Exploratory Data Analysis

Estimated time needed: **30** minutes

Objectives

After completing this lab you will be able to:

- Explore features or characteristics to predict price of car
- Analyze patterns and run descriptive statistical analysis
- Group data based on identified parameters and create pivot tables
- Identify the effect of independent attributes on price of cars

Import Data from Module 2

Import libraries:

```
#install specific version of libraries used in lab
#! mamba install pandas==1.3.3
#! mamba install numpy=1.21.2
#! mamba install scipy=1.7.1-y
#!  mamba install seaborn=0.9.0-y

import pandas as pd
import numpy as np
import piplite
await piplite.install('seaborn')
```

```
<ipython-input-2-cec0a3d86e2d>:1: DeprecationWarning: 
Pyarrow will become a required dependency of pandas in the next major release of pandas (pandas 3.0),
(to allow more performant data types, such as the Arrow string type, and better interoperability with other libraries)
but was not found to be installed on your system.
If this would cause problems for you,
please provide us feedback at https://github.com/pandas-dev/pandas/issues/54466
        
  import pandas as pd
```

Download the updated dataset by running the cell below.

The functions below will download the dataset into your browser and store it in dataframe `df`:

```
from pyodide.http import pyfetch

async def download(url, filename):
```

```
    response = await pyfetch(url)
    if response.status == 200:
        with open(filename, "wb") as f:
            f.write(await response.bytes())


file_path= "https://cf-courses-data.s3.us.cloud-object-
storage.appdomain.cloud/IBMDeveloperSkillsNetwork-DA0101EN-
SkillsNetwork/labs/Data%20files/automobileEDA.csv"

await download(file_path, "usedcars.csv")
file_name="usedcars.csv"

df = pd.read_csv(file_name, header=0)
```

Note: This version of the lab is working on JupyterLite, which requires the dataset to be downloaded to the interface.While working on the downloaded version of this notebook on their local machines(Jupyter Anaconda), the learners can simply **skip the steps above,** and simply use the URL directly in the `pandas.read_csv()` function. You can uncomment and run the statements in the cell below.

```
#filepath='https://cf-courses-data.s3.us.cloud-object-
storage.appdomain.cloud/IBMDeveloperSkillsNetwork-DA0101EN-
SkillsNetwork/labs/Data%20files/automobileEDA.csv'
#df = pd.read_csv(filepath, header=None)
```

View the first 5 values of the updated dataframe using `dataframe.head()`

```
df.head()

   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

  city-L/100km  horsepower-binned  diesel  gas
0    11.190476             Medium       0    1
1    11.190476             Medium       0    1
2    12.368421             Medium       0    1
3     9.791667             Medium       0    1
4    13.055556             Medium       0    1

[5 rows x 29 columns]
```

Analyzing Individual Feature Patterns Using Visualization

To install Seaborn we use pip, the Python package manager.

Import visualization packages "Matplotlib" and "Seaborn". Don't forget about "%matplotlib inline" to plot in a Jupyter notebook.

```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

# list the data types for each column
print(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
horsepower             float64
peak-rpm               float64
city-mpg                 int64
highway-mpg              int64
price                  float64
city-L/100km           float64
horsepower-binned       object
diesel                   int64
gas                      int64
dtype: object
```

```
# Write your code below and press Shift+Enter to execute
df['peak-rpm'].dtypes
```

