19722.871875
std	35902.589244	0.0	4337.519969
min	0.000000	85.0	5555.000000
25%	4800.000000	85.0	15990.000000
50%	20049.000000	85.0	19588.000000
75%	47800.000000	85.0	22692.500000
max	291800.000000	85.0	56100.000000

0.3.5 Display the columns and their respective data types.

[162]:
```
# displaying the columns and their respective data types
dataSet.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4800 entries, 0 to 4799
Data columns (total 8 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   make_model    4800 non-null   object
 1   body_type     4800 non-null   object
 2   Body Color    4796 non-null   object
 3   km            4797 non-null   float64
 4   hp            4800 non-null   int64
 5   Gearing Type  4800 non-null   object
 6   Extras        4800 non-null   object
 7   price         4800 non-null   int64
dtypes: float64(1), int64(2), object(5)
memory usage: 300.1+ KB
```

[163]:
```
# removing 'hp' attribute as 'hp' attribute has no variation
dataSet=dataSet.drop(columns=['hp'])
```

0.3.6 Convert the columns to appropriate data types

[164]:
```
#Convert categorical in values
dataSet.make_model = pd.Categorical(dataSet.make_model)
dataSet.body_type = pd.Categorical(dataSet.body_type)
dataSet["Body Color"]=pd.Categorical(dataSet['Body Color'])
```

```python
dataSet["Gearing Type"]=pd.Categorical(dataSet['Gearing Type'])
dataSet.Extras = pd.Categorical(dataSet.Extras)
dataSet.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4800 entries, 0 to 4799
Data columns (total 7 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   make_model    4800 non-null   category
 1   body_type     4800 non-null   category
 2   Body Color    4796 non-null   category
 3   km            4797 non-null   float64
 4   Gearing Type  4800 non-null   category
 5   Extras        4800 non-null   category
 6   price         4800 non-null   int64
dtypes: category(5), float64(1), int64(1)
memory usage: 99.9 KB
```

Write your observations from the above.

1- Sample Size: There are 4,797 entries for kilometers and 4,800 entries for price, indicating a very slight difference in the number of entries between the two
variables.

2- Statistical summary of km and price: (A)- Kilometers (km)

```
--> The mean (average) number of kilometers is 31,931.95.
--> The median (50th percentile) is 20,049 kilometers, indicating that half of the cars have be
    less than 20,049 kilometers.
--> The standard deviation is 35,902.59, which is relatively large compared to the mean,
    suggesting a wide range of values.
--> The minimum value is 0 kilometers, indicating some cars might be brand new or with very mi
--> The maximum value is 291,800 kilometers, indicating some cars have been driven extensively
```

(B)- Price

```
--> Price is target attribute for this data set.
--> The mean (average) price is 19,722.87.
--> The median (50th percentile) price is 19,588, suggesting that half of the cars are priced 
--> The standard deviation is 4,337.52, which shows moderate variability in prices.
--> The minimum price is 5,555, showing that some cars are quite inexpensive.
--> The maximum price is 56,100, indicating some cars are relatively high-priced.
```

(c)- hp

```
--> hp attribute does not have a significant impact on the target variable 'price ' ad 'hp' at
--> hp attribute can be removed to to simplify the model.
```

3- Data Consistency:

```
--> The slight difference in the count of entries for kilometers and price (4,797 vs. 4,800) mi
    or mismatched data points.
```

0.3.7 Check for Data Quality Issues (1.5M)

- duplicate data
- missing data
- data inconsistencies

[165]:
```
# Checking for duplicate records
duplicateValue_Count=dataSet.duplicated().sum()
print("Total no of duplicate records count:- {}".format(duplicateValue_Count))
```

```
Total no of duplicate records count:- 491
```

[166]:
```
#Finding total no. of missing values for attributes specific
missingValue_Count=dataSet.isnull().sum()
print(missingValue_Count)
```

```
make_model      0
body_type       0
Body Color      4
km              3
Gearing Type    0
Extras          0
price           0
dtype: int64
```

[167]:
```
#finding missing values attribues with counts
missingValue_attributes=missingValue_Count[missingValue_Count.
  ↪where(missingValue_Count.values>0).notnull()]
print(missingValue_attributes)
#Finding the attributes's key which have missing values
print("Below is the list of  missing values attributes:- ")
print(missingValue_attributes.keys())
```

```
Body Color    4
km            3
dtype: int64
Below is the list of  missing values attributes:-
Index(['Body Color', 'km'], dtype='object')
```

[168]:
```
#finding index for NULL valuees in 'Body Color' attribure
BodyColor_nullValues_index=dataSet[dataSet['Body Color'].isnull()].index
print("Body Color attribute null value index:- {}".
  ↪format(BodyColor_nullValues_index))
km_nullValues_index=dataSet[dataSet['km'].isnull()].index
```

```
print("km attribute null value index:- {}".format(km_nullValues_index))
```

```
Body Color attribute null value index:- Index([4606, 4607, 4608, 4609],
dtype='int64')
km attribute null value index:- Index([4636, 4637, 4638], dtype='int64')
```

[169]:
```
# checking for inconsistencies for categorical features
print('***** Data Inconsistency in Categorical features *****')
print('Column:make_model',dataSet['make_model'].unique())
print('Column:body_type',dataSet['body_type'].unique())
print('Column:Body Color',dataSet['Body Color'].unique())
print('Column:Gearing Type',dataSet['Gearing Type'].unique())
print('Column:Extras',dataSet['Extras'].unique())
```

```
***** Data Inconsistency in Categorical features *****
Column:make_model ['Audi A1', 'Audi A2', 'Audi A3']
Categories (3, object): ['Audi A1', 'Audi A2', 'Audi A3']
Column:body_type ['Sedans', 'Station wagon', 'Compact', 'Coupe', 'Other', 'Off-
Road', 'Convertible']
Categories (7, object): ['Compact', 'Convertible', 'Coupe', 'Off-Road', 'Other',
'Sedans', 'Station wagon']
Column:Body Color ['Black', 'Red', 'Brown', 'White', 'Grey', …, 'Yellow',
'Green', 'Bronze', 'Orange', NaN]
Length: 14
Categories (13, object): ['Beige', 'Black', 'Blue', 'Bronze', …, 'Silver',
'Violet', 'White', 'Yellow']
Column:Gearing Type ['Automatic', 'Manual', 'Semi-automatic']
Categories (3, object): ['Automatic', 'Manual', 'Semi-automatic']
Column:Extras ['Converter', 'Sport', 'Others']
Categories (3, object): ['Converter', 'Others', 'Sport']
```

[170]:
```
# checking data inconsistency for numerical features
# numerical list
num_cols = ['km']
num_cols
# outlier List
outliers_km=[]
dataSet_tmp = dataSet[num_cols]
dataSet_tmp

for item in num_cols:
    #print('item/col :' , item)
    mean_value= dataSet_tmp[item].mean()
    print('Column:' ,item,', Mean value :',mean_value)

# q1 = np.percentile(df_tmp[item],25)
    q1 = dataSet_tmp[item].quantile(0.25)
```

