ASSIGNMENT 2 - REPORT

Data manipulation + Exploratory data analysis + Regression Analysis

Chirag Hebbal Rudresh
ID: 1002160960

10/21/23

DASC 5301-002 Data Science

INDEX

ASSIGNMENT 2 REPORT

1. Data from our Lives

A situation where a regression model would be appropriate is predicting the final GPA at the end of the academic year.

The factors that can influence a student's GPA are:

1. **Exam Scores**: The results of midterm exams and assignments can be important predictors of the final GPA, as they reflect a student's performance.
2. **Study Hours**: The number of hours a student spends studying is a predictor of their GPA. More study time often results in better grades.
3. **Previous GPA**: A student's GPA from the previous year or semester is a strong predictor of their future performance.
4. **Use of Resources**: Utilizing available resources like tutoring, academic advisors, study groups, and academic support services can positively impact a student's performance.

This assignment focuses data cleaning by removing inaccurate data from the auto imports data set. Data wrangling focuses on transforming the data's format by converting "raw" data into usable format more suitable for getting insights. This assignment uses the auto_imports1 csv file. The data will be modified and insights from this data can be extracted.

2.1 Data Munging

```
from scipy import stats
from sklearn.linear_model import LinearRegression
from statsmodels.compat import lzip
from statsmodels.formula.api import ols
from statsmodels.stats.anova import anova_lm
from statsmodels.stats.outliers_influence import variance_inflation_factor
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import statsmodels.api as sm

%matplotlib inline
```

First, we import the libraries required for the process of data analysis.

Numpy module is used since it has efficient multidimensional array operations, and it has a wide range of mathematical function which are used for computing and data analysis. It is imported with the alias np.

Pandas is python library for data manipulation and analysis. It provides powerful and flexible tools for working with structured data, making it a fundamental library in the field of data science, data analysis, etc. It is mainly used for data exploration, cleaning, transformation, and analysis purposes. It is imported with the alias pd.

```
#Read in data
df =pd.read_csv('auto_imports1.csv')
df.head()
```

Scipy (stats) contains many probability distributions, summary and frequency statistics, correlation functions and statistical tests, masked statistics, kernel density estimation, quasi-Monte Carlo functionality, and more. It is also used in regression, linear models, time series analysis.

Sklearn (Linear Regression) is used to fit a linear model with coefficients w = (w1, …, wp) to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation.

statsmodels is a Python package that provides a complement to scipy for statistical computations including descriptive statistics and estimation and inference for statistical models.

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.

Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

The dataset to be analysed is first equated to pd.read_csv() i.e df= pd.read_csv((function that is used to get the data from the csv file into the notebook environment. If the csv file is in the same folder(location) as that of the python notebook, we can just give the name of the file as argument in the function. For example, in this case it will pd.read_csv('auto_imports1.csv"). In case the data file is at a different location on the computer, we need to provide the path for that file as argument in the function.

We can check if the data is properly imported into the notebook by checking the values of the file that is read using pandas. The df.head() function displays the first 5 rows of the data and the column names and data can be verified.

1. **To find out variable types**: To find the type of variables in the dataset, inbuilt function of df.info() is used. It provides information about the data types of all the columns in the dataset. In this case, it was found that the variables were of object data type, int data type and float data type.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 201 entries, 0 to 200
Data columns (total 18 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   fuel_type     201 non-null    object 
 1   body          201 non-null    object 
 2   wheel_base    201 non-null    float64
 3   length        201 non-null    float64
 4   width         201 non-null    float64
 5   heights       201 non-null    float64
 6   curb_weight   201 non-null    int64  
 7   engine_type   201 non-null    object 
 8   cylinders     201 non-null    object 
 9   engine_size   201 non-null    int64  
 10  bore          201 non-null    object 
 11  stroke        201 non-null    object 
 12  compression   201 non-null    float64
 13  horse_power   201 non-null    object 
 14  peak_rpm      201 non-null    object 
 15  city_mpg      201 non-null    int64  
 16  highway_mpg   201 non-null    int64  
 17  price         201 non-null    int64  
dtypes: float64(5), int64(5), object(8)
memory usage: 28.4+ KB
```

2. **Replace '?' with None**: First to find out which rows had the '?' as their value, df.eq('?').any() function was used.

```
rows = df[df.eq('?').any(axis=1)].index # getting index of rows
rows
```

```
Index([52, 53, 54, 55, 126, 127], dtype='int64')
```

This gives us the index of the rows with value '?'. But this is just used to cross-check the data after replacing. We can see in the image below; the bore and stroke columns have '?' as values. We need to replace these.

idth	heights	curb_weight	engine_type	cylinders	engine_size	bore	stroke	co
64.2	54.1	1950	ohc	four	91	3.08	3.15	
65.7	49.6	2380	rotor	two	70	?	?	
65.7	49.6	2380	rotor	two	70	?	?	
65.7	49.6	2385	rotor	two	70	?	?	
65.7	49.6	2500	rotor	two	80	?	?	
66.5	53.7	2385	ohc	four	122	3.39	3.39	

So, the df.replace() function is used to replace '?' with None.

```
df=df.replace('?', None) # replacing '?' values with None.
```

ders	engine_size	bore	stroke	comprassion
four	91	3.08	3.15	9.0
two	70	None	None	9.4
two	70	None	None	9.4
two	70	None	None	9.4
two	80	None	None	9.4
four	122	3.39	3.39	8.6

In the above picture, we can see the values have been replaced. After checking for location of null values again, we can make sure all the required values have been replaced.

```
df[df.eq('?').any(axis=1)].index # checking if any '?' values are remaining
```

```
Index([], dtype='int64')
```

3. **Change the variables: bore, stroke, horse_power, peak_rpm to float64**: Here, the following formula is used to set the column data types to float64.

```
## Changing datatype of above columns to float64 [NONE VALUES WILL BE CONVERTED TO NAN]
df=df.astype({'bore': 'float64','stroke': 'float64', 'horse_power': 'float64', 'peak_rpm': 'float64'})
```

In the screenshot below, we can see that the above columns data types have been updated from object type to float64 type.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 201 entries, 0 to 200
Data columns (total 18 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   fuel_type     201 non-null    object 
 1   body          201 non-null    object 
 2   wheel_base    201 non-null    float64
 3   length        201 non-null    float64
 4   width         201 non-null    float64
 5   heights       201 non-null    float64
 6   curb_weight   201 non-null    int64  
 7   engine_type   201 non-null    object 
 8   cylinders     201 non-null    object 
 9   engine_size   201 non-null    int64  
 10  bore          197 non-null    float64
 11  stroke        197 non-null    float64
 12  compression   201 non-null    float64
 13  horse_power   199 non-null    float64
 14  peak_rpm      199 non-null    float64
 15  city_mpg      201 non-null    int64  
 16  highway_mpg   201 non-null    int64  
 17  price         201 non-null    int64  
dtypes: float64(9), int64(5), object(4)
memory usage: 28.4+ KB
```

Simultaneously, the 'None' values that were replaced in the last step will be converted to 'NaN', since the column is of float data type.

rs	engine_size	bore	stroke	compression	h
ur	91	3.08	3.15	9.0	
vo	70	NaN	NaN	9.4	
vo	70	NaN	NaN	9.4	
vo	70	NaN	NaN	9.4	
vo	80	NaN	NaN	9.4	
ur	122	3.39	3.39	8.6	
ur	122	3.39	3.39	8.6	