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

For example, we can calculate the correlation between variables of type "int64" or "float64" using the method "corr":

```
numeric_df = df.select_dtypes(include=['number'])
correlation_matrix = numeric_df.corr()
print(correlation_matrix)
```

```
                   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             -0.008245           0.055563    0.158502  0.124139
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

                      width    height  curb-weight  engine-size
bore  \
symboling         -0.242423 -0.550160    -0.233118    -0.110581 -
0.140019
normalized-losses  0.086802 -0.373737     0.099404     0.112360 -
0.029862
wheel-base         0.814507  0.590742     0.782097     0.572027
0.493244
length             0.857170  0.492063     0.880665     0.685025
0.608971
width              1.000000  0.306002     0.866201     0.729436
0.544885
height             0.306002  1.000000     0.307581     0.074694
0.180449
curb-weight        0.866201  0.307581     1.000000     0.849072
0.644060
engine-size        0.729436  0.074694     0.849072     1.000000
0.572609
bore               0.544885  0.180449     0.644060     0.572609
1.000000
stroke             0.188829 -0.062704     0.167562     0.209523 -
0.055390
compression-ratio  0.189867  0.259737     0.156433     0.028889
0.001263
horsepower         0.615077 -0.087027     0.757976     0.822676
```

```
0.566936
peak-rpm          -0.245800 -0.309974    -0.279361    -0.256733 -
0.267392
city-mpg          -0.633531 -0.049800    -0.749543    -0.650546 -
0.582027
highway-mpg       -0.680635 -0.104812    -0.794889    -0.679571 -
0.591309
price              0.751265  0.135486     0.834415     0.872335
0.543155
city-L/100km       0.673363  0.003811     0.785353     0.745059
0.554610
diesel             0.244356  0.281578     0.221046     0.070779
0.054458
gas               -0.244356 -0.281578    -0.221046    -0.070779 -
0.054458

                     stroke  compression-ratio  horsepower  peak-
rpm  \
symboling         -0.008245          -0.182196    0.075819  0.279740

normalized-losses  0.055563          -0.114713    0.217299  0.239543

wheel-base         0.158502           0.250313    0.371147 -0.360305

length             0.124139           0.159733    0.579821 -0.285970

width              0.188829           0.189867    0.615077 -0.245800

height            -0.062704           0.259737   -0.087027 -0.309974

curb-weight        0.167562           0.156433    0.757976 -0.279361

engine-size        0.209523           0.028889    0.822676 -0.256733

bore              -0.055390           0.001263    0.566936 -0.267392

stroke             1.000000           0.187923    0.098462 -0.065713

compression-ratio  0.187923           1.000000   -0.214514 -0.435780

horsepower         0.098462          -0.214514    1.000000  0.107885

peak-rpm          -0.065713          -0.435780    0.107885  1.000000

city-mpg          -0.034696           0.331425   -0.822214 -0.115413

highway-mpg       -0.035201           0.268465   -0.804575 -0.058598

price              0.082310           0.071107    0.809575 -0.101616

city-L/100km       0.037300          -0.299372    0.889488  0.115830
```

```
diesel               0.241303            0.985231   -0.169053 -0.475812

gas                 -0.241303           -0.985231    0.169053  0.475812


                   city-mpg  highway-mpg     price  city-L/100km
diesel  \
symboling         -0.035527     0.036233 -0.082391      0.066171 -
0.196735
normalized-losses -0.225016    -0.181877  0.133999      0.238567 -
0.101546
wheel-base        -0.470606    -0.543304  0.584642      0.476153
0.307237
length            -0.665192    -0.698142  0.690628      0.657373
0.211187
width             -0.633531    -0.680635  0.751265      0.673363
0.244356
height            -0.049800    -0.104812  0.135486      0.003811
0.281578
curb-weight       -0.749543    -0.794889  0.834415      0.785353
0.221046
engine-size       -0.650546    -0.679571  0.872335      0.745059
0.070779
bore              -0.582027    -0.591309  0.543155      0.554610
0.054458
stroke            -0.034696    -0.035201  0.082310      0.037300
0.241303
compression-ratio  0.331425     0.268465  0.071107     -0.299372
0.985231
horsepower        -0.822214    -0.804575  0.809575      0.889488 -
0.169053
peak-rpm          -0.115413    -0.058598 -0.101616      0.115830 -
0.475812
city-mpg           1.000000     0.972044 -0.686571     -0.949713
0.265676
highway-mpg        0.972044     1.000000 -0.704692     -0.930028
0.198690
price             -0.686571    -0.704692  1.000000      0.789898
0.110326
city-L/100km      -0.949713    -0.930028  0.789898      1.000000 -
0.241282
diesel             0.265676     0.198690  0.110326     -0.241282
1.000000
gas               -0.265676    -0.198690 -0.110326      0.241282 -
1.000000

                        gas
symboling          0.196735
normalized-losses  0.101546
```

```
wheel-base          -0.307237
length              -0.211187
width               -0.244356
height              -0.281578
curb-weight         -0.221046
engine-size         -0.070779
bore                -0.054458
stroke              -0.241303
compression-ratio   -0.985231
horsepower           0.169053
peak-rpm             0.475812
city-mpg            -0.265676
highway-mpg         -0.198690
price               -0.110326
city-L/100km         0.241282
diesel              -1.000000
gas                  1.000000
```

The diagonal elements are always one; we will study correlation more precisely Pearson correlation in-depth at the end of the notebook.

```
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	0.187923	0.098462
compression-ratio	0.001263	0.187923	1.000000	-0.214514
horsepower	0.566936	0.098462	-0.214514	1.000000

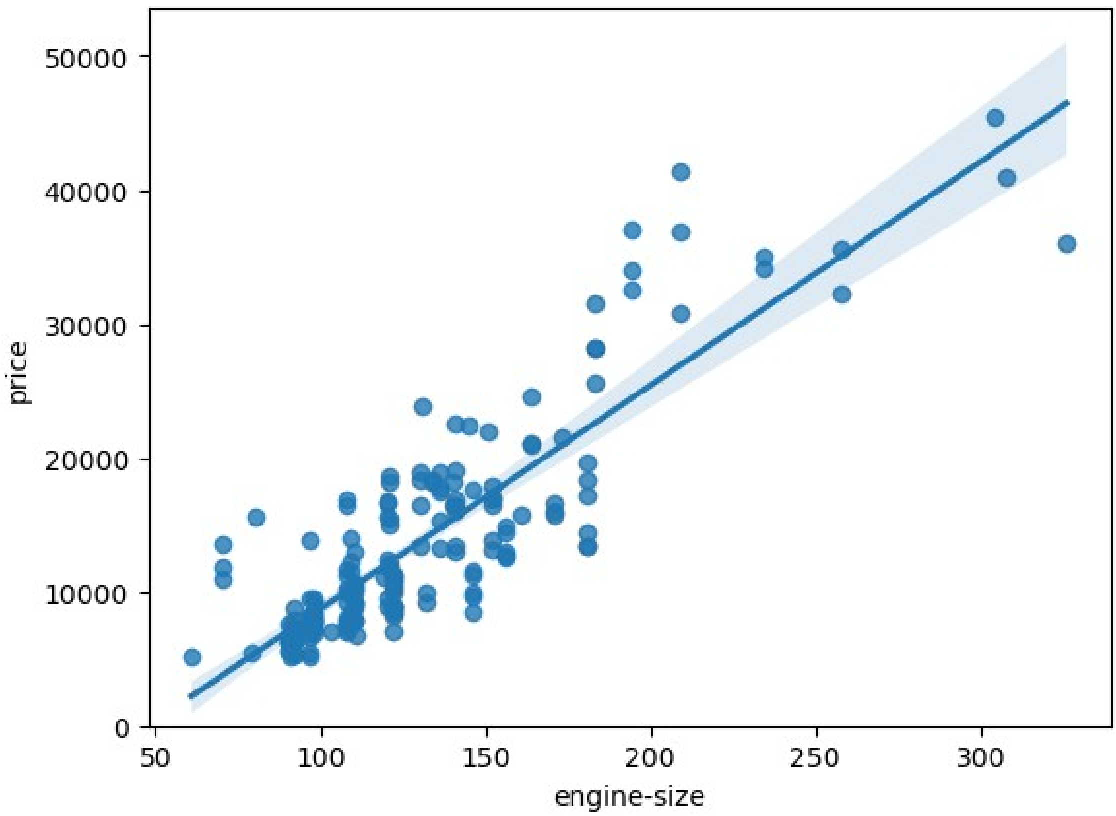
Let's see several examples of different linear relationships:

Positive Linear Relationship

Let's find the scatterplot of "engine-size" and "price".

