Meritorium Project-1

Importing Data

```
import pandas as pd
import numpy as np
```

```
# importing data
#Note :Because the data does not include headers, we can add an argument headers

path = r"https://archive.ics.uci.edu/ml/machine-learning-databases/autos/imports-
am_df = pd.read_csv(path, header=None)
```

```
am_df
```

	0	1	2	3	4	5	6	7	8	9	...	16	17	18	19	20	21	22	23	24	25
0	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	111	5000	21	27	13495
1	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	111	5000	21	27	16500
2	1	?	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	9.0	154	5000	19	26	16500
3	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	10.0	102	5500	24	30	13950
4	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.40	8.0	115	5500	18	22	17450

```
## NOTE : Point to Remember : pandas automatically set the header by an integer. To better describe our data we can introduce a heade
```

```
headers = ["symboling","normalized-losses","make","fuel-type","aspiration", "num-of-doors","body-style","drive-wheels","engine-locati
print("headers\n", headers)
```

```
headers
 ['symboling', 'normalized-losses', 'make', 'fuel-type', 'aspiration', 'num-of-doors', 'body-style', 'drive-wheels', 'engine-lo
```

```
am_df.columns = headers
```

```
am_df
```

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	s
0	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	
1	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	
2	1	?	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	
3	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	
4	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
200	-1	95	volvo	gas	std	four	sedan	rwd	front	109.1	...	141	mpfi	3.78	
201	-1	95	volvo	gas	turbo	four	sedan	rwd	front	109.1	...	141	mpfi	3.78	
204	-1	95	volvo	gas	turbo	four	sedan	rwd	front	109.1	...	141	mpfi	3.78	

Preprocessing

```
#checking type of data

type(am_df)
```

```
pandas.core.frame.DataFrame
```

```
#checking the datatype for ? using index location

type(am_df.iloc[0,1])
```

```
str
```

```
## NOTE : Relacing all '?' with NaN type

am_df=am_df.replace(['?'],np.nan)



#The info() method prints information about the DataFrame.
#The information contains the number of columns, column labels, column data types, memory usage, range index, and the number of cells

am_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 26 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   symboling          205 non-null    int64  
 1   normalized-losses  164 non-null    object 
 2   make               205 non-null    object 
 3   fuel-type          205 non-null    object 
 4   aspiration         205 non-null    object 
 5   num-of-doors       203 non-null    object 
 6   body-style         205 non-null    object 
 7   drive-wheels       205 non-null    object 
 8   engine-location    205 non-null    object 
 9   wheel-base         205 non-null    float64
 10  length             205 non-null    float64
 11  width              205 non-null    float64
 12  height             205 non-null    float64
 13  curb-weight        205 non-null    int64  
 14  engine-type        205 non-null    object 
 15  num-of-cylinders   205 non-null    object 
 16  engine-size        205 non-null    int64  
 17  fuel-system        205 non-null    object 
 18  bore               201 non-null    object 
 19  stroke             201 non-null    object 
 20  compression-ratio  205 non-null    float64
 21  horsepower         203 non-null    object 
 22  peak-rpm           203 non-null    object 
 23  city-mpg           205 non-null    int64  
```

```
 24  highway-mpg        205 non-null    int64
 25  price              201 non-null    object
dtypes: float64(5), int64(5), object(16)
memory usage: 41.8+ KB
```

```
#coverting the object datatype to numeric datatype using the information we get from above code

am_df[['price', 'peak-rpm', 'horsepower', 'stroke', 'bore', 'normalized-losses']] = am_df[['price', 'peak-rpm', 'horsepower', 'stroke
```

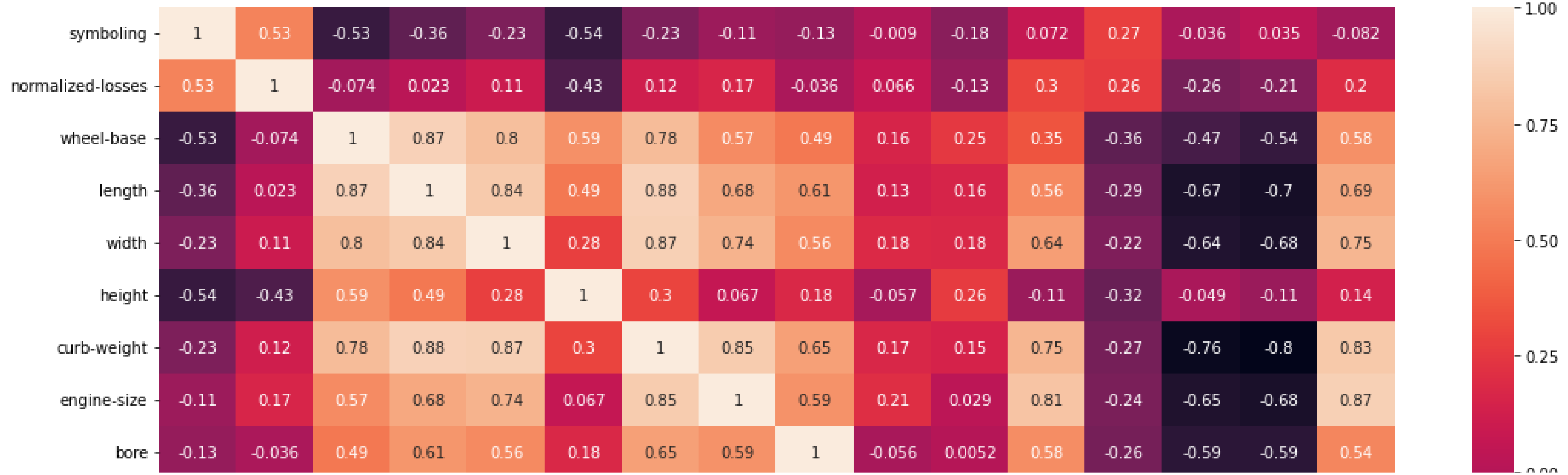
```
am_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 26 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   symboling          205 non-null    int64
 1   normalized-losses  164 non-null    float64
 2   make               205 non-null    object
 3   fuel-type          205 non-null    object
 4   aspiration         205 non-null    object
 5   num-of-doors       203 non-null    object
 6   body-style         205 non-null    object
 7   drive-wheels       205 non-null    object
 8   engine-location    205 non-null    object
 9   wheel-base         205 non-null    float64
 10  length             205 non-null    float64
 11  width              205 non-null    float64
 12  height             205 non-null    float64
 13  curb-weight        205 non-null    int64
 14  engine-type        205 non-null    object
 15  num-of-cylinders   205 non-null    object
 16  engine-size        205 non-null    int64
 17  fuel-system        205 non-null    object
 18  bore               201 non-null    float64
 19  stroke             201 non-null    float64
 20  compression-ratio  205 non-null    float64
 21  horsepower         203 non-null    float64
 22  peak-rpm           203 non-null    float64
```

```
 23  city-mpg           205 non-null    int64  
 24  highway-mpg        205 non-null    int64  
 25  price              201 non-null    float64
dtypes: float64(11), int64(5), object(10)
memory usage: 41.8+ KB
```

```python
list_num = [j for j in am_df.columns if am_df[j].dtype != 'object']     ##List of numberic columns headers
list_cat = [j for j in am_df.columns if am_df[j].dtype == 'object']     ##List of category column headers


#using Heatmap to see the relation between different variables present in one DataFrame and it can be helpful in data understanding

import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(18,10))
sns.heatmap(am_df[list_num].corr(), annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2e76c92940>
```



Maximum Null Values are in Normalized-losses and its co-relation with the target variable (price) is also weak (0.2 < 0.5), Hence we can drop this column. But, for now proceeding with same data.

