```
#Prediction of Car price
# import pandas library
import pandas as pd
import numpy as np
# Import pandas library
import pandas as pd

# Read the online file by the URL provides above, and assign it to variable "df"
other_path = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-DA0101EN-SkillsNetwork/labs/Data%2
df = pd.read_csv(other_path, header=None)
# show the first 5 rows using dataframe.head() method
print("The first 5 rows of the dataframe")
df.head(5)
df.tail(10)
# create headers list
headers = ["symboling","normalized-losses","make","fuel-type","aspiration", "num-of-doors","body-style",
         "drive-wheels","engine-location","wheel-base", "length","width","height","curb-weight","engine-type",
         "num-of-cylinders", "engine-size","fuel-system","bore","stroke","compression-ratio","horsepower",
         "peak-rpm","city-mpg","highway-mpg","price"]
print("headers\n", headers)
df.columns = headers
df.head(10)
print("_____---_____Dealing with missing values_____--")
df1=df.replace('?',np.NaN)    #We need to replace the "?" symbol with NaN so the dropna() can remove the missing values:
df=df1.dropna(subset=["price"], axis=0)    #We can drop missing values along the column "price"
df.head(20)
print(df.columns)                              #Find the name of the columns of the dataframe.
df.to_csv("automobile.csv", index=False)   # if you would save the dataframe df as automobile.csv to your local machine,
                                            #you may use the syntax below, where index = False means the row names will not be written.
df.dtypes
print(df.dtypes)
df.describe()
print("_____describe all :it provides the statistical summary of all the columns, including object-typed attributes._____")
df.describe(include = "all")
df[['length','compression-ratio']].describe()
df.info() #info :It provides a concise summary of your DataFrame.
#To add 1 to each “symboling” entry, use this command.
#This changes each value of the dataframe column by adding 1 to the current value.
df['symboling']=df['symboling']+1
df[['symboling']]

print("_____Correcting datatypes_____")
df1["price"]=df["price"].astype("int")
df.head()
#We may want to normalize these variables so that the range of the values is consistent.
#3 ways to normalise data
#1st method is called SIMPLE FEATURE SCALING
#x(new)=x(old)/x(max)
#The resulting new values range between 0 and 1.
#2nd method is MINMAX method
#3rd method is "Z SCORE METHOD" or "STANDARD METHOD"
df["length"]=(df["length"])/df["length"].max()
df.info()
"""Binning:Binning is when you group values together into bins. For example, you can bin “age”
into [0 to 5], [6 to 10], [11 to 15] and so on.
Sometimes, binning can improve accuracy of the predictive models.
In addition, sometimes we use data binning to group a set of numerical values into a
smaller number of bins to have a better understanding of the data distribution."""

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

```
dtypes: float64(5), int64(5), object(16)
memory usage: 42.4+ KB
```

Correcting datatypes

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 201 entries, 0 to 204
Data columns (total 26 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   symboling          201 non-null    int64  
 1   normalized-losses  164 non-null    object 
 2   make               201 non-null    object 
 3   fuel-type          201 non-null    object 
 4   aspiration         201 non-null    object 
 5   num-of-doors       199 non-null    object 
 6   body-style         201 non-null    object 
 7   drive-wheels       201 non-null    object 
 8   engine-location    201 non-null    object 
 9   wheel-base         201 non-null    float64
 10  length             201 non-null    float64
 11  width              201 non-null    float64
 12  height             201 non-null    float64
 13  curb-weight        201 non-null    int64  
 14  engine-type        201 non-null    object 
 15  num-of-cylinders   201 non-null    object 
 16  engine-size        201 non-null    int64  
 17  fuel-system        201 non-null    object 
 18  bore               197 non-null    object 
 19  stroke             197 non-null    object 
 20  compression-ratio  201 non-null    float64
 21  horsepower         199 non-null    object 
 22  peak-rpm           199 non-null    object 
 23  city-mpg           201 non-null    int64  
 24  highway-mpg        201 non-null    int64  
 25  price              201 non-null    object 
dtypes: float64(5), int64(5), object(16)
memory usage: 50.5+ KB
<ipython-input-104-22c80f240b28>:39: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a
  df['symboling']=df['symboling']+1
<ipython-input-104-22c80f240b28>:52: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a
```

```
#Identifying missing values
# replace "?" to NaN
df.replace("?",np.nan,inplace=True)
df.head()
```

```
/usr/local/lib/python3.8/dist-packages/pandas/core/frame.py:5238: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-co
  return super().replace(
```

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke	comp
0	4	NaN	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	
1	4	NaN	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	
2	2	NaN	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	
3	3	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	
4	3	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.40	

5 rows × 26 columns

```
#Evaluating mising data
missing_data=df.isnull()
```

```
missing_data.head()
#"True" means the value is a missing value while "False" means the value is not a missing value.
```

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke	compressi ra
0	False	True	False	False	False	False	False	False	False	False	...	False	False	False	False	F
1	False	True	False	False	False	False	False	False	False	False	...	False	False	False	False	F
2	False	True	False	False	False	False	False	False	False	False	...	False	False	False	False	F
3	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	F
4	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	F

5 rows × 26 columns

```
'''Count missing values in each column
Using a for loop in Python, we can quickly figure out the number of missing values in each column. As mentioned above,
"True" represents a missing value and "False" means the value is present in the dataset.
 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(" ")
```

```
highway-mpg
False    201
Name: highway-mpg, dtype: int64

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

```
#Calculate the mean value for the "normalized-losses" column
avg_norm_loss=df["normalized-losses"].astype("float").mean(axis=0)
print("average normalized-losses is:::",avg_norm_loss)
```

```
average normalized-losses is::: 122.0
```

```
#Replace NaN with avg_norm_loss
df["normalized-losses"].replace(np.nan,avg_norm_loss,inplace=True)
df.head()
```

```
/usr/local/lib/python3.8/dist-packages/pandas/core/generic.py:6619: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-cc
  return self._update_inplace(result)
```

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke	comp
0	4	122.0	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	
1	4	122.0	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	
2	2	122.0	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	
3	3	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	
4	3	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.40	

5 rows × 26 columns

```
avg_bore=df['bore'].astype('float').mean(axis=0)
print("Average of bore:", avg_bore)
df["bore"].replace(np.nan, avg_bore, inplace=True)
df.head()
```

```
Average of bore: 3.330710659898477
/usr/local/lib/python3.8/dist-packages/pandas/core/generic.py:6619: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-cc
  return self._update_inplace(result)
```

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke	comp
0	4	122.0	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	
1	4	122.0	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	
2	2	122.0	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	
3	3	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	
4	3	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.40	

5 rows × 26 columns

```
#Calculate the mean value for the "stroke" column and replacing "np.nan" with avg_stroke
avg_stroke=df["stroke"].astype("float").mean(axis=0)
print("average stroke:::",avg_stroke)
df["stroke"].replace(np.nan,avg_stroke,inplace=True)
df.head()
df.tail(20)
```