4. **Drop body,engine_type,cylinders columns:** The mentioned columns are dropped using df.drop() function and the resulting dataframe is saved into df2.

```
## Dropping above columns
df2=df.drop(['body','engine_type','cylinders'], axis=1)
```

```
df2.head()
```

	fuel_type	wheel_base	length	width	heights	curb_weight	engine_size	bore	stroke	compression	horse_power	peak_rpm	city_mpg	highway_mpg	price
0	gas	88.6	168.8	64.1	48.8	2548	130	3.47	2.68	9.0	111.0	5000.0	21	27	13495
1	gas	88.6	168.8	64.1	48.8	2548	130	3.47	2.68	9.0	111.0	5000.0	21	27	16500
2	gas	94.5	171.2	65.5	52.4	2823	152	2.68	3.47	9.0	154.0	5000.0	19	26	16500
3	gas	99.8	176.6	66.2	54.3	2337	109	3.19	3.40	10.0	102.0	5500.0	24	30	13950
4	gas	99.4	176.6	66.4	54.3	2824	136	3.19	3.40	8.0	115.0	5500.0	18	22	17450

5. **Drop all NaN values:** All the NaN values in the dataset are dropped using dropna() function. Then to verify all NaN values are removed, isnull().sum() is used to get the sum of null values in each column. This turns out to be. Hence, all null values are dropped.

```
## Dropping rows with NaN value
df2 = df2.dropna()
```

```
df2.isnull().sum() # checking count of Nan/null values in all columns
```

```
fuel_type      0
wheel_base     0
length         0
width          0
heights        0
curb_weight    0
engine_size    0
bore           0
stroke         0
compression    0
horse_power    0
peak_rpm       0
city_mpg       0
highway_mpg    0
price          0
dtype: int64
```

6. **Get dummy variables for fuel_type column:** Pandas.get_dummies() function is used to get the dummy variables for fuel_type column as shown in the following picture.

```
pd.get_dummies(df2['fuel_type'], drop_first=True)
```

	gas
0	True
1	True
2	True
3	True
4	True
...	...
196	True
197	True
198	True
199	False
200	True

195 rows × 1 columns

This will give us the final cleaned data set in variable df2 which can be used in next steps.

2.2 Exploratory Data Analysis

This part of the assignment deals exploratory data analysis for the Auto imports data set. EDA involves the exploration, visualization, and summarization of data to understand its main characteristics or features, detect patterns, identify anomalies, and gain insights into the underlying structure of the dataset.

First, we import all the required libraries which will be used for upcoming analysis: **import seaborn as sns; import pandas as pd; import matplotlib.pyplot as plt; %matplotlib inline**

By using %matplotlib inline, any Matplotlib plots created in code cells of the Jupyter Notebook will be displayed directly below the code cell that generates them. This makes it easier to study the plots and analyze the data represented in the charts.

STEP 1: To get informed about the structure of the data, .shape function is used which returns us the number of rows and columns in the dataset. After seeing the output, it's clear that auto_imports1.csv dataset has 195 rows and 15 columns of data. The columns of the dataset can be called as features or properties of fish that have been studied while collecting the data.

```
## Your EDA should start here
df2.shape # Number of rows and columns
```

```
(195, 15)
```

```
df2.info() #data types in the dataset
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 195 entries, 0 to 200
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   fuel_type    195 non-null    object 
 1   wheel_base   195 non-null    float64
 2   length       195 non-null    float64
 3   width        195 non-null    float64
 4   heights      195 non-null    float64
 5   curb_weight  195 non-null    int64  
 6   engine_size  195 non-null    int64  
 7   bore         195 non-null    float64
 8   stroke       195 non-null    float64
 9   compression  195 non-null    float64
 10  horse_power  195 non-null    float64
 11  peak_rpm     195 non-null    float64
 12  city_mpg     195 non-null    int64  
 13  highway_mpg  195 non-null    int64  
 14  price        195 non-null    int64  
dtypes: float64(9), int64(5), object(1)
memory usage: 24.4+ KB
```

The .describe() function is used to get some common metrics of the dataset which can be used for analysis. The image below shows the output of the describe function.

```
df2.describe() # basic statistics of the dataset
```

	wheel_base	length	width	heights	curb_weight	engine_size	bore	stroke	compression	horse_power	peak_rpm	city_mpg	highway_
count	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	195.0(
mean	98.896410	174.256923	65.886154	53.861538	2559.000000	127.938462	3.329385	3.250308	10.194974	103.271795	5099.487179	25.374359	30.8
std	6.132038	12.476443	2.132484	2.396778	524.715799	41.433916	0.271866	0.314115	4.062109	37.869730	468.271381	6.401382	6.8
min	86.600000	141.100000	60.300000	47.800000	1488.000000	61.000000	2.540000	2.070000	7.000000	48.000000	4150.000000	13.000000	16.0(
25%	94.500000	166.300000	64.050000	52.000000	2145.000000	98.000000	3.150000	3.110000	8.500000	70.000000	4800.000000	19.500000	25.0(
50%	97.000000	173.200000	65.400000	54.100000	2414.000000	120.000000	3.310000	3.290000	9.000000	95.000000	5100.000000	25.000000	30.0(
75%	102.400000	184.050000	66.900000	55.650000	2943.500000	145.500000	3.590000	3.410000	9.400000	116.000000	5500.000000	30.000000	35.0(
max	120.900000	208.100000	72.000000	59.800000	4066.000000	326.000000	3.940000	4.170000	23.000000	262.000000	6600.000000	49.000000	54.0(

It gives information about the mean, standard deviation, minimum, maximum, Quartiles(25%,50%,75%) of the different column values(features).

STEP 2: **Checking for missing/null values and duplicates**. We can use isnull().sum() function to check the count of null values. This returns the count of null values, but in this case, it shows 0 for all columns. So, it's safe to say there are no null values in the fish dataset and cleaning of null values is not required.

```
df2.isnull().sum() # checking if any null values exist.

fuel_type       0
wheel_base      0
length          0
width           0
heights         0
curb_weight     0
engine_size     0
bore            0
stroke          0
compression     0
horse_power     0
peak_rpm        0
city_mpg        0
highway_mpg     0
price           0
dtype: int64
```

Since we are exploring the dataset, we can check what are the unique fuel types that are present in the data. Here, .unique() function is run on the fuel_type column to get the different fuel type. Hence, we have discovered that the data contains 2 types of fuel.

```
df2.fuel_type.unique() # types of fuel in the dataset

array(['gas', 'diesel'], dtype=object)
```

Next step is to check for duplicates of the data in the dataset. Here we will be using two functions to verify if duplicates exist or not:

Drop_duplicates method: In this method, the code will execute and drop rows which are duplicates of other data and return the number of rows remaining in the dataset. Hence, we can know if there were any duplicates by checking if there is difference in number of rows after executing this command. In our case, there are 3 duplicates since the command execution returned 192 rows and 15 columns.

```
df2.drop_duplicates() # dropping dupliacte rows
```

	fuel_type	wheel_base	length	width	heights	curb_weight	engine_size	bore	stroke	compression	horse_power	peak_rpm	city_mpg	highway_mpg	price
0	gas	88.6	168.8	64.1	48.8	2548	130	3.47	2.68	9.0	111.0	5000.0	21	27	13495
1	gas	88.6	168.8	64.1	48.8	2548	130	3.47	2.68	9.0	111.0	5000.0	21	27	16500
2	gas	94.5	171.2	65.5	52.4	2823	152	2.68	3.47	9.0	154.0	5000.0	19	26	16500
3	gas	99.8	176.6	66.2	54.3	2337	109	3.19	3.40	10.0	102.0	5500.0	24	30	13950
4	gas	99.4	176.6	66.4	54.3	2824	136	3.19	3.40	8.0	115.0	5500.0	18	22	17450
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
196	gas	109.1	188.8	68.9	55.5	2952	141	3.78	3.15	9.5	114.0	5400.0	23	28	16845
197	gas	109.1	188.8	68.8	55.5	3049	141	3.78	3.15	8.7	160.0	5300.0	19	25	19045
198	gas	109.1	188.8	68.9	55.5	3012	173	3.58	2.87	8.8	134.0	5500.0	18	23	21485
199	diesel	109.1	188.8	68.9	55.5	3217	145	3.01	3.40	23.0	106.0	4800.0	26	27	22470
200	gas	109.1	188.8	68.9	55.5	3062	141	3.78	3.15	9.5	114.0	5400.0	19	25	22625

192 rows × 15 columns

STEP 3: **To see if the dataset is balanced**, we will check how many data points belong to certain fish species. The .value_counts() function will be used to count the number of fuel types for the column fuel_type. The table below shows that the data points for each fuel aren't equal, hence the dataset is not balanced.

