Data Wrangling Lab

January 10, 2024

1 Data Wrangling

Estimated time needed: **30** minutes

1.1 Objectives

- Handle missing values
- Correct data formatting
- Standardize and normalize data

Table of Contents

What is the purpose of data wrangling?

You use data wrangling to convert data from an initial format to a format that may be better for analysis.

What is the fuel consumption (L/100k) rate for the diesel car?

Import data

You can find the "Automobile Dataset" from the following link: https://archive.ics.uci.edu/ml/machine-learning-databases/autos/imports-85.data. You will be using this data set throughout this course.

Import pandas

[1]:
```
#install specific version of libraries used in lab
#! mamba install pandas==1.3.3
#! mamba install numpy=1.21.2
```

[2]:
```python
import pandas as pd
import matplotlib.pylab as plt
```

Reading the dataset from the URL and adding the related headers

This dataset was hosted on IBM Cloud object. Click HERE for free storage.

The functions below will download the dataset into your browser:

[3]:
```python
'''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())'''
```

[3]:
```
'from pyodide.http import pyfetch\n\nasync def download(url, filename):\n
response = await pyfetch(url)\n    if response.status == 200:\n        with
open(filename, "wb") as f:\n            f.write(await response.bytes())'
```

First, assign the URL of the data set to "filepath".

[4]:
```python
#file_path="https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/
↪IBMDeveloperSkillsNetwork-DA0101EN-SkillsNetwork/labs/Data%20files/auto.csv"
```

To obtain the dataset, utilize the download() function as defined above:

[5]:
```python
#await download(file_path, "usedcars.csv")
file_name="usedcars.csv"
```

Then, create a Python list headers containing name of headers.

[6]:
```python
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"]
```

Use the Pandas method read_csv() to load the data from the web address. Set the parameter "names" equal to the Python list "headers".

[7]:
```python
df = pd.read_csv('usedcars.csv', names = headers)
```

[8]:
```python
#filepath = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/
↪IBMDeveloperSkillsNetwork-DA0101EN-SkillsNetwork/labs/Data%20files/auto.csv"
#df = pd.read_csv(filepath, header=headers)    # Utilize the same header list␣
↪defined above
```

Use the method head() to display the first five rows of the dataframe.

[9]:
```python
# To see what the data set looks like, we'll use the head() method.
df.head()
```

[9]:
```
   symboling normalized-losses         make fuel-type aspiration num-of-doors  \
0          3                 ?  alfa-romero       gas        std          two
1          3                 ?  alfa-romero       gas        std          two
2          1                 ?  alfa-romero       gas        std          two
3          2               164         audi       gas        std         four
4          2               164         audi       gas        std         four

    body-style drive-wheels engine-location  wheel-base  …  engine-size  \
0  convertible          rwd           front        88.6  …          130
1  convertible          rwd           front        88.6  …          130
2    hatchback          rwd           front        94.5  …          152
3        sedan          fwd           front        99.8  …          109
4        sedan          4wd           front        99.4  …          136

   fuel-system  bore  stroke compression-ratio horsepower  peak-rpm city-mpg  \
0         mpfi  3.47    2.68               9.0        111      5000       21
1         mpfi  3.47    2.68               9.0        111      5000       21
2         mpfi  2.68    3.47               9.0        154      5000       19
3         mpfi  3.19    3.40              10.0        102      5500       24
4         mpfi  3.19    3.40               8.0        115      5500       18

  highway-mpg  price
0          27  13495
1          27  16500
2          26  16500
3          30  13950
4          22  17450

[5 rows x 26 columns]
```

As you can see, several question marks appeared in the data frame; those missing values may hinder further analysis.

So, how do we identify all those missing values and deal with them?

How to work with missing data?

Steps for working with missing data:

Identify missing data

Deal with missing data

Correct data format

2 Identify and handle missing values

2.0.1 Identify missing values

Convert "?" to NaN

In the car data set, missing data comes with the question mark "?". We replace "?" with NaN (Not a Number), Python's default missing value marker for reasons of computational speed and convenience. Use the function:

to replace A by B.

