Data Analysis

```
import pandas as pd
import matplotlib.pylab as plt
import seaborn as sns


df = pd.read_csv('dataset/final_data.csv')
df.head()
```

	symboling	normalized-losses	make	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	length	...	compression-ratio	hor
0	3	122	alfa-romero	std	two	convertible	rwd	front	88.6	0.811148	...	9.0	111
1	3	122	alfa-romero	std	two	convertible	rwd	front	88.6	0.811148	...	9.0	111
2	1	122	alfa-romero	std	two	hatchback	rwd	front	94.5	0.822681	...	9.0	154
3	2	164	audi	std	four	sedan	fwd	front	99.8	0.848630	...	10.0	102
4	2	164	audi	std	four	sedan	4wd	front	99.4	0.848630	...	8.0	115

5 rows × 29 columns

```
   symboling  normalized-losses         make aspiration num-of-doors  \
0          3                122  alfa-romero        std          two
1          3                122  alfa-romero        std          two
2          1                122  alfa-romero        std          two
3          2                164         audi        std         four
4          2                164         audi        std         four

    body-style drive-wheels engine-location  wheel-base    length  ...  \
0  convertible          rwd           front        88.6  0.811148  ...
1  convertible          rwd           front        88.6  0.811148  ...
2    hatchback          rwd           front        94.5  0.822681  ...
3        sedan          fwd           front        99.8  0.848630  ...
4        sedan          4wd           front        99.4  0.848630  ...

   compression-ratio  horsepower  peak-rpm city-mpg highway-mpg    price  \
0                9.0       111.0    5000.0       21          27  13495.0
1                9.0       111.0    5000.0       21          27  16500.0
2                9.0       154.0    5000.0       19          26  16500.0
3               10.0       102.0    5500.0       24          30  13950.0
4                8.0       115.0    5500.0       18          22  17450.0
```

Steps for working with missing data:

1. Identify missing data
2. Deal with missing data
3. Correct data format

```python
import numpy as np

df.replace("?", np.nan, inplace = True)
df.sample(5)
```

	symboling	normalized-losses	make	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	length	...	compression ratio
47	1	104	mazda	std	two	hatchback	fwd	front	93.1	0.764536	...	9.0
165	2	134	toyota	std	two	hatchback	rwd	front	98.4	0.846708	...	9.3
186	3	256	volkswagen	std	two	hatchback	fwd	front	94.5	0.796252	...	8.5
4	2	164	audi	std	four	sedan	4wd	front	99.4	0.848630	...	8.0
11	0	188	bmw	std	two	sedan	rwd	front	101.2	0.849592	...	9.0

5 rows × 29 columns

```
     symboling  normalized-losses        make aspiration num-of-doors  \
47           1                104       mazda        std          two   
165          2                134      toyota        std          two   
186          3                256  volkswagen        std          two   
4            2                164        audi        std         four   
11           0                188         bmw        std          two   

    body-style drive-wheels engine-location  wheel-base    length  ...  \
47   hatchback          fwd           front        93.1  0.764536  ...   
165  hatchback          rwd           front        98.4  0.846708  ...   
186  hatchback          fwd           front        94.5  0.796252  ...   
4        sedan          4wd           front        99.4  0.848630  ...   
11       sedan          rwd           front       101.2  0.849592  ...   

     compression-ratio  horsepower  peak-rpm city-mpg highway-mpg    price  \
47                 9.0        68.0    5000.0       30          31   5195.0   
165                9.3       116.0    4800.0       24          30   9989.0   
186                8.5        90.0    5500.0       24          29   9980.0   
4                  8.0       115.0    5500.0       18          22  17450.0   
11                 9.0       121.0    4250.0       21          28  20970.0   
```

Missing Data

The missing values are converted to Python's default. We use Python's built-in functions to identify these missing values. Methods to detect missing data:

1. **.isnull()**
2. **.notnull()**

The output is a boolean value indicating whether the value that is passed into the argument is in fact missing data.

```
missing_data = df.isnull()
missing_data.head(5)
```

	symboling	normalized-losses	make	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	length	...	compression-ratio	horsepower
0	False	False	False	False	False	False	False	False	False	False	...	False	False
1	False	False	False	False	False	False	False	False	False	False	...	False	False
2	False	False	False	False	False	False	False	False	False	False	...	False	False
3	False	False	False	False	False	False	False	False	False	False	...	False	False
4	False	False	False	False	False	False	False	False	False	False	...	False	False

5 rows × 29 columns

```
   symboling  normalized-losses   make  aspiration  num-of-doors  body-style  \
0      False              False  False       False         False       False   
1      False              False  False       False         False       False   
2      False              False  False       False         False       False   
3      False              False  False       False         False       False   
4      False              False  False       False         False       False   

   drive-wheels  engine-location  wheel-base  length  ...  compression-ratio  \
0         False            False       False   False  ...              False   
1         False            False       False   False  ...              False   
2         False            False       False   False  ...              False   
3         False            False       False   False  ...              False   
4         False            False       False   False  ...              False   

   horsepower  peak-rpm  city-mpg  highway-mpg  price  city-L/100km  \
0       False     False     False        False  False         False   
1       False     False     False        False  False         False   
2       False     False     False        False  False         False   
3       False     False     False        False  False         False   
4       False     False     False        False  False         False   
```

"True" stands for missing value, while "False" stands for not missing value.

Missing values in each column

Using a for loop in Python, we can quickly figure out the number of missing values. In the body of the for loop the method ".value_counts()" counts the number of "True" values.

```
for column in missing_data.columns.values.tolist():
    print(column)
    print(missing_data[column].value_counts())
    print("")
```

```
symboling
False    201
Name: symboling, dtype: int64

normalized-losses
False    201
Name: normalized-losses, dtype: int64

make
False    201
Name: make, dtype: int64

aspiration
False    201
Name: aspiration, dtype: int64

num-of-doors
False    201
Name: num-of-doors, dtype: int64
```

Each column has 205 rows of data and 7 columns containing missing data:

1. "normalized-losses": 41 missing data
2. "num-of-doors": 2 missing data
3. "bore": 4 missing data
4. "stroke" : 4 missing data
5. "horsepower": 2 missing data
6. "peak-rpm": 2 missing data

7. "price": 4 missing data

Deal with missing data

1. drop data
   a. drop the whole row
   b. drop the whole column
2. replace data
   a. replace it by mean
   b. replace it by frequency
   c. replace it based on other functions

Whole columns should be dropped only if most entries in the column are empty. We will apply each method to many different columns:

Replace by mean:

- "normalized-losses": 41 missing data, replace them with mean
- "stroke": 4 missing data, replace them with mean
- "bore": 4 missing data, replace them with mean
- "horsepower": 2 missing data, replace them with mean
- "peak-rpm": 2 missing data, replace them with mean

Replace by frequency:

- "num-of-doors": 2 missing data, replace them with "four".
  - Reason: 84% sedans is four doors. Since four doors is most frequent, it is most likely to occur

Drop the whole row:

- "price": 4 missing data, simply delete the whole row

```
avg_norm = df["normalized-losses"].astype("float").mean(axis=0)
print("Average of normalized-losses:", avg_norm)
```

```
Average of normalized-losses: 122.0
```

```
df["normalized-losses"].replace(np.nan,avg_norm,inplace=True)
```

```
avg_of_bore = df["bore"].astype("float").mean(axis=0)
print("Average of Bore Values:",avg_of_bore)
```

```
Average of Bore Values: 3.33069156704042
```

```
df["bore"].replace(np.nan,avg_of_bore,inplace=True)
```

```
avg_stroke = df["stroke"].astype("float").mean(axis=0)
print("Average of Stroke Values:",avg_of_bore)
```

```
Average of Stroke Values: 3.33069156704042
```

```
df["stroke"].replace(np.nan,avg_stroke,inplace = True)
```

```
avg_horsepower = df['horsepower'].astype("float").mean(axis=0)
print("Average horsepower:", avg_horsepower)
```

```
Average horsepower: 103.40553390682057
```

```
df['horsepower'].replace(np.nan, avg_horsepower, inplace=True)
```

```
avg_peak_rpm=df['peak-rpm'].astype('float').mean(axis=0)
print("Average peak rpm:", avg_peak_rpm)
```

```
Average peak rpm: 5117.665367742568
```

```
df['peak-rpm'].replace(np.nan, avg_peakrpm, inplace=True)
```

```
NameError: name 'avg_peakrpm' is not defined
```

To see which values are present in a particular column,use the ".value_counts()" method:

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

```
four    115
two      86
Name: num-of-doors, dtype: int64
```

We can see that four doors are the most common type.

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

```
'four'
```

```
#replace the missing 'num-of-doors' values by the most frequent
df['num-of-doors'].replace(np.nan,"four",inplace=True)
```

Finally, let's drop all rows that do not have price data:

```python
# simply drop whole row with NaN in "price" column
df.dropna(subset=["price"],axis=0,inplace=True)

# reset index, because we droped two rows
df.reset_index(drop=True,inplace=True)
```

```python
df.head()
```

	symboling	normalized-losses	make	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	length	...	compression-ratio	hor
0	3	122	alfa-romero	std	two	convertible	rwd	front	88.6	0.811148	...	9.0	111
1	3	122	alfa-romero	std	two	convertible	rwd	front	88.6	0.811148	...	9.0	111
2	1	122	alfa-romero	std	two	hatchback	rwd	front	94.5	0.822681	...	9.0	154
3	2	164	audi	std	four	sedan	fwd	front	99.8	0.848630	...	10.0	102
4	2	164	audi	std	four	sedan	4wd	front	99.4	0.848630	...	8.0	115

5 rows × 29 columns

```
   symboling  normalized-losses         make aspiration num-of-doors  \
0          3                122  alfa-romero        std          two
1          3                122  alfa-romero        std          two
2          1                122  alfa-romero        std          two
3          2                164         audi        std         four
4          2                164         audi        std         four

    body-style drive-wheels engine-location  wheel-base    length  ...   \
0  convertible          rwd           front        88.6  0.811148  ...
1  convertible          rwd           front        88.6  0.811148  ...
2    hatchback          rwd           front        94.5  0.822681  ...
3        sedan          fwd           front        99.8  0.848630  ...
4        sedan          4wd           front        99.4  0.848630  ...

   compression-ratio  horsepower  peak-rpm city-mpg highway-mpg    price  \
0                9.0       111.0    5000.0       21          27  13495.0
1                9.0       111.0    5000.0       21          27  16500.0
2                9.0       154.0    5000.0       19          26  16500.0
3               10.0       102.0    5500.0       24          30  13950.0
4                8.0       115.0    5500.0       18          22  17450.0
```

Now, the dataset with no missing values is obtained.

Convert data types to proper format

```
df[["bore", "stroke"]] = df[["bore", "stroke"]].astype("float")
df[["normalized-losses"]] = df[["normalized-losses"]].astype("int")
df[["price"]] = df[["price"]].astype("float")
df[["peak-rpm"]] = df[["peak-rpm"]].astype("float")
```

Columns after the conversion

```
df.dtypes
```

```
symboling              int64
normalized-losses      int64
make                  object
aspiration            object
num-of-doors          object
body-style            object
drive-wheels          object
engine-location       object
wheel-base           float64
length               float64
width                float64
height               float64
curb-weight            int64
engine-type           object
num-of-cylinders      object
engine-size            int64
fuel-system           object
bore                 float64
stroke               float64
compression-ratio    float64
```

Finally the cleaned dataset is obtained with no missing values and all data in its proper format.

Data Standardization

```
df.head()
```

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	fuel-system	bore	stroke
0	3	122	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	mpfi	3.47	2.68
1	3	122	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	mpfi	3.47	2.68
2	1	122	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	mpfi	2.68	3.47
3	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	mpfi	3.19	3.40
4	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...	mpfi	3.19	3.40

5 rows × 27 columns

```
   symboling  normalized-losses         make fuel-type aspiration  \
0          3                122  alfa-romero       gas        std
1          3                122  alfa-romero       gas        std
2          1                122  alfa-romero       gas        std
3          2                164         audi       gas        std
4          2                164         audi       gas        std

  num-of-doors   body-style drive-wheels engine-location  wheel-base  \
0          two  convertible          rwd           front        88.6
1          two  convertible          rwd           front        88.6
2          two    hatchback          rwd           front        94.5
3         four        sedan          fwd           front        99.8
4         four        sedan          4wd           front        99.4

       ...       fuel-system  bore  stroke  compression-ratio horsepower  \
0      ...              mpfi  3.47    2.68                9.0        111
1      ...              mpfi  3.47    2.68                9.0        111
2      ...              mpfi  2.68    3.47                9.0        154
3      ...              mpfi  3.19    3.40               10.0        102
4      ...              mpfi  3.19    3.40                8.0        115
```

```
df["city-L/100km"] = 235 / df["city-mpg"]

df.head()
```

	symboling	normalized-losses	make	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	length	...	compression-ratio	hor
0	3	122	alfa-romero	std	two	convertible	rwd	front	88.6	0.811148	...	9.0	111
1	3	122	alfa-romero	std	two	convertible	rwd	front	88.6	0.811148	...	9.0	111
2	1	122	alfa-romero	std	two	hatchback	rwd	front	94.5	0.822681	...	9.0	154
3	2	164	audi	std	four	sedan	fwd	front	99.8	0.848630	...	10.0	102
4	2	164	audi	std	four	sedan	4wd	front	99.4	0.848630	...	8.0	115