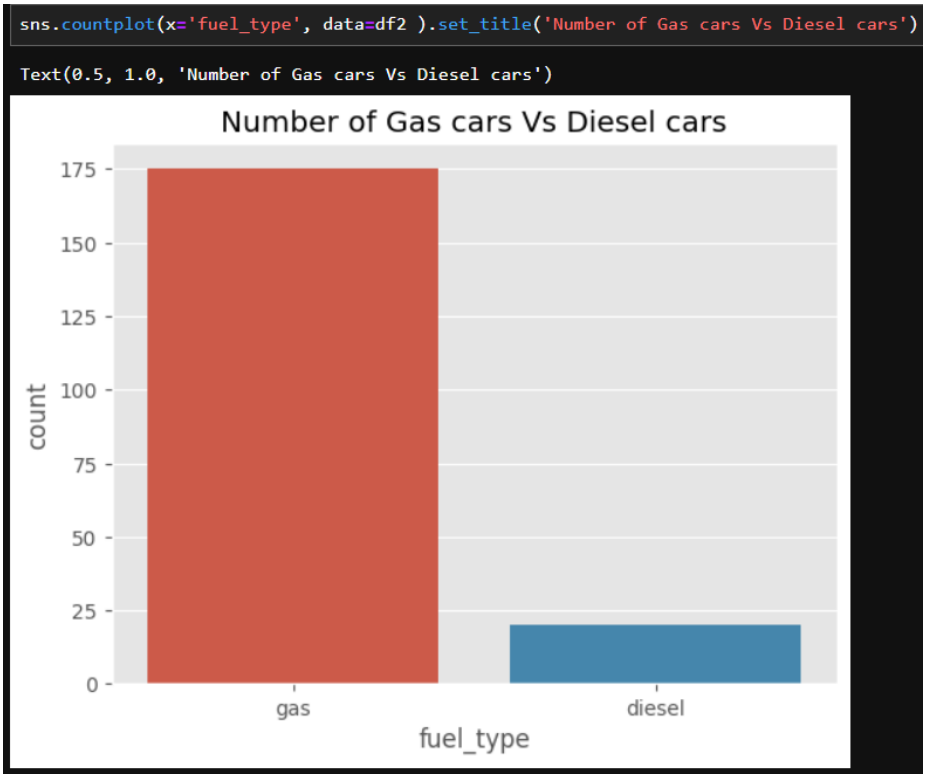
```
print('Number of gas vehicel data: ',df2['fuel_type'].value_counts()['gas'])# chec
print('Number of diesel vehicel data: ',df2['fuel_type'].value_counts()['diesel'])
```

```
Number of gas vehicel data:  175
Number of diesel vehicel data:  20
```

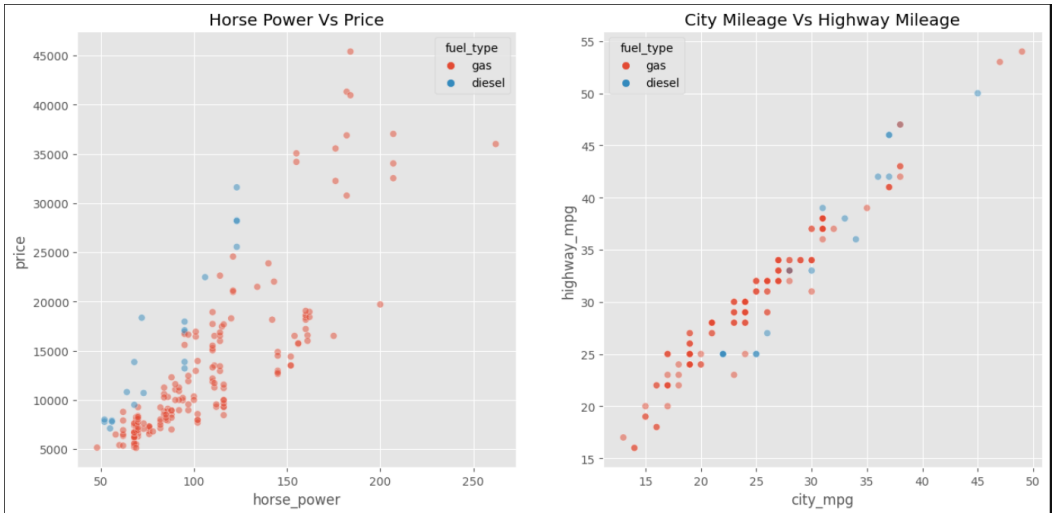
The same data can be visualized using seaborn countplot(). By using plots, it becomes easier to spot trends, outliers, abnormalities, and aids in the analysis process.

```
sns.countplot(x='fuel_type', data=df2 ).set_title('Number of Gas cars Vs Diesel cars')
```

```
Text(0.5, 1.0, 'Number of Gas cars Vs Diesel cars')
```

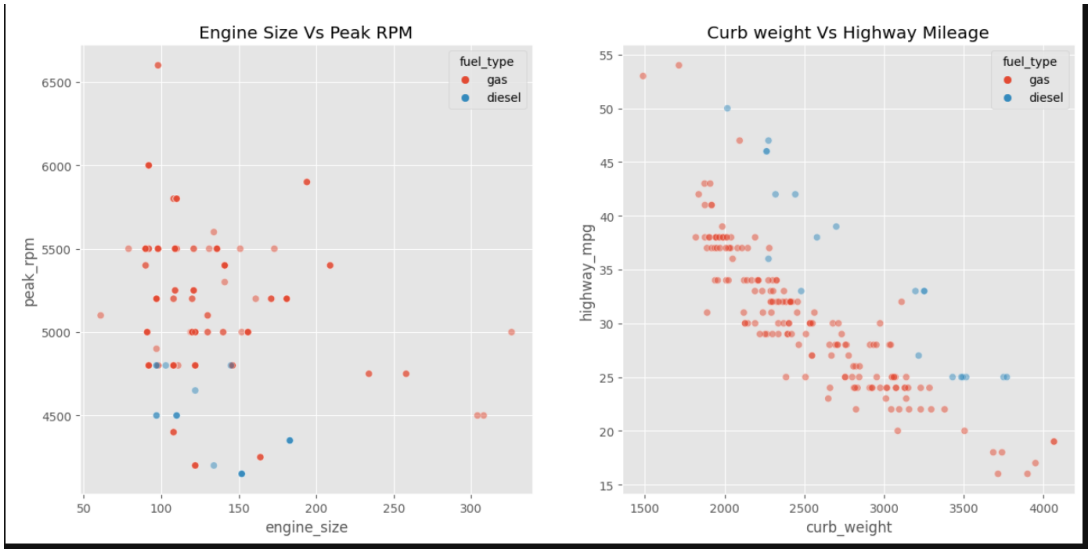


STEP 4: Finding Relations between the features of the data set. This will help us understand the trends in the data and in some cases predict features which are unknown. Here, we will try to see what the relation is between horsepower, price, and other features. For this process we will use scatter plot.



From the above graph, it can be concluded that diesel vehicles generally have less horsepower and are less expensive when compared to gas operated vehicles. But we cannot be sure, since this data is unbalanced not enough data is present. It can be said that gas operated vehicles are available in various price brackets. The relation between city mileage and highway mileage is mostly linear.