[10]:
```
import numpy as np

# replace "?" to NaN
df.replace("?", np.nan, inplace = True)
df.head(5)
```

[10]:
```
   symboling normalized-losses         make fuel-type aspiration num-of-doors  \
0          3               NaN  alfa-romero       gas        std          two
1          3               NaN  alfa-romero       gas        std          two
2          1               NaN  alfa-romero       gas        std          two
3          2               164         audi       gas        std         four
4          2               164         audi       gas        std         four

    body-style drive-wheels engine-location  wheel-base  …  engine-size  \
0  convertible          rwd           front        88.6  …          130
1  convertible          rwd           front        88.6  …          130
2    hatchback          rwd           front        94.5  …          152
3        sedan          fwd           front        99.8  …          109
4        sedan          4wd           front        99.4  …          136

   fuel-system  bore  stroke compression-ratio horsepower  peak-rpm city-mpg  \
0         mpfi  3.47    2.68               9.0        111      5000       21
1         mpfi  3.47    2.68               9.0        111      5000       21
2         mpfi  2.68    3.47               9.0        154      5000       19
3         mpfi  3.19    3.40              10.0        102      5500       24
4         mpfi  3.19    3.40               8.0        115      5500       18

  highway-mpg  price
0          27  13495
1          27  16500
2          26  16500
3          30  13950
4          22  17450

[5 rows x 26 columns]
```

Evaluating for Missing Data

The missing values are converted by default. Use the following functions to identify these missing values. You can use two methods to detect missing data:

.isnull()

.notnull()

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

[11]:
```
missing_data = df.isnull()
missing_data.head(5)
```

[11]:
```
   symboling  normalized-losses   make  fuel-type  aspiration  num-of-doors  \
0      False               True  False      False       False         False
1      False               True  False      False       False         False
2      False               True  False      False       False         False
3      False              False  False      False       False         False
4      False              False  False      False       False         False

   body-style  drive-wheels  engine-location  wheel-base  …  engine-size  \
0       False         False            False       False  …        False
1       False         False            False       False  …        False
2       False         False            False       False  …        False
3       False         False            False       False  …        False
4       False         False            False       False  …        False

   fuel-system   bore  stroke  compression-ratio  horsepower  peak-rpm  \
0        False  False   False              False       False     False
1        False  False   False              False       False     False
2        False  False   False              False       False     False
3        False  False   False              False       False     False
4        False  False   False              False       False     False

   city-mpg  highway-mpg  price
0     False        False  False
1     False        False  False
2     False        False  False
3     False        False  False
4     False        False  False

[5 rows x 26 columns]
```

"True" means the value is a missing value while "False" means the value is not a missing value.

Count missing values in each column

Using a for loop in Python, you 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 data set. In the body of the for loop the method ".value_counts()" counts the number of "True" values.

[12]:
```
for column in missing_data.columns.values.tolist():
    print(column)
    print (missing_data[column].value_counts())
    print("")
```

```
symboling
symboling
False    205
Name: count, dtype: int64

normalized-losses
normalized-losses
False    164
True      41
Name: count, dtype: int64

make
make
False    205
Name: count, dtype: int64

fuel-type
fuel-type
False    205
Name: count, dtype: int64

aspiration
aspiration
False    205
Name: count, dtype: int64

num-of-doors
num-of-doors
False    203
True       2
Name: count, dtype: int64

body-style
body-style
False    205
Name: count, dtype: int64

drive-wheels
drive-wheels
False    205
Name: count, dtype: int64

engine-location
```

```
engine-location
False    205
Name: count, dtype: int64

wheel-base
wheel-base
False    205
Name: count, dtype: int64

length
length
False    205
Name: count, dtype: int64

width
width
False    205
Name: count, dtype: int64

height
height
False    205
Name: count, dtype: int64

curb-weight
curb-weight
False    205
Name: count, dtype: int64

engine-type
engine-type
False    205
Name: count, dtype: int64

num-of-cylinders
num-of-cylinders
False    205
Name: count, dtype: int64

engine-size
engine-size
False    205
Name: count, dtype: int64

fuel-system
fuel-system
False    205
Name: count, dtype: int64
```

```
bore
bore
False    201
True       4
Name: count, dtype: int64

stroke
stroke
False    201
True       4
Name: count, dtype: int64

compression-ratio
compression-ratio
False    205
Name: count, dtype: int64

horsepower
horsepower
False    203
True       2
Name: count, dtype: int64

peak-rpm
peak-rpm
False    203
True       2
Name: count, dtype: int64

city-mpg
city-mpg
False    205
Name: count, dtype: int64

highway-mpg
highway-mpg
False    205
Name: count, dtype: int64

price
price
False    201
True       4
Name: count, dtype: int64
```

Based on the summary above, each column has 205 rows of data and seven of the columns containing missing data:

“normalized-losses”: 41 missing data

“num-of-doors”: 2 missing data

“bore”: 4 missing data

“stroke” : 4 missing data

“horsepower”: 2 missing data

“peak-rpm”: 2 missing data

“price”: 4 missing data

2.0.2 Deal with missing data

How should you deal with missing data?