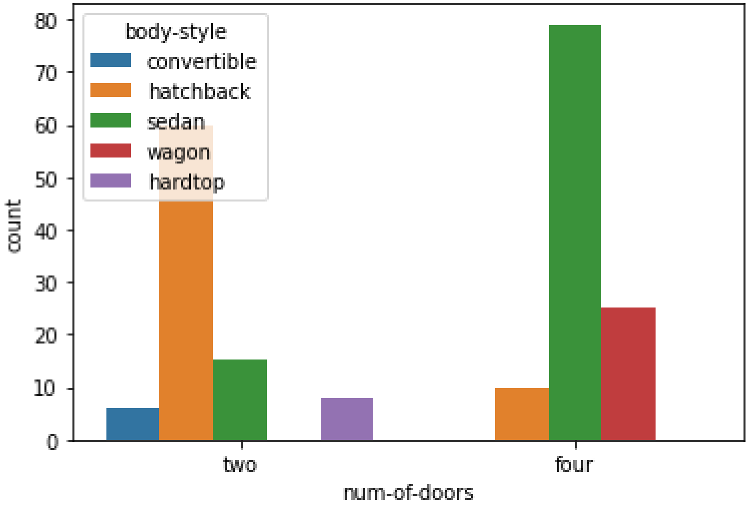

```
am_df[list_cat].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 10 columns):
 #   Column            Non-Null Count  Dtype 
---  ------            --------------  ----- 
 0   make              205 non-null    object
 1   fuel-type         205 non-null    object
 2   aspiration        205 non-null    object
 3   num-of-doors      203 non-null    object
 4   body-style        205 non-null    object
 5   drive-wheels      205 non-null    object
 6   engine-location   205 non-null    object
 7   engine-type       205 non-null    object
 8   num-of-cylinders  205 non-null    object
 9   fuel-system       205 non-null    object
```

```
dtypes: object(10)
memory usage: 16.1+ KB
```

```
import seaborn as sns
sns.countplot(data=am_df, x='num-of-doors', hue='body-style')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2e6c5863d0>
```



Replacing all the Numberic missing values with "Mean", because all numeric missing values are continous type and Categorical missing value with "Mode" (most frequent). Ideally replacement should be done based on data understanding. Ex: num-of-doors missing value should be done based on co-relation with body-style as shown in above figure.

```
am_df['num-of-doors'].mode()
```

```
0    four
dtype: object
```

```
#Filling misssing values

for j in am_df.columns:
```

```
  if am_df[j].dtype != 'object':
    #am_df[j] = am_df[j].fillna(am_df[j].mode())
  #else:
    am_df[j] = am_df[j].fillna(am_df[j].mean())
```

```
am_df['num-of-doors'].value_counts()
```

```
     four    114
     two      89
     Name: num-of-doors, dtype: int64
```

```
am_df.loc[:,'num-of-doors'].fillna('four', inplace=True)
```

```
am_df.isnull().any()
```

```
     symboling            False
     normalized-losses    False
     make                 False
     fuel-type            False
     aspiration           False
     num-of-doors         False
     body-style           False
     drive-wheels         False
     engine-location      False
     wheel-base           False
     length               False
     width                False
     height               False
     curb-weight          False
     engine-type          False
     num-of-cylinders     False
     engine-size          False
     fuel-system          False
     bore                 False
     stroke               False
     compression-ratio    False
     horsepower           False
```

```
    peak-rpm             False
    city-mpg             False
    highway-mpg          False
    price                False
    dtype: bool
```

```
'''
NOTE :

am_df.mean()
am_df.mode()
am_df.median()

this will give mean and median for dtype- numeric and mode for dtype - numeric and object
but this will give type error from the next revision

'''
```

```
    '\nNOTE : \n\nam_df.mean()\nam_df.mode()\nam_df.median()\n\nthis will give mean and median for dtype- numeric and mode for dtyp
    e - numeric and object\nbut this will give type error from the next revision\n\n'
```

```
am_df[list_num].mean()      #provide mean for all numeric dtypes
```

```
    symboling               0.834146
    normalized-losses     122.000000
    wheel-base             98.756585
    length                174.049268
    width                  65.907805
    height                 53.724878
    curb-weight          2555.565854
    engine-size           126.907317
    bore                    3.329751
    stroke                  3.255423
    compression-ratio      10.142537
    horsepower            104.256158
    peak-rpm             5125.369458
    city-mpg               25.219512
    highway-mpg            30.751220
```

```
price                13207.129353
dtype: float64
```

```
am_df[list_num].median()
```

```
symboling               1.00
normalized-losses     122.00
wheel-base             97.00
length                173.20
width                  65.50
height                 54.10
curb-weight          2414.00
engine-size           120.00
bore                    3.31
stroke                  3.29
compression-ratio       9.00
horsepower             95.00
peak-rpm             5200.00
city-mpg               24.00
highway-mpg            30.00
price               10595.00
dtype: float64
```

```
am_df.mode()
```

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke
0	0.0	122.0	toyota	gas	std	four	sedan	fwd	front	94.5	...	92	mpfi	3.62	3.4
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	122	NaN	NaN	NaN

2 rows × 26 columns

```
#changing units for ['city-mpg', 'highway-mpg'] columns

am_df['city-mpg'] = 235 / am_df['city-mpg']
am_df['highway-mpg'] = 235 / am_df['highway-mpg']


#we have to change  the header name to reflect the unit change

am_df = am_df.rename(columns={'city-mpg': 'city-litres/100km', 'highway-mpg': 'highway-litres/100km'})


am_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 26 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   symboling          205 non-null    int64
 1   normalized-losses  205 non-null    float64
 2   make               205 non-null    object
 3   fuel-type          205 non-null    object
 4   aspiration         205 non-null    object
 5   num-of-doors       205 non-null    object
 6   body-style         205 non-null    object
 7   drive-wheels       205 non-null    object
 8   engine-location    205 non-null    object
 9   wheel-base         205 non-null    float64
 10  length             205 non-null    float64
 11  width              205 non-null    float64
 12  height             205 non-null    float64
 13  curb-weight        205 non-null    int64
 14  engine-type        205 non-null    object
 15  num-of-cylinders   205 non-null    object
 16  engine-size        205 non-null    int64
 17  fuel-system        205 non-null    object
 18  bore               205 non-null    float64
 19  stroke             205 non-null    float64
 20  compression-ratio  205 non-null    float64
 21  horsepower         205 non-null    float64
```

```
 22  peak-rpm              205 non-null    float64
 23  city-litres/100km     205 non-null    float64
 24  highway-litres/100km  205 non-null    float64
 25  price                 205 non-null    float64
dtypes: float64(13), int64(3), object(10)
memory usage: 41.8+ KB
```

```
 #renewing old list_num variable as column heading for the numeric variable is changed

list_num = [j for j in am_df.columns if am_df[j].dtype != 'object']


# Standardising numerical variables using MinMaxScaler which scales all values in between a range of 0-1

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

#fit_transform() - calculates mean and standard dev of each column - (x-mean)/std
am_df_numerical_scaled = scaler.fit_transform(am_df[list_num].to_numpy())     ## .to_numpy converts dataframe to numpy array


am_df_numerical_scaled
```

```
array([[1.        , 0.29842932, 0.05830904, ..., 0.48148148, 0.42105263,
        0.20795889],
       [1.        , 0.29842932, 0.05830904, ..., 0.48148148, 0.42105263,
        0.28255797],
       [0.6       , 0.29842932, 0.2303207 , ..., 0.57017544, 0.4534413 ,
        0.28255797],
       ...,
       [0.2       , 0.15706806, 0.65597668, ..., 0.62191358, 0.56750572,
        0.40631051],
       [0.2       , 0.15706806, 0.65597668, ..., 0.31944444, 0.42105263,
        0.43076312],
       [0.2       , 0.15706806, 0.65597668, ..., 0.57017544, 0.48842105,
        0.43461099]])
```

```
am_df_numerical_scaled.shape, am_df[list_num].shape
```

```
((205, 16), (205, 16))
```

```
#converting numpy array back to dataframe

am_df_numerical_scaled = pd.DataFrame(am_df_numerical_scaled, columns = list_num)
am_df_numerical_scaled.head(2)
```