```
average stroke::: 3.256903553299492
/usr/local/lib/python3.8/dist-packages/pandas/core/generic.py:6619: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-cc
  return self._update_inplace(result)
```

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke
185	3	94	volkswagen	gas	std	four	sedan	fwd	front	97.3	...	109	mpfi	3.19	3.40
186	3	94	volkswagen	gas	std	four	sedan	fwd	front	97.3	...	109	mpfi	3.19	3.40
187	3	94	volkswagen	diesel	turbo	four	sedan	fwd	front	97.3	...	97	idi	3.01	3.40
188	3	94	volkswagen	gas	std	four	sedan	fwd	front	97.3	...	109	mpfi	3.19	3.40
189	4	122.0	volkswagen	gas	std	two	convertible	fwd	front	94.5	...	109	mpfi	3.19	3.40
190	4	256	volkswagen	gas	std	two	hatchback	fwd	front	94.5	...	109	mpfi	3.19	3.40
191	1	122.0	volkswagen	gas	std	four	sedan	fwd	front	100.4	...	136	mpfi	3.19	3.40
192	1	122.0	volkswagen	diesel	turbo	four	sedan	fwd	front	100.4	...	97	idi	3.01	3.40
193	1	122.0	volkswagen	gas	std	four	wagon	fwd	front	100.4	...	109	mpfi	3.19	3.40
194	-1	103	volvo	gas	std	four	sedan	rwd	front	104.3	...	141	mpfi	3.78	3.15
195	0	74	volvo	gas	std	four	wagon	rwd	front	104.3	...	141	mpfi	3.78	3.15
196	-1	103	volvo	gas	std	four	sedan	rwd	front	104.3	...	141	mpfi	3.78	3.15
197	0	74	volvo	gas	std	four	wagon	rwd	front	104.3	...	141	mpfi	3.78	3.15
198	-1	103	volvo	gas	turbo	four	sedan	rwd	front	104.3	...	130	mpfi	3.62	3.15
199	0	74	volvo	gas	turbo	four	wagon	rwd	front	104.3	...	130	mpfi	3.62	3.15
200	0	95	volvo	gas	std	four	sedan	rwd	front	109.1	...	141	mpfi	3.78	3.15
201	0	95	volvo	gas	turbo	four	sedan	rwd	front	109.1	...	141	mpfi	3.78	3.15
202	0	95	volvo	gas	std	four	sedan	rwd	front	109.1	...	173	mpfi	3.58	2.87
203	0	95	volvo	diesel	turbo	four	sedan	rwd	front	109.1	...	145	idi	3.01	3.40
204	0	95	volvo	gas	turbo	four	sedan	rwd	front	109.1	...	141	mpfi	3.78	3.15

20 rows × 26 columns

```
#Calculate the mean value for the "horsepower" column and replacing "np.nan" with avg_horsepower
avg_hp=df["horsepower"].astype("float").mean(axis=0)
print("average horsepower:::",avg_hp)
df["horsepower"].replace(np.nan,avg_hp,inplace=True)
df.head()
df.tail(20)
```

```
average horsepower::: 103.39698492462311
/usr/local/lib/python3.8/dist-packages/pandas/core/generic.py:6619: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-cc
  return self._update_inplace(result)
```

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke
185	3	94	volkswagen	gas	std	four	sedan	fwd	front	97.3	...	109	mpfi	3.19	3.40
186	3	94	volkswagen	gas	std	four	sedan	fwd	front	97.3	...	109	mpfi	3.19	3.40
187	3	94	volkswagen	diesel	turbo	four	sedan	fwd	front	97.3	...	97	idi	3.01	3.40
188	3	94	volkswagen	gas	std	four	sedan	fwd	front	97.3	...	109	mpfi	3.19	3.40
189	4	122.0	volkswagen	gas	std	two	convertible	fwd	front	94.5	...	109	mpfi	3.19	3.40
190	4	256	volkswagen	gas	std	two	hatchback	fwd	front	94.5	...	109	mpfi	3.19	3.40
191	1	122.0	volkswagen	gas	std	four	sedan	fwd	front	100.4	...	136	mpfi	3.19	3.40
192	1	122.0	volkswagen	diesel	turbo	four	sedan	fwd	front	100.4	...	97	idi	3.01	3.40
193	1	122.0	volkswagen	gas	std	four	wagon	fwd	front	100.4	...	109	mpfi	3.19	3.40
194	-1	103	volvo	gas	std	four	sedan	rwd	front	104.3	...	141	mpfi	3.78	3.15
195	0	74	volvo	gas	std	four	wagon	rwd	front	104.3	...	141	mpfi	3.78	3.15
196	-1	103	volvo	gas	std	four	sedan	rwd	front	104.3	...	141	mpfi	3.78	3.15
197	0	74	volvo	gas	std	four	wagon	rwd	front	104.3	...	141	mpfi	3.78	3.15
198	-1	103	volvo	gas	turbo	four	sedan	rwd	front	104.3	...	130	mpfi	3.62	3.15
199	0	74	volvo	gas	turbo	four	wagon	rwd	front	104.3		130	mpfi	3.62	3.15

```
#Calculate the mean value for the "peak-rpm" column and replacing "np.nan" with avg_rpm

avg_peakrpm=df['peak-rpm'].astype('float').mean(axis=0)
print("Average peak rpm:", avg_peakrpm)
df['peak-rpm'].replace(np.nan, avg_peakrpm, inplace=True)
df.head()
```

```
Average peak rpm: 5117.587939698493
/usr/local/lib/python3.8/dist-packages/pandas/core/generic.py:6619: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-cc
  return self._update_inplace(result)
```

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke	comp
0	4	122.0	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	
1	4	122.0	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	
2	2	122.0	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	
3	3	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	
4	3	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.40	

5 rows × 26 columns

```
#To see which values are present in a particular column, we can use the ".value_counts()" method:
df["num-of-doors"].value_counts()
```

```
four    113
two      86
Name: num-of-doors, dtype: int64
```

```
#We can see that four doors are the most common type. We can also use the ".idxmax()" method to calculate the most common type automatically:
df["num-of-doors"].value_counts().idxmax()
```

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

```
'four'
```

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

```
/usr/local/lib/python3.8/dist-packages/pandas/core/generic.py:6619: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-co
  return self._update_inplace(result)
```

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke	comp
0	4	122.0	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	
1	4	122.0	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	
2	2	122.0	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	
3	3	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	
4	3	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.40	

5 rows × 26 columns

```
## 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)
df.head()
```

```
/usr/local/lib/python3.8/dist-packages/pandas/util/_decorators.py:311: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-co
  return func(*args, **kwargs)
```

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke	comp
0	4	122.0	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	
1	4	122.0	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	
2	2	122.0	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	
3	3	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	
4	3	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.40	

5 rows × 26 columns

```
#Let's list the data types for each column
df.dtypes
```

```
symboling             int64
normalized-losses    object
make                 object
fuel-type            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                  object
stroke                object
compression-ratio    float64
horsepower            object
peak-rpm              object
city-mpg               int64
highway-mpg            int64
price                 object
dtype: object
```

```
#Convert data types to proper format
#Here,bore n stroke are objects.so we shall convert it to float and other conversions are made
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")
df.dtypes
```

```
/usr/local/lib/python3.8/dist-packages/pandas/core/frame.py:3641: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-cc
  self[k1] = value[k2]
symboling              int64
normalized-losses      int64
make                  object
fuel-type             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
horsepower            object
peak-rpm             float64
city-mpg               int64
highway-mpg            int64
price                float64
dtype: object
```

Data standardisation

```
#Standardization is the process of transforming data into a common format, allowing the researcher to make the meaningful comparison.
#Transform mpg to L/100km which is in miles per gallon(mpg) ["fuel consumption columns "]
#The formula for unit conversion is:
#L/100km = 235 / mpg
## Convert mpg to L/100km by mathematical operation (235 divided by mpg)
df["city-L/100km"]=235/df["city-mpg"]
df.head()
```

```
<ipython-input-120-737e1ecedf54>:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-co
  df["city-L/100km"]=235/df["city-mpg"]
```

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	fuel-system	bore	stroke	compression-ratio
0	4	122	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	mpfi	3.47	2.68	9.0
1	4	122	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	mpfi	3.47	2.68	9.0
2	2	122	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	mpfi	2.68	3.47	9.0
3	3	164	audi	gas	std	four	sedan	fwd	front	99.8	...	mpfi	3.19	3.40	10.0
4	3	164	audi	gas	std	four	sedan	4wd	front	99.4	...	mpfi	3.19	3.40	8.0

5 rows × 27 columns

```
# transform mpg to L/100km in the column of "highway-mpg" and change the name of column to "highway-L/100km".
df["highway-L/100km"]=235/df["highway-mpg"]
df.head()
```

```
<ipython-input-121-b3f7589453fe>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-co
  df["highway-L/100km"]=235/df["highway-mpg"]
```

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	bore	stroke	compression-ratio	horsepo
0	4	122	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	3.47	2.68	9.0	
1	4	122	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	3.47	2.68	9.0	
2	2	122	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	2.68	3.47	9.0	
3	3	164	audi	gas	std	four	sedan	fwd	front	99.8	...	3.19	3.40	10.0	
4	3	164	audi	gas	std	four	sedan	4wd	front	99.4	...	3.19	3.40	8.0	

5 rows × 28 columns

Data Normalisation