Drop data a. Drop the whole row b. Drop the whole column

Replace data a. Replace it by mean b. Replace it by frequency c. Replace it based on other functions

You should only drop whole columns if most entries in the column are empty. In the data set, none of the columns are empty enough to drop entirely. You have some freedom in choosing which method to replace data; however, some methods may seem more reasonable than others. Apply each method to different columns:

Replace by mean:

“normalized-losses”: 41 missing data, replace them with mean

“stroke”: 4 missing data, replace them with mean

“bore”: 4 missing data, replace them with mean

“horsepower”: 2 missing data, replace them with mean

“peak-rpm”: 2 missing data, replace them with mean

Replace by frequency:

“num-of-doors”: 2 missing data, replace them with “four”.

Reason: 84% sedans are four doors. Since four doors is most frequent, it is most likely to occur

Drop the whole row:

“price”: 4 missing data, simply delete the whole row

Reason: You want to predict price. You cannot use any data entry without price data for prediction; therefore any row now without price data is not useful to you.

Calculate the mean value for the “normalized-losses” column

[13]:
```
avg_norm_loss = df["normalized-losses"].astype("float").mean(axis=0)
print("Average of normalized-losses:", avg_norm_loss)
```

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

Replace "NaN" with mean value in "normalized-losses" column

[14]:
```
df["normalized-losses"].replace(np.nan, avg_norm_loss, inplace=True)
```

Calculate the mean value for the "bore" column

[15]:
```
avg_bore=df['bore'].astype('float').mean(axis=0)
print("Average of bore:", avg_bore)
```

```
Average of bore: 3.3297512437810943
```

Replace "NaN" with the mean value in the "bore" column

[16]:
```
df["bore"].replace(np.nan, avg_bore, inplace=True)
```

Question #1:

Based on the example above, replace NaN in "stroke" column with the mean value.

[17]:
```
#Calculate the mean vaule for "stroke" column
avg_stroke = df["stroke"].astype("float").mean(axis = 0)
print("Average of stroke:", avg_stroke)

# replace NaN by mean value in "stroke" column
df["stroke"].replace(np.nan, avg_stroke, inplace = True)
```

```
Average of stroke: 3.255422885572139
```

Calculate the mean value for the "horsepower" column

[18]:
```
avg_horsepower = df['horsepower'].astype('float').mean(axis=0)
print("Average horsepower:", avg_horsepower)
```

```
Average horsepower: 104.25615763546799
```

Replace "NaN" with the mean value in the "horsepower" column

[19]:
```
df['horsepower'].replace(np.nan, avg_horsepower, inplace=True)
```

Calculate the mean value for "peak-rpm" column

[20]:
```
avg_peakrpm=df['peak-rpm'].astype('float').mean(axis=0)
print("Average peak rpm:", avg_peakrpm)
```

```
Average peak rpm: 5125.369458128079
```

Replace "NaN" with the mean value in the "peak-rpm" column

[21]:
```
df['peak-rpm'].replace(np.nan, avg_peakrpm, inplace=True)
```

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

[22]:
```
df['num-of-doors'].value_counts()
```

[22]:
```
num-of-doors
four    114
two      89
Name: count, dtype: int64
```

You can see that four doors is the most common type. We can also use the ".idxmax()" method to calculate the most common type automatically:

[23]:
```
df['num-of-doors'].value_counts().idxmax()
```

[23]:
```
'four'
```

The replacement procedure is very similar to what you have seen previously:

[24]:
```
#replace the missing 'num-of-doors' values by the most frequent
df["num-of-doors"].replace(np.nan, "four", inplace=True)
```