5 rows × 29 columns

```
   symboling  normalized-losses         make aspiration num-of-doors  \
0          3                122  alfa-romero        std          two
1          3                122  alfa-romero        std          two
2          1                122  alfa-romero        std          two
3          2                164         audi        std         four
4          2                164         audi        std         four

    body-style drive-wheels engine-location  wheel-base    length  ...  \
0  convertible          rwd           front        88.6  0.811148  ...
1  convertible          rwd           front        88.6  0.811148  ...
2    hatchback          rwd           front        94.5  0.822681  ...
3        sedan          fwd           front        99.8  0.848630  ...
4        sedan          4wd           front        99.4  0.848630  ...

   compression-ratio  horsepower  peak-rpm city-mpg highway-mpg    price  \
0                9.0       111.0    5000.0       21          27  13495.0
1                9.0       111.0    5000.0       21          27  16500.0
2                9.0       154.0    5000.0       19          26  16500.0
3               10.0       102.0    5500.0       24          30  13950.0
4                8.0       115.0    5500.0       18          22  17450.0
```

```
df["highway-mpg"] = 235/df["highway-mpg"]
df.rename(columns={'"highway-mpg"':'highway-L/100km'}, inplace=True)
```

Data Normalization

Why normalization?

Normalization is the process of transforming values of several variables into a similar range. Typical normalizations include scaling the variable so the variable average is 0, scaling the variable so the variance is 1, or scaling variable so the variable values range from 0 to 1

```
# replace (original value) by (original value)/(maximum value)
df["length"] = df["length"]/df["length"].max()
df["width"] = df["width"]/df["width"].max()
df['height'] = df['height']/df['height'].max()
df[["length","width","height"]].head()
```

	length	width	height
0	0.811148	0.890278	0.816054
1	0.811148	0.890278	0.816054
2	0.822681	0.909722	0.876254
3	0.848630	0.919444	0.908027
4	0.848630	0.922222	0.908027


```
     length     width    height
0  0.811148  0.890278  0.816054
1  0.811148  0.890278  0.816054
2  0.822681  0.909722  0.876254
3  0.848630  0.919444  0.908027
4  0.848630  0.922222  0.908027
```


Here we can see, we've normalized "length", "width" and "height" in the range of [0,1].

Binning

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

```
df["horsepower"] = df["horsepower"].astype(int,copy=True)
```

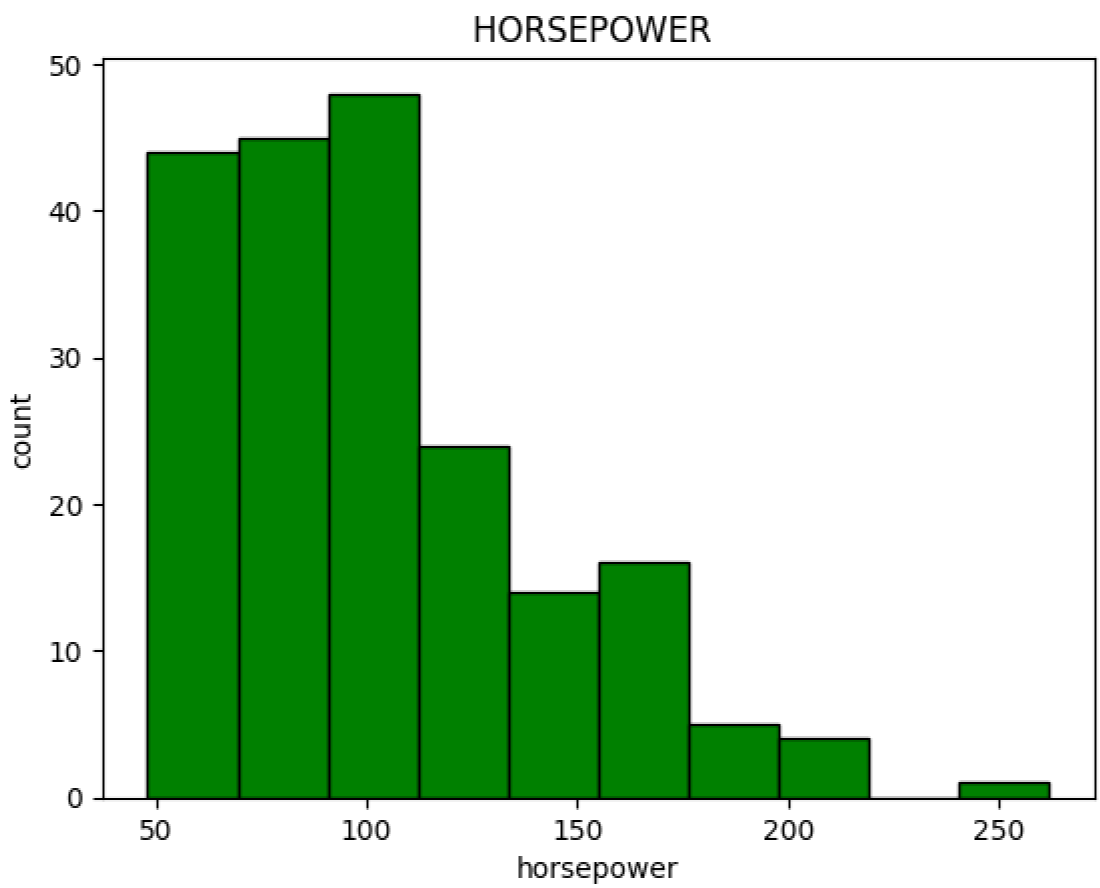
Plot the histogram of horspower in order to see what the distribution of horsepower looks like.

```
%matplotlib inline
import matplotlib.pyplot as plt
plt.hist(df["horsepower"], color='green', edgecolor='black', bins=10)

plt.xlabel("horsepower")
plt.ylabel("count")
plt.title("HORSEPOWER ")
```

```
Text(0.5, 1.0, 'HORSEPOWER ')
```

Download




```
<Figure size 640x480 with 1 Axes>
```

```
bins = np.linspace(min(df["horsepower"]),max(df["horsepower"]),4)
bins
```

```
array([ 48.        , 119.33333333, 190.66666667, 262.        ])
```