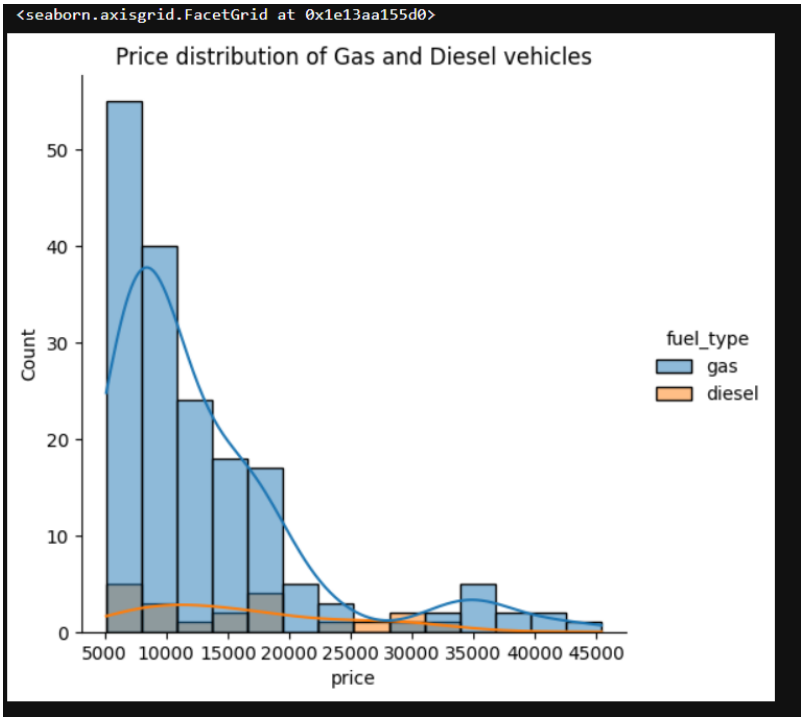
Similarly, other plots were plotted to analyze relationships between features.



From the above plot, we see that engine size of most cars (gas/diesel) are between 50-175 units and gas vehicles have higher rpm compared to diesel vehicles. The curb weight and mileage are inversely related to each other. This means that, with an increase in weight, mileage decreases.

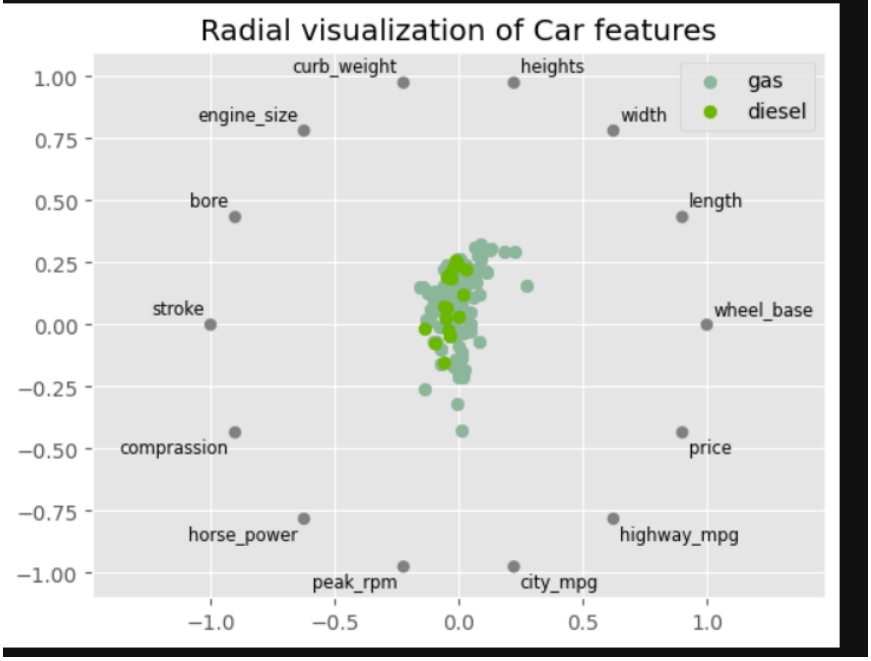
Following is distribution plot for the price of Gas and diesel cars. It is seen that most gas cars cost around 10,000 units. From the limited available data, diesel cars seem to have equal representation in all cost ranges.

```
<seaborn.axisgrid.FacetGrid at 0x1e13aa155d0>
```



From the plot below, we can say maximum mileage and wheelbase belong to gas operated cars. The radial plot is used for analysis.

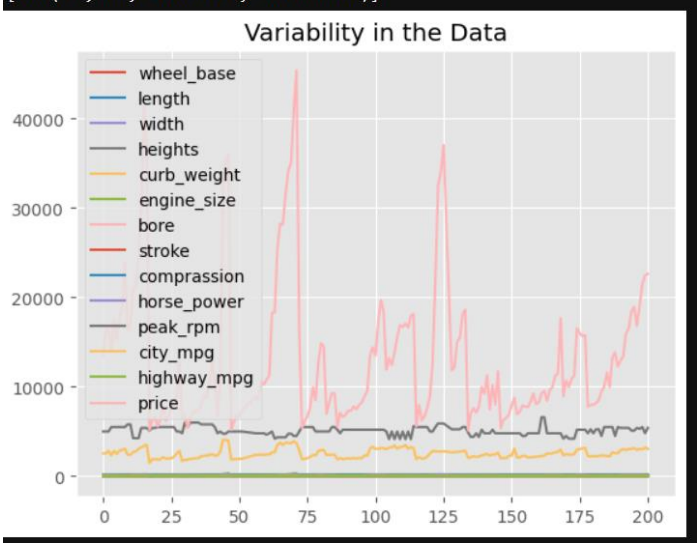
```
import pandas.plotting as pdplt
pdplt.radviz(df2, 'fuel_type').set(title='Radial visualization of Car features')

[Text(0.5, 1.0, 'Radial visualization of Car features')]
```



To check variability of the features in the dataset, the following graph is plotted. It is observed that price is the most varying feature in the dataset.

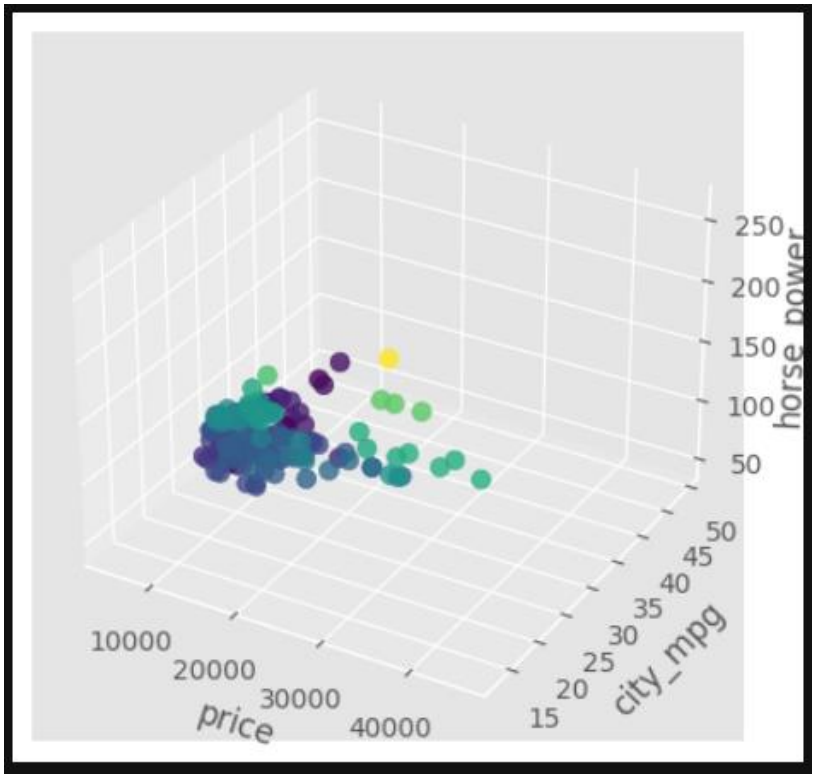
```
# visualizing Variability
df2.plot().set(title='Variability in the Data')
```

```
[Text(0.5, 1.0, 'Variability in the Data')]
```