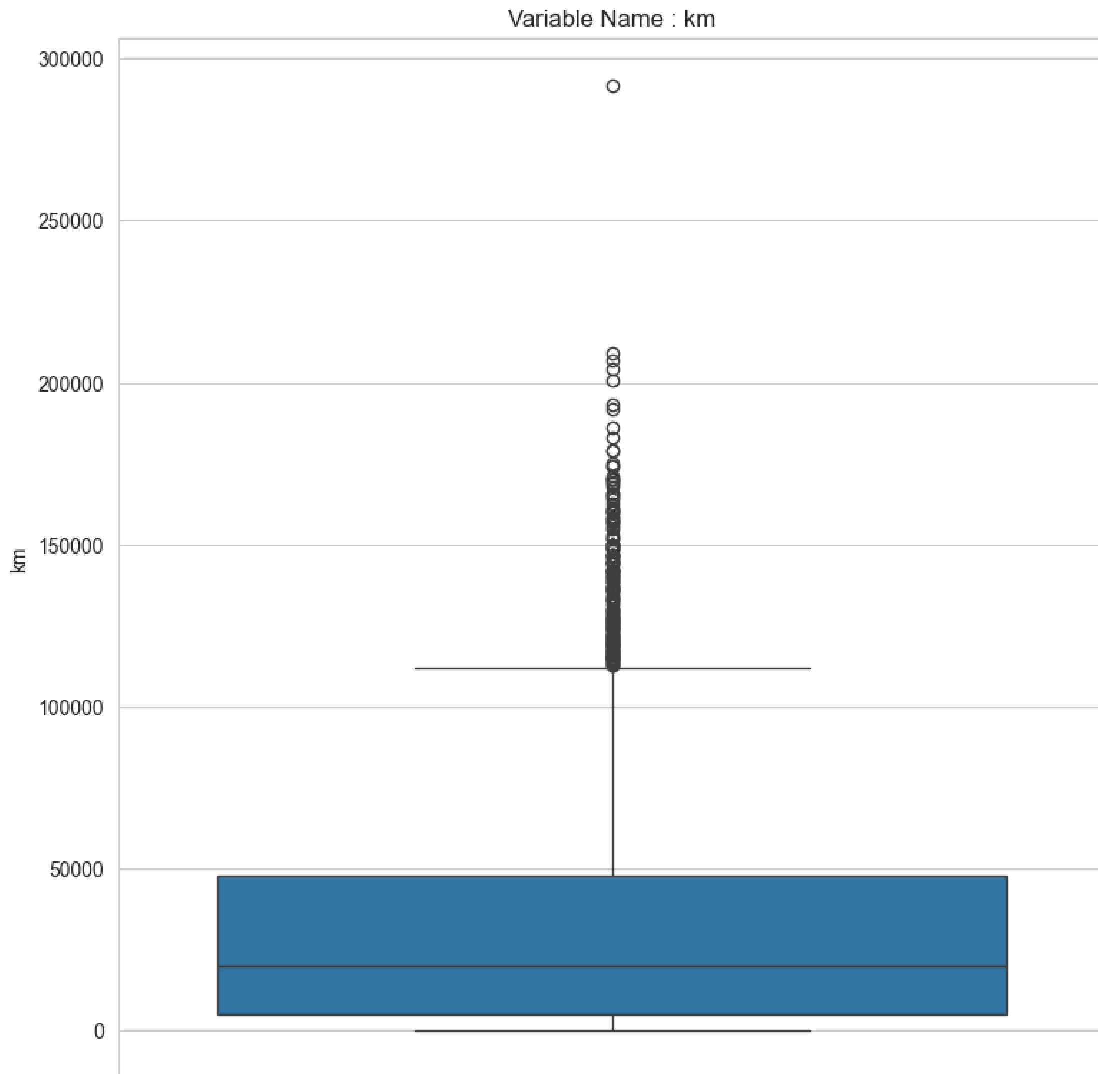
```python
    print('q1:',q1)

    #q3 = np.percentile(df_tmp[item],75)
    q3 = dataSet_tmp[item].quantile(0.75)
    print('q3:',q3)

    iqr = q3 - q1
    print('iqr:',iqr)

    upper_value = q3 + (1.5 * iqr)
    print('upper_value:',upper_value)
    lower_value = q1 - (1.5 * iqr)
    print('lower_value:',lower_value)

    for i in dataSet_tmp[item]:
        if i > upper_value or i < lower_value:
            if item == 'km':
                outliers_km.append(i)
                upper_value_km = upper_value
                lower_value_km = lower_value
print('------------------------------------------------------------')
print('upper_value_km:',upper_value_km,' lower_value_km:',lower_value_km)
print('------------------------------------------------------------')
print('outliers_km_values:',outliers_km)
```

```
Column: km , Mean value : 31931.949134875966
q1: 4800.0
q3: 47800.0
iqr: 43000.0
upper_value: 112300.0
lower_value: -59700.0
------------------------------------------------------------
upper_value_km: 112300.0  lower_value_km: -59700.0
------------------------------------------------------------
outliers_km_values: [115000.0, 126000.0, 192000.0, 130000.0, 137066.0, 115900.0,
146140.0, 129550.0, 125000.0, 115000.0, 118200.0, 168000.0, 137145.0, 146710.0,
124521.0, 148257.0, 136000.0, 141147.0, 158300.0, 120000.0, 121650.0, 144000.0,
122381.0, 120522.0, 120121.0, 161205.0, 115137.0, 122000.0, 123750.0, 133000.0,
171000.0, 136100.0, 115000.0, 115000.0, 144000.0, 146503.0, 124136.0, 142000.0,
170231.0, 129000.0, 120454.0, 123748.0, 135600.0, 138700.0, 126192.0, 120497.0,
123230.0, 134950.0, 124950.0, 144572.0, 119100.0, 158500.0, 114573.0, 169501.0,
117000.0, 133900.0, 126100.0, 150000.0, 148818.0, 115802.0, 121200.0, 127324.0,
119500.0, 175000.0, 126000.0, 120000.0, 118366.0, 170000.0, 160385.0, 162553.0,
144995.0, 174100.0, 113490.0, 118032.0, 140000.0, 140000.0, 121510.0, 127335.0,
145000.0, 155000.0, 141000.0, 163871.0, 186000.0, 126200.0, 207000.0, 132414.0,
119000.0, 126416.0, 119529.0, 179000.0, 142011.0, 119427.0, 170000.0, 291800.0,
160000.0, 140229.0, 113000.0, 138000.0, 120000.0, 144450.0, 116460.0, 200814.0,
```

```
150000.0, 124197.0, 160000.0, 174000.0, 119000.0, 149000.0, 115694.0, 138717.0,
142000.0, 158000.0, 127435.0, 149170.0, 141939.0, 165650.0, 165650.0, 117000.0,
179150.0, 135887.0, 114000.0, 113200.0, 151800.0, 130000.0, 127896.0, 152000.0,
131290.0, 112750.0, 128429.0, 128429.0, 154910.0, 160000.0, 138653.0, 124800.0,
146900.0, 123561.0, 165081.0, 183000.0, 209000.0, 204000.0, 193000.0, 156000.0,
122189.0, 135936.0, 152818.0, 136819.0, 116000.0, 136552.0, 149999.0, 144000.0,
157000.0, 133990.0, 133000.0, 127000.0, 119900.0, 149000.0, 114230.0, 121000.0,
122000.0, 125000.0, 165000.0, 169000.0, 161000.0, 150000.0, 157000.0, 150000.0,
152000.0, 142036.0, 116593.0, 142800.0, 129800.0, 118982.0, 146839.0, 122300.0,
124511.0, 127892.0, 116000.0, 117000.0, 116557.0, 146357.0, 157544.0, 133000.0,
136000.0, 117647.0, 117000.0, 112700.0, 139879.0, 131900.0, 115000.0, 127000.0,
114550.0]
```

[171]:

```
# importing required package
import seaborn as sbn
import matplotlib.pyplot as plt
# All columns are not numeric, need to select the columns explicitly
list = ['km']
dataSet_new = dataSet[list]
# Boxplot
sbn.set_style("whitegrid")
columns = dataSet_new.columns
# Plot build for each variable
plt.figure(figsize=(15, 30) )
for count, item in enumerate(columns, 1):
    plt.subplot(4, 2, count)
    sbn.boxplot(dataSet_new[item])
    plt.title(f"Variable Name : {item}")

plt.tight_layout()
plt.show()
```



0.3.8 Handling the data quality issues(1.5M)

Apply techniques * to remove duplicate data * to impute or remove missing data * to remove data inconsistencies Give detailed explanation for each column how you handle the data quality issues.

0.3.9 Task 1:- to remove duplicate data

[172]:
```
# removing duplicate records
print("Total no. of rows:-                        {} ".format(dataSet.shape[0:
↪2][0]))
print("Total no of duplicate records count:-      {}".
↪format(duplicateValue_Count))
dataSet = dataSet.drop_duplicates(subset=None,keep='first')
print("Total No. of Unique or unduplicated rows:- {}".format(dataSet.shape[0]))
```

```
Total no. of rows:-                        4800
Total no of duplicate records count:-      491
Total No. of Unique or unduplicated rows:- 4309
```

0.3.10 Task 2 :- to impute or remove missing data

[173]:
```
# Handling & imputing missing values for "Body Color" attribute
# calculating mode of "Body Colorl" attribute as this is categorical values
print("Body Color attribute mode:- {}".format(dataSet['Body Color'].mode()[0]))
```

```
Body Color attribute mode:- Black
```

[174]:
```
# Imputing missing values by mode for "Body Color" attribute
BodyColor_mode=dataSet['Body Color'].mode()[0]
dataSet['Body Color'].fillna(BodyColor_mode,inplace=True)
```

[175]:
```
# Verifing that missing values of 'Body Color' attribute filled with mode
dataSet['Body Color'].loc[BodyColor_nullValues_index]
```

[175]:
```
4606    Black
4607    Black
4608    Black
4609    Black
Name: Body Color, dtype: category
Categories (13, object): ['Beige', 'Black', 'Blue', 'Bronze', …, 'Silver',
'Violet', 'White', 'Yellow']
```