We set group names:

```
group_names = ['Low', 'Medium', 'High']
```

We apply the function "cut" the determine what each value of "df['horsepower']" belongs to.

```
df['horsepower-binned'] = pd.cut(df['horsepower'], bins, labels=group_names ,include_lowes
df[['horsepower','horsepower-binned']].head(10)
```

	horsepower	horsepower-binned
0	111	Low
1	111	Low
2	154	Medium
3	102	Low
4	115	Low
5	110	Low
6	110	Low
7	110	Low
8	140	Medium
9	101	Low

```
   horsepower horsepower-binned
0         111               Low
1         111               Low
2         154            Medium
3         102               Low
4         115               Low
```

```
5          110              Low
6          110              Low
7          110              Low
8          140           Medium
9          101              Low
```

Lets see the number of vehicles in each bin.

```
df["horsepower-binned"].value_counts()
```

```
Low       153
Medium     43
High        5
Name: horsepower-binned, dtype: int64
```

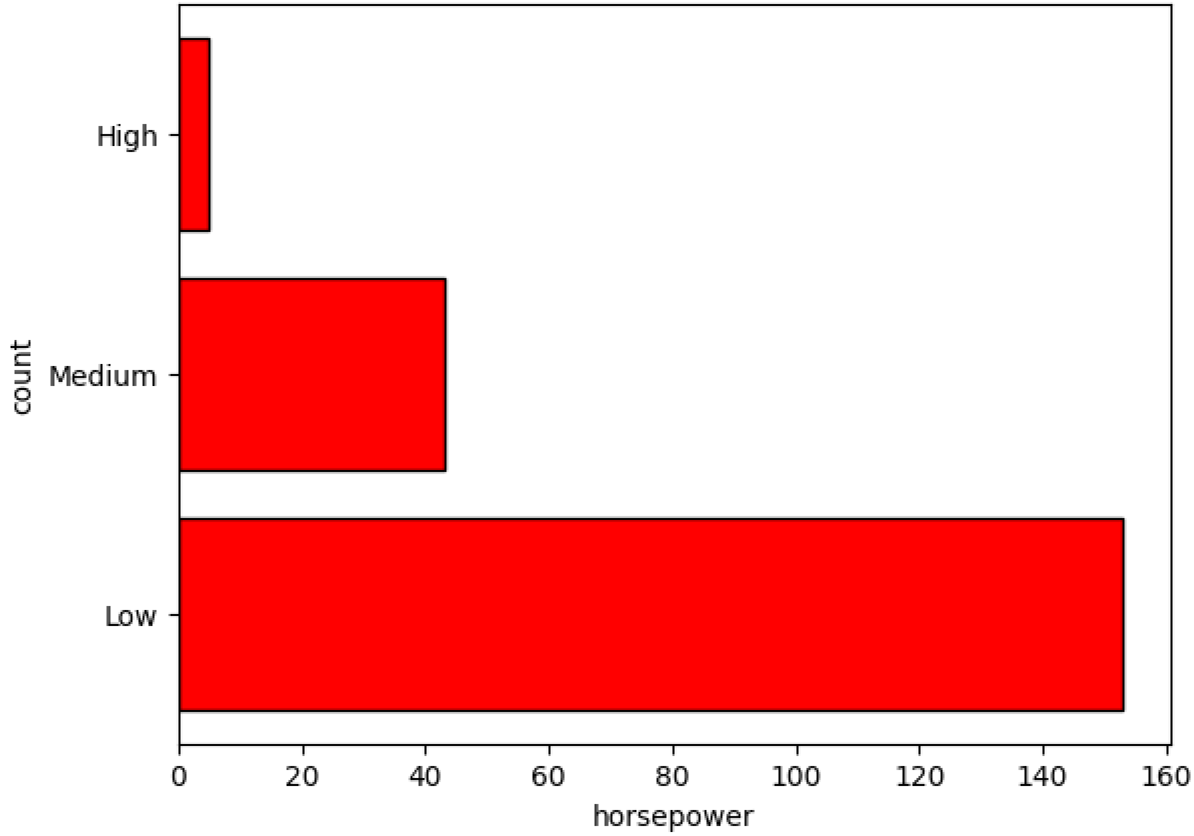
Lets plot the distribution of each bin.

```
%matplotlib inline
import matplotlib as plt
from matplotlib import pyplot
pyplot.barh(group_names, df["horsepower-binned"].value_counts() , color='red', edgecolor='

plt.pyplot.xlabel("horsepower")
plt.pyplot.ylabel("count")
```