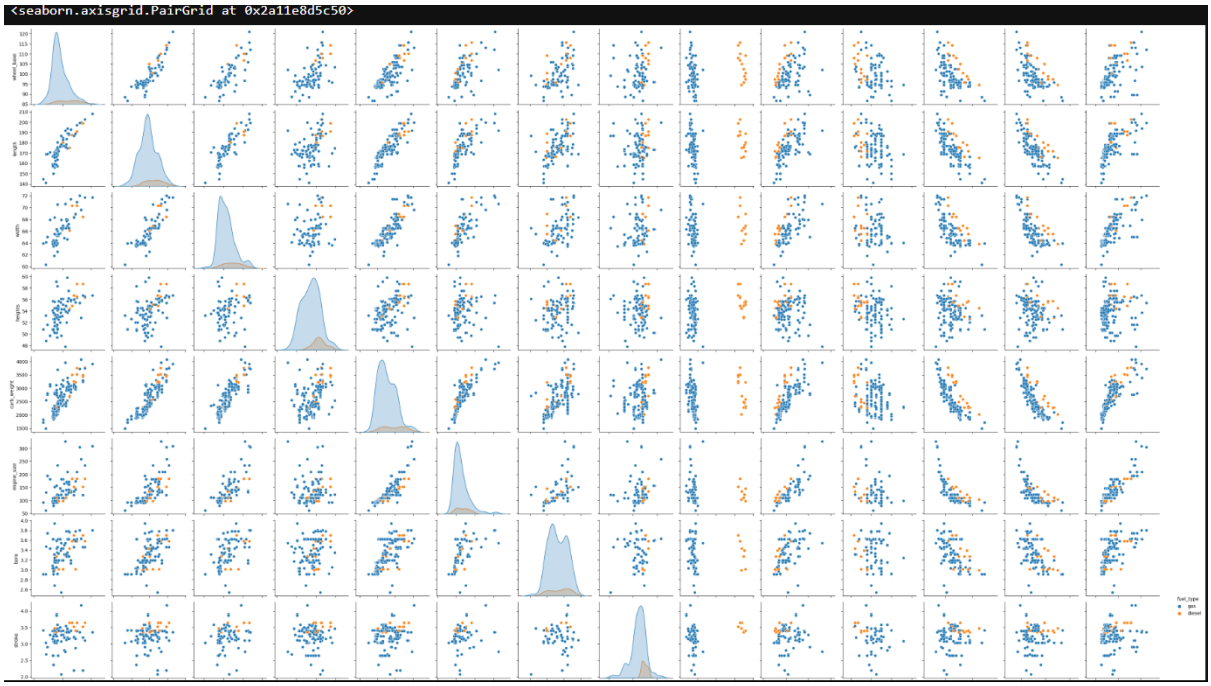


The following graph shows a 3d representation of price vs city mileage vs horsepower of the car.

```
x=df2['price']
y=df2['city_mpg']
z=df2['horse_power']
fig = plt.figure()
ax = plt.axes(projection='3d')
ax.set_xlabel('price')
ax.set_ylabel('city_mpg')
ax.set_zlabel('horse_power',labelpad=-0.4)
ax.scatter3D(x, y, z, c=z, cmap='viridis',s=50, alpha=0.8);
```



A pair plot was plotted for the dataset, but there are many attributes hence the plots are very small in size.



STEP 5: **Correlations-** It is a statistical measure that quantifies the degree to which two variables are related or associated with each other. Correlation coefficients range from -1 to 1. A correlation of 1 indicates a perfect positive linear relationship, where both variables move together. A correlation of 0 suggests no linear relationship.

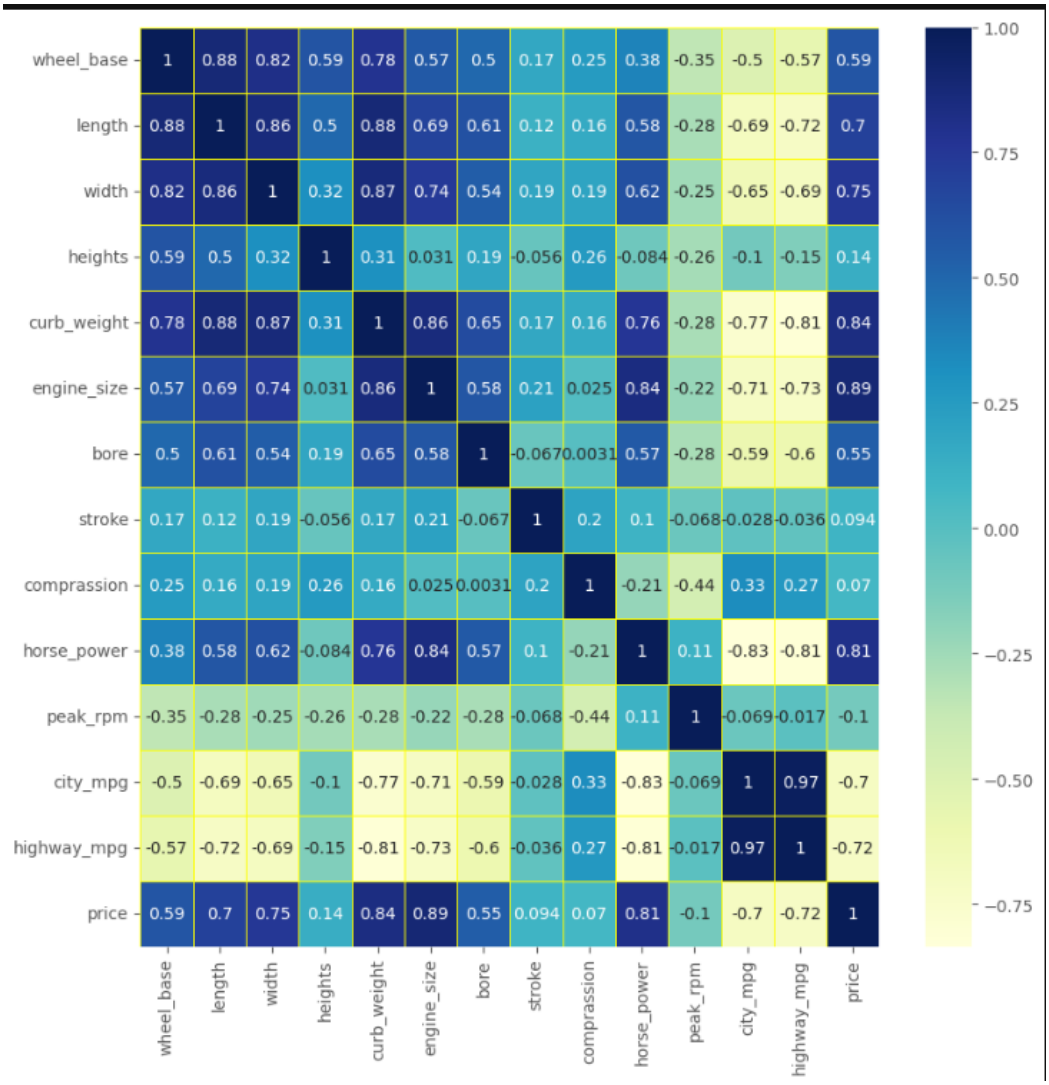
For this we will use the inbuilt Pearson correlation method.

```
df2_corr.corr(method='pearson')
```