```
'''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 the variable so the variable values range from 0 to 1.
To demonstrate normalization, let's say we want to scale the columns "length", "width" and "height".
Target: would like to normalize those variables so their value ranges from 0 to 1
Approach: 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.head()
```

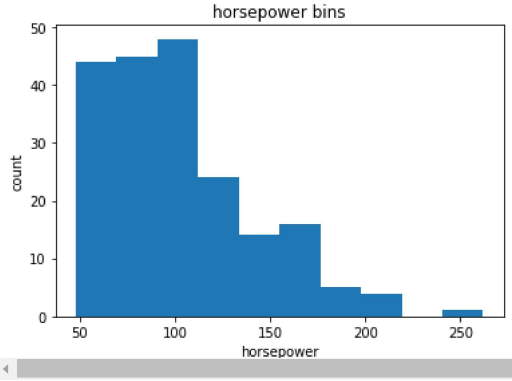
```
<ipython-input-122-b4facc2633f9>:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-co
  df["length"]=df["length"]/df["length"].max()
<ipython-input-122-b4facc2633f9>:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-co
  df["width"]=df["width"]/df["width"].max()
<ipython-input-122-b4facc2633f9>:10: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-co
  df["height"]=df["height"]/df["height"].max()
```

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	bore	stroke	compression-ratio	horsepo
0	4	122	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	3.47	2.68	9.0	
1	4	122	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	3.47	2.68	9.0	
2	2	122	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	2.68	3.47	9.0	
3	3	164	audi	gas	std	four	sedan	fwd	front	99.8	...	3.19	3.40	10.0	
4	3	164	audi	gas	std	four	sedan	4wd	front	99.4	...	3.19	3.40	8.0	

Binning

```
''' Why binning?
Binning is a process of transforming continuous numerical variables into discrete categorical 'bins' for grouped analysis.
Example:
In our dataset, "horsepower" is a real valued variable ranging from 48 to 288 and it has 59 unique values. What if we only care
about the price difference between cars with high horsepower, medium horsepower, and little horsepower (3 types)?
Can we rearrange them into three ‘bins' to simplify analysis?
We will use the pandas method 'cut' to segment the 'horsepower' column into 3 bins.'''
df["horsepower"]=df["horsepower"].astype(int, copy=True)
#converting into correct data format
%matplotlib inline
import matplotlib as plt
from matplotlib import pyplot
plt.pyplot.hist(df["horsepower"])
#Name x n y labels with title
plt.pyplot.xlabel("horsepower")
plt.pyplot.ylabel("count")
plt.pyplot.title("horsepower bins")
```

```
<ipython-input-123-73bb2f6b6121>:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-co
  df["horsepower"]=df["horsepower"].astype(int, copy=True)
Text(0.5, 1.0, 'horsepower bins')
```



```
'''We would like 3 bins of equal size bandwidth so we use numpy's
linspace(start_value, end_value, numbers_generated function.'''
''' Since we want to include the minimum value of horsepower, we want to set start_value = min(df["horsepower"]).
Since we want to include the maximum value of horsepower, we want to set end_value = max(df["horsepower"]).
Since we are building 3 bins of equal length, there should be 4 dividers, so numbers_generated = 4.'''
bins=np.linspace(min(df["horsepower"]),max(df["horsepower"]),4)
bins
```

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

```
#set group names
group_names=['low','medium','high']
```

```
#We apply the function "cut" to determine what each value of df['horsepower'] belongs to.
df["horsepower"]=df["horsepower"].astype(int, copy=True)
df["horsepower-binned"]=pd.cut(df["horsepower"],bins,labels=group_names,include_lowest=True)
df[["horsepower","horsepower-binned"]].head(30)
```

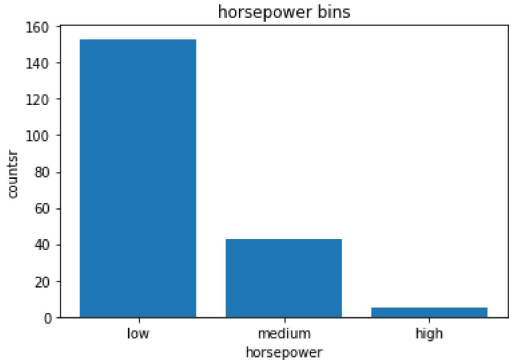
```
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a
  df["horsepower"]=df["horsepower"].astype(int, copy=True)
<ipython-input-126-f6b147709304>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
```

```
#Lets count number of vehicles in each bin
df["horsepower-binned"].value_counts()
```

```
low       153
medium     43
high        5
Name: horsepower-binned, dtype: int64
```

```
#Let's plot the distribution of each bin:
%matplotlib inline
import matplotlib as plt
from matplotlib import pyplot
pyplot.bar(group_names,df["horsepower-binned"].value_counts())
#label x n y axis with title
plt.pyplot.xlabel("horsepower")
plt.pyplot.ylabel("countsr")
plt.pyplot.title("horsepower bins")
```

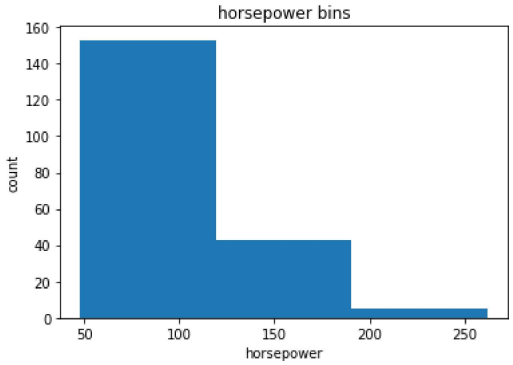
```
Text(0.5, 1.0, 'horsepower bins')
```



19 70 low

```
#Bins Visualization
'''Normally, a histogram is used to visualize the distribution of bins we created above.
%matplotlib inline'''
import matplotlib as plt
from matplotlib import pyplot


# draw historgram of attribute "horsepower" with bins = 3
plt.pyplot.hist(df["horsepower"], bins = 3)

# set x/y labels and plot title
plt.pyplot.xlabel("horsepower")
plt.pyplot.ylabel("count")
plt.pyplot.title("horsepower bins")
```

```
Text(0.5, 1.0, 'horsepower bins')
```



```
'''Indicator Variable (or Dummy Variable)
What is an indicator variable?
An indicator variable (or dummy variable) is a numerical variable used to label categories. They are called 'dummies' because the numbers the
Why we use indicator variables?
We use indicator variables so we can use categorical variables for regression analysis in the later modules.
Example
We see the column "fuel-type" has two unique values: "gas" or "diesel". Regression doesn't understand words, only numbers. To use this attrib
We will use pandas' method 'get_dummies' to assign numerical values to different categories of fuel type.'''
df.columns
```

```
Index(['symboling', 'normalized-losses', 'make', 'fuel-type', 'aspiration',
       'num-of-doors', 'body-style', 'drive-wheels', 'engine-location',
       'wheel-base', 'length', 'width', 'height', 'curb-weight', 'engine-type',
       'num-of-cylinders', 'engine-size', 'fuel-system', 'bore', 'stroke',
       'compression-ratio', 'horsepower', 'peak-rpm', 'city-mpg',
       'highway-mpg', 'price', 'city-L/100km', 'highway-L/100km',
       'horsepower-binned'],
      dtype='object')
```

```
df.dtypes
```

```
symboling              int64
normalized-losses      int64
make                  object
fuel-type             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
horsepower             int64
peak-rpm             float64
city-mpg               int64
highway-mpg            int64
price                float64
city-L/100km         float64
highway-L/100km      float64
horsepower-binned   category
dtype: object
```

```
dummy_variable_1 = pd.get_dummies(df["fuel-type"])
dummy_variable_1.head()
```

	diesel	gas
0	0	1
1	0	1
2	0	1
3	0	1
4	0	1