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

Download




```
<Figure size 640x480 with 1 Axes>
```

```
df.corr()
```

	symboling	normalized-losses	wheel-base	length	width	height	curb-weight	engine-size	bore	stroke
symboling	1.000000	0.466264	-0.535987	-0.365404	-0.242423	-0.550160	-0.233118	-0.110581	-0.140019	-0.007992
normalized-losses	0.466264	1.000000	-0.056661	0.019424	0.086802	-0.373737	0.099404	0.112360	-0.029862	0.055127
wheel-base	-0.535987	-0.056661	1.000000	0.876024	0.814507	0.590742	0.782097	0.572027	0.493244	0.157964
length	-0.365404	0.019424	0.876024	1.000000	0.857170	0.492063	0.880665	0.685025	0.608971	0.123913
width	-0.242423	0.086802	0.814507	0.857170	1.000000	0.306002	0.866201	0.729436	0.544885	0.188814
height	-0.550160	-0.373737	0.590742	0.492063	0.306002	1.000000	0.307581	0.074694	0.180449	-0.060822
curb-weight	-0.233118	0.099404	0.782097	0.880665	0.866201	0.307581	1.000000	0.849072	0.644060	0.167412
engine-size	-0.110581	0.112360	0.572027	0.685025	0.729436	0.074694	0.849072	1.000000	0.572609	0.205806
bore	-0.140019	-0.029862	0.493244	0.608971	0.544885	0.180449	0.644060	0.572609	1.000000	-0.055390
stroke	-0.007992	0.055127	0.157964	0.123913	0.188814	-0.060822	0.167412	0.205806	-0.055390	1.000000
compression-ratio	-0.182196	-0.114713	0.250313	0.159733	0.189867	0.259737	0.156433	0.028889	0.001263	0.187854
horsepower	0.075819	0.217299	0.371147	0.579821	0.615077	-0.087027	0.757976	0.822676	0.566936	0.098282
peak-rpm	0.279740	0.239543	-0.360305	-0.285970	-0.245800	-0.309974	-0.279361	-0.256733	-0.267392	-0.063388
city-mpg	-0.035527	-0.225016	-0.470606	-0.665192	-0.633531	-0.049800	-0.749543	-0.650546	-0.582027	-0.034079
highway-mpg	0.036233	-0.181877	-0.543304	-0.698142	-0.680635	-0.104812	-0.794889	-0.679571	-0.591309	-0.034741
price	-0.082391	0.133999	0.584642	0.690628	0.751265	0.135486	0.834415	0.872335	0.543155	0.082267
city-L/100km	0.066171	0.238567	0.476153	0.657373	0.673363	0.003811	0.785353	0.745059	0.554610	0.036285
diesel	-0.196735	-0.101546	0.307237	0.211187	0.244356	0.281578	0.221046	0.070779	0.054458	0.241033
gas	0.196735	0.101546	-0.307237	-0.211187	-0.244356	-0.281578	-0.221046	-0.070779	-0.054458	-0.241033

```
                   symboling  normalized-losses  wheel-base    length  \
symboling           1.000000           0.466264   -0.535987 -0.365404
normalized-losses   0.466264           1.000000   -0.056661  0.019424
wheel-base         -0.535987          -0.056661    1.000000  0.876024
length             -0.365404           0.019424    0.876024  1.000000
width              -0.242423           0.086802    0.814507  0.857170
height             -0.550160          -0.373737    0.590742  0.492063
curb-weight        -0.233118           0.099404    0.782097  0.880665
engine-size        -0.110581           0.112360    0.572027  0.685025
bore               -0.140019          -0.029862    0.493244  0.608971
stroke             -0.007992           0.055127    0.157964  0.123913
compression-ratio  -0.182196          -0.114713    0.250313  0.159733
horsepower          0.075819           0.217299    0.371147  0.579821
peak-rpm            0.279740           0.239543   -0.360305 -0.285970
city-mpg           -0.035527          -0.225016   -0.470606 -0.665192
highway-mpg         0.036233          -0.181877   -0.543304 -0.698142
price              -0.082391           0.133999    0.584642  0.690628
city-L/100km        0.066171           0.238567    0.476153  0.657373
diesel             -0.196735          -0.101546    0.307237  0.211187
gas                 0.196735           0.101546   -0.307237 -0.211187
```

```
/tmp/ipykernel_6935/1134722465.py:1: FutureWarning: The default value of numeric_only in
  df.corr()
```

```python
df[['bore','stroke','compression-ratio','horsepower']].corr()
```

	bore	stroke	compression-ratio	horsepower
bore	1.000000	-0.055390	0.001263	0.566936
stroke	-0.055390	1.000000	0.187854	0.098282
compression-ratio	0.001263	0.187854	1.000000	-0.214514
horsepower	0.566936	0.098282	-0.214514	1.000000


```
                       bore    stroke  compression-ratio  horsepower
bore               1.000000 -0.055390           0.001263    0.566936
stroke            -0.055390  1.000000           0.187854    0.098282
compression-ratio  0.001263  0.187854           1.000000   -0.214514
horsepower         0.566936  0.098282          -0.214514    1.000000
```

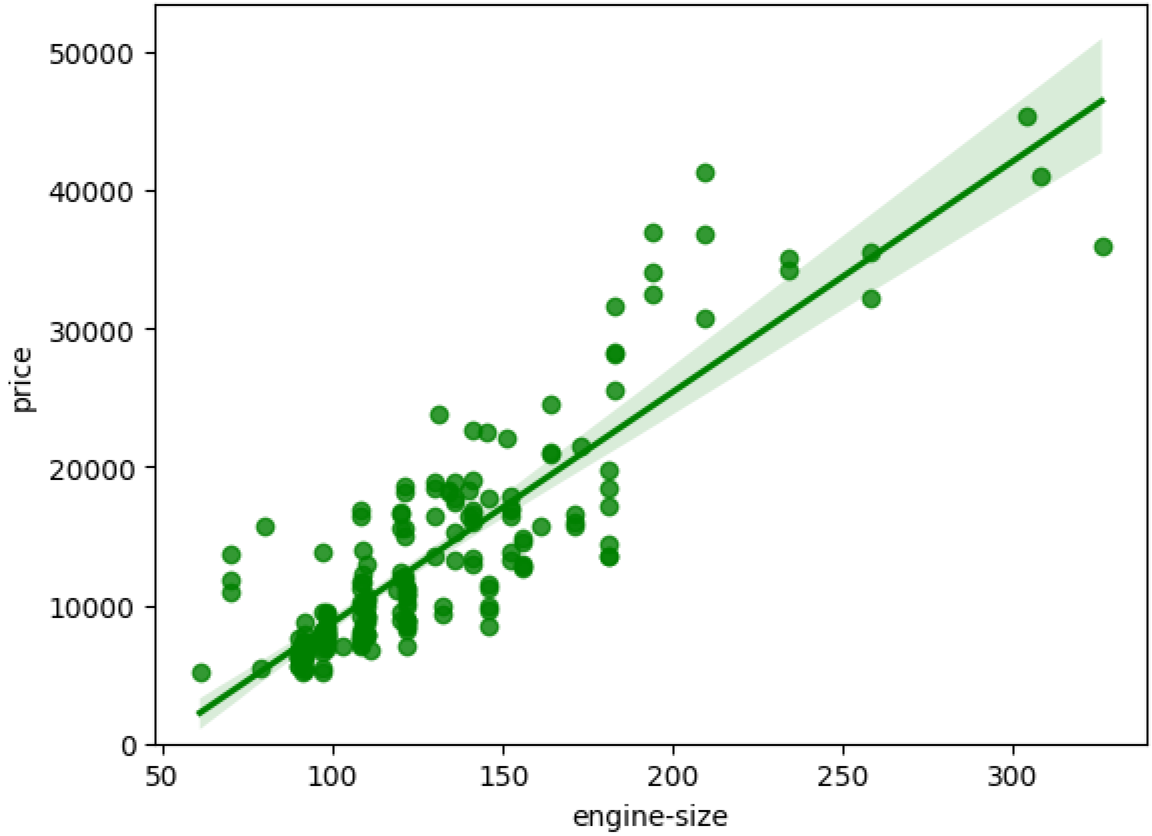

Positive linear relationship

Scatterplot of "engine-size" and "price"

```python
sns.regplot(x="engine-size", y="price", data=df ,color='green')
plt.ylim(0,)
```

```
(0.0, 53417.19589840475)
```

Download




```
<Figure size 640x480 with 1 Axes>
```

```
df[["engine-size", "price"]].corr()
```