	wheel_base	length	width	heights	curb_weight	engine_size	bore	stroke	compression	horse_power	peak_rpm	city_mpg	highway_m
wheel_base	1.000000	0.879222	0.819009	0.592500	0.782720	0.569704	0.498228	0.171722	0.247730	0.375541	-0.352331	-0.499126	-0.5663
length	0.879222	1.000000	0.858084	0.496218	0.881665	0.687479	0.609437	0.118664	0.160172	0.583813	-0.280986	-0.689660	-0.7193
width	0.819009	0.858084	1.000000	0.315834	0.867315	0.740320	0.544311	0.186432	0.190997	0.616779	-0.251627	-0.647099	-0.6922
heights	0.592500	0.496218	0.315834	1.000000	0.307732	0.031286	0.189283	-0.055525	0.261160	-0.084412	-0.264078	-0.102367	-0.1511
curb_weight	0.782720	0.881665	0.867315	0.307732	1.000000	0.857573	0.645806	0.172785	0.155382	0.760285	-0.278944	-0.772171	-0.8127
engine_size	0.569704	0.687479	0.740320	0.031286	0.857573	1.000000	0.583091	0.211989	0.024617	0.842691	-0.219008	-0.710624	-0.7321
bore	0.498228	0.609437	0.544311	0.189283	0.645806	0.583091	1.000000	-0.066793	0.003057	0.568527	-0.277662	-0.591950	-0.6000
stroke	0.171722	0.118664	0.186432	-0.055525	0.172785	0.211989	-0.066793	1.000000	0.199882	0.100040	-0.068300	-0.027641	-0.0364
compression	0.247730	0.160172	0.190997	0.261160	0.155382	0.024617	0.003057	0.199882	1.000000	-0.214401	-0.444582	0.331413	0.2679
horse_power	0.375541	0.583813	0.616779	-0.084412	0.760285	0.842691	0.568527	0.100040	-0.214401	1.000000	0.105654	-0.834117	-0.8129
peak_rpm	-0.352331	-0.280986	-0.251627	-0.264078	-0.278944	-0.219008	-0.277662	-0.068300	-0.444582	0.105654	1.000000	-0.069493	-0.0169
city_mpg	-0.499126	-0.689660	-0.647099	-0.102367	-0.772171	-0.710624	-0.591950	-0.027641	0.331413	-0.834117	-0.069493	1.000000	0.9723
highway_mpg	-0.566355	-0.719324	-0.692220	-0.151188	-0.812710	-0.732138	-0.600040	-0.036453	0.267941	-0.812917	-0.016950	0.972350	1.0000
price	0.585793	0.695331	0.754273	0.138291	0.835729	0.888942	0.546873	0.093746	0.069500	0.811027	-0.104333	-0.702685	-0.7155

This table is not the best way to study correlation. So, heat-map plot will be used to get more insights regarding correlation between all features of the fish.

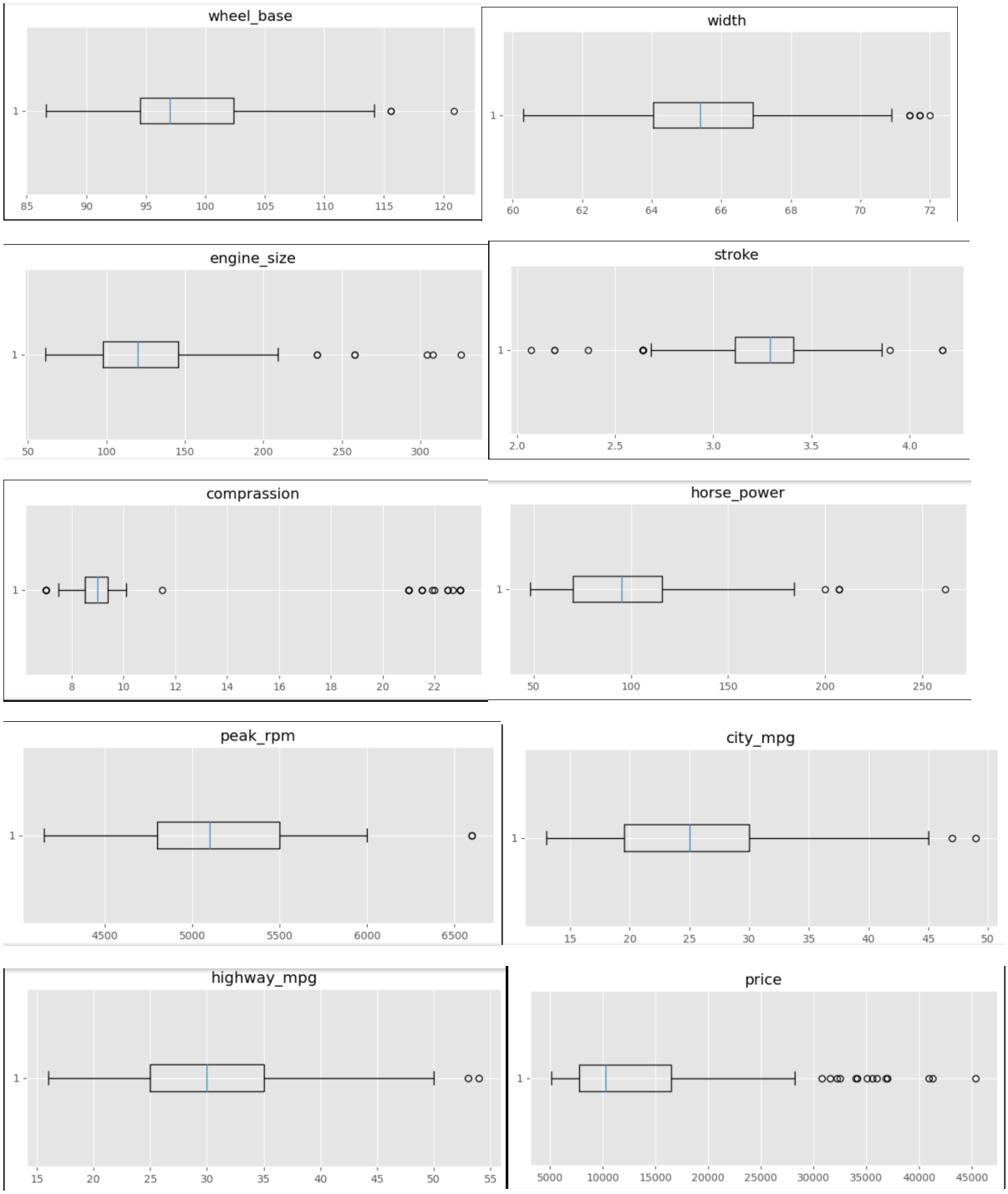
- sns.heatmap(df2_corr.corr(method='pearson'),annot=True,cmap="YlGnBu",linewidths=0.5,linecolor='yellow')



From the above heatmap, we can deduce that engine size and price of the cars are very much positively correlated. Meaning, if there is an increase in engine size of the car, the price of the car will also increase and vice versa. Similarly, we can see that curb weight and price of the fish are also positively correlated.

On the other hand, it is seen that there is a negative relationship between mileage and weight of car, implying increase in weight will result in decrease in mileage. It is also evident that there is virtually no relation between stroke and horsepower.

STEP 6: Finding and treating outliers. We will use boxplot to determine which columns(features) have outliers and treat these outliers. From the plot, the following features have been identified for having outliers.



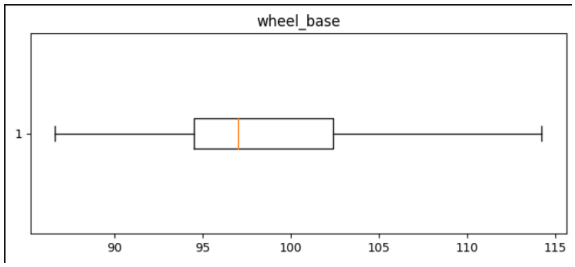
<u>Finding the outliers in the data- IQR METHOD</u>:

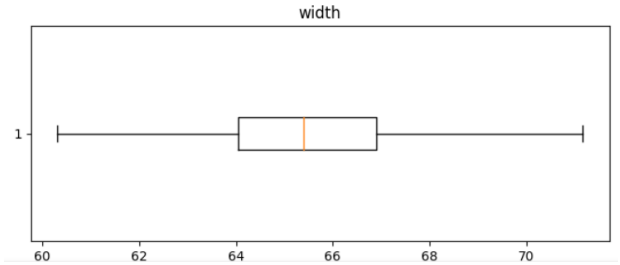
In this method, we will find the first (Q1) and third (Q3) quartiles of each feature and find the IQR using the formula IQR= Q3-Q1. Then we set the upper and lower limits of the values that will help in filtering data.