```
#Change the column names for clarity:

dummy_variable_1.rename(columns={'gas':'fuel-type-gas', 'diesel':'fuel-type-diesel'}, inplace=True)
dummy_variable_1.head()
```

	fuel-type-diesel	fuel-type-gas
0	0	1

```
df.columns
df.head()
#now fuel_type column has been renamed as fuel-type-gas and fiel-type-diesel
```

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	stroke	compression-ratio	horsepower	
0	4	122	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	2.68	9.0	111	5
1	4	122	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	2.68	9.0	111	5
2	2	122	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	3.47	9.0	154	5
3	3	164	audi	gas	std	four	sedan	fwd	front	99.8	...	3.40	10.0	102	5
4	3	164	audi	gas	std	four	sedan	4wd	front	99.4	...	3.40	8.0	115	5

5 rows × 29 columns

```
# get indicator variables of aspiration and assign it to data frame "dummy_variable_2"
dummy_variable_2 = pd.get_dummies(df['aspiration'])

# change column names for clarity
dummy_variable_2.rename(columns={'std':'aspiration-std', 'turbo': 'aspiration-turbo'}, inplace=True)

# show first 5 instances of data frame "dummy_variable_1"
dummy_variable_2.head()
```

	aspiration-std	aspiration-turbo
0	1	0
1	1	0
2	1	0
3	1	0
4	1	0

```
# merge the new dataframe to the original datafram
df = pd.concat([df, dummy_variable_2], axis=1)

# drop original column "aspiration" from "df"
df.drop('aspiration', axis = 1, inplace=True)


df.dtypes
```

```
symboling              int64
normalized-losses      int64
make                  object
fuel-type             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
horsepower              int64
peak-rpm              float64
city-mpg                int64
highway-mpg             int64
price                 float64
city-L/100km          float64
highway-L/100km       float64
horsepower-binned    category
aspiration-std          uint8
aspiration-turbo        uint8
dtype: object
```

```
#Converting or saving this data table as 'clean_df.csv' file after applying all these wanglings
df.to_csv('clean_df.csv')
```

```
#What is the data type of the column "peak-rpm"?
df["peak-rpm"].dtype
```

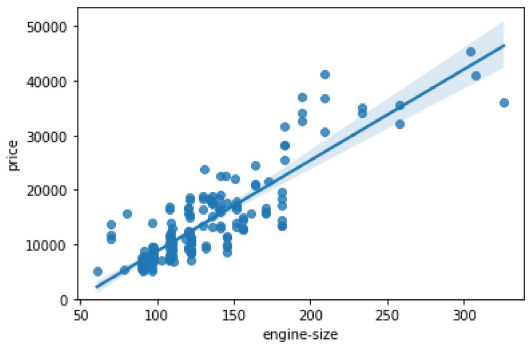
```
dtype('float64')
```

```
#For example, we can calculate the correlation between variables of type "int64" or "float64" using the method "corr":
df.corr()
df[['bore','stroke','compression-ratio','horsepower','engine-size','price']].corr()
#Diagonal element values are always 1 in correalation table
```

	bore	stroke	compression-ratio	horsepower	engine-size	price
bore	1.000000	-0.055390	0.001250	0.566786	0.572516	0.543154
stroke	-0.055390	1.000000	0.187854	0.097598	0.205806	0.082267
compression-ratio	0.001250	0.187854	1.000000	-0.214392	0.028889	0.071107
horsepower	0.566786	0.097598	-0.214392	1.000000	0.822636	0.809729
engine-size	0.572516	0.205806	0.028889	0.822636	1.000000	0.872335
price	0.543154	0.082267	0.071107	0.809729	0.872335	1.000000

```
#Positive linear relationship
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
sns.regplot(x='engine-size',y='price',data=df)
plt.ylim(0,)
```

```
(0.0, 53458.34099175837)
```

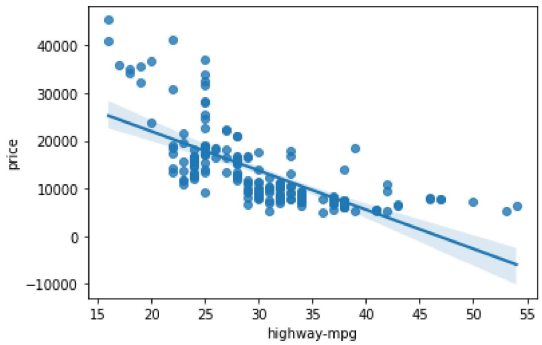



```
df[['engine-size','price']].corr()
'''As the engine-size goes up, the price goes up: this indicates a positive direct correlation between these two variables.
 Engine size seems like a pretty good predictor of price since the regression line is almost a perfect diagonal line.
We can examine the correlation between 'engine-size' and 'price' and see that it's approximately 0.87.'''
```

```
'As the engine-size goes up, the price goes up: this indicates a positive direct correlation between these two variables.\n Engine size
```

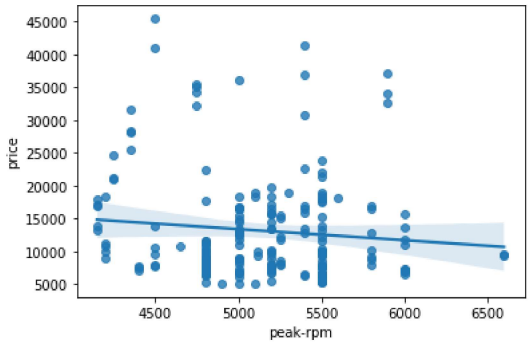
```
#negative linear relationship
sns.regplot(x='highway-mpg',y='price',data=df)
'''As highway-mpg goes up, the price goes down: this indicates an inverse/negative relationship
between these two variables. Highway mpg could potentially be a predictor of price.'''
df[['highway-mpg','price']].corr()
```

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



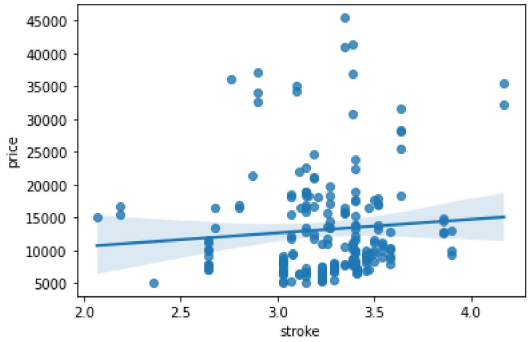

```
#Weak linear relationship
sns.regplot(x="peak-rpm", y="price", data=df)
df[['peak-rpm','price']].corr()
```

	peak-rpm	price
peak-rpm	1.000000	-0.101542
price	-0.101542	1.000000



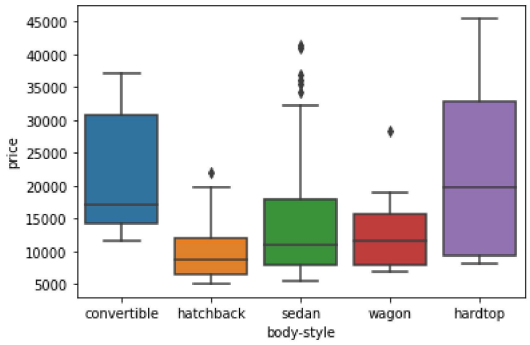

```
df[["stroke","price"]].corr()
sns.regplot(x="stroke", y="price", data=df)
#There is a weak correlation between the variable 'stroke' and 'price.' as such regression will not work well. We can see this using "regplot
```

```
<AxesSubplot:xlabel='stroke', ylabel='price'>
```



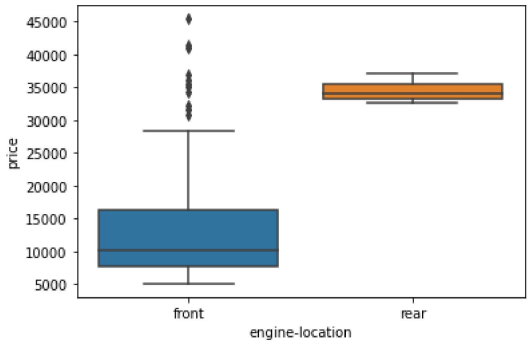
```
#Boxplot correlation
sns.boxplot(x='body-style',y='price',data=df)
'''We see that the distributions of price between the different body-style categories have a significant overlap, so body-style would not be
```