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

[25]:
```
# 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)
```

[26]:
```
df.head()
```

[26]:
```
   symboling normalized-losses         make fuel-type aspiration num-of-doors  \
0          3             122.0  alfa-romero       gas        std          two
1          3             122.0  alfa-romero       gas        std          two
2          1             122.0  alfa-romero       gas        std          two
3          2               164         audi       gas        std         four
4          2               164         audi       gas        std         four

    body-style drive-wheels engine-location  wheel-base  …  engine-size  \
0  convertible          rwd           front        88.6  …          130
1  convertible          rwd           front        88.6  …          130
2    hatchback          rwd           front        94.5  …          152
3        sedan          fwd           front        99.8  …          109
4        sedan          4wd           front        99.4  …          136

   fuel-system  bore  stroke compression-ratio horsepower  peak-rpm city-mpg  \
0         mpfi  3.47    2.68               9.0        111      5000       21
1         mpfi  3.47    2.68               9.0        111      5000       21
2         mpfi  2.68    3.47               9.0        154      5000       19
```

```
3          mpfi   3.19    3.40              10.0        102       5500         24
4          mpfi   3.19    3.40               8.0        115       5500         18

   highway-mpg  price
0           27  13495
1           27  16500
2           26  16500
3           30  13950
4           22  17450

[5 rows x 26 columns]
```

Now, you have a data set with no missing values.

2.0.3 Correct data format

We are almost there!

The last step in data cleaning is checking and making sure that all data is in the correct format (int, float, text or other).

In Pandas, you use:

.dtype() to check the data type

.astype() to change the data type

Let's list the data types for each column

[27]:
```
df.dtypes
```

[27]:
```
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
```

As you can see above, some columns are not of the correct data type. Numerical variables should have type 'float' or 'int', and variables with strings such as categories should have type 'object'. For example, the numerical values 'bore' and 'stroke' describe the engines, so you should expect them to be of the type 'float' or 'int'; however, they are shown as type 'object'. You have to convert data types into a proper format for each column using the "astype()" method.

Convert data types to proper format

[28]:
```
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")
```

Let us list the columns after the conversion

[29]:
```
df.dtypes
```

[29]:
```
symboling              int64
normalized-losses      int32
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
```

Now you finally obtained the cleansed data set with no missing values and with all data in its proper format.

2.1 Data Standardization

You usually collect data from different agencies in different formats. (Data standardization is also a term for a particular type of data normalization where you subtract the mean and divide by the standard deviation.)

What is standardization?

Standardization is the process of transforming data into a common format, allowing the researcher to make the meaningful comparison.

Example

Transform mpg to L/100km:

In your data set, the fuel consumption columns "city-mpg" and "highway-mpg" are represented by mpg (miles per gallon) unit. Assume you are developing an application in a country that accepts the fuel consumption with L/100km standard.

You will need to apply data transformation to transform mpg into L/100km.

Use this formula for unit conversion:

L/100km = 235 / mpg

You can do many mathematical operations directly using Pandas.

[30]:
```
df.head()
```

[30]:
```
   symboling  normalized-losses         make fuel-type aspiration  \
0          3                122  alfa-romero       gas        std
1          3                122  alfa-romero       gas        std
2          1                122  alfa-romero       gas        std
3          2                164         audi       gas        std
4          2                164         audi       gas        std

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

   engine-size  fuel-system  bore  stroke compression-ratio horsepower  \
0          130         mpfi  3.47    2.68               9.0        111
```

```
1          130         mpfi  3.47    2.68                 9.0         111
2          152         mpfi  2.68    3.47                 9.0         154
3          109         mpfi  3.19    3.40                10.0         102
4          136         mpfi  3.19    3.40                 8.0         115

   peak-rpm city-mpg  highway-mpg    price
0    5000.0       21           27  13495.0
1    5000.0       21           27  16500.0
2    5000.0       19           26  16500.0
3    5500.0       24           30  13950.0
4    5500.0       18           22  17450.0

[5 rows x 26 columns]
```

[31]:

```
# Convert mpg to L/100km by mathematical operation (235 divided by mpg)
df['city-L/100km'] = 235/df["city-mpg"]

# check your transformed data
df.head()
```

[31]:

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

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

   fuel-system  bore  stroke  compression-ratio horsepower peak-rpm  city-mpg  \
0         mpfi  3.47    2.68                9.0        111   5000.0        21
1         mpfi  3.47    2.68                9.0        111   5000.0        21
2         mpfi  2.68    3.47                9.0        154   5000.0        19
3         mpfi  3.19    3.40               10.0        102   5500.0        24
4         mpfi  3.19    3.40                8.0        115   5500.0        18

  highway-mpg    price  city-L/100km
0          27  13495.0     11.190476
1          27  16500.0     11.190476
2          26  16500.0     12.368421
3          30  13950.0      9.791667
4          22  17450.0     13.055556
```