[176]:
```
# Handling & imputing missing values for "km" attribute
# calculating mean of "km" attribute as this is numerical values
print("km attribute mean:- {}".format(round(dataSet['km'].mean(),0)))
```

```
km attribute mean:- 33010.0
```

[177]:
```
# Imputing missing values by mean for "km" attribute
km_mean=round(dataSet['km'].mean())
dataSet['km'].fillna(km_mean,inplace=True)
```

[178]:
```
# Verifing that missing values of 'km' attribute filled with mode
dataSet['km'].loc[km_nullValues_index]
```

[178]:
```
4636    33010.0
4637    33010.0
4638    33010.0
Name: km, dtype: float64
```

[179]:
```
#Verifying that there is no missing values in dataset after imputing missing␣
↪values
dataSet.isnull().sum()
```

[179]:
```
make_model      0
body_type       0
Body Color      0
km              0
Gearing Type    0
Extras          0
price           0
dtype: int64
```

[180]:
```
dataSet.shape[0:2]
```

[180]:
```
(4309, 7)
```

[181]:
```
# checking outliers records in 'km' features
dataSet[(dataSet['km']<=lower_value_km) | (dataSet['km']>=upper_value_km)]
```

[181]:
```
     make_model      body_type Body Color        km Gearing Type  Extras  \
24      Audi A1         Sedans      White  115000.0       Manual  Others
32      Audi A1         Sedans      White  126000.0       Manual  Others
59      Audi A1         Sedans      Black  192000.0       Manual  Others
60      Audi A1        Compact      White  130000.0       Manual  Others
291     Audi A1        Compact       Blue  137066.0    Automatic   Sport
…          …              …         …        …            …       …
3682    Audi A3  Station wagon      Black  139879.0       Manual   Sport
3684    Audi A3         Sedans      Black  131900.0       Manual   Sport
3686    Audi A3  Station wagon      White  115000.0       Manual   Sport
3687    Audi A3         Sedans      White  127000.0       Manual  Others
3695    Audi A3         Sedans      White  114550.0       Manual   Sport

      price
24     8999
32    11900
59    10000
60    10490
291   14998
…       …
3682  16900
3684  16888
```

```
3686   16900
3687   16900
3695   19797

[188 rows x 7 columns]
```

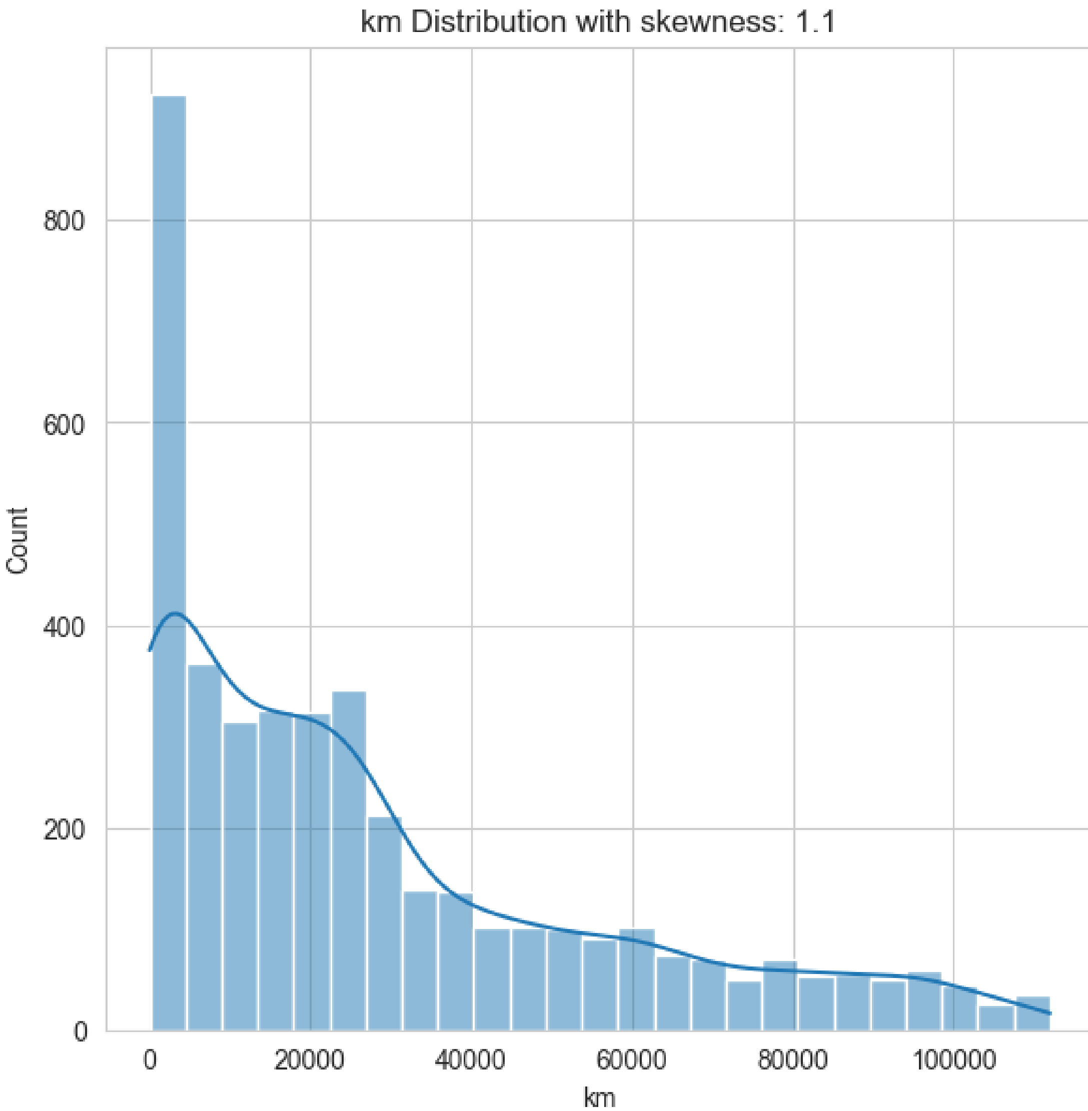
[182]:
```
# removing outliers records in 'km' features
dataSet_without_outliers=dataSet[(dataSet['km']>=lower_value_km) &␣
↪(dataSet['km']<=upper_value_km)]
```

[183]:
```
# # checking that outliers records are removed in 'km' features
dataSet_without_outliers[(dataSet['km']<=lower_value_km) |␣
↪(dataSet['km']>=upper_value_km)]
```

[183]:
```
Empty DataFrame
Columns: [make_model, body_type, Body Color, km, Gearing Type, Extras, price]
Index: []
```

[184]:
```
#Histogram Plot for Numerical column "km"
plt.figure(figsize=(15,15))
plt.subplot(2,2,1)
sbn.histplot(dataSet_without_outliers['km'],bins=25,kde=True)
skewness = round(dataSet_without_outliers['km'].skew(),1)
plt.title(f"km Distribution with skewness: {skewness}")
```

[184]:
```
Text(0.5, 1.0, 'km Distribution with skewness: 1.1')
```



0.3.11 Standardise the data (1M)

Standardization is the process of transforming data into a common format which you to make the meaningful comparison.

Obervation:- In car_pricing_prediction dataset 'km' is numerical attribute which does not show normal distribution as per above histogram plot.hence standardization has not been applied on dataset

0.3.12 Normalise the data wherever necessary(1M)

[185]:
```
# Min Max Scaler is used to perform normalization only for one numerical␣
 ↪varaibles which is km,
# and hp attribute is alredy dropped as 'hp' attribute has no variation
# and  pricing is a target varaible in dataset
# hence normalization has not been applied on hp & pricing
#importing MinMaxScaler library
from sklearn.preprocessing import MinMaxScaler
# taking list of all numerocal attributes
numerical_attributes_list = ['km']
scalling=MinMaxScaler()
dataSet_without_outliers[numerical_attributes_list] = scalling.
 ↪fit_transform(dataSet_without_outliers[numerical_attributes_list])
print(dataSet_without_outliers.head())
```

```
  make_model body_type Body Color        km Gearing Type     Extras  price
0    Audi A1    Sedans      Black  0.500116    Automatic  Converter  15770
1    Audi A1    Sedans        Red  0.714286    Automatic      Sport  14500
2    Audi A1    Sedans      Black  0.745089    Automatic     Others  14640
3    Audi A1    Sedans      Brown  0.651786    Automatic      Sport  14500
4    Audi A1    Sedans      Black  0.144643    Automatic      Sport  16790
```

0.3.13 Perform Binning (1M)

Binning is a process of transforming continuous numerical variables into discrete categorical 'bins', for grouped analysis.