```
'We see that the distributions of price between the different body-style categories have a significant overlap, so body-style would not be a good predictor of price. Let\'s examine engine "engine-location" and "price":'
```



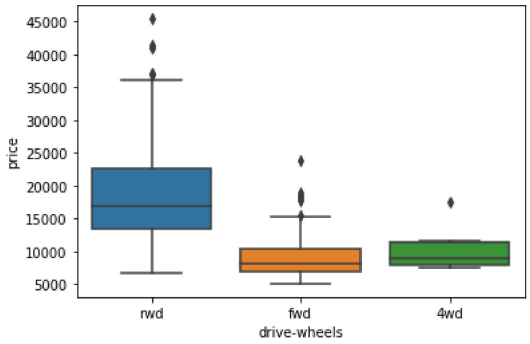

```
sns.boxplot(x="engine-location", y="price", data=df)
'''Here we see that the distribution of price between these two engine-location categories, front and rear, are distinct enough to take engin
```

```
'Here we see that the distribution of price between these two engine-location categories, front and rear, are distinct enough to take engine-location as a potential good predictor of price.'
```




```
# drive-wheels
sns.boxplot(x="drive-wheels", y="price", data=df)
'''Here we see that the distribution of price between the different drive-wheels categories differs. As such, drive-wheels could potentially
```

```
'Here we see that the distribution of price between the different drive-wheels categories differs. As such, drive-wheels could potentially be a predictor of price.'
```




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

	make	fuel-type	num-of-doors	body-style	drive-wheels	engine-location	engine-type	num-of-cylinders	fuel-system
count	201	201	201	201	201	201	201	201	201
unique	22	2	2	5	3	2	6	7	8
top	toyota	gas	four	sedan	fwd	front	ohc	four	mpfi
freq	32	181	115	94	118	198	145	157	92

```
df["drive-wheels"].value_counts()
```

```
fwd    118
rwd     75
4wd      8
Name: drive-wheels, dtype: int64
```

```
df["drive-wheels"].value_counts().to_frame()
```

	drive-wheels
fwd	118
rwd	75
4wd	8

```
#Let's repeat the above steps but save the results to the dataframe "drive_wheels_counts" and rename the column 'drive-wheels' to 'value_coun
drive_wheel_counts=df["drive-wheels"].value_counts().to_frame()
drive_wheel_counts.rename(columns={"drive-wheels":"value_counts"},inplace=True)
drive_wheel_counts
```

	value_counts
fwd	118
rwd	75
4wd	8

```
#Now let's rename the index to 'drive-wheels':
drive_wheel_counts.index.name='drive-wheels'
drive_wheel_counts
```

	value_counts
drive-wheels	
fwd	118
rwd	75
4wd	8

```
#We can repeat the above process for the variable 'engine-location'.

# engine-location as variable
engine_loc_counts = df['engine-location'].value_counts().to_frame()
engine_loc_counts.rename(columns={'engine-location': 'value_counts'}, inplace=True)
engine_loc_counts.index.name = 'engine-location'
engine_loc_counts.head(10)
```

	value_counts
engine-location	
front	198
rear	3

```
df['drive-wheels'].unique()
```

```
array(['rwd', 'fwd', '4wd'], dtype=object)
```

```
#We can select the columns 'drive-wheels', 'body-style' and 'price', then assign it to the variable "df_group_one".
df_group_one = df[['drive-wheels','body-style','price']]
#We can then calculate the average price for each of the different categories of data.
# grouping results
df_group_one = df_group_one.groupby(['drive-wheels'],as_index=False).mean()
df_group_one
```

	drive-wheels	price
0	4wd	10241.000000

```
# grouping results
df_gptest = df[['drive-wheels','body-style','price']]
grouped_test1 = df_gptest.groupby(['drive-wheels','body-style'],as_index=False).mean()
grouped_test1
```

	drive-wheels	body-style	price
0	4wd	hatchback	7603.000000
1	4wd	sedan	12647.333333
2	4wd	wagon	9095.750000
3	fwd	convertible	11595.000000
4	fwd	hardtop	8249.000000
5	fwd	hatchback	8396.387755
6	fwd	sedan	9811.800000
7	fwd	wagon	9997.333333
8	rwd	convertible	23949.600000
9	rwd	hardtop	24202.714286
10	rwd	hatchback	14337.777778
11	rwd	sedan	21711.833333
12	rwd	wagon	16994.222222

```
'''This grouped data is much easier to visualize when it is made into a pivot table. A pivot table is like an Excel spreadsheet,
 with one variable along the column and another along the row. We can convert the dataframe to a pivot table using the method "pivot"
 to create a pivot table from the groups.
In this case, we will leave the drive-wheels variable as the rows of the table, and pivot body-style to become the columns of the table:'''

grouped_pivot = grouped_test1.pivot(index='drive-wheels',columns='body-style')
grouped_pivot
```

	price				
body-style	convertible	hardtop	hatchback	sedan	wagon
drive-wheels					
4wd	NaN	NaN	7603.000000	12647.333333	9095.750000
fwd	11595.0	8249.000000	8396.387755	9811.800000	9997.333333
rwd	23949.6	24202.714286	14337.777778	21711.833333	16994.222222

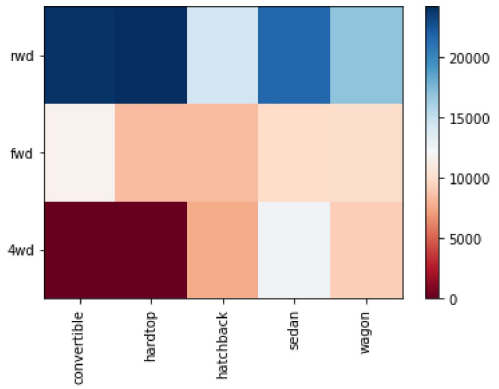
```
grouped_pivot = grouped_pivot.fillna(0) #fill missing values with 0
grouped_pivot
```

	price				
body-style	convertible	hardtop	hatchback	sedan	wagon
drive-wheels					
4wd	0.0	0.000000	7603.000000	12647.333333	9095.750000
fwd	11595.0	8249.000000	8396.387755	9811.800000	9997.333333
rwd	23949.6	24202.714286	14337.777778	21711.833333	16994.222222

```
# grouping results for price n body style
df_gptest2 = df[['body-style','price']]
grouped_test_bodystyle = df_gptest2.groupby(['body-style'],as_index= False).mean()
grouped_test_bodystyle
```

	body-style	price
0	convertible	21890.500000
1	hardtop	22208.500000
2	hatchback	9957.441176

```
fig, ax = plt.subplots()
im = ax.pcolor(grouped_pivot, cmap='RdBu')

#label names
row_labels = grouped_pivot.columns.levels[1]
col_labels = grouped_pivot.index

#move ticks and labels to the center
ax.set_xticks(np.arange(grouped_pivot.shape[1]) + 0.5, minor=False)
ax.set_yticks(np.arange(grouped_pivot.shape[0]) + 0.5, minor=False)

#insert labels
ax.set_xticklabels(row_labels, minor=False)
ax.set_yticklabels(col_labels, minor=False)

#rotate label if too long
plt.xticks(rotation=90)

fig.colorbar(im)
plt.show()
```




```
#To get a better measure of the important characteristics, we look at the correlation of these variables with the car price.
#In other words: how is the car price dependent on this variable?
'''Correlation and Causation
Correlation: a measure of the extent of interdependence between variables.

Causation: the relationship between cause and effect between two variables.

It is important to know the difference between these two. Correlation does not imply causation. Determining correlation is much simpler the d

Pearson Correlation

The Pearson Correlation measures the linear dependence between two variables X and Y.