```
# Engine size as potential predictor variable of price
sns.regplot(x="engine-size", y="price", data=df)
plt.ylim(0,)
```

```
(0.0, 53529.821087193704)
```



We can examine the correlation between 'engine-size' and 'price' and see that it's approximately 0.87.

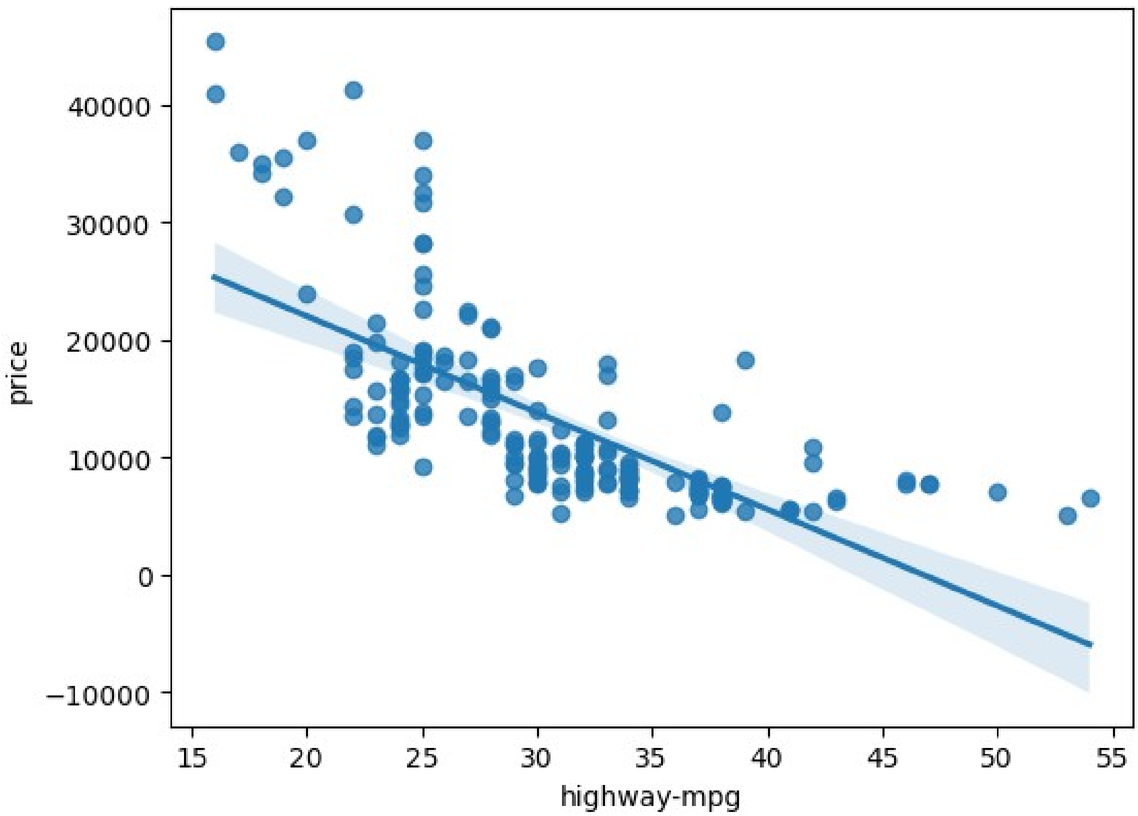
```
df[["engine-size", "price"]].corr()
```

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

Highway mpg is a potential predictor variable of price. Let's find the scatterplot of "highway-mpg" and "price".

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

```
<AxesSubplot:xlabel='highway-mpg', ylabel='price'>
```



We can examine the correlation between 'highway-mpg' and 'price' and see it's approximately -0.704.

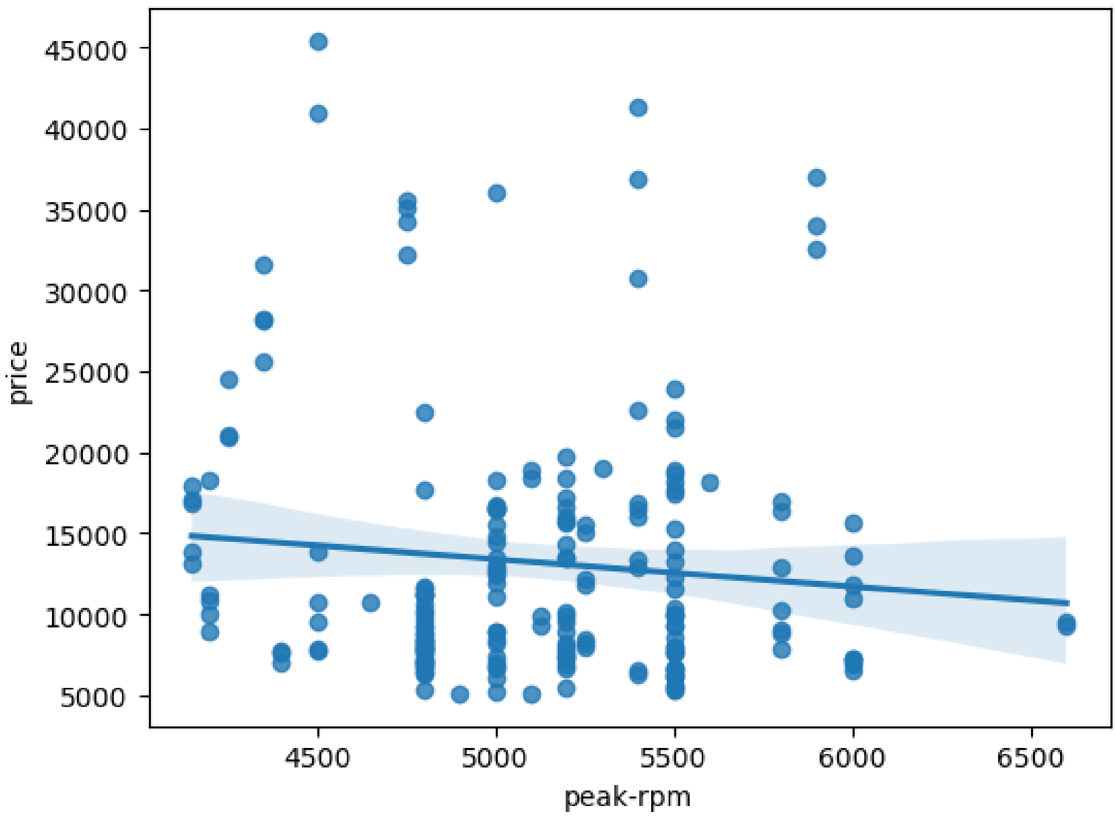
```
df[['highway-mpg', 'price']].corr()
```

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

Let's see if "peak-rpm" is a predictor variable of "price".

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