[186]:
```
#Binning of 'km'
dataSet_without_outliers['km_binned'] = pd.cut(dataSet_without_outliers['km'],␣
 ↪bins=20, labels=False)
print(f'Maximum of km by Bin :- {dataSet_without_outliers.
 ↪groupby("km_binned")["km"].max()}')
print(f'Minimum of km by Bin :- {dataSet_without_outliers.
 ↪groupby("km_binned")["km"].min()}')

#Binning of 'pricing'
dataSet_without_outliers['price_binned'] = pd.
 ↪cut(dataSet_without_outliers['price'], bins=20, labels=False)
print(f'Maximum of price by Bin :- {dataSet_without_outliers.
 ↪groupby("price_binned")["price"].max()}')
print(f'Minimum of price by Bin :- {dataSet_without_outliers.
 ↪groupby("price_binned")["price"].min()}')
```

```
Maximum of km by Bin :- km_binned
0      0.049875
1      0.100000
2      0.150000
```

```
3     0.199955
4     0.250000
5     0.299107
6     0.350000
7     0.399732
8     0.448929
9     0.500000
10    0.549839
11    0.598929
12    0.649652
13    0.699813
14    0.749589
15    0.799679
16    0.848884
17    0.898384
18    0.950000
19    1.000000
Name: km, dtype: float64
Minimum of km by Bin :- km_binned
0     0.000000
1     0.050009
2     0.100045
3     0.150179
4     0.200179
5     0.250348
6     0.301339
7     0.350411
8     0.400714
9     0.450330
10    0.500116
11    0.550188
12    0.600696
13    0.651152
14    0.700893
15    0.750000
16    0.801893
17    0.850313
18    0.900286
19    0.952286
Name: km, dtype: float64
Maximum of price by Bin :- price_binned
0      5555
1     10600
2     13100
3     15660
4     18185
5     20700
6     23240
```

```
7      25750
8      28300
9      30000
10     33250
11     35500
12     38000
13     39575
14     41495
19     56100
Name: price, dtype: int64
Minimum of price by Bin :- price_binned
0       5555
1      10300
2      10800
3      13200
4      15667
5      18200
6      20730
7      23250
8      25780
9      28390
10     30900
11     33800
12     35900
13     38495
14     41495
19     56100
Name: price, dtype: int64
```

[187]:

```
dataSet_without_outliers
```

[187]:

```
      make_model      body_type Body Color        km Gearing Type     Extras  \
0        Audi A1         Sedans      Black  0.500116    Automatic  Converter
1        Audi A1         Sedans        Red  0.714286    Automatic      Sport
2        Audi A1         Sedans      Black  0.745089    Automatic     Others
3        Audi A1         Sedans      Brown  0.651786    Automatic      Sport
4        Audi A1         Sedans      Black  0.144643    Automatic      Sport
…            …              …          …         …            …
4793     Audi A3         Sedans        Red  0.000089       Manual      Sport
4794     Audi A3         Sedans     Silver  0.000089       Manual     Others
4795     Audi A3         Sedans      White  0.000482       Manual      Sport
4796     Audi A3         Sedans      White  0.000446       Manual     Others
4797     Audi A3  Station wagon     Silver  0.059518       Manual     Others

      price  km_binned  price_binned
0     15770         10             4
1     14500         14             3
```

```
2      14640         14             3
3      14500         13             3
4      16790          2             4
…      …        …             …
4793   24987          0             7
4794   24980          0             7
4795   25000          0             7
4796   24980          0             7
4797   24980          1             7

[4121 rows x 9 columns]
```

0.3.14 Perform encoding (1M)

[188]:
```python
# importing LabelEncoder package
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
categorical = ['make_model', 'body_type', 'Body Color','Gearing Type','Extras']
for feature in categorical:
        le = LabelEncoder()
        dataSet_without_outliers[feature] = le.
 ↪fit_transform(dataSet_without_outliers[feature])
```

[189]:
```python
dataSet_without_outliers
```

[189]:
```
      make_model  body_type  Body Color        km  Gearing Type  Extras  \
0              0          5           1  0.500116             0       0
1              0          5           8  0.714286             0       2
2              0          5           1  0.745089             0       1
3              0          5           4  0.651786             0       2
4              0          5           1  0.144643             0       2
…          …         …          …       …             …      …
4793           2          5           8  0.000089             1       2
4794           2          5           9  0.000089             1       1
4795           2          5          11  0.000482             1       2
4796           2          5          11  0.000446             1       1
4797           2          6           9  0.059518             1       1

      price  km_binned  price_binned
0     15770         10             4
1     14500         14             3
2     14640         14             3
3     14500         13             3
4     16790          2             4
…     …        …             …
4793  24987          0             7
4794  24980          0             7
```

```
4795   25000          0             7
4796   24980          0             7
4797   24980          1             7

[4121 rows x 9 columns]
```

0.3.15 Perform Data Discretization(2M)

Not required as alredy performed binning process .

0.3.16 EDA using Visuals(3M)

Use any 3 or more visualisation methods (Boxplot,Scatterplot,histogram,….etc) to perform Exploratory data analysis and briefly give interpretations from each visual.

(a) Library Import

[190]:
```python
# importing required package
# importing required packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sbn
import warnings as wrn
from datetime import datetime
wrn.filterwarnings('ignore')
```

(b) Read DataSet

[191]:
```python
# defining dataSet github raw url for csv dataSet file
github_filePath="https://raw.githubusercontent.com/shreenath197/
↪car_pricing_dataSet/main/car_price.csv"
# loading dataSet
dataSet=pd.read_csv(github_filePath)
```

(c) Incpect the data

[192]:
```python
#feature hp gets dropped since is 85 for each row
dataSet.drop(columns=["hp"], inplace=True)
# Checking quick overview of the dataSet
print(dataSet.head())
# Checking descriptive statistic of the dataSet
print(dataSet.describe())
#checking summary of the dataSet
print(dataSet.info())
```

```
  make_model body_type Body Color       km Gearing Type     Extras  price
0    Audi A1    Sedans      Black  56013.0    Automatic  Converter  15770
1    Audi A1    Sedans        Red  80000.0    Automatic      Sport  14500
```

```
2    Audi A1    Sedans      Black  83450.0    Automatic     Others  14640
3    Audi A1    Sedans      Brown  73000.0    Automatic      Sport  14500
4    Audi A1    Sedans      Black  16200.0    Automatic      Sport  16790
                  km         price
count    4797.000000   4800.000000
mean    31931.949135  19722.871875
std     35902.589244   4337.519969
min         0.000000   5555.000000
25%      4800.000000  15990.000000
50%     20049.000000  19588.000000
75%     47800.000000  22692.500000
max    291800.000000  56100.000000
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4800 entries, 0 to 4799
Data columns (total 7 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   make_model    4800 non-null   object
 1   body_type     4800 non-null   object
 2   Body Color    4796 non-null   object
 3   km            4797 non-null   float64
 4   Gearing Type  4800 non-null   object
 5   Extras        4800 non-null   object
 6   price         4800 non-null   int64
dtypes: float64(1), int64(1), object(5)
memory usage: 262.6+ KB
None
```

(d) Check for missing , duplicates & unique values

[193]:
```
#Calculating the number of missing values in each column
print(dataSet.isnull().sum())
# Handeling missing values
print(dataSet.dropna())
# Dropping Duplicates if exists
dataSet_duplicate = dataSet.drop_duplicates()
print(dataSet_duplicate.shape)
# Checking unique values columswise
print(dataSet.nunique())
```