The resulting coefficient is a value between -1 and 1 inclusive, where:

1: Perfect positive linear correlation.
0: No linear correlation, the two variables most likely do not affect each other.
-1: Perfect negative linear correlation.
Pearson Correlation is the default method of the function "corr". Like before, we can calculate the Pearson Correlation of the of the 'int64'
from scipy import stats
#Wheel-Base vs. Price
#Let's calculate the Pearson Correlation Coefficient and P-value of 'wheel-base' and 'price'.
#Here,correlation coefficient should be more and p value should be less then we can say that there is significant correlation between variabl
pearson_coef,p_value=stats.pearsonr(df["wheel-base"],df["price"])
print("The pearson coefficient is",pearson_coef,"with p-value as",p_value)


#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).
```

```
The pearson coefficient is 0.584641822265508 with p-value as 8.076488270733218e-20
```

```
'''Horsepower vs. Price
Let's calculate the Pearson Correlation Coefficient and P-value of 'horsepower' 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_value)
The Pearson Correlation Coefficient is 0.8095745670036559  with a P-value of P =  6.369057428260101e-48
Conclusion:
Since the p-value is  <0.001, the correlation between horsepower and price is statistically significant, and the linear relationship is quite

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_value)
The Pearson Correlation Coefficient is 0.6906283804483643  with a P-value of P =  8.01647746615853e-30
Conclusion:
Since the p-value is  < 0.001, the correlation between length and price is statistically significant, and the linear relationship is moderate

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_value )
The Pearson Correlation Coefficient is 0.7512653440522666  with a P-value of P = 9.200335510483739e-38
Conclusion:
Since the p-value is < 0.001, the correlation between width and price is statistically significant, and the linear relationship is quite stro

Curb-Weight vs. Price
Let's calculate the Pearson Correlation Coefficient and P-value of 'curb-weight' and '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 = ", p_value)
The Pearson Correlation Coefficient is 0.8344145257702845  with a P-value of P =  2.189577238893816e-53
Conclusion:
Since the p-value is  <0.001, the correlation between curb-weight and price is statistically significant, and the linear relationship is quit

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_value)
The Pearson Correlation Coefficient is 0.8723351674455188  with a P-value of P = 9.265491622196808e-64
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
Let's calculate the Pearson Correlation Coefficient and P-value of 'bore' and '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 =  ", p_value )
The Pearson Correlation Coefficient is 0.54315538326266  with a P-value of P =   8.049189483935489e-17
Conclusion:
Since the p-value is  < 0.001, the correlation between bore and price is statistically significant, but the linear relationship is only moder

We can relate the process for each 'city-mpg' and 'highway-mpg':

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_value)
The Pearson Correlation Coefficient is -0.6865710067844684  with a P-value of P =  2.3211320655672453e-29
Conclusion:
Since the p-value is  <0.001, the correlation between city-mpg and price is statistically significant, and the coefficient of about -0.687 sh
  negative and moderately strong.

Highway-mpg vs. Price
pearson_coef, p_value = stats.pearsonr(df['highway-mpg'], df['price'])
print( "The Pearson Correlation Coefficient is", pearson_coef, " with a P-value of P = ", p_value )
The Pearson Correlation Coefficient is -0.7046922650589534  with a P-value of P =  1.749547114447437e-31
Conclusion:
Since the p-value is < 0.001, the correlation between highway-mpg and price is statistically significant, and the coefficient of about -0.705
 that the relationship is negative and moderately strong.'''
```

```
'Horsepower vs. Price\nLet\'s calculate the Pearson Correlation Coefficient and P-value of \'horsepower\' and \'price\'.\n\npearson_coe
f, p_value = stats.pearsonr(df[\'horsepower\'], df[\'price\'])\nprint("The Pearson Correlation Coefficient is", pearson_coef, " with a
P-value of P = ", p_value)  \nThe Pearson Correlation Coefficient is 0.8095745670036559  with a P-value of P =  6.369057428260101e-48\n
Conclusion:\nSince the p-value is  <0.001, the correlation between horsepower and price is statistically significant, and the linear re
lationship is quite strong (~0.809, close to 1).\n\nLength vs. Price\nLet\'s calculate the Pearson Correlation Coefficient and P-value
of \'length\' and \'price\'.\n\npearson_coef, p_value = stats.pearsonr(df[\'length\'], df[\'price\'])\nprint("The Pearson Correlation C
```

```
'''ANOVA: Analysis of Variance
The Analysis of Variance (ANOVA) is a statistical method used to test whether there are significant differences between the means of
 two or more groups. ANOVA returns two parameters:

F-test score: ANOVA assumes the means of all groups are the same, calculates how much the actual means deviate from the assumption,
and reports it as the F-test score. A larger score means there is a larger difference between the means.

P-value: P-value tells how statistically significant our calculated score value is.

If our price variable is strongly correlated with the variable we are analyzing, we expect ANOVA to return a sizeable F-test score and a smal

Drive Wheels
Since ANOVA analyzes the difference between different groups of the same variable, the groupby function will come in handy. Because the ANOVA
algorithm averages the data automatically, we do not need to take the average before hand.

To see if different types of 'drive-wheels' impact 'price', we group the data.'''
grouped_test2=df_gptest[['drive-wheels', 'price']].groupby(['drive-wheels'])
grouped_test2.head()
```

	drive-wheels	price
0	rwd	13495.0
1	rwd	16500.0
2	rwd	16500.0
3	fwd	13950.0
4	4wd	17450.0
5	fwd	15250.0
6	fwd	17710.0
7	fwd	18920.0
8	fwd	23875.0
9	rwd	16430.0
10	rwd	16925.0
136	4wd	7603.0
140	4wd	9233.0
141	4wd	11259.0
144	4wd	8013.0

```
df_gptest
```

```
          drive-wheels  body-style    price
#We can obtain the values of the method group using the method "get_group".

grouped_test2.get_group('4wd')['price']

     4      17450.0
     136     7603.0
     140     9233.0
     141    11259.0
     144     8013.0
     145    11694.0
     150     7898.0
     151     8778.0
     Name: price, dtype: float64


#We can use the function 'f_oneway' in the module 'stats' to obtain the F-test score and P-value.

# ANOVA
f_val, p_val = stats.f_oneway(grouped_test2.get_group('fwd')['price'], grouped_test2.get_group('rwd')['price'], grouped_test2.get_group('4wd'

print( "ANOVA results: F=", f_val, ", P =", p_val)

     ANOVA results: F= 67.95406500780399 , P = 3.3945443577151245e-23


'''This is a great result with a large F-test score showing a strong correlation and a P-value of almost 0 implying almost certain
statistical significance. But does this mean all three tested groups are all this highly correlated?

Let's examine them separately.

fwd and rwd
f_val, p_val = stats.f_oneway(grouped_test2.get_group('fwd')['price'], grouped_test2.get_group('rwd')['price'])

print( "ANOVA results: F=", f_val, ", P =", p_val )
ANOVA results: F= 130.5533160959111 , P = 2.2355306355677845e-23
Let's examine the other groups.

4wd and rwd
f_val, p_val = stats.f_oneway(grouped_test2.get_group('4wd')['price'], grouped_test2.get_group('rwd')['price'])

print( "ANOVA results: F=", f_val, ", P =", p_val)
ANOVA results: F= 8.580681368924756 , P = 0.004411492211225333
4wd and fwd
f_val, p_val = stats.f_oneway(grouped_test2.get_group('4wd')['price'], grouped_test2.get_group('fwd')['price'])

print("ANOVA results: F=", f_val, ", P =", p_val)
ANOVA results: F= 0.665465750252303 , P = 0.41620116697845655
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.'''