```
<AxesSubplot:xlabel='peak-rpm', ylabel='price'>
```



We can examine the correlation between 'peak-rpm' and 'price' and see it's approximately -0.101616.

```
df[['peak-rpm','price']].corr()
```

	peak-rpm	price
peak-rpm	1.000000	-0.101616
price	-0.101616	1.000000

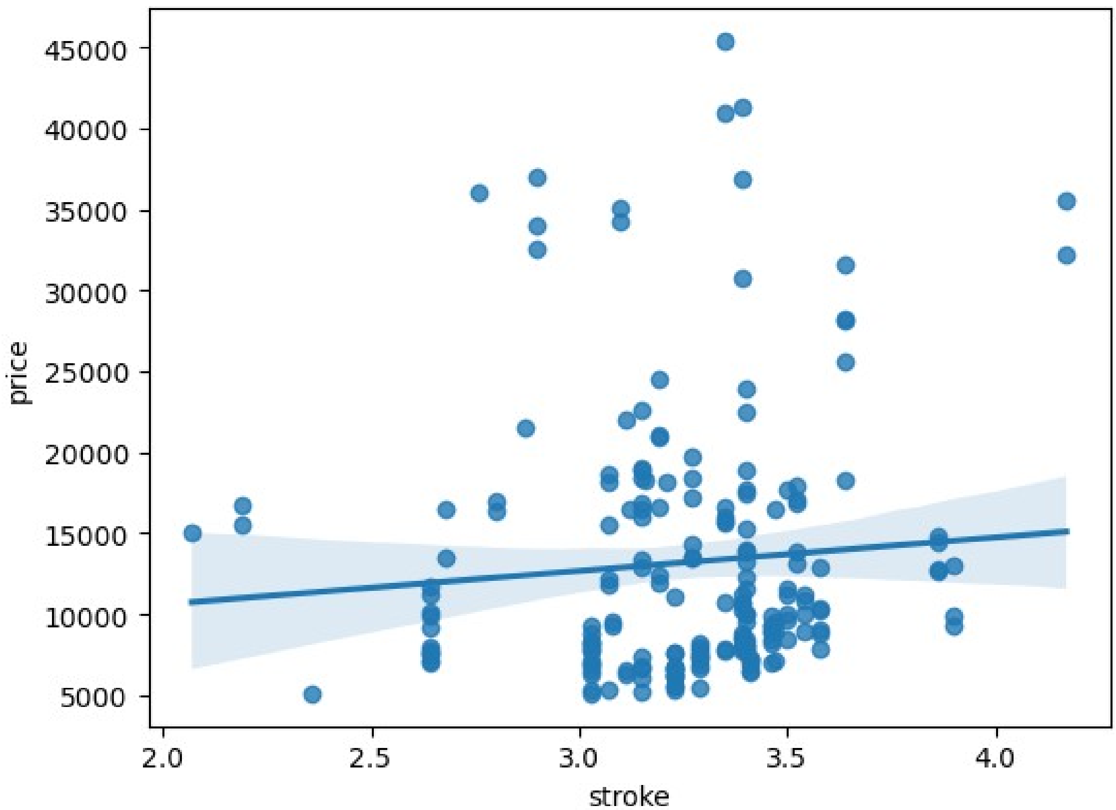
Question 3 a):

```
# Write your code below and press Shift+Enter to execute
df[["stroke","price"]].corr()
```

	stroke	price
stroke	1.00000	0.08231
price	0.08231	1.00000