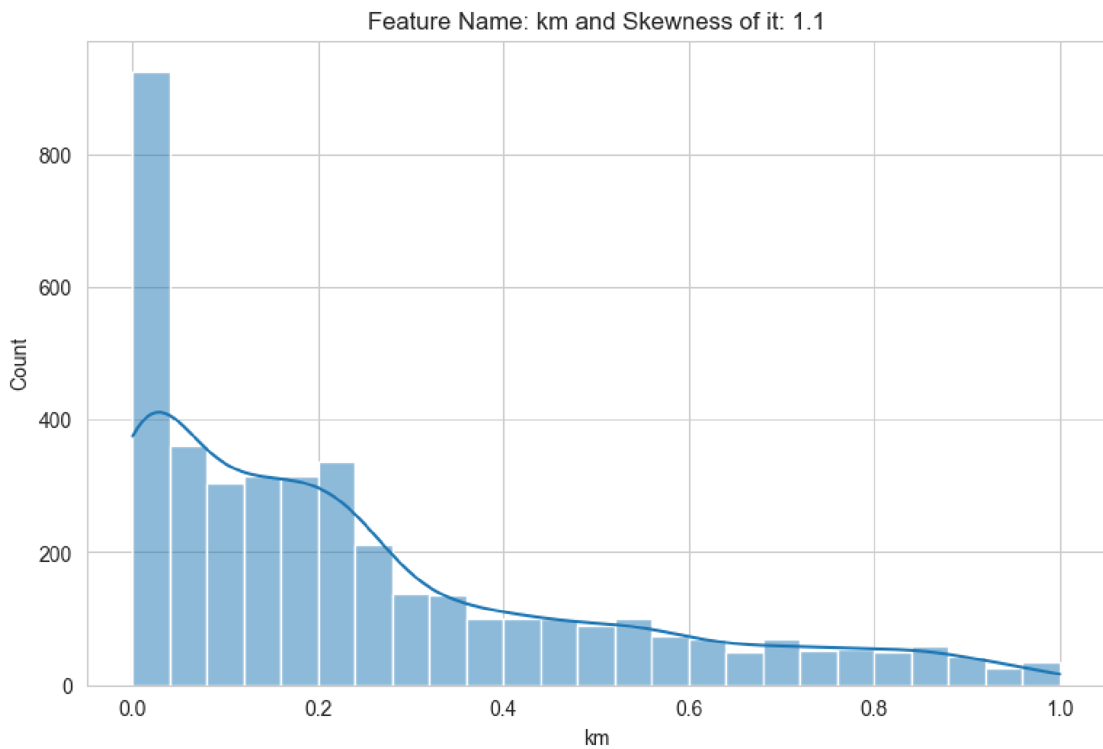
```
make_model      0
body_type       0
Body Color      4
km              3
Gearing Type    0
Extras          0
price           0
dtype: int64
```

```
     make_model      body_type Body Color       km Gearing Type     Extras  \
0       Audi A1         Sedans      Black  56013.0    Automatic  Converter
1       Audi A1         Sedans        Red  80000.0    Automatic      Sport
2       Audi A1         Sedans      Black  83450.0    Automatic     Others
3       Audi A1         Sedans      Brown  73000.0    Automatic      Sport
4       Audi A1         Sedans      Black  16200.0    Automatic      Sport
…         …            …        …        …          …          …
4795    Audi A3         Sedans      White     54.0       Manual      Sport
4796    Audi A3         Sedans      White     50.0       Manual     Others
4797    Audi A3  Station wagon     Silver   6666.0       Manual     Others
4798    Audi A3         Sedans     Silver     10.0       Manual     Others
4799    Audi A3         Sedans     Silver     10.0       Manual     Others

      price
0     15770
1     14500
2     14640
3     14500
4     16790
…     …
4795  25000
4796  24980
4797  24980
4798  24980
4799  24980

[4793 rows x 7 columns]
(4309, 7)
make_model         3
body_type          7
Body Color        13
km              2443
Gearing Type       3
Extras             3
price           1155
dtype: int64
```

(c) Histogram Plot

[194]:
```
# Set the parameters that control the general style of the plots.
sbn.set_style(style="whitegrid")
#dropping dependent feature column
X=dataSet_without_outliers.drop('price',axis=1)
# Selecting numerical independent feature columns names
columns=['km']
#Building Histogram plot for each independent features
plt.figure(figsize=(15, 30) )
for index, columns_name in enumerate(columns, 1):
```

```
    plt.subplot(6, 2,index)
    sbn.histplot(X[columns_name], kde=True)
    skewness = round(X[columns_name].skew(),1)
    plt.title("Feature Name: {} and Skewness of it: {}".
 ↪format(columns_name,skewness))
plt.tight_layout()
plt.show()
```

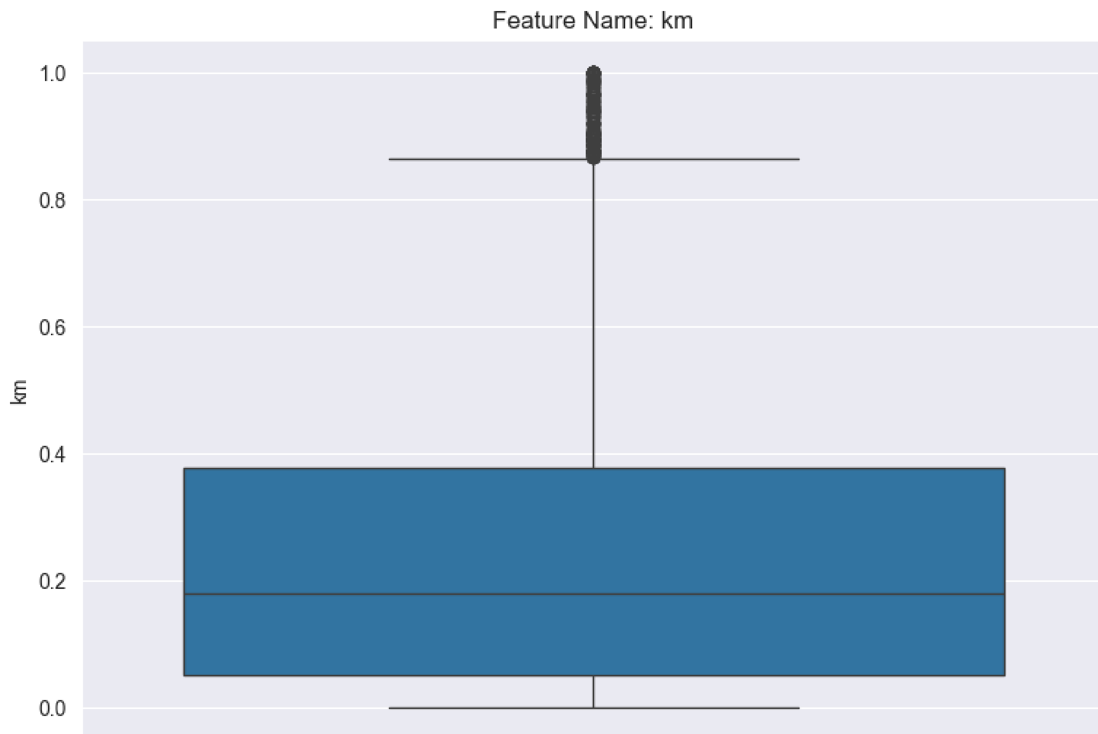


0.3.17 skewness and histogram shows that except Widht column all the columns are skewed & do not follow perfectly normal distribution

(d) Box Plot

[195]:
```
# Set the parameters that control the general style of the plots.
sbn.set_style("darkgrid")
#dropping dependent feature column
X=dataSet_without_outliers.drop('price',axis=1)
# Selecting numerical independent feature columns names
columns=['km']
#Building Box plot for each independent features
plt.figure(figsize=(15, 30) )
for index, columns_name in enumerate(columns, 1):
    plt.subplot(6, 2, index)
```

```
    sbn.boxplot(X[columns_name])
    plt.title("Feature Name: {}".format(columns_name))
plt.tight_layout()
plt.show()
```