```
[5 rows x 27 columns]
```

Question #2:

According to the example above, transform mpg to L/100km in the column of "highway-mpg" and change the name of column to "highway-L/100km".

[32]:
```python
# transform mpg to L/100km by mathematical operation (235 divided by mpg)
df["highway-mpg"] = 235/df["highway-mpg"]

# rename column name from "highway-mpg" to "highway-L/100km"
df.rename(columns={'"highway-mpg"':'highway-L/100km'}, inplace=True)

# check your transformed data
df.head()
```

[32]:
```
   symboling  normalized-losses         make fuel-type aspiration  \
0          3                122  alfa-romero       gas        std
1          3                122  alfa-romero       gas        std
2          1                122  alfa-romero       gas        std
3          2                164         audi       gas        std
4          2                164         audi       gas        std

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

   fuel-system  bore  stroke  compression-ratio horsepower peak-rpm city-mpg  \
0         mpfi  3.47    2.68                9.0        111   5000.0       21
1         mpfi  3.47    2.68                9.0        111   5000.0       21
2         mpfi  2.68    3.47                9.0        154   5000.0       19
3         mpfi  3.19    3.40               10.0        102   5500.0       24
4         mpfi  3.19    3.40                8.0        115   5500.0       18

  highway-mpg    price  city-L/100km
0    8.703704  13495.0     11.190476
1    8.703704  16500.0     11.190476
2    9.038462  16500.0     12.368421
3    7.833333  13950.0      9.791667
4   10.681818  17450.0     13.055556

[5 rows x 27 columns]
```

2.2 Data Normalization

Why normalization?

Normalization is the process of transforming values of several variables into a similar range. Typical normalizations include

scaling the variable so the variable average is 0

scaling the variable so the variance is 1

scaling the variable so the variable values range from 0 to 1

Example

To demonstrate normalization, say you want to scale the columns "length", "width" and "height".

Target: normalize those variables so their value ranges from 0 to 1

Approach: replace the original value by (original value)/(maximum value)

[33]:
```
# replace (original value) by (original value)/(maximum value)
df['length'] = df['length']/df['length'].max()
df['width'] = df['width']/df['width'].max()
```

Question #3:

According to the example above, normalize the column "height".

[34]:
```
df['height'] = df['height']/df['height'].max()