```
sns.regplot(x="stroke", y="price", data=df)
```

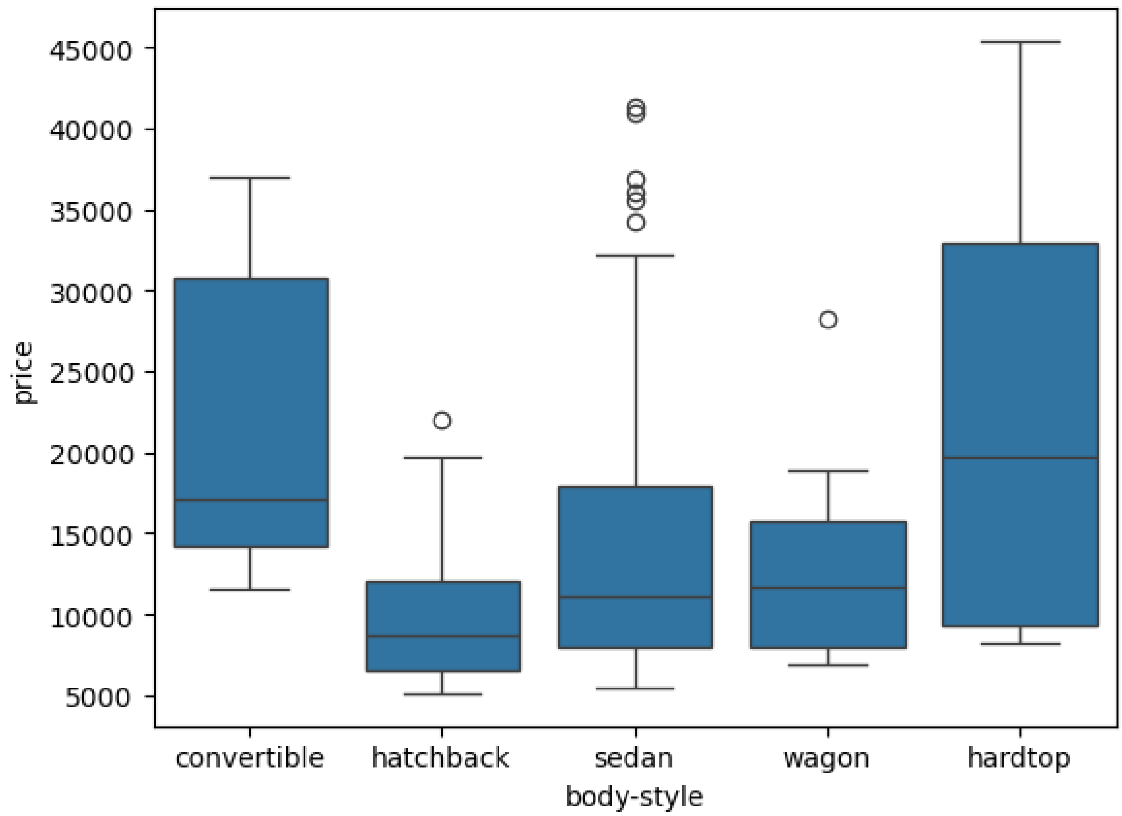
```
<AxesSubplot:xlabel='stroke', ylabel='price'>
```



Let's look at the relationship between "body-style" and "price".

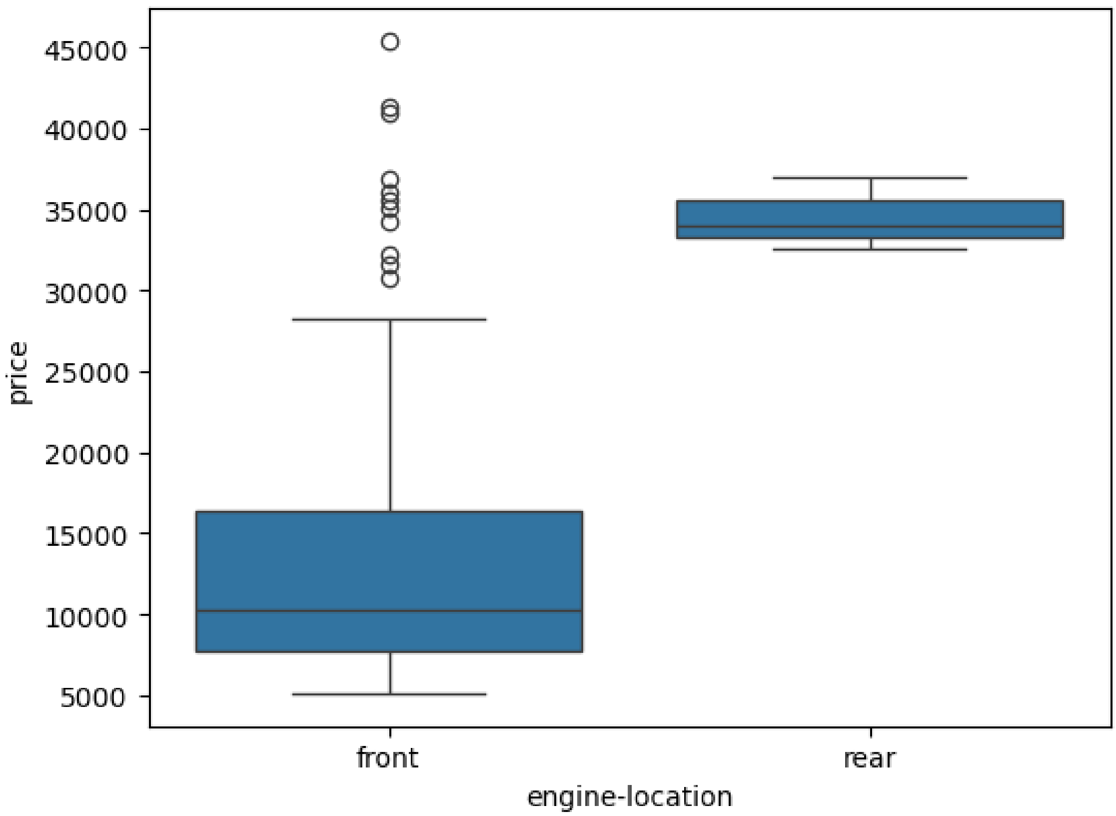
```
sns.boxplot(x="body-style", y="price", data=df)
```