	symboling	normalized-losses	wheel-base	length	width	height	curb-weight	engine-size	bore	stroke	compression-ratio	horsepower
0	1.0	0.298429	0.058309	0.413433	0.316667	0.083333	0.411171	0.260377	0.664286	0.290476	0.125	0.2625
1	1.0	0.298429	0.058309	0.413433	0.316667	0.083333	0.411171	0.260377	0.664286	0.290476	0.125	0.2625

```
#creating dummies for one hot encoding categorical variables

am_df_categorical_ohe = pd.get_dummies(data = am_df[list_cat], columns = list_cat)
```

```
am_df['fuel-system'].unique()   #checking no. of classes present in column specified in code
```

```
array(['mpfi', '2bbl', 'mfi', '1bbl', 'spfi', '4bbl', 'idi', 'spdi'],
      dtype=object)
```

```
am_df_categorical_ohe.head()
```

	make_alfa-romero	make_audi	make_bmw	make_chevrolet	make_dodge	make_honda	make_isuzu	make_jaguar	make_mazda	make_mercedes-benz
0	1	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0
3	0	1	0	0	0	0	0	0	0	0
4	0	1	0	0	0	0	0	0	0	0

```
#adding numerical and categorical together to proceed further

am_df_scaled = pd.concat([am_df_numerical_scaled, am_df_categorical_ohe], axis = 1)


am_df_scaled
```

	symboling	normalized-losses	wheel-base	length	width	height	curb-weight	engine-size	bore	stroke	...	num-of-cylinders_twelve
0	1.0	0.298429	0.058309	0.413433	0.316667	0.083333	0.411171	0.260377	0.664286	0.290476	...	0
1	1.0	0.298429	0.058309	0.413433	0.316667	0.083333	0.411171	0.260377	0.664286	0.290476	...	0
2	0.6	0.298429	0.230321	0.449254	0.433333	0.383333	0.517843	0.343396	0.100000	0.666667	...	0

```
#Creating Bins

am_df_scaled['Bin']=am_df_scaled['horsepower'].apply(lambda x: "Low" if x<=.33 else ("Medium" if 0.33<x<=0.66 else "High"))
am_df_scaled['Bin'].value_counts()
```

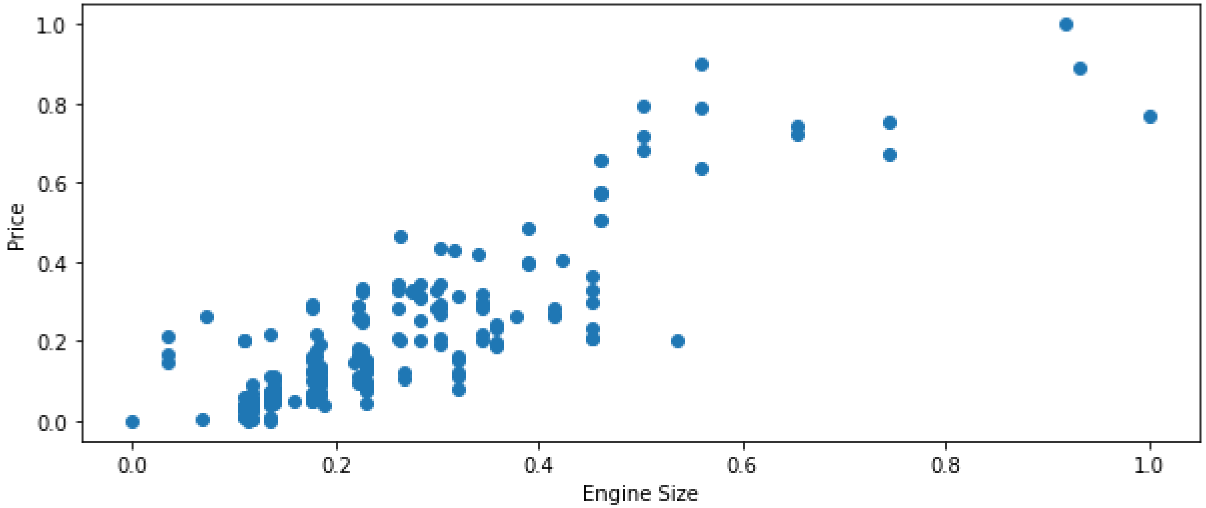
```
Low       163
Medium     37
High        5
Name: Bin, dtype: int64
```

```
# heatmap using bore, stroke, comrpession-ratio, horsepower input features
sns.heatmap(am_df_scaled[['bore', 'stroke', 'compression-ratio', 'horsepower']].corr(), annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2e6a7692b0>
```


```
#creating scatter plots

import matplotlib.pyplot as plt

fig, ax = plt.subplots(1,1,figsize=(10,4))
ax.scatter(am_df_scaled['engine-size'], am_df_scaled['price'])
ax.set_xlabel("Engine Size")
ax.set_ylabel("Price")
```

```
Text(0, 0.5, 'Price')
```




```
am_df_scaled.info()
```