# show the scaled columns
df[["length","width","height"]].head()
```

[34]:

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

Here you've normalized "length", "width" and "height" to fall in the range of [0,1].

2.3 Binning

Why binning?

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

Example:

In your data set, "horsepower" is a real valued variable ranging from 48 to 288 and it has 59 unique values. What if you only care about the price difference between cars with high horsepower, medium horsepower, and little horsepower (3 types)? You can rearrange them into three 'bins' to simplify analysis.

Use the Pandas method 'cut' to segment the 'horsepower' column into 3 bins.

Example of Binning Data In Pandas

Convert data to correct format:

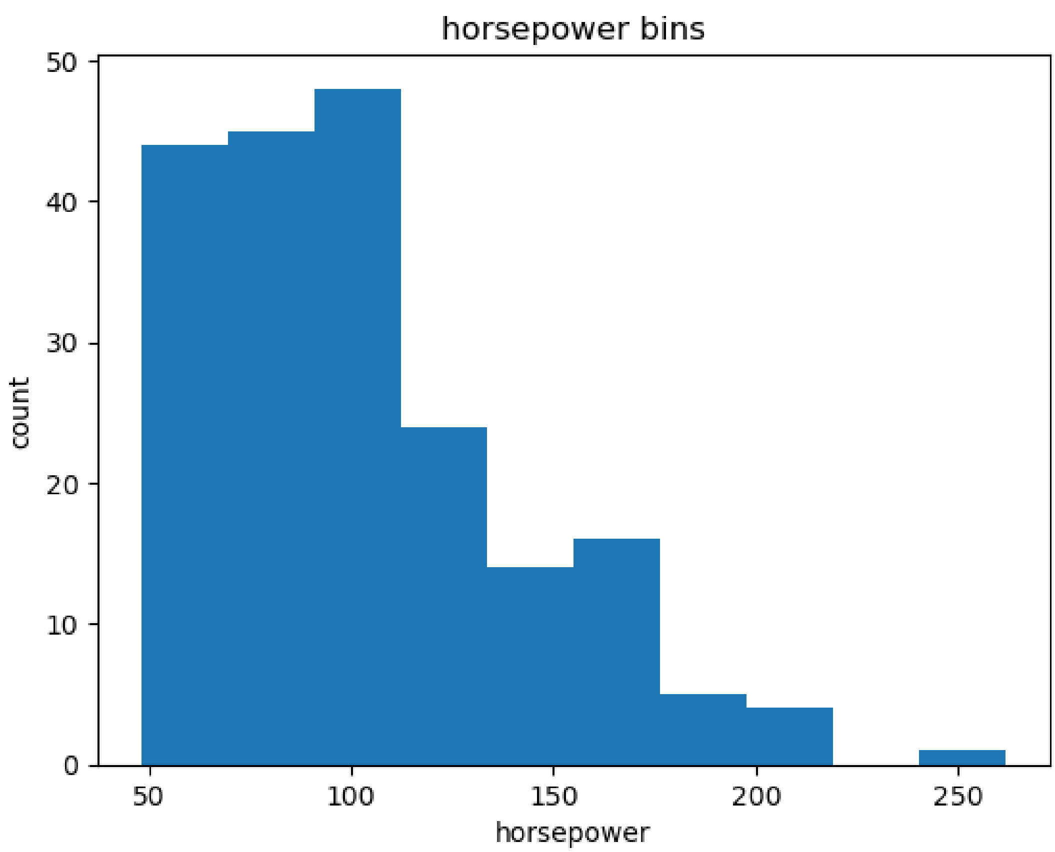
[35]:
```
df["horsepower"]=df["horsepower"].astype(int, copy=True)
```

Plot the histogram of horsepower to see the distribution of horsepower.

[36]:
```
%matplotlib inline
import matplotlib as plt
from matplotlib import pyplot
plt.pyplot.hist(df["horsepower"])

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

[36]:
```
Text(0.5, 1.0, 'horsepower bins')
```



Find 3 bins of equal size bandwidth by using Numpy's linspace(start_value, end_value, numbers_generated function.

Since you want to include the minimum value of horsepower, set start_value = min(df["horsepower"]).

Since you want to include the maximum value of horsepower, set end_value = max(df["horsepower"]).

Since you are building 3 bins of equal length, you need 4 dividers, so numbers_generated = 4.

Build a bin array with a minimum value to a maximum value by using the bandwidth calculated above. The values will determine when one bin ends and another begins.

[37]:
```python
bins = np.linspace(min(df["horsepower"]), max(df["horsepower"]), 4)
bins
```

[37]:
```
array([ 48.        , 119.33333333, 190.66666667, 262.        ])
```

Set group names:

[38]:
```python
group_names = ['Low', 'Medium', 'High']
```

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

[39]:
```python
df['horsepower-binned'] = pd.cut(df['horsepower'], bins, labels=group_names,␣
↪include_lowest=True )
df[['horsepower','horsepower-binned']].head(20)
```

[39]:

	horsepower	horsepower-binned
0	111	Low
1	111	Low
2	154	Medium
3	102	Low
4	115	Low
5	110	Low
6	110	Low
7	110	Low
8	140	Medium
9	101	Low
10	101	Low
11	121	Medium
12	121	Medium
13	121	Medium
14	182	Medium
15	182	Medium
16	182	Medium
17	48	Low
18	70	Low
19	70	Low

See the number of vehicles in each bin:

[40]:
```
df["horsepower-binned"].value_counts()
```

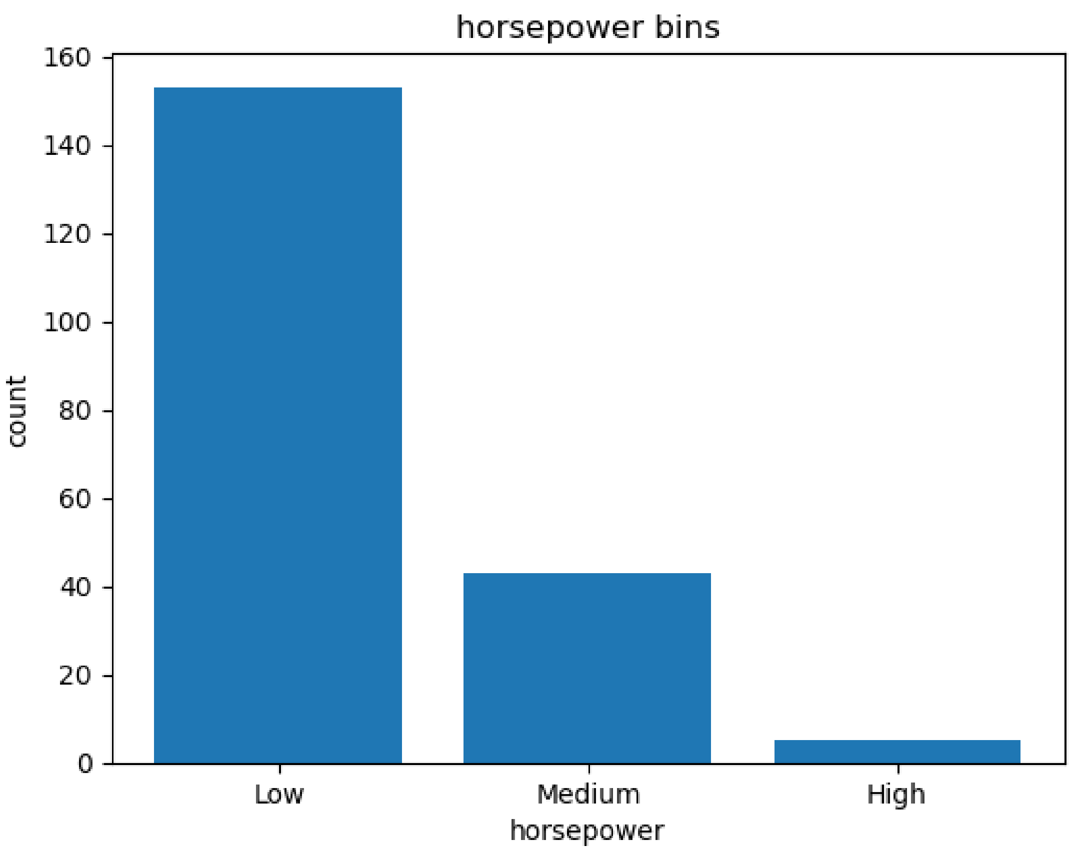
[40]:
```
horsepower-binned
Low       153
Medium     43
High        5
Name: count, dtype: int64
```

Plot the distribution of each bin:

[41]:
```
%matplotlib inline
import matplotlib as plt
from matplotlib import pyplot
pyplot.bar(group_names, df["horsepower-binned"].value_counts())

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

[41]:
```
Text(0.5, 1.0, 'horsepower bins')
```



Look at the data frame above carefully. You will find that the last column provides the bins for “horsepower” based on 3 categories (“Low”, “Medium” and “High”).

You successfully narrowed down the intervals from 59 to 3!

Bins Visualization

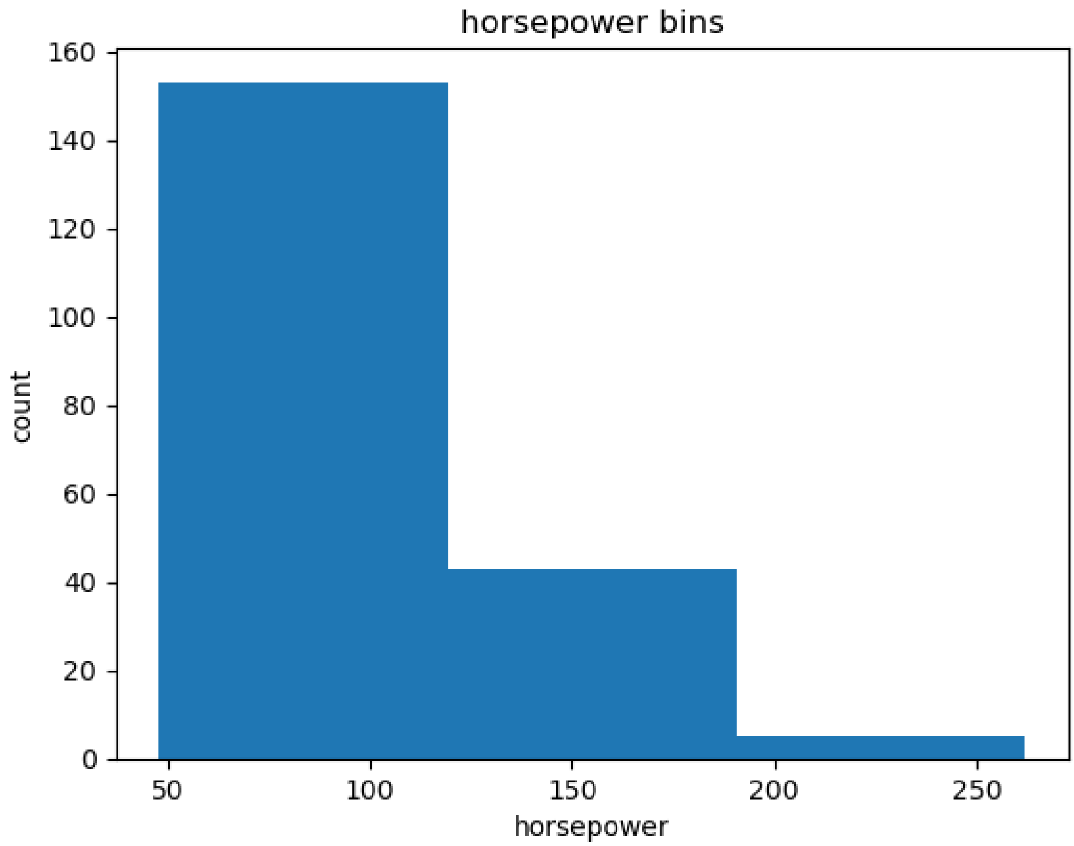
Normally, you use a histogram to visualize the distribution of bins we created above.

[42]:
```
%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")
```

[42]:
```
Text(0.5, 1.0, 'horsepower bins')
```



The plot above shows the binning result for the attribute "horsepower".

2.4 Indicator 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 themselves don't have inherent meaning.

Why use indicator variables?

You use indicator variables so you can use categorical variables for regression analysis in the later modules.

Example

The column "fuel-type" has two unique values: "gas" or "diesel". Regression doesn't understand words, only numbers. To use this attribute in regression analysis, you can convert "fuel-type" to indicator variables.

Use the Panda method 'get_dummies' to assign numerical values to different categories of fuel type.

[43]:
```
df.columns
```

[43]:
```
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', 'horsepower-binned'],
      dtype='object')
```

Get the indicator variables and assign it to data frame "dummy_variable_1":

[44]:
```
dummy_variable_1 = pd.get_dummies(df["fuel-type"])
dummy_variable_1.head()
```

[44]:

	diesel	gas
0	False	True
1	False	True
2	False	True
3	False	True
4	False	True

Change the column names for clarity:

[45]:
```
dummy_variable_1.rename(columns={'gas':'fuel-type-gas', 'diesel':
 ↪'fuel-type-diesel'}, inplace=True)
dummy_variable_1.head()
```

[45]:

	fuel-type-diesel	fuel-type-gas
0	False	True
1	False	True
2	False	True
3	False	True
4	False	True

In the data frame, column 'fuel-type' now has values for 'gas' and 'diesel' as 0s and 1s.

[46]:
```
# merge data frame "df" and "dummy_variable_1"
df = pd.concat([df, dummy_variable_1], axis=1)

# drop original column "fuel-type" from "df"
df.drop("fuel-type", axis = 1, inplace=True)
```

[47]:
```
df.head()
```

[47]:

	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    5000.0       21    8.703704  13495.0
1                9.0         111    5000.0       21    8.703704  16500.0
2                9.0         154    5000.0       19    9.038462  16500.0
3               10.0         102    5500.0       24    7.833333  13950.0
4                8.0         115    5500.0       18   10.681818  17450.0

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

[5 rows x 29 columns]
```

The last two columns are now the indicator variable representation of the fuel-type variable. They're all 0s and 1s now.

Question #4:

Similar to before, create an indicator variable for the column "aspiration"

[48]:
```python
# 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()
```

[48]:
```
   aspiration-std  aspiration-turbo
0            True             False
1            True             False
2            True             False
```

```
3             True              False
4             True              False
```

Question #5:

Merge the new dataframe to the original dataframe, then drop the column 'aspiration'.

[49]:
```
# 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)
```

Save the new csv:

[50]:
```
df.to_csv('cleaned_used_cars.csv')
```

[ ]:
```

```