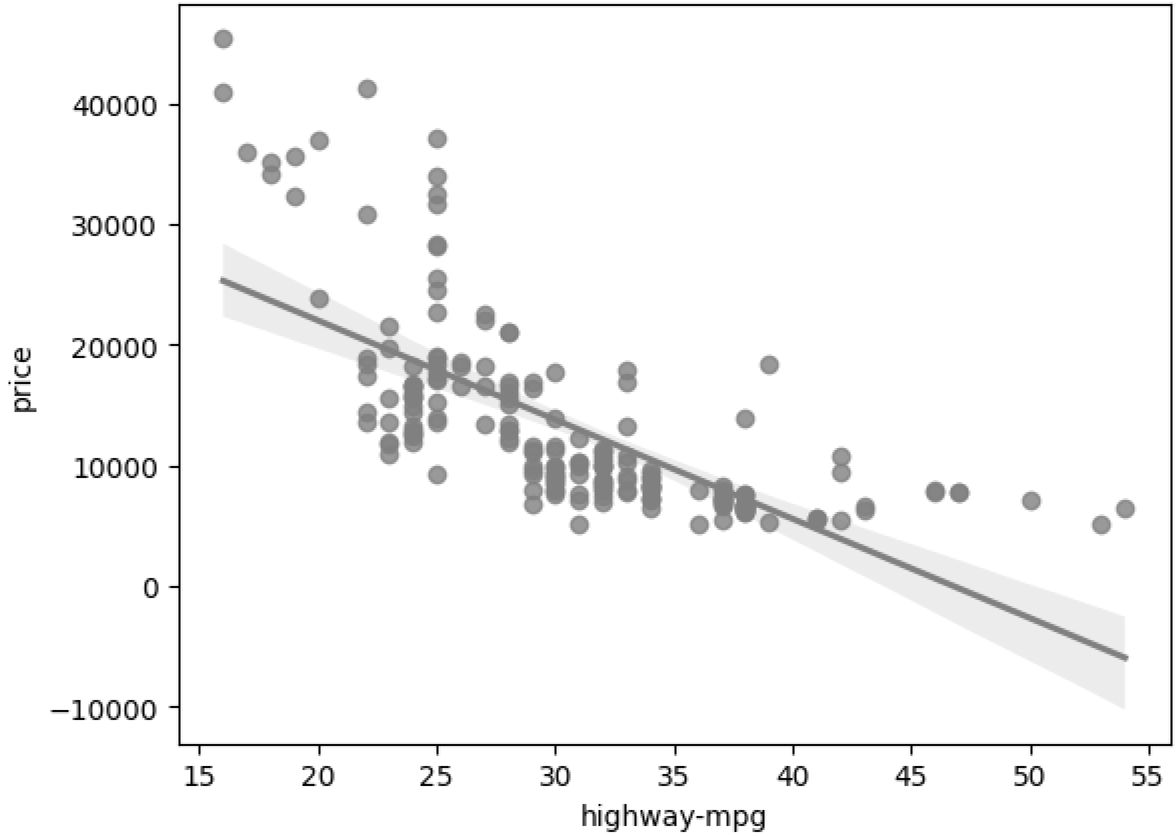
	engine-size	price
engine-size	1.000000	0.872335
price	0.872335	1.000000

```
             engine-size     price
engine-size     1.000000  0.872335
price           0.872335  1.000000
```

Thus, Highway mpg is a potential predictor variable of price

```
sns.regplot(x="highway-mpg", y="price", data=df, color='grey');
```

Download




```
<Figure size 640x480 with 1 Axes>
```

```
df[['highway-mpg', 'price']].corr()
```

	highway-mpg	price
highway-mpg	1.000000	-0.704692
price	-0.704692	1.000000

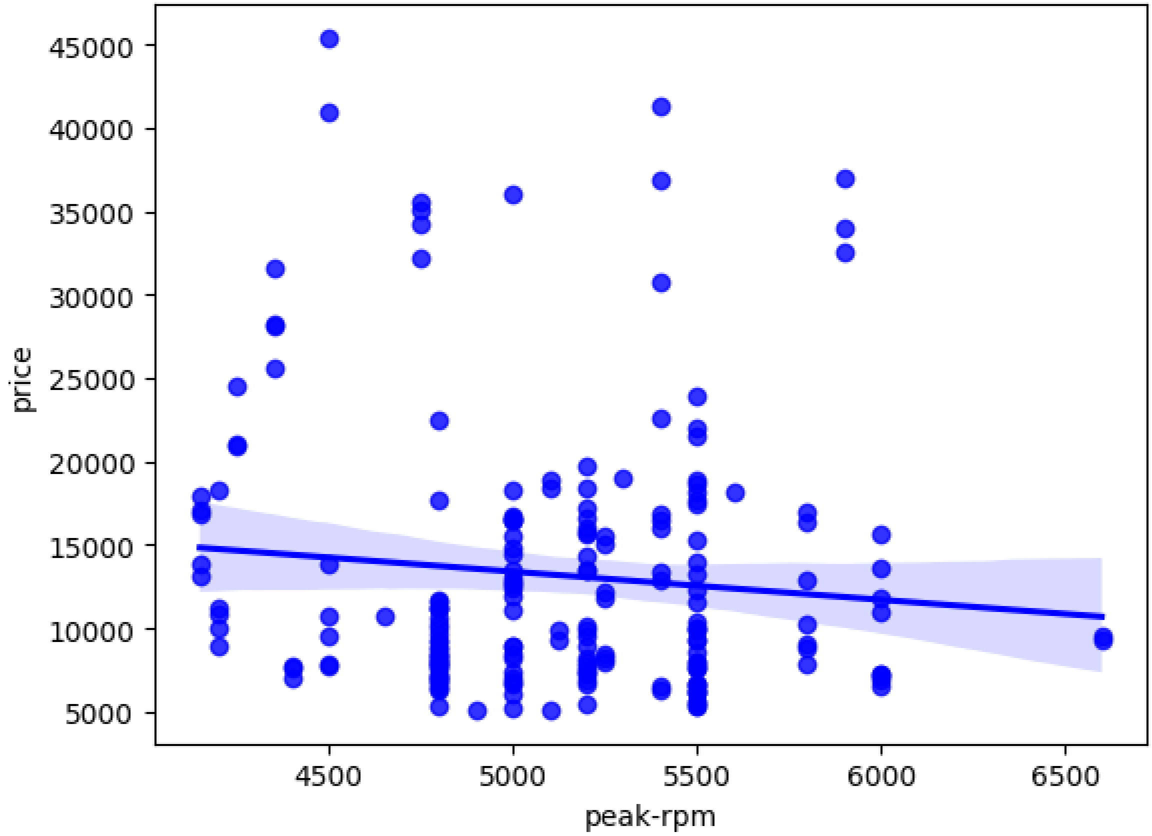
```
             highway-mpg     price
highway-mpg     1.000000 -0.704692
price          -0.704692  1.000000
```