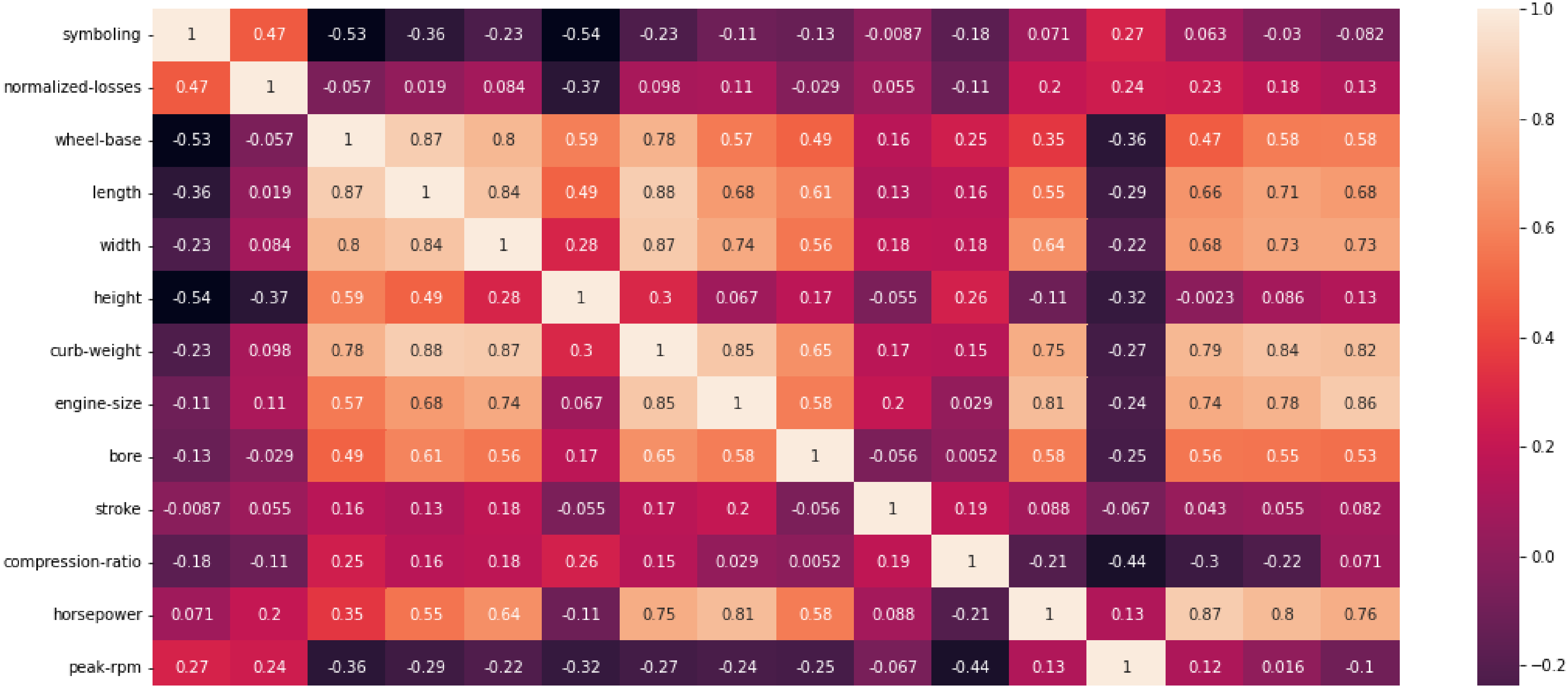
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 77 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
```

```
0   symboling             205 non-null    float64
1   normalized-losses     205 non-null    float64
2   wheel-base            205 non-null    float64
3   length                205 non-null    float64
4   width                 205 non-null    float64
5   height                205 non-null    float64
6   curb-weight           205 non-null    float64
7   engine-size           205 non-null    float64
8   bore                  205 non-null    float64
9   stroke                205 non-null    float64
10  compression-ratio     205 non-null    float64
11  horsepower            205 non-null    float64
12  peak-rpm              205 non-null    float64
13  city-litres/100km     205 non-null    float64
14  highway-litres/100km  205 non-null    float64
15  price                 205 non-null    float64
16  make_alfa-romero      205 non-null    uint8
17  make_audi             205 non-null    uint8
18  make_bmw              205 non-null    uint8
19  make_chevrolet        205 non-null    uint8
20  make_dodge            205 non-null    uint8
21  make_honda            205 non-null    uint8
22  make_isuzu            205 non-null    uint8
23  make_jaguar           205 non-null    uint8
24  make_mazda            205 non-null    uint8
25  make_mercedes-benz    205 non-null    uint8
26  make_mercury          205 non-null    uint8
27  make_mitsubishi       205 non-null    uint8
28  make_nissan           205 non-null    uint8
29  make_peugot           205 non-null    uint8
30  make_plymouth         205 non-null    uint8
31  make_porsche          205 non-null    uint8
32  make_renault          205 non-null    uint8
33  make_saab             205 non-null    uint8
34  make_subaru           205 non-null    uint8
35  make_toyota           205 non-null    uint8
36  make_volkswagen       205 non-null    uint8
37  make_volvo            205 non-null    uint8
38  fuel-type_diesel      205 non-null    uint8
39  fuel-type_gas         205 non-null    uint8
40  aspiration_std        205 non-null    uint8
```

```
41  aspiration_turbo         205 non-null    uint8
42  num-of-doors_four        205 non-null    uint8
43  num-of-doors_two         205 non-null    uint8
44  body-style_convertible   205 non-null    uint8
45  body-style_hardtop       205 non-null    uint8
46  body-style_hatchback     205 non-null    uint8
47  body-style_sedan         205 non-null    uint8
48  body-style_wagon         205 non-null    uint8
49  drive-wheels_4wd         205 non-null    uint8
50  drive-wheels_fwd         205 non-null    uint8
51  drive-wheels_rwd         205 non-null    uint8
```

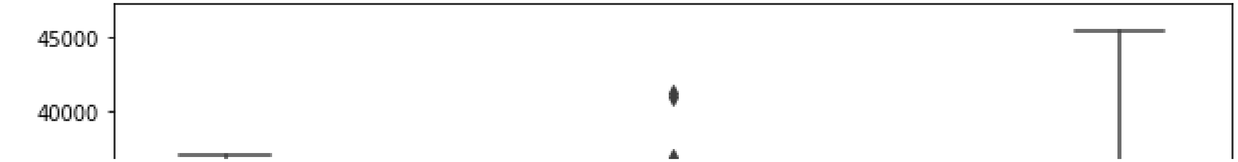
```python
plt.figure(figsize=(18,10))
sns.heatmap(am_df_scaled[list_num].corr(), annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2e6a5c8ee0>
```



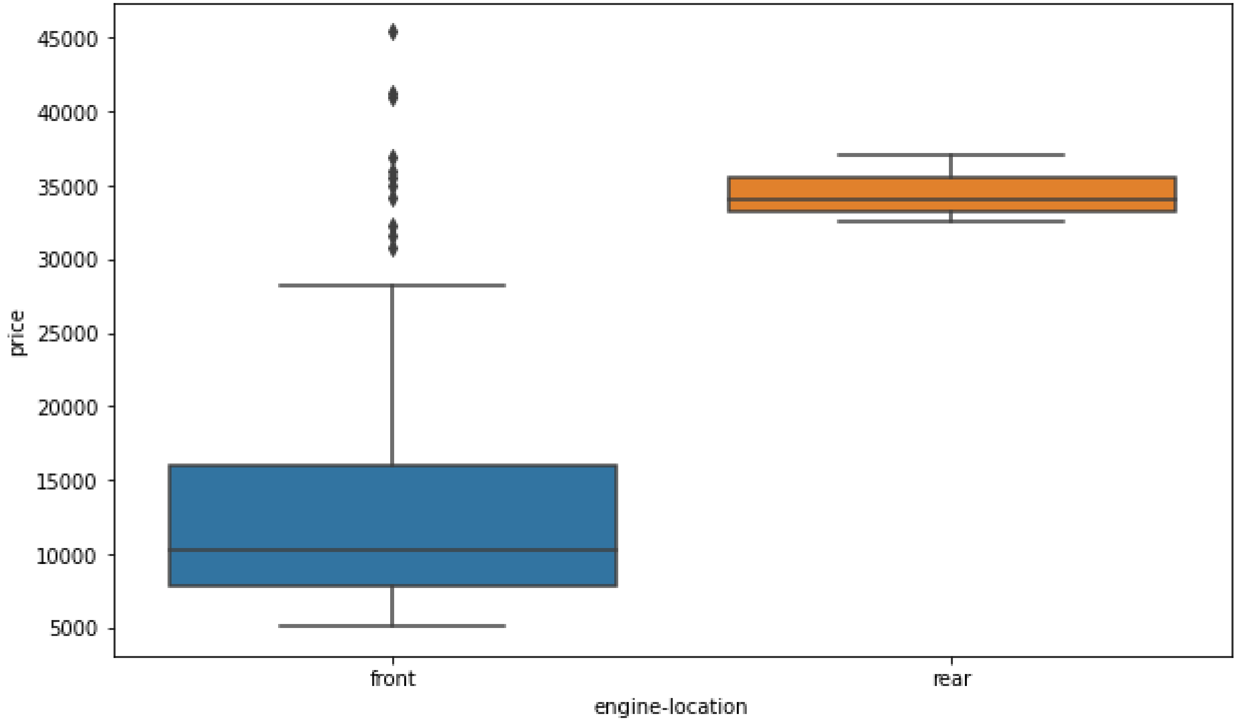
```
#Creating Boxplots
#Box and whisker plots, sometimes known as box plots, are a great chart to use when showing the distribution of data points across a
#These charts display ranges within variables measured. This includes the outliers, the median, the mode, and where the majority of t