     'This is a great result with a large F-test score showing a strong correlation and a P-value of almost 0 implying almost certain \nstat
     istical significance. But does this mean all three tested groups are all this highly correlated?\n\nLet\'s examine them separately.\n\n
     fwd and rwd\nf_val, p_val = stats.f_oneway(grouped_test2.get_group(\'fwd\')[\'price\'], grouped_test2.get_group(\'rwd\')[\'price\'])
     \n \nprint( "ANOVA results: F=", f_val, ", P =", p_val )\nANOVA results: F= 130.5533160959111 , P = 2.2355306355677845e-23\nLet\'s exam
     ine the other groups.\n\n4wd and rwd\nf_val, p_val = stats.f_oneway(grouped_test2.get_group(\'4wd\')[\'price\'], grouped_test2.get_grou
     p(\'rwd\')[\'price\'])  \n   \nprint( "ANOVA results: F=", f_val, ", P =", p_val)   \nANOVA results: F= 8.580681368924756 , P = 0.00441
     1492211225333\n4wd and fwd\nf_val, p_val = stats.f_oneway(grouped_test2.get_group(\'4wd\')[\'price\'], grouped_test2.get_group(\'fwd\')
     [\'price\'])  \n \nprint("ANOVA results: F=", f_val, ", P =
```

```
#Linear Regression
from sklearn.linear_model import LinearRegression
lm = LinearRegression()
lm
X = df[['highway-mpg']]
Y = df['price']
lm.fit(X,Y)
Yhat=lm.predict(X)
Yhat[0:5]
```

```
array([16236.50464347, 16236.50464347, 17058.23802179, 13771.3045085 ,
       20345.17153508])
```

```
lm.intercept_
```

```
38423.3058581574
```

```
lm.coef_
```

```
array([-821.73337832])
```

```
#Plugging in the actual values we get:
#Price = 38423.31 - 821.73 x highway-mpg
```

```
#Multiple linear regression
Z=df[['horsepower', 'curb-weight', 'engine-size', 'highway-mpg']]
lm.fit(Z, df['price'])
print(lm.intercept_)
print(lm.coef_)
#4 variables and 4 coefficients
#Price = -15678.742628061467 + 52.65851272 x horsepower + 4.69878948 x curb-weight + 81.95906216 x engine-size + 33.58258185 x highway-mpg
```

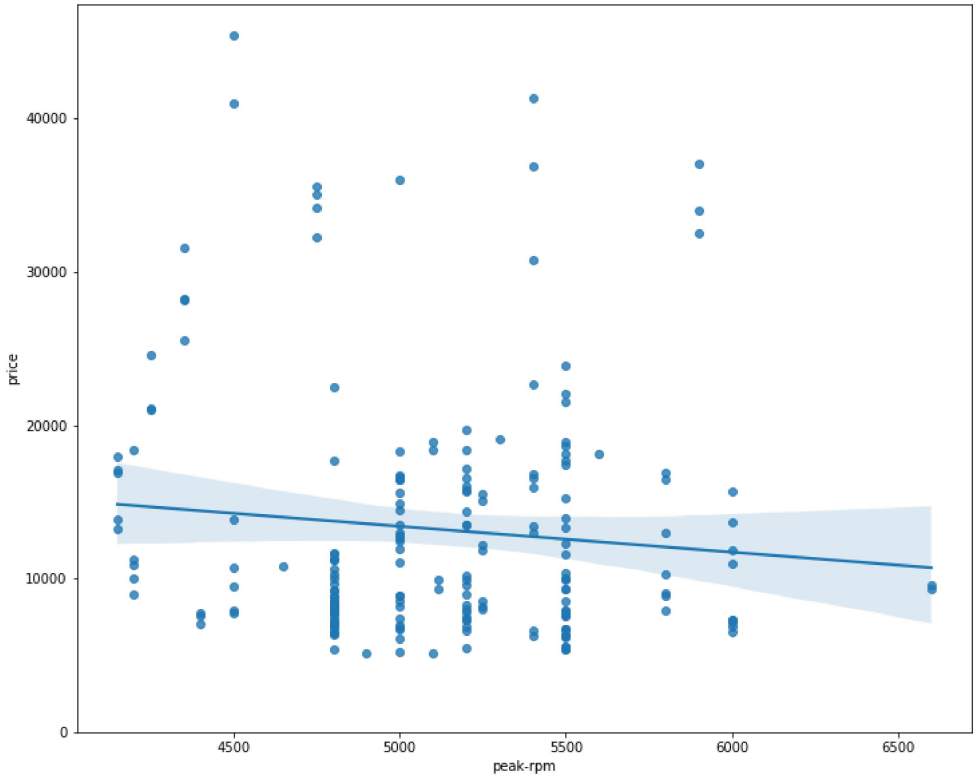
```
-15831.93096029948
[53.66247317  4.70938694 81.44600167 36.55016267]
```

```
#Regression plot
#Let's visualize highway-mpg as potential predictor variable of price:

width = 12
height = 10
plt.figure(figsize=(width, height))
sns.regplot(x="highway-mpg", y="price", data=df)
plt.ylim(0,)
```

```
(0.0, 48168.76753168157)
```

```
#Let's compare this plot to the regression plot of "peak-rpm".

plt.figure(figsize=(width, height))
sns.regplot(x="peak-rpm", y="price", data=df)
plt.ylim(0,)
```

```
(0.0, 47414.1)
```

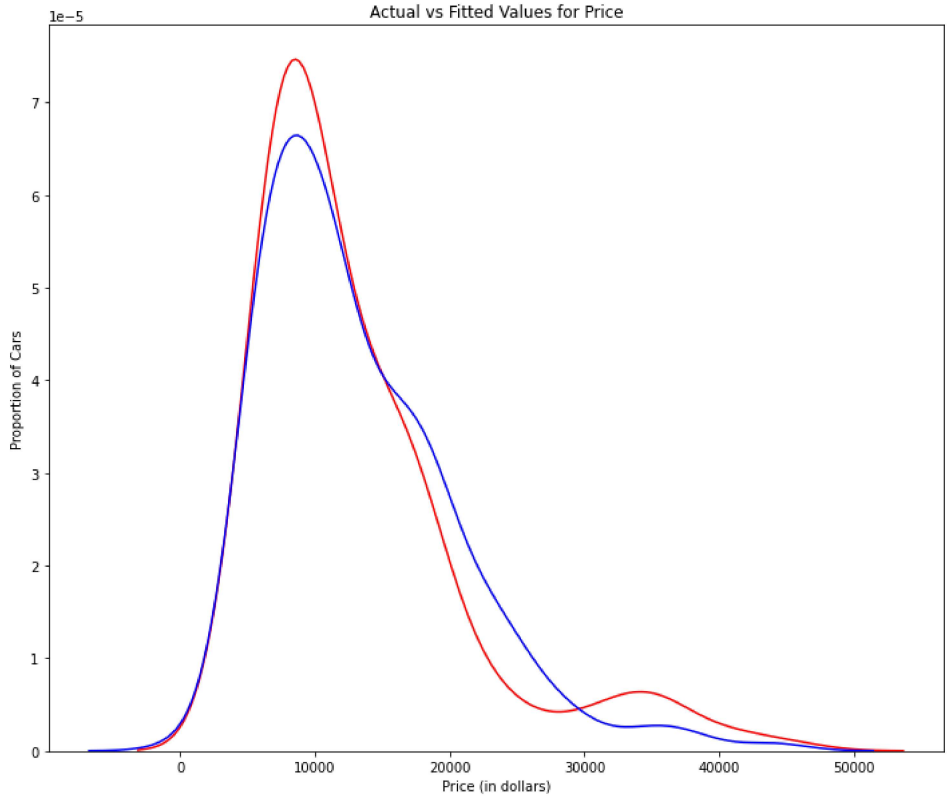



```
#Residual plot
width = 12
height = 10
plt.figure(figsize=(width, height))
sns.residplot(x=df['highway-mpg'],y=df['price'])
plt.show()
```


```
#Distribution plot
#Lets make prediction first
Y_hat = lm.predict(Z)
```

```
plt.figure(figsize=(width, height))


ax1 = sns.distplot(df['price'], hist=False, color="r", label="Actual Value")
sns.distplot(Y_hat, hist=False, color="b", label="Fitted Values" , ax=ax1)


plt.title('Actual vs Fitted Values for Price')
plt.xlabel('Price (in dollars)')
plt.ylabel('Proportion of Cars')

plt.show()
plt.close()
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be rem
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.8/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be rem
  warnings.warn(msg, FutureWarning)
```

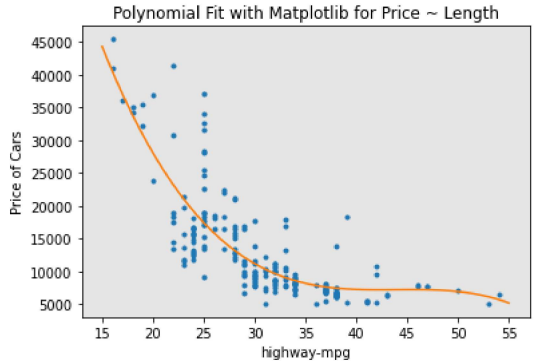