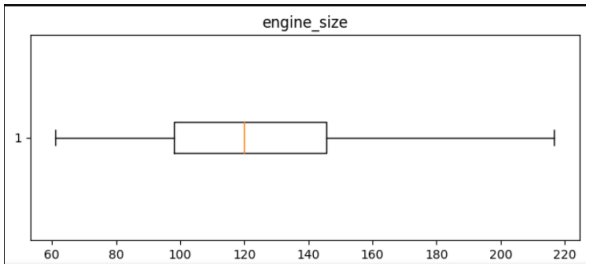
Next, we will use capping and filtering function using loc (label-based indexing). The code is iterated over the dataframe's columns to make the code cleaner and simpler.

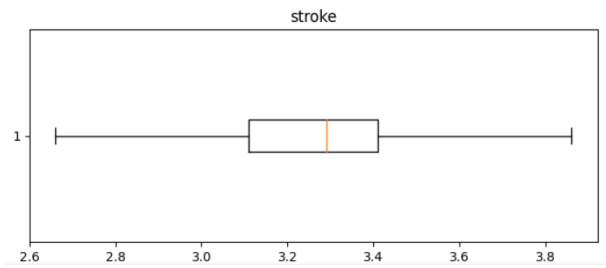
```python
# Finding quartiles and IQR
for i in df3.columns:
    q1= df3[i].quantile(0.25) #First quartile
    q3=df3[i].quantile(0.75)# 3rd quartile
    iqr=q3 - q1  # finding IQR
    upper= q3 + (1.5*iqr) #upper limit formula
    lower= q1 - (1.5*iqr) #lower limit formula
    print(upper,lower)
    #capping and Filterting data based in upper and lower limits
    df3.loc[(df3[i]> upper), i ]=upper
    df3.loc[(df3[i]< lower), i ]=lower
```

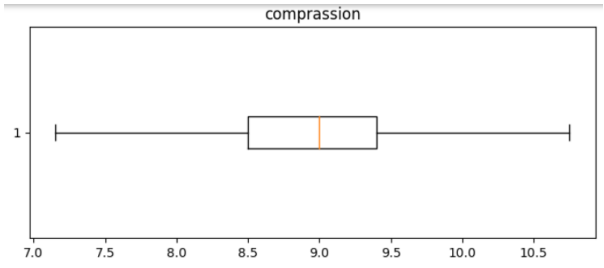
Then we plot the same data again to view updated box plots where outliers are removed.

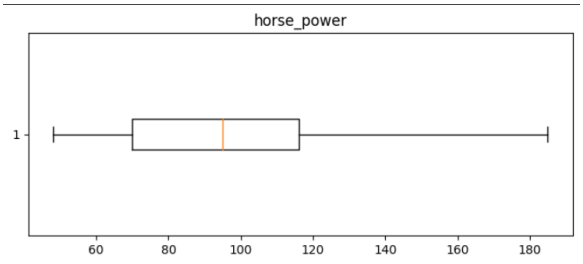


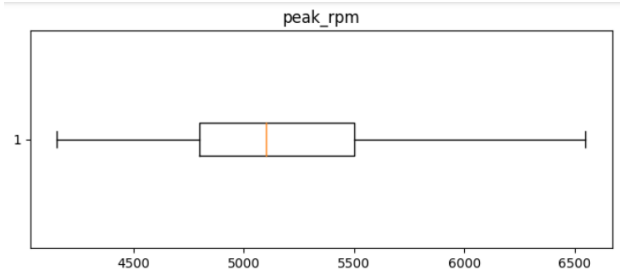


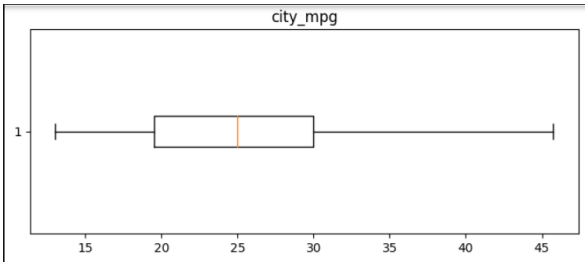


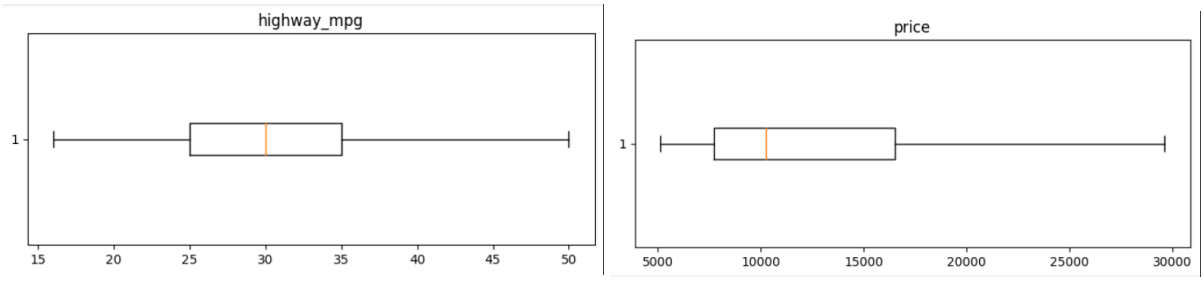












In the above plot we observe that all the outliers have been removed. Outliers can have a significant impact on the performance of statistical models. If a model is trained on this data, by removing outliers the model can be trained on a cleaner, more representative subset of the data, leading to better predictive accuracy and generalization. In situations where outliers are the result of data entry errors or measurement errors, removing them can help maintain the integrity and quality of the dataset. Also, removing outliers can reduce the risk of overfitting and improve a model's ability to generalize to new data.

3. Multiple regression Analysis

First, the dependent variable columns i.e., price is set as y and the independent variables are set to X for creating a model.

```
X=df3.drop(columns=['price']) # dropping price column since it is dependent variable and wont be used for training.
y=df3['price'] #target variable is price column
```

Intercept values give an estimate of the dependent variable when all independent variables are zero.

The dataset can be split into test and train set in 70:30 ratio. This is used for prediction, but we aren't doing that in this case.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 0)# Train-test split is 70-30.
```

Next, the model 1 is fitted by calling the following code snippet.

```
import statsmodels.api as sm
import statsmodels.formula. api as smf
model1 =smf.ols(formula="price ~ wheel_base +length+width+heights+curb_weight+engine_size+bore+stroke+comprassion+horse_power+peak_rpm+city_mpg+highway_m
# model1=sm.OLS(y_train,X_train).fit()
```

The summary of the model 1 is obtained, and further analysis is done using these values.

```
model1.summary(alpha=0.1)
```

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.863
Model:	OLS	**Adj. R-squared:**	0.854
Method:	Least Squares	**F-statistic:**	87.95
Date:	Sun, 22 Oct 2023	**Prob (F-statistic):**	6.56e-71
Time:	19:24:49	**Log-Likelihood:**	-1801.8
No. Observations:	195	**AIC:**	3632.
Df Residuals:	181	**BIC:**	3677.
Df Model:	13		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.05	0.95]
Intercept	-2.665e+04	6583.768	-4.047	0.000	-3.75e+04	-1.58e+04
wheel_base	99.1414	86.256	1.149	0.252	-43.467	241.750
length	-92.9557	47.114	-1.973	0.050	-170.849	-15.062
width	619.4968	213.867	2.897	0.004	265.907	973.087
heights	166.2881	114.055	1.458	0.147	-22.281	354.858
curb_weight	3.7947	1.398	2.715	0.007	1.484	6.105
engine_size	93.8753	17.696	5.305	0.000	64.619	123.132
bore	-1957.0100	1015.432	-1.927	0.056	-3635.839	-278.181
stroke	-3151.9768	783.571	-4.023	0.000	-4447.467	-1856.486
compression	756.6830	333.926	2.266	0.025	204.598	1308.767
horse_power	42.4487	16.767	2.532	0.012	14.728	70.170
peak_rpm	1.0672	0.564	1.893	0.060	0.135	1.999
city_mpg	-242.0323	146.915	-1.647	0.101	-484.929	0.864

Variance of this model was also calculated using the code below.