plt.figure(figsize=(10,6))
sns.boxplot(x='body-style', y='price', data=am_df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2e6a3b3d30>
```



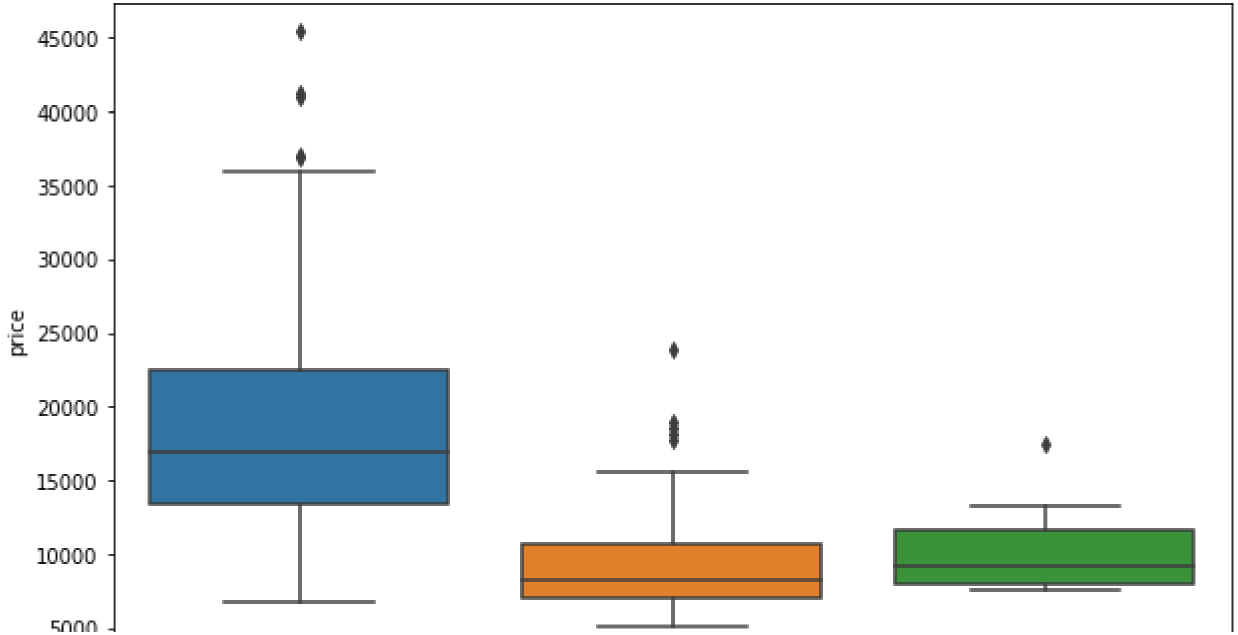

```
plt.figure(figsize=(10,6))
sns.boxplot(x='engine-location', y='price', data=am_df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2e6a299fa0>
```




```
plt.figure(figsize=(10,6))
sns.boxplot(x='drive-wheels', y='price', data=am_df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2e6a36e100>
```




```
am_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 26 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   symboling          205 non-null    int64  
 1   normalized-losses  205 non-null    float64
 2   make               205 non-null    object 
 3   fuel-type          205 non-null    object 
 4   aspiration         205 non-null    object 
 5   num-of-doors       205 non-null    object 
 6   body-style         205 non-null    object 
 7   drive-wheels       205 non-null    object 
 8   engine-location    205 non-null    object 
 9   wheel-base         205 non-null    float64
 10  length             205 non-null    float64
 11  width              205 non-null    float64
 12  height             205 non-null    float64
```

```
 13  curb-weight           205 non-null    int64
 14  engine-type           205 non-null    object
 15  num-of-cylinders      205 non-null    object
 16  engine-size           205 non-null    int64
 17  fuel-system           205 non-null    object
 18  bore                  205 non-null    float64
 19  stroke                205 non-null    float64
 20  compression-ratio     205 non-null    float64
 21  horsepower            205 non-null    float64
 22  peak-rpm              205 non-null    float64
 23  city-litres/100km     205 non-null    float64
 24  highway-litres/100km  205 non-null    float64
 25  price                 205 non-null    float64
dtypes: float64(13), int64(3), object(10)
memory usage: 41.8+ KB
```

```
# groupby is used to separate identical data into groups to allow for further aggregation and analysis.

am_df[['body-style', 'price']].groupby('body-style').agg(np.mean)
```

	price
body-style	
convertible	21890.500000
hardtop	22208.500000
hatchback	10050.289410
sedan	14433.658945
wagon	12371.960000

```
gk = am_df.groupby('make')
```

```
# Let's print the first entries
# in all the groups formed.
gk.first()
```

	symboling	normalized-losses	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	length	...	engine-size	fuel-system
make													
alfa-romero	3	122.0	gas	std	two	convertible	rwd	front	88.6	168.8	...	130	mpfi
audi	2	164.0	gas	std	four	sedan	fwd	front	99.8	176.6	...	109	mpfi
bmw	2	192.0	gas	std	two	sedan	rwd	front	101.2	176.8	...	108	mpfi
chevrolet	2	121.0	gas	std	two	hatchback	fwd	front	88.4	141.1	...	61	2bbl
dodge	1	118.0	gas	std	two	hatchback	fwd	front	93.7	157.3	...	90	2bbl
honda	2	137.0	gas	std	two	hatchback	fwd	front	86.6	144.6		92	1bbl

```
# Finding the values contained in the "audi" group
gk.get_group('audi')
```

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke
3	2	164.0	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.4
4	2	164.0	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.4
5	2	122.0	audi	gas	std	two	sedan	fwd	front	99.8	...	136	mpfi	3.19	3.4
6	1	158.0	audi	gas	std	four	sedan	fwd	front	105.8	...	136	mpfi	3.19	3.4
7	1	122.0	audi	gas	std	four	wagon	fwd	front	105.8	...	136	mpfi	3.19	3.4
8	1	158.0	audi	gas	turbo	four	sedan	fwd	front	105.8	...	131	mpfi	3.13	3.4
9	0	122.0	audi	gas	turbo	two	hatchback	4wd	front	99.5	...	131	mpfi	3.13	3.4

7 rows × 26 columns

toyota	1	87.6	gas	std	two	hatchback	fwd	front	95.7	166.7	...	62	2bbl

```
# First grouping based on "make"
# Within each team we are grouping based on "fuel-type"
gkk = am_df.groupby(['make', 'fuel-type'])