```
#Polynomial Regression
def PlotPolly(model, independent_variable, dependent_variabble, Name):
    x_new = np.linspace(15, 55, 100)
    y_new = model(x_new)

    plt.plot(independent_variable, dependent_variabble, '.', x_new, y_new, '-')
    plt.title('Polynomial Fit with Matplotlib for Price ~ Length')
    ax = plt.gca()
    ax.set_facecolor((0.898, 0.898, 0.898))
    fig = plt.gcf()
    plt.xlabel(Name)
    plt.ylabel('Price of Cars')
```

```
    plt.show()
    plt.close()


#Let's get the variables:

x = df['highway-mpg']
y = df['price']


#Let's fit the polynomial using the function polyfit, then use the function poly1d to display the polynomial function.

# Here we use a polynomial of the 3rd order (cubic)
f = np.polyfit(x, y, 3)
p = np.poly1d(f)
print(p)
```

```
            3         2
    -1.557 x + 204.8 x - 8965 x + 1.379e+05
```

```
#Let's plot the function:

PlotPolly(p, x, y, 'highway-mpg')
```

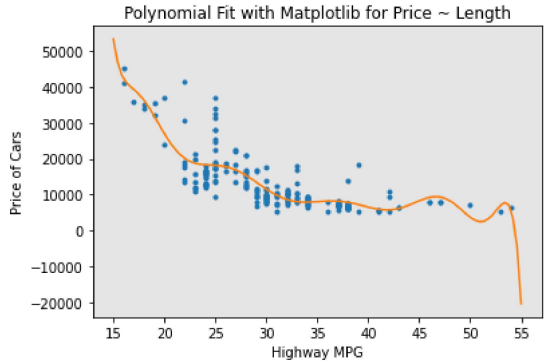



```
np.polyfit(x, y, 3)
```

```
    array([-1.55663829e+00,  2.04754306e+02, -8.96543312e+03,  1.37923594e+05])
```

```
#Create 11 order polynomial model with the variables x and y from above.
f1 = np.polyfit(x, y, 11)
p1 = np.poly1d(f1)
print(p1)
PlotPolly(p1,x,y, 'Highway MPG')
```

```
             11             10             9           8         7
    -1.243e-08 x  + 4.722e-06 x  - 0.0008028 x + 0.08056 x - 5.297 x
              6        5             4             3             2
     + 239.5 x - 7588 x + 1.684e+05 x - 2.565e+06 x + 2.551e+07 x - 1.491e+08 x + 3.879e+08
```




```
#We can perform a polynomial transform on multiple features. First, we import the module:

from sklearn.preprocessing import PolynomialFeatures
#We create a PolynomialFeatures object of degree 2:
```

```
pr=PolynomialFeatures(degree=2)
pr   PolynomialFeatures()


Z_pr=pr.fit_transform(Z)


Z.shape

     (201, 4)


#After the transformation, there are 201 samples and 15 features.

Z_pr.shape

     (201, 15)


#Pipeline
#Data Pipelines simplify the steps of processing the data. We use the module Pipeline to create a pipeline. We also use StandardScaler as a s

from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
#We create the pipeline by creating a list of tuples including the name of the model or estimator and its corresponding constructor.

Input=[('scale',StandardScaler()), ('polynomial', PolynomialFeatures(include_bias=False)), ('model',LinearRegression())]



#We input the list as an argument to the pipeline constructor:

pipe=Pipeline(Input)
pipe

     Pipeline(steps=[('scale', StandardScaler()),
                     ('polynomial', PolynomialFeatures(include_bias=False)),
                     ('model', LinearRegression())])


#First, we convert the data type Z to type float to avoid conversion warnings that may appear as a result of StandardScaler taking float inpu

#Then, we can normalize the data, perform a transform and fit the model simultaneously.

Z = Z.astype(float)
pipe.fit(Z,y)

     Pipeline(steps=[('scale', StandardScaler()),
                     ('polynomial', PolynomialFeatures(include_bias=False)),
                     ('model', LinearRegression())])


ypipe=pipe.predict(Z)
ypipe[0:4]

     array([13103.67557905, 13103.67557905, 18229.84126783, 10394.17656982])


#Measures for In-Sample Evaluation
#R^2 / R-squared
#Mean Squared Error (MSE)
#highway_mpg_fit
lm.fit(X, Y)
# Find the R^2
print('The R-square is: ', lm.score(X, Y))

     The R-square is:  0.4965911884339176


#Model 1:Simple linear regression
#Let's calculate the MSE:
#We can predict the output i.e., "yhat" using the predict method, where X is the input variable:

Yhat=lm.predict(X)
print('The output of the first four predicted value is: ', Yhat[0:4])
#Let's import the function mean_squared_error from the module metrics:

from sklearn.metrics import mean_squared_error
#We can compare the predicted results with the actual results:

mse = mean_squared_error(df['price'], Yhat)
```

```
print('The mean square error of price and predicted value is: ', mse)
```

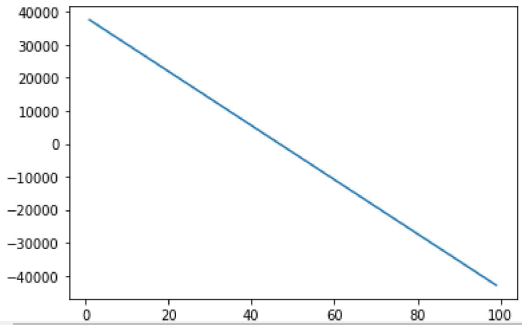
```
The output of the first four predicted value is:  [16236.50464347 16236.50464347 17058.23802179 13771.3045085 ]
The mean square error of price and predicted value is:  31635042.944639888
```

```
#Model 2: Multiple Linear Regression
#Let's calculate the R^2:

# fit the model
lm.fit(Z, df['price'])
# Find the R^2
print('The R-square is: ', lm.score(Z, df['price']))
#We can say that ~80.896 % of the variation of price is explained by this multiple linear regression "multi_fit".

#Let's calculate the MSE.

#We produce a prediction:

Y_predict_multifit = lm.predict(Z)
#We compare the predicted results with the actual results:

print('The mean square error of price and predicted value using multifit is: ', \
      mean_squared_error(df['price'], Y_predict_multifit))
```

```
The R-square is:  0.80943904228153
The mean square error of price and predicted value using multifit is:  11975165.99330355
```

```
# Prediction and Decision Making
#Prediction
#In the previous section, we trained the model using the method fit. Now we will use the method predict to produce a prediction. Lets import

import matplotlib.pyplot as plt
import numpy as np

%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
#Create a new input:

new_input=np.arange(1, 100, 1).reshape(-1, 1)
#Fit the model:

lm.fit(X, Y)
lm

#Produce a prediction:

yhat=lm.predict(new_input)
yhat[0:5]
#We can plot the data:

plt.plot(new_input, yhat)
plt.show()
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/base.py:450: UserWarning: X does not have valid feature names, but LinearRegression was f
  warnings.warn(
```




```
'''Comparing these three models, we conclude that the MLR model is the best model to be able to predict price from our dataset.
This result makes sense since we have 27 variables in total and
```

```
we know that more than one of those variables are potential predictors of the final car price.'''
```

```
'Comparing these three models, we conclude that the MLR model is the best model to be able to predict price from our dataset. \nThis re
sult makes sense since we have 27 variables in total and\n we know that more than one of those variables are potential predictors of th
e final car price.'
```

✓ 0s completed at 7:25 PM