```
<AxesSubplot:xlabel='body-style', ylabel='price'>
```




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

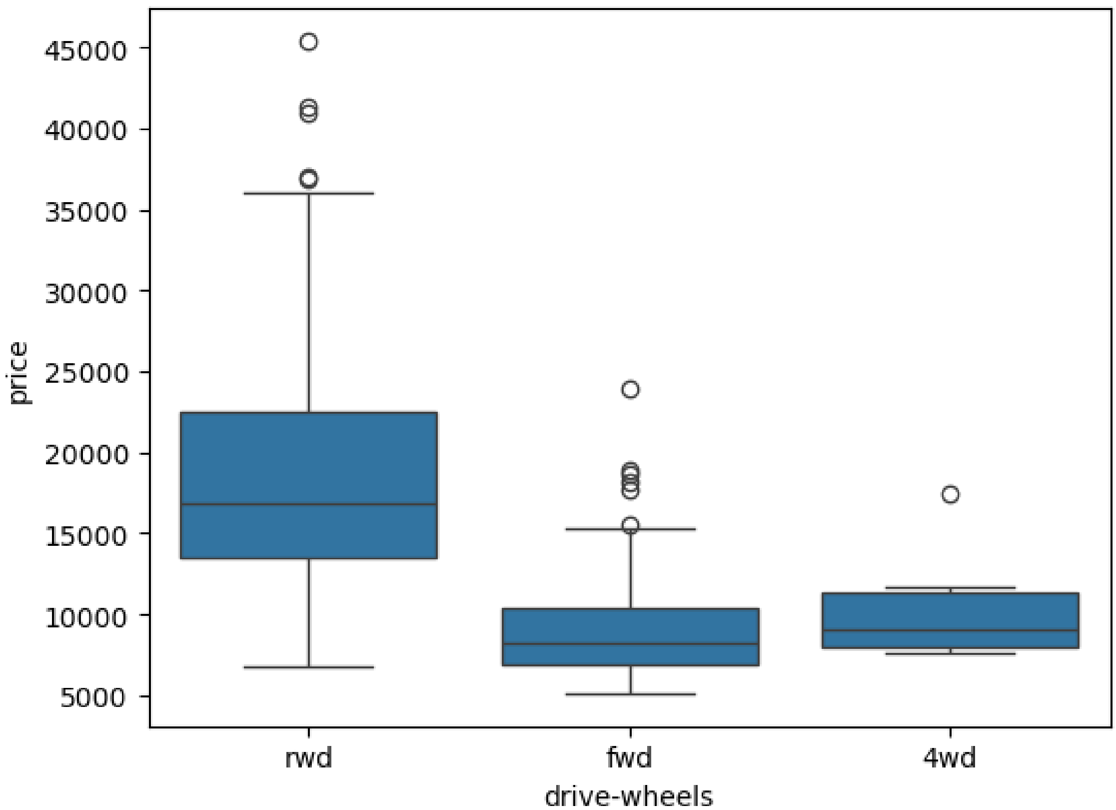
```
<AxesSubplot:xlabel='engine-location', ylabel='price'>
```



Let's examine "drive-wheels" and "price".

```
# drive-wheels
sns.boxplot(x="drive-wheels", y="price", data=df)
```

```
<AxesSubplot:xlabel='drive-wheels', ylabel='price'>
```



Descriptive Statistical Analysis

This will show: the count of that variable the mean the standard deviation (std) the minimum value the IQR (Interquartile Range: 25%, 50% and 75%) the maximum value

We can apply the method "describe" as follows:

```
df.describe()

        symboling  normalized-losses  wheel-base      length
width  \
count  201.000000          201.00000  201.000000  201.000000
201.000000
mean     0.840796          122.00000   98.797015    0.837102
0.915126
std      1.254802           31.99625    6.066366    0.059213
0.029187
min     -2.000000           65.00000   86.600000    0.678039
0.837500
25%      0.000000          101.00000   94.500000    0.801538
0.890278
50%      1.000000          122.00000   97.000000    0.832292
0.909722
75%      2.000000          137.00000  102.400000    0.881788
```

```
0.925000
max      3.000000          256.00000  120.900000    1.000000
1.000000

             height  curb-weight  engine-size        bore      stroke  \
count    201.000000   201.000000   201.000000  201.000000  197.000000
mean      53.766667  2555.666667   126.875622    3.330692    3.256904
std        2.447822   517.296727    41.546834    0.268072    0.319256
min       47.800000  1488.000000    61.000000    2.540000    2.070000
25%       52.000000  2169.000000    98.000000    3.150000    3.110000
50%       54.100000  2414.000000   120.000000    3.310000    3.290000
75%       55.500000  2926.000000   141.000000    3.580000    3.410000
max       59.800000  4066.000000   326.000000    3.940000    4.170000

       compression-ratio  horsepower     peak-rpm    city-mpg
highway-mpg  \
count         201.000000  201.000000   201.000000  201.000000
201.000000
mean           10.164279  103.405534  5117.665368   25.179104
30.686567
std             4.004965   37.365700   478.113805    6.423220
6.815150
min             7.000000   48.000000  4150.000000   13.000000
16.000000
25%             8.600000   70.000000  4800.000000   19.000000
25.000000
50%             9.000000   95.000000  5125.369458   24.000000
30.000000
75%             9.400000  116.000000  5500.000000   30.000000
34.000000
max            23.000000  262.000000  6600.000000   49.000000
54.000000

              price  city-L/100km      diesel         gas
count    201.000000    201.000000  201.000000  201.000000
mean   13207.129353      9.944145    0.099502    0.900498
std     7947.066342      2.534599    0.300083    0.300083
min     5118.000000      4.795918    0.000000    0.000000
25%     7775.000000      7.833333    0.000000    1.000000
50%    10295.000000      9.791667    0.000000    1.000000
75%    16500.000000     12.368421    0.000000    1.000000
max    45400.000000     18.076923    1.000000    1.000000
```

The default setting of "describe" skips variables of type object. We can apply the method "describe" on the variables of type 'object' as follows:

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

```
          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
```

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

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

We can convert the series to a dataframe as follows:

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

```
              count
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_counts'.

```
drive_wheels_counts = df['drive-wheels'].value_counts().to_frame()
drive_wheels_counts.rename(columns={'drive-wheels': 'value_counts'},
inplace=True)
drive_wheels_counts
```

```
              count
drive-wheels
fwd             118
rwd              75
4wd               8
```

Now let's rename the index to 'drive-wheels':

```
drive_wheels_counts.index.name = 'drive-wheels'
drive_wheels_counts
```

	count
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)
```

	count
engine-location	
front	198
rear	3

Basics of Grouping

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

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

```
df_group_one = df[['drive-wheels','body-style','price']]
```

We can then calculate the average price for each of the different categories of data.