# Print the first value in each group
gkk.first()
```

make	fuel-type	symboling	normalized-losses	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	length	width	...	engine-size
alfa-romero	**gas**	3	122.0	std	two	convertible	rwd	front	88.6	168.8	64.1	...	130
audi	**gas**	2	164.0	std	four	sedan	fwd	front	99.8	176.6	66.2	...	109
bmw	**gas**	2	192.0	std	two	sedan	rwd	front	101.2	176.8	64.8	...	108
chevrolet	**gas**	2	121.0	std	two	hatchback	fwd	front	88.4	141.1	60.3	...	61
dodge	**gas**	1	118.0	std	two	hatchback	fwd	front	93.7	157.3	63.8	...	90
honda	**gas**	2	137.0	std	two	hatchback	fwd	front	86.6	144.6	63.9	...	92
isuzu	**gas**	0	122.0	std	four	sedan	rwd	front	94.3	170.7	61.8	...	111
jaguar	**gas**	0	145.0	std	four	sedan	rwd	front	113.0	199.6	69.6	...	258
mazda	**diesel**	0	122.0	std	four	sedan	fwd	front	98.8	177.8	66.5	...	122
	gas	1	104.0	std	two	hatchback	fwd	front	93.1	159.1	64.2	...	91
mercedes-benz	**diesel**	-1	93.0	turbo	four	sedan	rwd	front	110.0	190.9	70.3	...	183
	gas	-1	122.0	std	four	sedan	rwd	front	115.6	202.6	71.7	...	234
mercury	**gas**	1	122.0	turbo	two	hatchback	rwd	front	102.7	178.4	68.0	...	140
mitsubishi	**gas**	2	161.0	std	two	hatchback	fwd	front	93.7	157.3	64.4	...	92
nissan	**diesel**	1	128.0	std	two	sedan	fwd	front	94.5	165.3	63.8	...	103
peugot	**diesel**	0	161.0	turbo	four	sedan	rwd	front	107.9	186.7	68.4	...	152

```
dd = am_df.groupby(['make', 'body-style'])
```

```
dd.first()
```

make	body-style	symboling	normalized-losses	fuel-type	aspiration	num-of-doors	drive-wheels	engine-location	wheel-base	length	width	...	engine-siz
alfa-romero	**convertible**	3	122.0	gas	std	two	rwd	front	88.6	168.8	64.1	...	13
	hatchback	1	122.0	gas	std	two	rwd	front	94.5	171.2	65.5	...	15
audi	**hatchback**	0	122.0	gas	turbo	two	4wd	front	99.5	178.2	67.9	...	13
	sedan	2	164.0	gas	std	four	fwd	front	99.8	176.6	66.2	...	10
	wagon	1	122.0	gas	std	four	fwd	front	105.8	192.7	71.4	...	13
bmw	**sedan**	2	192.0	gas	std	two	rwd	front	101.2	176.8	64.8	...	10
chevrolet	**hatchback**	2	121.0	gas	std	two	fwd	front	88.4	141.1	60.3	...	6
	sedan	0	81.0	gas	std	four	fwd	front	94.5	158.8	63.6	...	9
dodge	**hatchback**	1	118.0	gas	std	two	fwd	front	93.7	157.3	63.8	...	9
	sedan	1	148.0	gas	std	four	fwd	front	93.7	157.3	63.8	...	9
	wagon	-1	110.0	gas	std	four	fwd	front	103.3	174.6	64.6	...	12
honda	**hatchback**	2	137.0	gas	std	two	fwd	front	86.6	144.6	63.9	...	9
	sedan	0	110.0	gas	std	four	fwd	front	96.5	163.4	64.0	...	9
	wagon	0	78.0	gas	std	four	fwd	front	96.5	157.1	63.9	...	9
isuzu	**hatchback**	2	122.0	gas	std	two	rwd	front	96.0	172.6	65.2	...	11
	sedan	0	122.0	gas	std	four	rwd	front	94.3	170.7	61.8	...	11
jaguar	**sedan**	0	145.0	gas	std	four	rwd	front	113.0	199.6	69.6	...	25
mazda	**hatchback**	1	104.0	gas	std	two	fwd	front	93.1	159.1	64.2	...	9
	sedan	1	113.0	gas	std	four	fwd	front	93.1	166.8	64.2	...	9

mercedes-benz	**convertible**	3	142.0	gas	std	two	rwd	front	96.6	180.3	70.5	...	23
	hardtop	0	93.0	diesel	turbo	two	rwd	front	106.7	187.5	70.3	...	18

Pearson-corelation coefficient

```
import pandas as pd
import scipy.stats as stats
```

```
r = stats.pearsonr(am_df['wheel-base'], am_df['price'])
print(r)
```

```
(0.5831681499789549, 4.527625545686636e-20)
```

```
r1 = stats.pearsonr(am_df['horsepower'], am_df['price'])
print(r1)
```

```
(0.7579169537498178, 1.6076703978129875e-39)
```

```
r2 = stats.pearsonr(am_df['length'], am_df['price'])
print(r2)
```

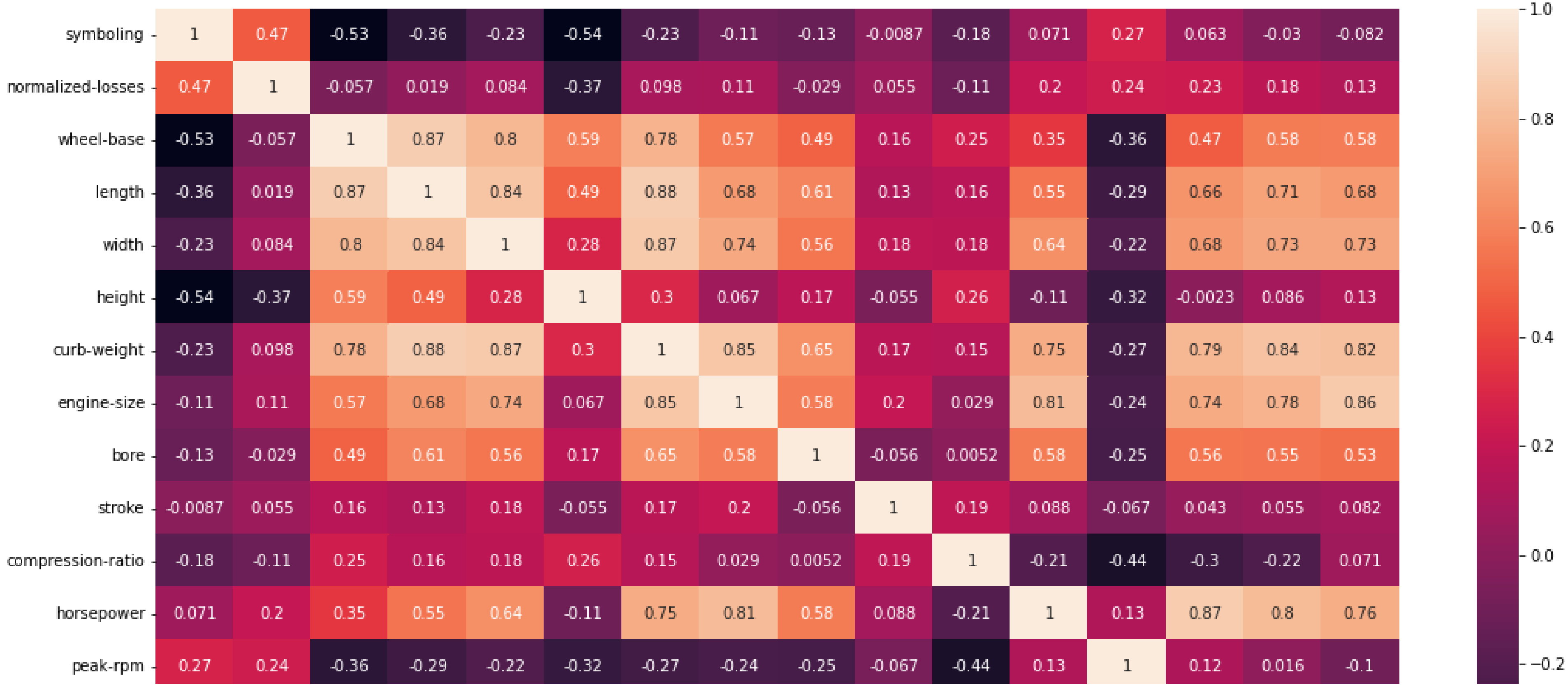
```
(0.6829862954386219, 1.6498873291218535e-29)
```