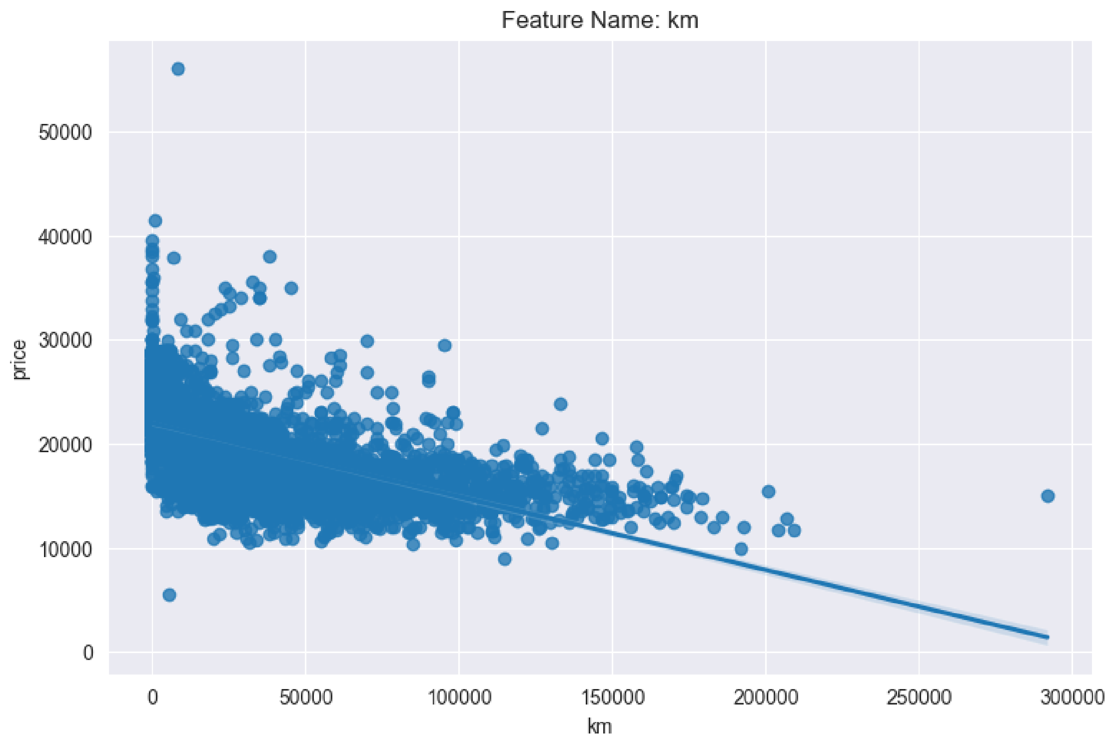


Boxplot shows that in km numerical independent attribute has outliers

(c) Scatter Plot with Regression Line The dataset has almost exclusively categorical data, (except for km attribute). Being the numerosity of unique elements (for each categorical feature) not too high, we could think of One-Hot encode each one of them.

[196]:
```
# Set the parameters that control the general style of the plots.
sbn.set_style("darkgrid")
#dropping dependent feature column
X=dataSet.drop('price',axis=1)
# Selecting numerical independent feature columns names
columns=['km']
#Building Box plot for each independent features
plt.figure(figsize=(15, 30) )
for index, columns_name in enumerate(columns, 1):
    plt.subplot(6, 2, index)
    # draw regplot
    sbn.regplot(data=dataSet,x=dataSet[columns_name],y='price')
```

```
    plt.title("Feature Name: {}".format(columns_name))
plt.tight_layout()
# show the plot
plt.show()
```



The km column probably refers to used cars. If we look at the scatter plot price-km, there is a negative correlation between the two attributes. We can also plot the average price for a car, given a certain body type, color, gearing and model. Looks like orange cars are cheaper!

0.3.18 Feature Selection(2M)

Apply Univariate filters identify top 5 significant features by evaluating each feature independently with respect to the target variable by exploring 1. Mutual Information (Information Gain) 2. Gini index 3. Gain Ratio 4. Chi-Squared test 5. Fisher Score (From the above 5 you are required to use any two)

(a) 1st Filter Method - Mutual Information Classifier Method for Categorical variables

[197]:
```
from sklearn.feature_selection import SelectKBest, mutual_info_classif
# X & Y intialization
categorical_var_list = ['make_model', 'body_type', 'Body Color','Gearing␣
↪Type','Extras','km_binned']
```

```
X= dataSet_without_outliers[categorical_var_list]
Y = dataSet_without_outliers["price_binned"]
# mutual info function fit
mut_info = SelectKBest(score_func=mutual_info_classif,k=5)
mut_info.fit_transform(X,Y)
# dataframe creation to contain mutual info score with variable names
mut_info_score = pd.DataFrame(mut_info.scores_,index=X.
↪columns,columns=['Mutual_Info_Score'])
# sorting mutual infor score in descending order
mut_info_score.sort_values(by=['Mutual_Info_Score'],ascending=False)
```

[197]:

	Mutual_Info_Score
km_binned	0.303887
Gearing Type	0.134836
make_model	0.077278
Body Color	0.045767
Extras	0.021127
body_type	0.011607

(b) 2nd Filter Method : Chi Square Test among categorical variables

[198]:

```
from scipy.stats import chi2_contingency
# categorical variable list define
categorical_var_list = ['make_model', 'body_type', 'Body Color','Gearing␣
↪Type','Extras','km_binned']
list1 = []

# calculation p_value for chi sqaure test for all categorical variable present␣
↪in the list above
for column in categorical_var_list:
    contingency_table = pd.crosstab(dataSet_without_outliers[column],␣
↪dataSet_without_outliers['price_binned'])
    chi2, p, degrees_of_freedom, expected_result =␣
↪chi2_contingency(contingency_table)
    list1.append(round(p,6))

# dataframe creation to contain variable names and their correspondi p_vales␣
↪for chi sqaure
df2 = pd.DataFrame()
df2['Column_Name'] = categorical_var_list
df2['Chi_Sq_P_Value'] = list1

df2.sort_values(by=['Chi_Sq_P_Value'])
```

[198]:

	Column_Name	Chi_Sq_P_Value
0	make_model	0.0
1	body_type	0.0

```
2    Body Color              0.0
3  Gearing Type              0.0
4        Extras              0.0
5     km_binned              0.0
```

0.3.19 Report observations (2M)

Write your observations from the results of each of the above method(1M). Clearly justify your choice of the method.(1M)

Obersvations:-

Mutual Information :

- Mutual Information Classfier is used to perform this filter method for feature selection.
- km has also been binned for Mutual Information method.
- pricing, the target variable is contineous as a numerical variable is needed for mututal information classifier method.
- We can see the top 5 varaibles who have got the most mutual information with pricing_binned (pricing_binned Field) variable are -
    1. km_binned

    2. Gearing Type
    3. make_model
    4. Extras
    5. Body Color

Chi Square :

- As Chi Square needs the target of dependent variable a categorical field , hence pricing field is numerical contineous variable to perform chi square
- km has also been binned for Mutual Information method.
- Since all p-values are 0.0, it indicates a strong relationship with the target variable. we can say that top 5 features will be
    1. km_binned

    2. Gearing Type
    3. make_model
    4. Extras
    5. Body Color
- In Chi Square we usually take those features whose p values are less than 0.05 and if p values are less than 0.05 we reject the null hypothesis.

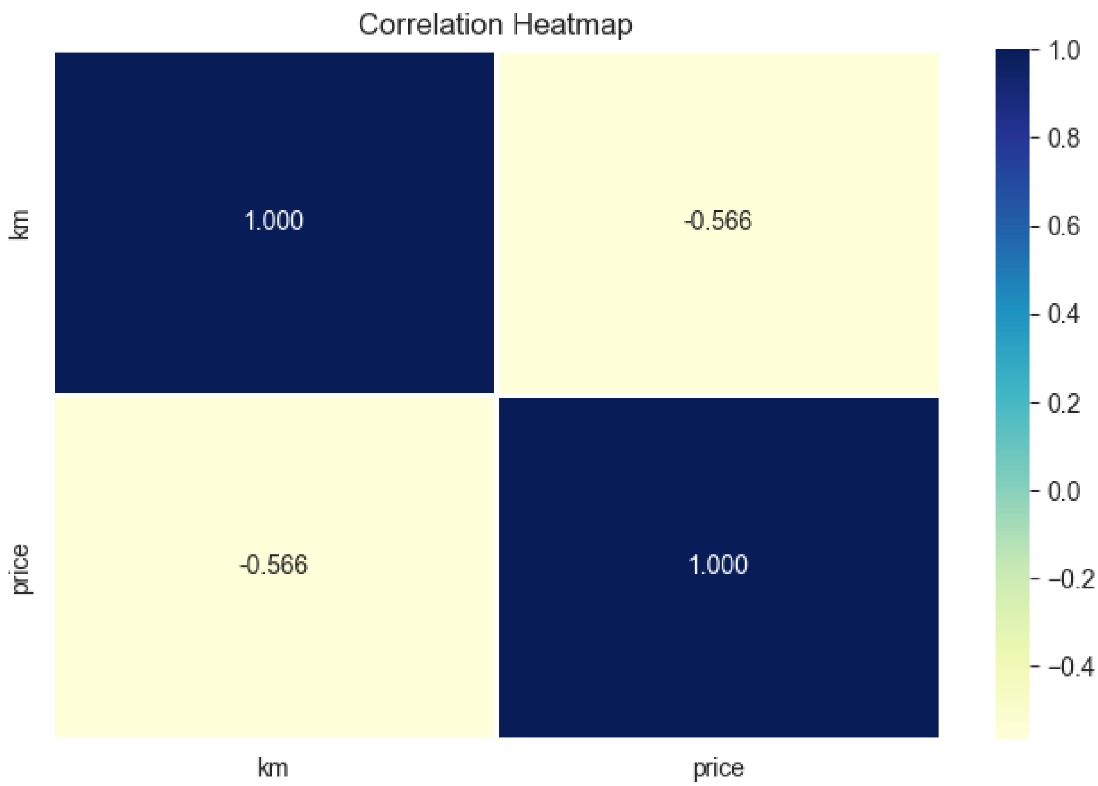
0.3.20 Correlation Analysis (3 M)