Weak Linear Relationship

Let's see if "Peak-rpm" as a predictor variable of "price".

```
sns.regplot(x="peak-rpm", y="price", data=df ,color='blue');
```

Download



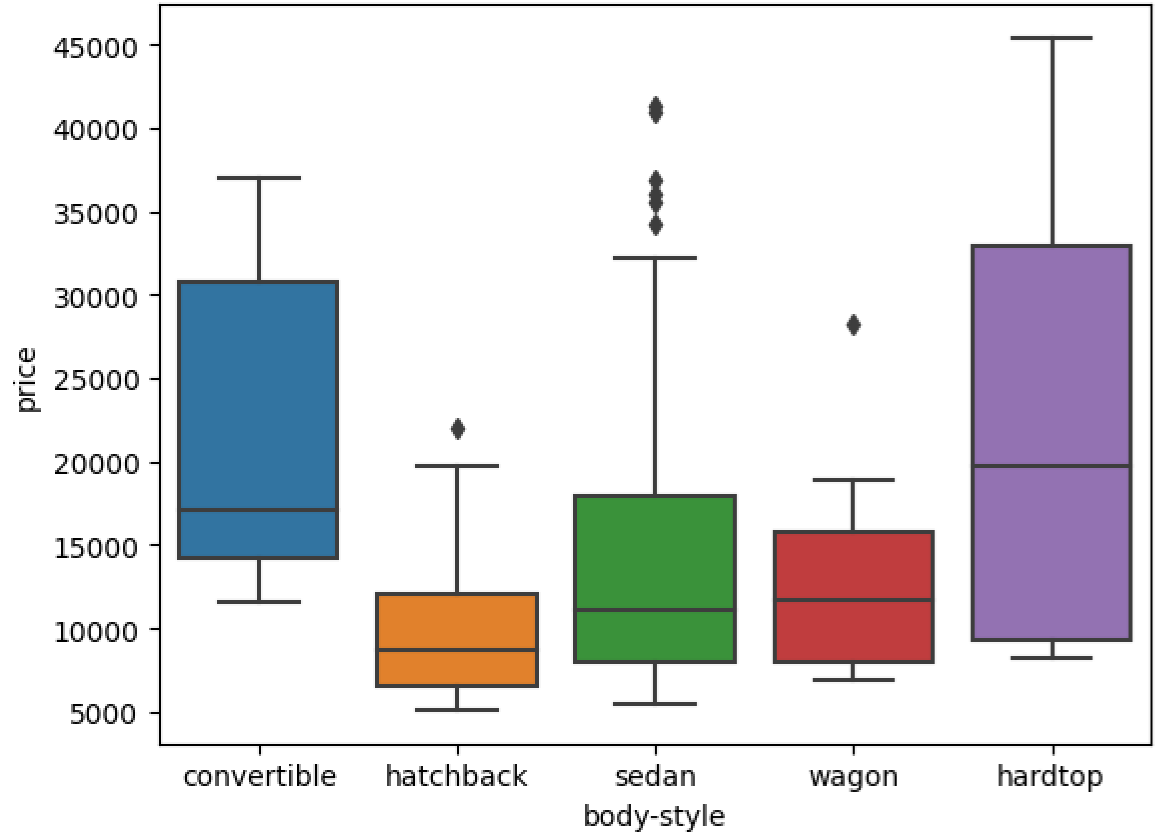

```
<Figure size 640x480 with 1 Axes>
```

Categorical variables

Relationship between "body-style" and "price"

```
sns.boxplot(x="body-style", y="price", data=df);
```

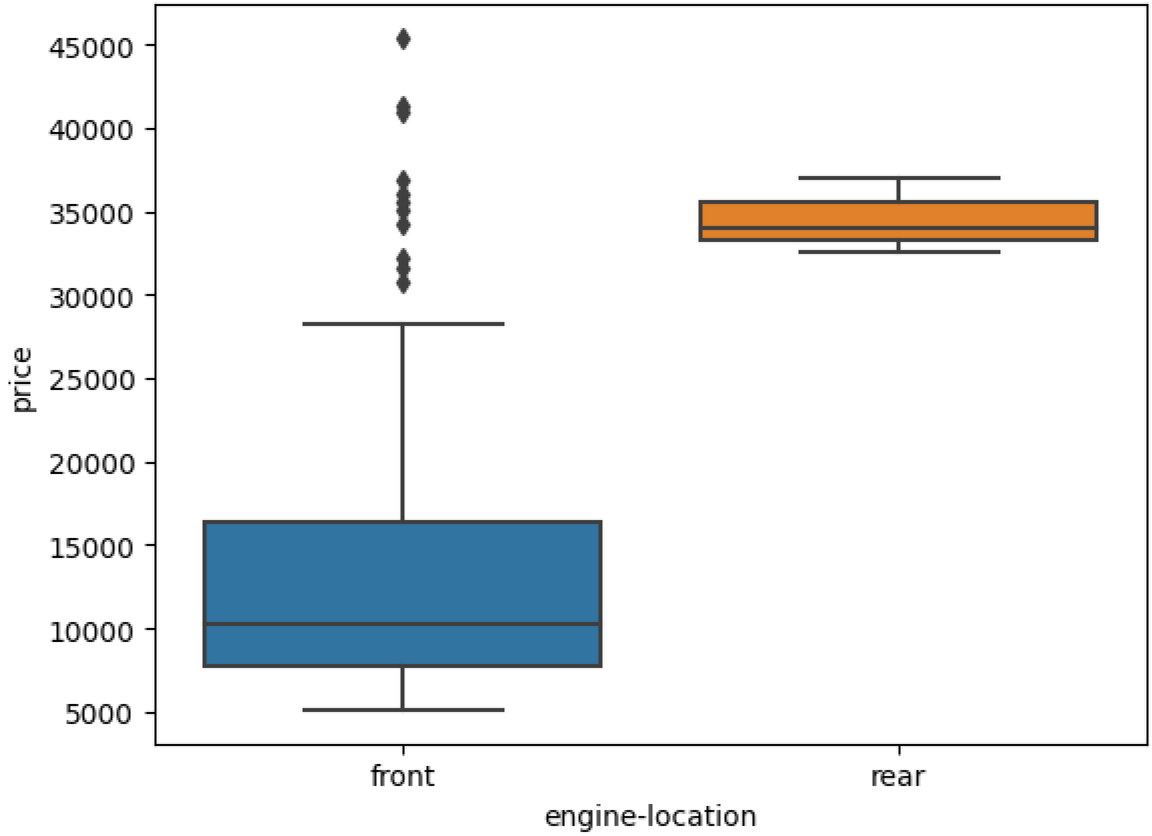
Download




```
<Figure size 640x480 with 1 Axes>
```

```
sns.boxplot(x="engine-location", y="price", data=df);
```