```
r3 = stats.pearsonr(am_df['width'], am_df['price'])
print(r3)
```

```
(0.7286988175931842, 3.214520483804299e-35)
```

```
plt.figure(figsize=(18,10))
sns.heatmap(am_df.corr(), annot=True)
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f2e6a355520>



Pearson's r value Strength of relationship

0 - No linear relationship, 0.1 to 0.3 : Weak linear relationship, 0.3 to 0.5 : Moderate linear relationship, 0.5 to 0.7 : Strong linear relationship, 0.7 to 1 : Very strong linear relationship, -0.1 to -0.3 : Weak negative linear relationship, -0.3 to -0.5 : Moderate negative linear relationship, -0.5 to -0.7 : Strong negative linear relationship, -0.7 to -1 : Very strong negative linear relationship

ANN model

```
am_df
```

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	s
0	3	122.0	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	
3	2	164.0	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	

205 rows × 26 columns

```
X = am_df_scaled.drop(['price', 'Bin'],axis=1)   #independent/ feature variables

y = am_df.iloc[:, -1]   #dependent / target variable-('price')

#converting to numpy array

X = np.array(X)
y = np.array(y)
```

```
#Splitting data

from sklearn.model_selection import train_test_split

Xtrain, Xtest, ytrain, ytest = train_test_split(X, y , test_size=0.2, random_state=42, shuffle=True)
```

```
Xtrain.shape
```

```
(164, 75)
```

```
Xtest.shape
```

```
(41, 75)
```

```
from keras.models import Sequential
from keras.layers import Dense
```

```
am_df_ANN = Sequential()


#Note:     sigmoid activation - binary [2 class classification]
#          softmax activation - multi  [2+ class classification]
#          relu activation    - regression

#Note:      output layer: number of units=number of class - multi  [2+ class classification]
#                       : number of units=1 - binary [2 class classification]
#                       : number of units=1 - regression


#hidden layer
am_df_ANN.add(Dense(units=1000, activation = 'relu'))

#hidden layer
am_df_ANN.add(Dense(units=400, activation = 'relu'))

#hidden layer
am_df_ANN.add(Dense(units=100, activation = 'relu'))

#output layer
am_df_ANN.add(Dense(units=1,  activation = 'relu'))


am_df_ANN.compile(loss='mean_squared_error', optimizer='adam', metrics='mean_absolute_percentage_error')


history = am_df_ANN.fit(Xtrain, ytrain, epochs=300)
```

```
Epoch 1/300
6/6 [==============================] - 2s 21ms/step - loss: 230089104.0000 - mean_absolute_percentage_error: 99.9878
Epoch 2/300
6/6 [==============================] - 0s 14ms/step - loss: 229853360.0000 - mean_absolute_percentage_error: 99.8983
Epoch 3/300
6/6 [==============================] - 0s 17ms/step - loss: 228971472.0000 - mean_absolute_percentage_error: 99.6045
Epoch 4/300
6/6 [==============================] - 0s 15ms/step - loss: 226668320.0000 - mean_absolute_percentage_error: 98.7822
```

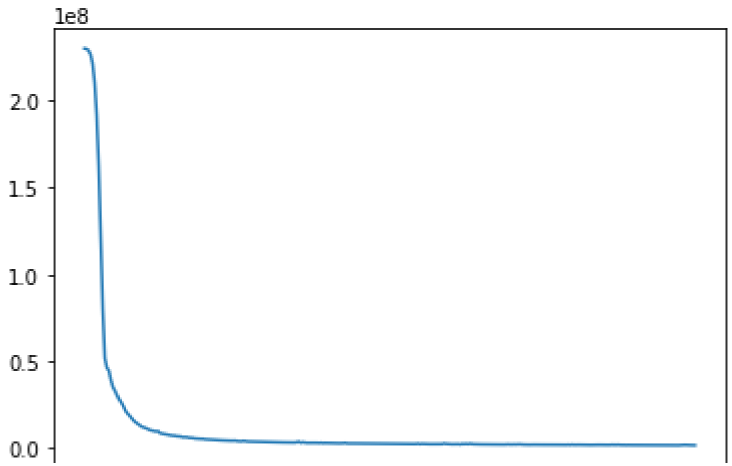
```
Epoch 5/300
6/6 [==============================] - 0s 14ms/step - loss: 221353312.0000 - mean_absolute_percentage_error: 96.9291
Epoch 6/300
6/6 [==============================] - 0s 16ms/step - loss: 210529824.0000 - mean_absolute_percentage_error: 93.1588
Epoch 7/300
6/6 [==============================] - 0s 20ms/step - loss: 191879424.0000 - mean_absolute_percentage_error: 86.0822
Epoch 8/300
6/6 [==============================] - 0s 19ms/step - loss: 162784496.0000 - mean_absolute_percentage_error: 73.8620
Epoch 9/300
6/6 [==============================] - 0s 15ms/step - loss: 123099624.0000 - mean_absolute_percentage_error: 54.7552
Epoch 10/300
6/6 [==============================] - 0s 15ms/step - loss: 79744728.0000 - mean_absolute_percentage_error: 32.4173
Epoch 11/300
6/6 [==============================] - 0s 19ms/step - loss: 51647740.0000 - mean_absolute_percentage_error: 36.8631
Epoch 12/300
6/6 [==============================] - 0s 18ms/step - loss: 46007192.0000 - mean_absolute_percentage_error: 50.7941
Epoch 13/300
6/6 [==============================] - 0s 21ms/step - loss: 44307868.0000 - mean_absolute_percentage_error: 52.8434
Epoch 14/300
6/6 [==============================] - 0s 18ms/step - loss: 38758276.0000 - mean_absolute_percentage_error: 44.6752
Epoch 15/300
6/6 [==============================] - 0s 14ms/step - loss: 34951880.0000 - mean_absolute_percentage_error: 37.3639
Epoch 16/300
6/6 [==============================] - 0s 18ms/step - loss: 32815790.0000 - mean_absolute_percentage_error: 32.9438
Epoch 17/300
6/6 [==============================] - 0s 20ms/step - loss: 30103658.0000 - mean_absolute_percentage_error: 32.5627
Epoch 18/300
6/6 [==============================] - 0s 17ms/step - loss: 27835776.0000 - mean_absolute_percentage_error: 35.3171
Epoch 19/300
6/6 [==============================] - 0s 18ms/step - loss: 25998098.0000 - mean_absolute_percentage_error: 36.6158
Epoch 20/300
6/6 [==============================] - 0s 21ms/step - loss: 23891402.0000 - mean_absolute_percentage_error: 34.7671
Epoch 21/300
6/6 [==============================] - 0s 18ms/step - loss: 21252942.0000 - mean_absolute_percentage_error: 28.6776
Epoch 22/300
6/6 [==============================] - 0s 15ms/step - loss: 19748678.0000 - mean_absolute_percentage_error: 23.2254
Epoch 23/300
6/6 [==============================] - 0s 21ms/step - loss: 18483666.0000 - mean_absolute_percentage_error: 20.4627
Epoch 24/300
6/6 [==============================] - 0s 16ms/step - loss: 16968760.0000 - mean_absolute_percentage_error: 20.1814
Epoch 25/300
```

```
6/6 [==============================] - 0s 18ms/step - loss: 15657019.0000 - mean_absolute_percentage_error: 20.3108
Epoch 26/300
6/6 [==============================] - 0s 26ms/step - loss: 14665452.0000 - mean_absolute_percentage_error: 19.1738
Epoch 27/300
6/6 [==============================] - 0s 22ms/step - loss: 13856100.0000 - mean_absolute_percentage_error: 19.1754
Epoch 28/300
6/6 [==============================] - 0s 20ms/step - loss: 12902683.0000 - mean_absolute_percentage_error: 18.4006
Epoch 29/300
```