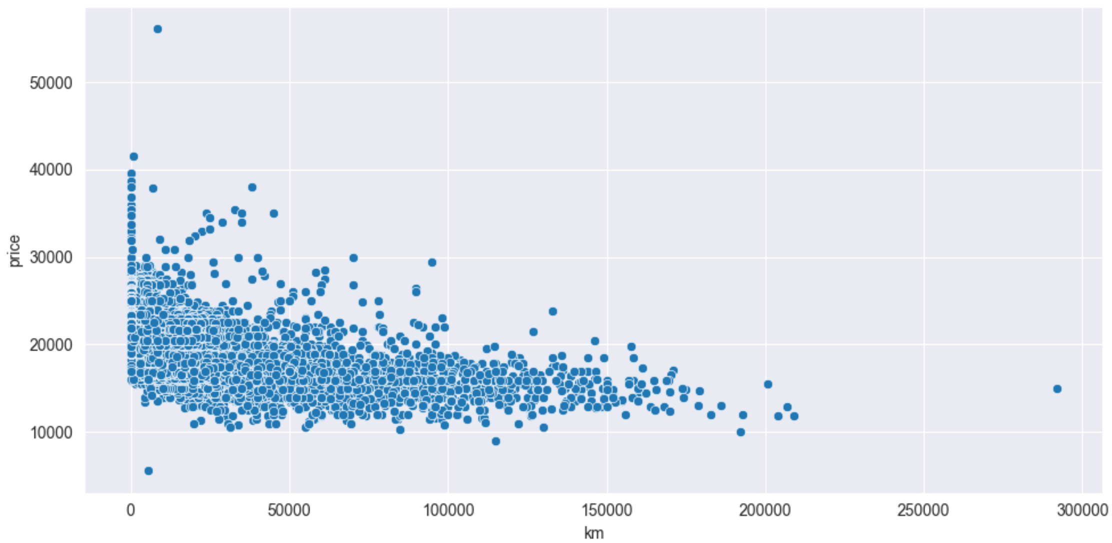
Perform correlation analysis(1M) and plot the visuals(1M).Briefly explain each process,why is it used and interpret the result(1M).

[199]:

```
plt.figure(figsize=(8, 5))
# Using Seaborn to create a heatmap
columns=dataSet_without_outliers[['price','km']]
plt.figure(figsize=(8, 5))
# Using Seaborn to create a heatmap
sbn.heatmap(dataSet_without_outliers[['km','price']].corr(method='pearson'),␣
↪annot=True, fmt='.3f', cmap='YlGnBu', linewidths=2)
plt.title('Correlation Heatmap')
plt.show()
dataSet_without_outliers[['price','km']].corr(method='pearson')
#scattered plot
plt.figure(figsize=(10,5))
sbn.scatterplot(x = dataSet['km'], y=dataSet['price'])
plt.tight_layout()
plt.show()
```

```
<Figure size 800x500 with 0 Axes>
```





Pearson Correlation :

```
* Pearson correaltion method has been applied on only numerical features and there are only 1 
  1- km . Target variable is Pricing.
* The correaltion coeffecient between km and pricing is super strong  -0.566. we can say there 
  relationship between km and pricing.
* The more the km will be , the less pricing will be
```

0.3.21 Model Building and Prediction (4M)

Fit a linear regression model using the most important features identified(1M).Plot the visuals(1M).Briefly explain the regression model,equation (1M) and perform one prediction using the same(1M).

0.3.22 Model Building

Split data in to features & target

[200]:
```python
# importing required package
from sklearn.model_selection import train_test_split
#Dropping all columns except make_model        , km, Gearing Type
X=dataSet_without_outliers.
 ↪drop(['price','km_binned','price_binned','Extras','body_type','Body␣
 ↪Color'],axis=1)
#Dropping all columns except Cost
Y=dataSet_without_outliers['price']
#Splitting train & test data
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.
 ↪28,random_state=42)
```

[201]:
```
Y.head()
```

[201]:
```
0    15770
1    14500
2    14640
3    14500
4    16790
Name: price, dtype: int64
```

[202]:
```
Y_train=Y_train.reset_index(drop=True)
type(y_pred_train)
```

[202]:
```
numpy.ndarray
```

[203]:
```
y_pred_train=y_pred_train.tolist()
```

Linear Regression Model using OLS

[204]:
```
from sklearn.linear_model import LinearRegression
from datetime import datetime
# Fit the model Linear Regression
start_time = datetime.now()
model = LinearRegression()
model.fit(X_train, Y_train)
OLS_time = datetime.now() - start_time
print(f'Time Taken : {datetime.now() - start_time}')

# (********************On Train data ******************)
print("***** On Train Data *****")
OLS_score = model.score(X_train, Y_train)
print(f"Model score: {OLS_score}")
print(f"Model X Coef :  {model.coef_} ")
print(f"Model Intercept :  {model.intercept_} ")

# Prediction on Train Data
y_pred_train = model.predict(X_train)
y_train_list = Y_train.to_list()
# Perform Squared error Calculation
se_sum_train = 0
for i, j in zip( y_train_list, y_pred_train):
  se_sum_train = se_sum_train +  (i - j)**2

OLS_mse_train = se_sum_train/len(y_train_list)
OLS_rmse_train = (se_sum_train / len(y_train_list)) ** .5

print(f"Model MSE : {OLS_mse_train}")
print(f"Model RMSE :  {OLS_rmse_train} ")
```

```python
# (*********************On Test data *******************)
print("***** On Test Data *****")
score = model.score(X_train, Y_train)
y_pred_test = model.predict(X_test)
y_test_list = Y_test.to_list()

# Squared error Calculation
se_sum_test = 0
for i, j in zip( y_test_list, y_pred_test.tolist()):
  se_sum_test = se_sum_test +  (i - j)**2
OLS_mse_test = se_sum_test/len(y_test_list)
OLS_rmse_test = (se_sum_test / len(y_test_list)) ** .5

print(f"Model MSE : {OLS_mse_test}")
print(f"Model RMSE :  {OLS_rmse_test} ")
```

```
Time Taken : 0:00:00.004008
***** On Train Data *****
Model score: 0.48291081141845493
Model X Coef :   [  1720.28880179 -10417.14571866   -740.63119602]
Model Intercept :  21014.645912202057
Model MSE : 9371349.886188282
Model RMSE :  3061.2660593598007
***** On Test Data *****
Model MSE : 9345393.220616017
Model RMSE :  3057.0235884951912
```

0.3.23 Linear regression using Gradient Descent

[205]:
```python
from sklearn.metrics import r2_score

class GD:

    def __init__(self,lr=0.01,loops=100):

        self.coef_ = None
        self.intercept_ = None
        self.lr = lr
        self.loops = loops

    def fit_gd(self,X_train,y_train):


        start_time = datetime.now()
        # Intercept is initialized as 0
```

```python
        self.intercept = 0
        # All coefficients are defined as 1 using numpy.
        self.coef = np.ones(X_train.shape[1])

        for i in range(self.loops):
            # intercept and coefficient modified here
            y_pred = np.dot(X_train,self.coef) + self.intercept
            intercept_derivative = -2 * np.mean(y_train - y_pred)
            self.intercept = self.intercept - (self.lr * intercept_derivative)

            coef_derivative = -2 * np.dot((y_train - y_pred),X_train)/X_train.
 ↪shape[0]
            self.coef = self.coef - (self.lr * coef_derivative)

        print(f'Time Taken : {datetime.now() - start_time}')
        time_taken = datetime.now() - start_time
        return self.intercept,self.coef,time_taken

    def predict_gd(self,data):
        #  predicting result value
        return np.dot(data,self.coef) + self.intercept



# On Training␣
 ↪Data-----------------------------------------------------------------------------------
# Class Object initialization (Learning Rate and number of epochs defined)
gd_obj = GD(loops=2000,lr=0.01)
# Model Fit
coef,intercept,GD_time = gd_obj.fit_gd((X_train.to_numpy()),Y_train)
# Train Data Prediction
y_pred_train_gd = gd_obj.predict_gd((X_train.to_numpy()))
# Model Score
GD_score = r2_score(Y_train,y_pred_train_gd)
print('***** On Train Data *****')
print(f"Model score: {GD_score}")
print(f"Model X Coef :  {coef} ")
print(f"Model Intercept :  {intercept} ")


y_train_list = Y_train.tolist()

# Squared error Calculation
se_sum_train = 0
for i, j in zip( y_train_list, y_pred_train_gd.tolist()):
  se_sum_train = se_sum_train +  (i - j)**2
```

```
GD_mse_train = se_sum_train/len(y_train_list)
GD_rmse_train = (se_sum_train / len(y_train_list)) ** .5

print(f"Model MSE : {GD_mse_train}")
print(f"Model RMSE :  {GD_rmse_train} ")


# On Test␣
  ↪Data-----------------------------------------------------------------------------------
print('***** On Test Data *****')
y_pred_test_gd = gd_obj.predict_gd((X_test.to_numpy()))
y_test_list = Y_test.tolist()

# Squared error Calculation
se_sum_test = 0
for i, j in zip( y_test_list, y_pred_test_gd.tolist()):
  se_sum_test = se_sum_test +  (i - j)**2

GD_mse_test = se_sum_test/len(y_test_list)
GD_rmse_test = (se_sum_test / len(y_test_list)) ** .5

print(f"Model MSE : {GD_mse_test}")
print(f"Model RMSE :  {GD_rmse_test} ")
```

```
Time Taken : 0:00:00.393773
***** On Train Data *****
Model score: 0.47698029592184543
Model X Coef :  20744.759013525625
Model Intercept :  [ 1675.51952351 -9098.1738816   -788.9062358 ]
Model MSE : 9478830.253118116
Model RMSE :  3078.77089974524
***** On Test Data *****
Model MSE : 9520327.784223542
Model RMSE :  3085.50284138964
```