```
# Get the residuals from the model
residuals1 = model1.resid
# Calculate the variance of the residuals
residual_variance1 = residuals1.var()
print("Variance of Residuals in Model 1:", residual_variance1) #variance
```

```
Variance of Residuals in Model 1: 6242654.592851117
```

From the OLS summary of model 1 we can arrive at the following conclusions:

1. The intercept is the average price of the car when all other features are 0. It is Intercept -5.329e+04.
2. From the OLS summary, if the t value can be greater or less than 0. If we take the confidence level to be 90%, then the features which have a P value less than 0.1 are considered

statistically significant. So, in this case wheel_base,heights and city_mpg columns are less significant. So, 11 columns are statistically significant.

3. The variance of model 1 was found to be 6242654.59.
4. The coefficient of determination is represented as R-squared in the OLS summary. If the linear regression model fits well, the R-squared value will be closer to 1. In this case, R-squared values is 0.863 which means the model is a good fit. If there is a lot of difference between R-squared value and adjusted R-squared value, this means that the dataset has features that are irrelevant.
5. F-statistic is used to analyze the overall statistical significance of the model. If the F-statistic value is large and P value i.e., Prob(F-statistics) is close to zero, then we can reject Null hypothesis. Hence, it can be said, there is a relationship between the dependent variable and the other features in the dataset. In our case F-statistics value is 87.95 and P value is 6.56e-71. Therefore, this is an acceptable model.

For the second model, the statistically less significant columns are dropped, and the model is trained again, and OLS summary is obtained.

```
model2 =smf.ols(formula="price ~ length+width+curb_weight+engine_size+bore+stroke+compression+horse_power+peak_rpm+highway_mpg+fuel_type_gas", data=df3).
# model2=sm.OLS(y2_train,X2_train).fit()
model2.summary(alpha=0.1)
```

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.858
Model:	OLS	**Adj. R-squared:**	0.851
Method:	Least Squares	**F-statistic:**	111.4
Date:	Sun, 22 Oct 2023	**Prob (F-statistic):**	1.58e-72
Time:	19:29:48	**Log-Likelihood:**	-1805.3
No. Observations:	195	**AIC:**	3633.
Df Residuals:	184	**BIC:**	3669.
Df Model:	10		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.05	0.95]
Intercept	-2.36e+04	6082.109	-3.881	0.000	-3.37e+04	-1.36e+04
length	-21.5676	37.281	-0.579	0.564	-83.200	40.065
width	618.5090	200.911	3.079	0.002	286.368	950.651
curb_weight	4.1649	1.365	3.052	0.003	1.909	6.421
engine_size	90.3298	17.542	5.149	0.000	61.330	119.330
bore	-1799.9144	1021.541	-1.762	0.080	-3488.703	-111.126
stroke	-3138.9116	779.065	-4.029	0.000	-4426.845	-1850.978
compression	611.2012	323.255	1.891	0.060	76.803	1145.599
horse_power	37.7073	16.171	2.332	0.021	10.974	64.441
peak_rpm	1.1792	0.564	2.091	0.038	0.247	2.111
highway_mpg	58.7025	74.889	0.784	0.434	-65.103	182.508
fuel_type_gas	-2.36e+04	6082.109	-3.881	0.000	-3.37e+04	-1.36e+04

Omnibus:	6.374	**Durbin-Watson:**	0.652
Prob(Omnibus):	0.041	**Jarque-Bera (JB):**	10.049
Skew:	-0.071	**Prob(JB):**	0.00658
Kurtosis:	4.103	**Cond. No.**	7.88e+18

The variance of model 2 is also calculated.

```
# Get the residuals from the model
residuals2 = model2.resid
# Calculate the variance of the residuals
residual_variance2 = residuals2.var()
print("Variance of Residuals in model 2:", residual_variance2) # variance
```

```
Variance of Residuals in model 2: 6474684.987766562
```

From the OLS summary of model 2 we can arrive at the following conclusions:

1. The intercept is the average price of the car when all other features are 0. It increased to Intercept -4.721e+04.
2. So, in this case length and highway_mpg columns are less significant. Hence 9 columns are statistically significant.
3. The variance of model 2 was found to be 6474684.98.
4. In this case, R-squared values is 0.858 that is minutely different from model 1 and the model is a good fit. If there is a lot of difference between R-squared value and adjusted R-squared value, this means that the dataset has features that are irrelevant.
5. F-statistic is used to analyze the overall statistical significance of the model. If the F-statistic value is large and P value i.e., Prob(F-statistics) is close to zero, then we can reject Null hypothesis. Hence, it can be said, there is a relationship between the dependent variable and the other features in the dataset. In our case F-statistics value is 111.4 and P values is 1.58e-72. Therefore, this is a good model. There might be a possibility where model two is better than model 1 because of the F-statistic value.

The two models are compared using ANOVA as depicted in the image below.

```
#comparing the two models
anova = sm.stats.anova_lm(model2, model1, test="F", typ=1)
print(anova)
```

	df_resid	ssr	df_diff	ss_diff	F	Pr(>F)
0	184.0	1.256089e+09	0.0	NaN	NaN	NaN
1	181.0	1.211075e+09	3.0	4.501390e+07	2.242502	0.084912

Since the P-value is 0.084912, This Is less than the confidence level of 0.1.

So, it may be concluded that **null hypothesis can be rejected**.

Null hypothesis is that all features can be used to predcit the price. But it was found that wheel_base,heights,city_mpg,length and highway_mpg were not statistically significant for prediction of dependent variable(price).¶

Furthermore, a multicollinearity check was done to check if the independent variables were related to each other.

```
#Checking for multicollinearity
from statsmodels.stats.outliers_influence import variance_inflation_factor
# X should be the matrix of your independent variables
vif = pd.DataFrame()
vif["Variable"] = X2.columns
vif["VIF"] = [variance_inflation_factor(X2.values, i) for i in range(X2.shape[1])]
```

```
vif.round(2)
```

	Variable	VIF
0	const	4226.69
1	length	6.15
2	width	5.01
3	curb_weight	14.57
4	engine_size	10.30
5	bore	2.19
6	stroke	1.32
7	comprassion	2.13
8	horse_power	9.37
9	peak_rpm	1.97
10	highway_mpg	7.19

From the above results, it can be concluded that there is multicollinearity between the features of the dataset.

4.RESOURCES

- Pandas: https://pandas.pydata.org/docs/reference/index.html
- Numpy: https://numpy.org/doc/stable/reference/routines.html
-
- Matplotlib: https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.plot.html
- Seaborn: https://seaborn.pydata.org/generated/seaborn.displot.html
- https://www.kdnuggets.com/2022/06/plotting-data-visualization-data-science.html#:~:text=Geometric%20Component%3AHere%20is%20where,heatmaps%2C%20pie%20charts%2C%20etc.
- https://chartio.com/learn/charts/essential-chart-types-for-data-visualization/
- https://www.geeksforgeeks.org/how-to-plot-a-normal-distribution-with-matplotlib-in-python/
- https://www.statology.org/seaborn-heatmap-size/
- https://python-graph-gallery.com/92-control-color-in-seaborn-heatmaps/