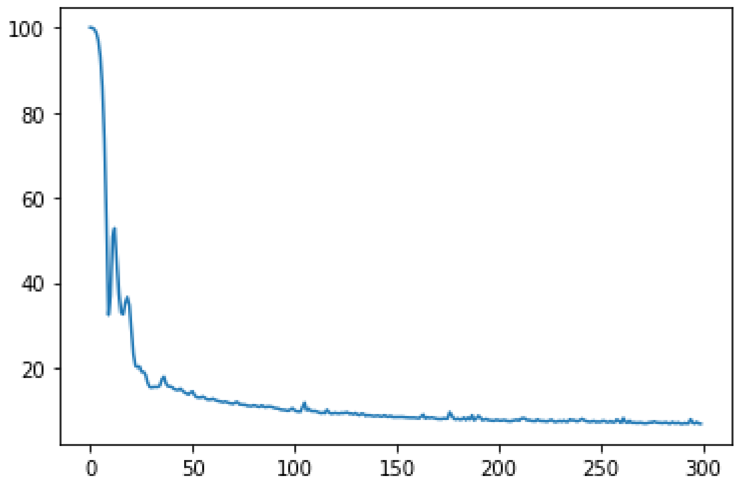
```
am_df_ANN.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 1000)              76000

 dense_1 (Dense)             (None, 400)               400400

 dense_2 (Dense)             (None, 100)               40100

 dense_3 (Dense)             (None, 1)                 101

=================================================================
Total params: 516,601
Trainable params: 516,601
Non-trainable params: 0
_________________________________________________________________
```

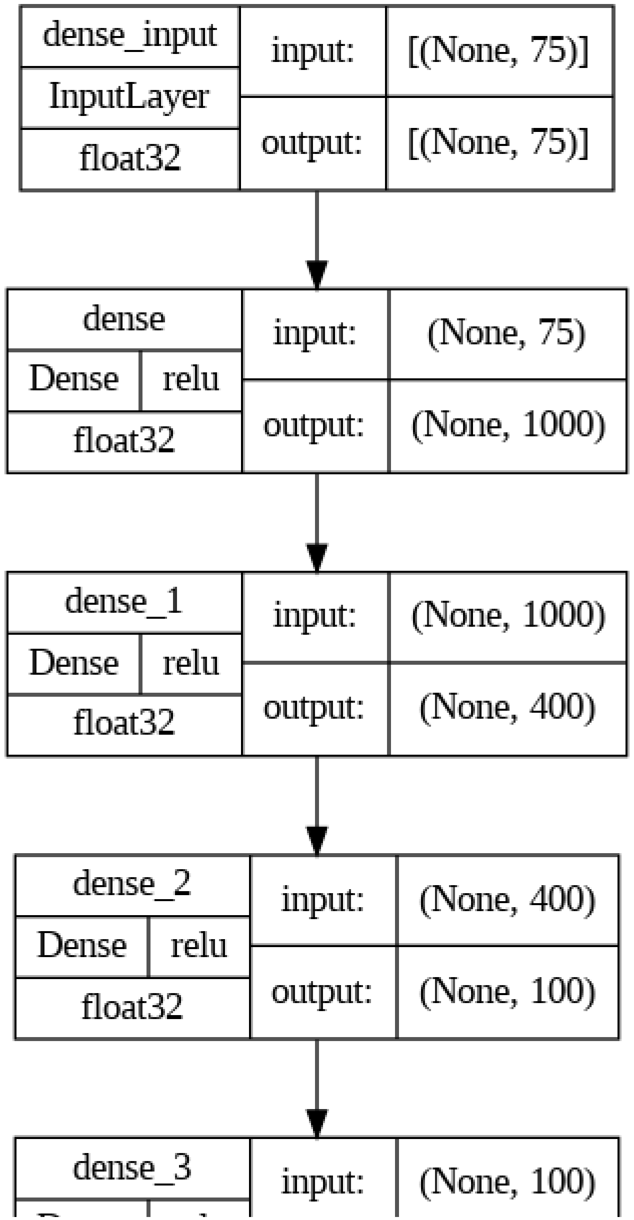
```
#plotting graph

import matplotlib.pyplot as plt


plt.plot(history.history['loss'])
plt.show()
```




```
plt.plot(history.history['mean_absolute_percentage_error'])
plt.show()
```




```
from tensorflow.keras.utils import plot_model
plot_model(am_df_ANN, show_shapes=True, show_dtype=True, show_layer_activations=True, show_layer_names=True)
```




```
ypred = am_df_ANN.predict(Xtest)
```

```
2/2 [==============================] - 0s 12ms/step
```

ypred

```
array([[28775.596 ],
       [21453.652 ],
       [ 8687.918 ],
       [15152.876 ],
       [29039.51  ],
       [ 6841.043 ],
       [ 8211.549 ],
       [ 7617.6216],
       [ 8465.913 ],
       [ 8038.9126],
       [12781.753 ],
       [ 7672.5493],
       [17832.82  ],
       [10321.077 ],
       [40202.03  ],
       [ 6953.777 ],
       [ 1808.3495],
       [13309.161 ],
       [ 8192.34  ],
       [ 8563.055 ],
       [10216.857 ],
       [15347.728 ],
       [ 7472.5464],
       [ 4546.2915],
       [ 6286.6885],
       [28770.111 ],
       [11268.868 ],
       [15512.438 ],
       [ 6675.3003],
       [16056.0205],
       [30615.496 ],
       [ 6693.7173],
       [ 7804.5864],
       [23308.877 ],
       [ 8717.314 ],
       [32922.477 ],
       [11339.835 ],
       [12711.351 ],
       [ 9054.7705],
```

```
       [13427.232 ],
       [ 7427.4077]], dtype=float32)
```

```
ypred.shape
```

```
(41, 1)
```

```
am_df_ANN.evaluate(Xtest, ytest)
```

```
2/2 [==============================] - 0s 8ms/step - loss: 8961007.0000 - mean_absolute_percentage_error: 14.1550
[8961007.0, 14.154963493347168]
```