```
# Ensure all columns except 'drive-wheels' are numeric
numeric_columns = df_group_one.columns[df_group_one.dtypes !=
'object']
numeric_columns = numeric_columns.drop('drive-wheels')

# Convert these columns to numeric, ignoring errors
df_group_one[numeric_columns] =
df_group_one[numeric_columns].apply(pd.to_numeric, errors='coerce')

# Now perform the groupby operation
df_group_one = df_group_one.groupby(['drive-wheels'],
as_index=False).mean()
print(df_group_one)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[55], line 3
      1 # Ensure all columns except 'drive-wheels' are numeric
      2 numeric_columns = df_group_one.columns[df_group_one.dtypes != 'object']
----> 3 numeric_columns = numeric_columns.drop('drive-wheels')
      5 # Convert these columns to numeric, ignoring errors
      6 df_group_one[numeric_columns] = df_group_one[numeric_columns].apply(pd.to_numeric, errors='coerce')

File /lib/python3.12/site-packages/pandas/core/indexes/base.py:7069, in Index.drop(self, labels, errors)
   7067 if mask.any():
   7068     if errors != "ignore":
-> 7069         raise KeyError(f"{labels[mask].tolist()} not found in axis")
   7070     indexer = indexer[~mask]
   7071 return self.delete(indexer)

KeyError: "['drive-wheels'] not found in axis"
```

```
# 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

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

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


body-style           wagon
drive-wheels
4wd            9095.750000
fwd            9997.333333
rwd           16994.222222
```

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

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


body-style           wagon
drive-wheels
4wd            9095.750000
fwd            9997.333333
rwd           16994.222222
```

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

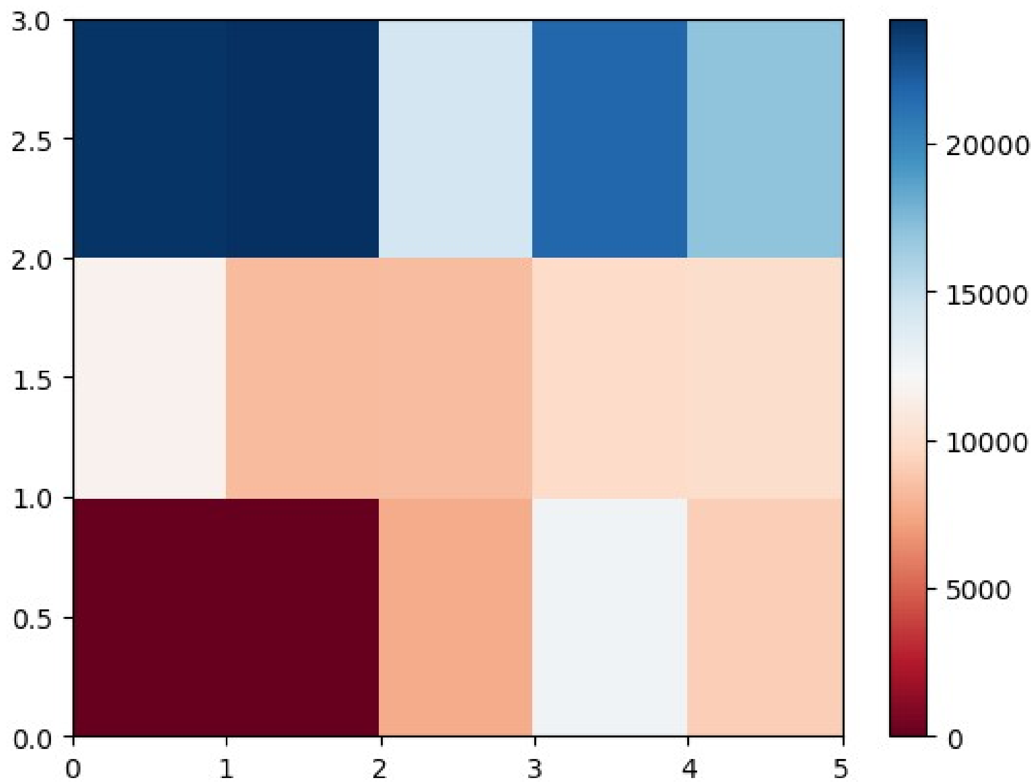
```
    body-style         price
0  convertible  21890.500000
1      hardtop  22208.500000
2    hatchback   9957.441176
3        sedan  14459.755319
4        wagon  12371.960000
```

If you did not import "pyplot", let's do it again.