Download




```
<Figure size 640x480 with 1 Axes>
```

```
df.describe(include=['object'])
```

	make	aspiration	num-of-doors	body-style	drive-wheels	engine-location	engine-type	num-of-cylinders	fuel-system	horsepower-binned
count	201	201	201	201	201	201	201	201	201	200
unique	22	2	2	5	3	2	6	7	8	3
top	toyota	std	four	sedan	fwd	front	ohc	four	mpfi	Low
freq	32	165	115	94	118	198	145	157	92	115

```
          make aspiration num-of-doors body-style drive-wheels  \
count      201        201          201        201          201
unique      22          2            2          5            3
top     toyota        std         four      sedan          fwd
freq        32        165          115         94          118

       engine-location engine-type num-of-cylinders fuel-system  \
count              201         201              201         201
unique               2           6                7           8
top              front         ohc             four        mpfi
freq               198         145              157          92

       horsepower-binned
count                200
unique                 3
```

```
top                     Low
freq                    115
```

P-value:

What is this P-value? The P-value is the probability value that the correlation between these two variables is statistically significant. Normally, we choose a significance level of 0.05, which means that we are 95% confident that the correlation between the variables is significant.

By convention, when the

- p-value is $<$ 0.001: we say there is strong evidence that the correlation is significant.
- the p-value is $<$ 0.05: there is moderate evidence that the correlation is significant.
- the p-value is $<$ 0.1: there is weak evidence that the correlation is significant.
- the p-value is $>$ 0.1: there is no evidence that the correlation is significant.

```python
from scipy import stats
```

Wheel-base vs Price

Let's calculate the Pearson Correlation Coefficient and P-value of 'wheel-base' and 'price'.

```python
pearson_coef, p_value = stats.pearsonr(df['wheel-base'], df['price'])
print("The Pearson Correlation Coefficient is", pearson_coef, " with a P-value of P =", p_
```

```
The Pearson Correlation Coefficient is 0.584641822265508  with a P-value of P = 8.076488
```

Conclusion:

Since the p-value is $<$ 0.001, the correlation between wheel-base and price is statistically significant, although the linear relationship isn't extremely strong (~0.585)

Horse-Power vs Price

Let's calculate the Pearson Correlation Coefficient and P-value of 'Horse-Power' and 'price'.

```
pearson_coef, p_value = stats.pearsonr(df['horsepower'], df['price'])
print("The Pearson Correlation Coefficient is", pearson_coef, " with a P-value of P =", p_
```

```
The Pearson Correlation Coefficient is 0.8095745670036562  with a P-value of P = 6.36905
```

Conclusion:

Since the p-value is $<$ 0.001, the correlation between horsepower and price is statistically significant, and the linear relationship is quite strong (~0.809, close to 1)

Length vs Price

Let's calculate the Pearson Correlation Coefficient and P-value of 'length' and 'price'.

```
pearson_coef, p_value = stats.pearsonr(df['length'], df['price'])
print("The Pearson Correlation Coefficient is", pearson_coef, " with a P-value of P = ", p
```

```
The Pearson Correlation Coefficient is 0.690628380448364  with a P-value of P =  8.01647
```

Conclusion:

Since the p-value is < 0.001, the correlation between length and price is statistically significant, and the linear relationship is moderately strong (~0.691).

Width vs Price

Let's calculate the Pearson Correlation Coefficient and P-value of 'width' and 'price':

```
pearson_coef, p_value = stats.pearsonr(df['width'], df['price'])
print("The Pearson Correlation Coefficient is", pearson_coef, " with a P-value of P =", p_
```

```
The Pearson Correlation Coefficient is 0.7512653440522674  with a P-value of P = 9.20033
```

Conclusion:

Since the p-value is < 0.001, the correlation between width and price is statistically significant, and the linear relationship is quite strong (~0.751).

Curb-weight vs Price

```
pearson_coef, p_value = stats.pearsonr(df['curb-weight'], df['price'])
print( "The Pearson Correlation Coefficient is", pearson_coef, " with a P-value of P = ",
```

```
The Pearson Correlation Coefficient is 0.8344145257702846  with a P-value of P =  2.1895
```

Conclusion:

Since the p-value is $<$ 0.001, the correlation between curb-weight and price is statistically significant, and the linear relationship is quite strong (~0.834).

Engine-size vs Price

Let's calculate the Pearson Correlation Coefficient and P-value of 'engine-size' and 'price':

```
pearson_coef, p_value = stats.pearsonr(df['engine-size'], df['price'])
print("The Pearson Correlation Coefficient is", pearson_coef, " with a P-value of P =", p_
```

```
The Pearson Correlation Coefficient is 0.8723351674455185  with a P-value of P = 9.26549
```

Conclusion:

Since the p-value is $<$ 0.001, the correlation between engine-size and price is statistically significant, and the linear relationship is very strong (~0.872).

Bore vs Price

```
pearson_coef, p_value = stats.pearsonr(df['bore'], df['price'])
print("The Pearson Correlation Coefficient is", pearson_coef, " with a P-value of P =  ",
```

```
The Pearson Correlation Coefficient is 0.5431553832626602  with a P-value of P =   8.049
```

Conclusion:

Since the p-value is $<$ 0.001, the correlation between bore and price is statistically significant, but the linear relationship is only moderate (~0.521).

City-mpg vs Price

```
pearson_coef, p_value = stats.pearsonr(df['city-mpg'], df['price'])
print("The Pearson Correlation Coefficient is", pearson_coef, " with a P-value of P = ", p
```

```
The Pearson Correlation Coefficient is -0.6865710067844677  with a P-value of P =  2.321
```

Conclusion:

Since the p-value is $<$ 0.001, the correlation between city-mpg and price is statistically significant, and the coefficient of ~ -0.687 shows that the relationship is negative and moderately strong.

Conclusion: Important Variables

We now have a better idea of what our data looks like and which variables are important to take into account when predicting the car price. We have narrowed it down to the following variables:

Continuous numerical variables:

- Length
- Width
- Curb-weight
- Engine-size
- Horsepower
- City-mpg
- Highway-mpg
- Wheel-base

- Bore

Categorical variables:

- Drive-wheels

As we now move into building machine learning models to automate our analysis, feeding the model with variables that meaningfully affect our target variable will improve our model's prediction performance.