0.3.24 Linear regression using Stochastic Gradient Descent

[206]:
```
from sklearn.metrics import r2_score
class SGD:

    def __init__(self,lr=0.01,loops=100):

        self.coef = None
        self.intercept = None
        self.lr = lr
        self.loops = loops
```

```python
    def fit_sgd(self,X_train,y_train):

        start_time = datetime.now()
        # Intercept is initialized as 0
        self.intercept = 0
        # All coefficients are defined as 1 using numpy.
        self.coef = np.ones(X_train.shape[1])

        for i in range(self.loops):
            for j in range(X_train.shape[0]):
                indx = np.random.randint(0,X_train.shape[0])
                # intercept and coefficient modified here
                y_hat = np.dot(X_train[indx],self.coef) + self.intercept

                intercept_der = -2 * (y_train[indx] - y_hat)

                self.intercept = self.intercept - (self.lr * intercept_der)

                coef_der = -2 * np.dot((y_train[indx] - y_hat),X_train[indx])
                self.coef = self.coef - (self.lr * coef_der)

        print(f'Time Taken : {datetime.now() - start_time}')
        time_taken = datetime.now() - start_time
        return self.intercept,self.coef,time_taken

    def predict_sgd(self,X_test):
        return np.dot(X_test,self.coef) + self.intercept




# ****************************** On Training Data␣
 ↪**********************************
# Class Object initialization (Learning Rate and number of epochs defined)
print(' ****************SGD ******************** ')
print('***** On Train Data *****')
sgd_obj = SGD(loops=30,lr=0.01)
# Model Fit

coef,intercept,SGD_time = sgd_obj.fit_sgd((X_train.to_numpy()),Y_train)
# Train Data Prediction
y_pred_train_sgd = sgd_obj.predict_sgd((X_train.to_numpy()))
# # Model Score
SGD_score = r2_score(Y_train,y_pred_train_sgd)

print(f"Model score: {SGD_score}")
print(f"Model X Coef :  {coef} ")
```

```
print(f"Model Intercept :  {intercept} ")


y_train_list = Y_train.tolist()

# # Squared error Calculation
se_sum_train = 0
for i, j in zip( y_train_list, y_pred_train_sgd.tolist()):
  se_sum_train = se_sum_train +  (i - j)**2

SGD_mse_train = se_sum_train/len(y_train_list)

SGD_rmse_train = (se_sum_train / len(y_train_list)) ** .5

print(f"Model MSE : {SGD_mse_train}")
print(f"Model RMSE :  {SGD_rmse_train} ")


# On Test␣
↪Data-----------------------------------------------------------------------------------
print('***** On Test Data *****')
y_pred_test_sgd = sgd_obj.predict_sgd((X_test.to_numpy()))
y_test_list = Y_test.tolist()

# Squared error Calculation
se_sum_test = 0
for i, j in zip( y_test_list, y_pred_test_sgd.tolist()):
  se_sum_test = se_sum_test +  (i - j)**2

SGD_mse_test = se_sum_test/len(y_test_list)
SGD_rmse_test = (se_sum_test / len(y_test_list)) ** .5

print(f"Model MSE : {SGD_mse_test}")
print(f"Model RMSE :  {SGD_rmse_test} ")
```

```
 ******************SGD **********************
***** On Train Data *****
Time Taken : 0:00:01.713199
Model score: 0.46182582415860307
Model X Coef :  20706.077458505704
Model Intercept :  [  2139.95817923 -10713.37339295   -274.81419802]
Model MSE : 9753478.921800004
Model RMSE :  3123.0560228404493
***** On Test Data *****
Model MSE : 9635172.38145858
Model RMSE :  3104.0574062762726
```

0.3.25 Observations and Conclusions(1M)

0.3.26 Observations

Feature Relevance: Identify and select features that have a significant impact on the target variable. For example, car km,makemodel,Gearing Type are typically highly relevant for price prediction.

Data Distribution: Analyze the distribution of the data to understand the range and frequency of different values. This can help identify any skewness or outliers that may affect the model's performance.

Correlation Analysis: Perform correlation analysis to determine how features are related to each other and to the target variable. This helps in selecting the most impactful features and reducing multicollinearity.

Model Performance: Evaluate different machine learning models to find the one that best fits the data. Common models for car prediction include linear regression, decision trees, random forests, and gradient boosting machines.

Evaluation Metrics: Use appropriate evaluation metrics such as Mean Absolute Error (MAE), Mean Squared Error (MSE), and R-squared to measure the performance of the models.

Overfitting and Underfitting: Monitor for overfitting (model performs well on training data but poorly on test data) and underfitting (model performs poorly on both training and test data). Techniques such as cross-validation, regularization, and pruning can help address these issues.

Data Preprocessing: Ensure data is preprocessed correctly, including handling missing values, encoding categorical variables, scaling numerical features, and splitting data into training and testing sets.

Conclusions Model Selection: Based on the evaluation metrics, choose the model that provides the best balance of accuracy and generalization. For instance, if a random forest model has the lowest MAE, it might be the best choice for predicting car prices.

Feature Importance: Identify which features are most important for the prediction. For example, the year of manufacture, mileage, and brand might be the top predictors of car price.

Predictive Accuracy: Conclude on the predictive accuracy of the model. If the model's predictions closely match actual values with low error metrics, it indicates good performance.

Insights for Improvement: Highlight areas for further improvement, such as collecting more data, engineering additional features, or trying more advanced models.

Practical Applications: Discuss practical applications of the model, such as using it for pricing strategy in car dealerships, recommending prices to sellers on online platforms, or evaluating trade-in values for customers.

Limitations: Acknowledge any limitations of the model, such as the potential impact of market fluctuations, regional price differences, or the condition of the car not being fully captured by the available data.

Future Work: Suggest directions for future work, such as incorporating real-time data, exploring more complex models like deep learning, or integrating additional features like market demand and supply trends.

By following this structured approach, you can effectively observe and conclude on car prediction models, ensuring they are robust, accurate, and useful for practical applications.

0.3.27 Solution (1M)

What is the solution that is proposed to solve the business problem discussed in the beginning. Also share your learnings while working through solving the problem in terms of challenges, observations, decisions made etc.

0.3.28 Solution:

The solution proposed is a predictive model for car prices based on various car attributes. The model helps manufacturers, dealers, and customers make informed decisions regarding car pricing. Learnings:

• Challenges: Handling missing data and data inconsistencies was challenging.

• Observations: Feature selection helped in identifying the most influential features for predicting car prices.

• Decisions Made: Standardization and normalization were applied to ensure data was in a common format.

Summary

1. Business Understanding: Identified the need for a car price prediction model.
2. Data Requirements and Collection: Collected data from Kaggle and identified key attributes.
3. Data Quality Checks: Handled duplicates, missing data, and inconsistencies.
4. Data Standardization and Normalization: Applied techniques to standardize and normalize data.
5. Exploratory Data Analysis: Used visuals to understand data distribution and relationships.
6. Feature Selection: Applied univariate filters to identify significant features.
7. Correlation Analysis: Analyzed correlations between features.
8. Model Building: Built and evaluated a linear regression model.
9. Solution and Learnings: Proposed a solution for car price prediction and shared learnings.