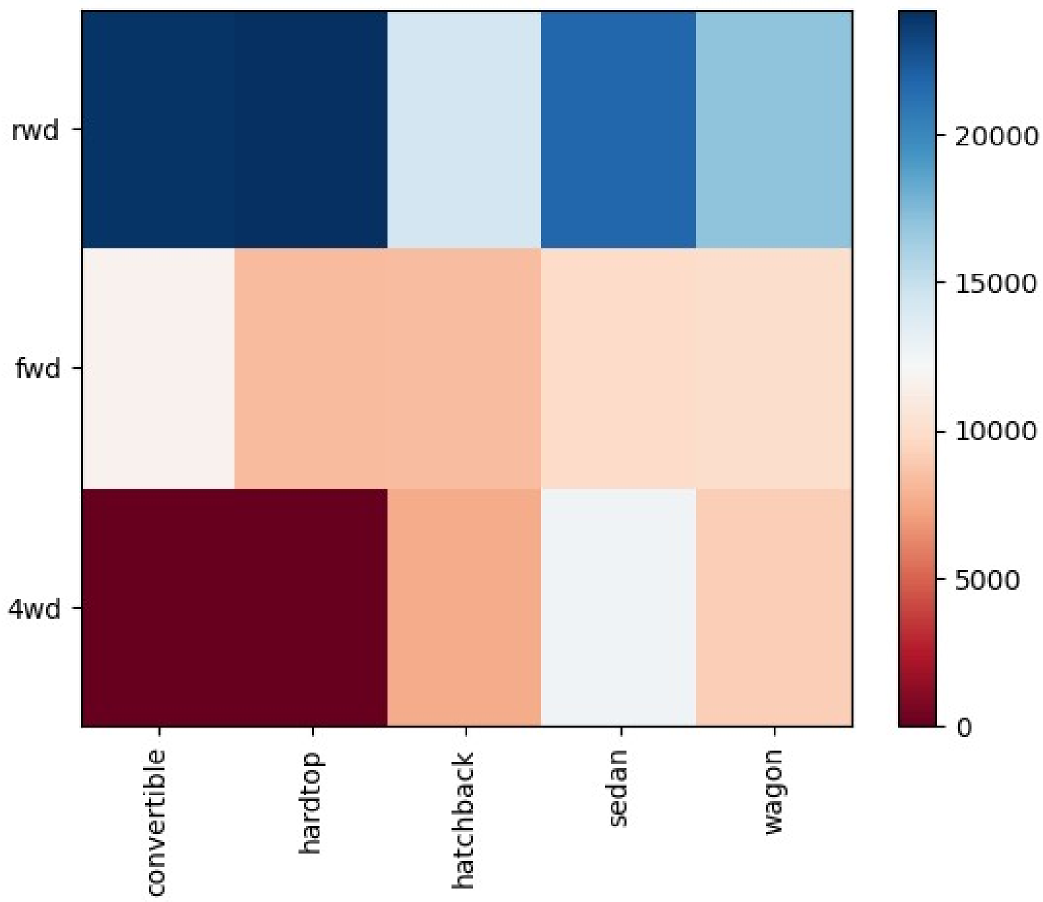
```
import matplotlib.pyplot as plt
%matplotlib inline
```

Let's use a heat map to visualize the relationship between Body Style vs Price.

```
#use the grouped results
plt.pcolor(grouped_pivot, cmap='RdBu')
plt.colorbar()
plt.show()
```




```
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()
```



Correlation and Causation

```
df.corr()

------------------------------------------------------------------------
-----
ValueError                                Traceback (most recent call
last)
Cell In[42], line 1
----> 1 df.corr()

File /lib/python3.12/site-packages/pandas/core/frame.py:11022, in
DataFrame.corr(self, method, min_periods, numeric_only)
  11020 cols = data.columns
  11021 idx = cols.copy()
> 11022 mat = data.to_numpy(dtype=float, na_value=np.nan, copy=False)
  11024 if method == "pearson":
  11025     correl = libalgos.nancorr(mat, minp=min_periods)

File /lib/python3.12/site-packages/pandas/core/frame.py:1981, in
DataFrame.to_numpy(self, dtype, copy, na_value)
   1979 if dtype is not None:
```

```
   1980     dtype = np.dtype(dtype)
-> 1981 result = self._mgr.as_array(dtype=dtype, copy=copy,
na_value=na_value)
   1982 if result.dtype is not dtype:
   1983     result = np.array(result, dtype=dtype, copy=False)

File
/lib/python3.12/site-packages/pandas/core/internals/managers.py:1693,
in BlockManager.as_array(self, dtype, copy, na_value)
   1691         arr.flags.writeable = False
   1692 else:
-> 1693     arr = self._interleave(dtype=dtype, na_value=na_value)
   1694     # The underlying data was copied within _interleave, so no
need
   1695     # to further copy if copy=True or setting na_value
   1697 if na_value is lib.no_default:

File
/lib/python3.12/site-packages/pandas/core/internals/managers.py:1752,
in BlockManager._interleave(self, dtype, na_value)
   1750     else:
   1751         arr = blk.get_values(dtype)
-> 1752     result[rl.indexer] = arr
   1753     itemmask[rl.indexer] = 1
   1755 if not itemmask.all():

ValueError: could not convert string to float: 'alfa-romero'
```

Sometimes we would like to know the significant of the correlation estimate.

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.

We can obtain this information using "stats" module in the "scipy" library.

```
from scipy import stats
```

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

```
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_value)
```

```
The Pearson Correlation Coefficient is 0.5846418222655085  with a P-value of P = 8.076488270732338e-20
```

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 =  6.36905742825956e-48
```

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 =  8.016477466158871e-30
```

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 0.7512653440522663  with a P-value of P = 9.200335510485071e-38
```

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

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 =  2.1895772388939654e-53
```

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 = 9.265491622195 82e-64
```

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 0.5431553832626601  with a P-value of P =   8.049189483935384e-17
```

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

```
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 =  2.321132065567 2357e-29
```

```
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 -0.7046922650589532  with a P-value of P =  1.7495471144475574e-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 shows that the relationship is negative and moderately strong.

Continuous numerical variables: Length Width Curb-weight Engine-size Horsepower City-mpg Highway-mpg Wheel-base Bore

Categorical variables: Drive-wheels

Thank you for completing this lab!

Author

Joseph Santarcangelo

Other Contributors

Mahdi Noorian PhD

Bahare Talayian

Eric Xiao

Steven Dong

Parizad

Hima Vasudevan

Fiorella Wenver

Yi Yao.

Abhishek Gagneja

<!--

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2023-09-28	2.2	Abhishek Gagneja	Updated instructions
2020-10-30	2.1	Lakshmi	changed URL of csv
2020-08-27	2.0	Lavanya	Moved lab to course repo in GitLab

--!>