data_wrangling

December 20, 2018

Link
Data Analysis with Python

1 Module 2 : Data Wrangling

1.0.1 Welcome!

By the end of this notebook, you will have learned the basics of Data Wrangling!

1.1 Table of contents

Identify and handle missing values
- Identify missing values
- Deal with missing values
- Correct data format

Data standardization
Data Normalization (centring/scaling)
Binning
Indicator variable
Estimated Time Needed: 30 min

1.2 What is the purpose of Data Wrangling?

Data Wrangling is the process of converting data from the initial format to a format that may be better for analysis.

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

1.2.2 Import data

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

Import pandas

```
In [1]: import pandas as pd
```

1.3 Reading the data set from the URL and adding the related headers.

URL of dataset

```
In [2]: filename = 'https://archive.ics.uci.edu/ml/machine-learning-databases/autos/imports-85.d
        print('URL read and saved as "filename"')
```

```
URL read and saved as "filename"
```

Python list "headers" containing name of headers

```
In [3]: headers = ["symboling","normalized-losses","make","fuel-type","aspiration", "num-of-door
                   "drive-wheels","engine-location","wheel-base", "length","width","height","curb-
                   "num-of-cylinders", "engine-size","fuel-system","bore","stroke","compression-ra
                   "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".

```
In [4]: df = pd.read_csv(filename, names = headers)
        print("Done")
```

```
Done
```

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

```
In [5]: # To see what the data set looks like, we'll use the head() method.
        df.head()
```

```
Out[5]:    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 we can see, several question marks appeared in the dataframe; those are missing values which may hinder our 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: 1. Identify missing data 2. Deal with missing data 3. Correct data format

1. Identify and handle missing values

Convert "?" to NaN In the car dataset, missing data comes with the question mark "?". We replace "?" with NaN (Not a Number), which is Python's default missing value marker, for reasons of computational speed and convenience. Here we use the function:

to replace A by B

```
In [6]: import numpy as np

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

```
Out[6]:    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]
```

1.3.1 Evaluating for Missing Data

The missing values are converted to Python's default. We use Python's built-in functions to identify these missing values. There are two methods to detect missing data: 1. **.isnull()** 2. **.notnull()** The output is a boolean value indicating whether the dataframe is missing data.

```
In [7]: missing_data = df.isnull()
        missing_data.head(5)
```

```
Out[7]:    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" stands for missing value, while "False" stands for not missing value.

1.3.2 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, "False" means the value is present in the dataset. In the body of the for loop the method ".value_couts()" counts the number of "True" values.

```
In [8]: for column in missing_data.columns.values.tolist():
            print(column)
            print (missing_data[column].value_counts())
            print("")
```

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

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

make
False    205
Name: make, dtype: int64

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

aspiration
False    205
Name: aspiration, dtype: int64

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

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

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

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

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

length
False    205
Name: length, dtype: int64

width
False    205
Name: width, dtype: int64

height
False    205
Name: height, dtype: int64

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

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

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

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

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

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

stroke
False    201
True       4
```

```
Name: stroke, dtype: int64

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

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

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

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

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

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

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

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

Deal with missing data **How to deal with missing data:**

```
1. Drop data
    a. drop the whole row
    b. drop the whole column
2. Replace data
    a. replace it by mean
```

```
    b. replace it by frequency
    c. replace it based on other functions
```

Whole columns should be dropped only if most entries in the column are empty.
In our dataset, none of the columns are empty enough to drop entirely.
We have some freedom in choosing which method to replace data; however, some methods may seem more reasonable than others. We will apply each method to many different columns:

Replace by mean:

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

Replace by frequency:

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

Drop the whole row:

```
"price": 4 missing data, simply delete the whole row
    * Reason: price is what we want to predict. Any data entry without price data cannot be used
```

Calculate the average of the column:

```
In [9]: avg_1 = df["normalized-losses"].astype("float").mean(axis = 0)
```

Replace "NaN" by mean value in "normalized-losses" column:

```
In [10]: df["normalized-losses"].replace(np.nan, avg_1, inplace = True)
```

Calculate the mean value for 'bore' column:

```
In [11]: avg_2=df['bore'].astype('float').mean(axis=0)
```

Replace NaN by mean value:

```
In [12]: df['bore'].replace(np.nan, avg_2, inplace= True)
```

Question #1:
According to the example above, replace NaN in "stroke" column by mean:

```
In [13]: avg3=df['stroke'].astype('float').mean(axis=0)
         df['stroke'].replace(np.nan, avg3, inplace=True)
         df['stroke']
```

Out[13]:
```
0      2.68
1      2.68
2      3.47
3      3.40
4      3.40
5      3.40
6      3.40
7      3.40
8      3.40
9      3.40
10     2.80
11     2.80
12     3.19
13     3.19
14     3.19
15     3.39
16     3.39
17     3.39
18     3.03
19     3.11
20     3.11
21     3.23
22     3.23
23     3.39
24     3.23
25     3.23
26     3.23
27     3.39
28     3.46
29     3.90
       ...
175    3.54
176    3.54
177    3.54
178    3.35
179    3.35
180    3.35
181    3.35
182    3.40
183    3.40
184    3.40
185    3.40
186    3.40
187    3.40
188    3.40
189    3.40
190    3.40
191    3.40
```

```
192    3.40
193    3.40
194    3.15
195    3.15
196    3.15
197    3.15
198    3.15
199    3.15
200    3.15
201    3.15
202    2.87
203    3.40
204    3.15
Name: stroke, Length: 205, dtype: object
```

Question #1 Answer:
Run the code below! Did you get the right code?
Click here for the solution

```
# calculate the mean vaule for "stroke" column
avg_3 = df["stroke"].astype("float").mean(axis = 0)

# replace NaN by mean value in "stroke" column
df["stroke"].replace(np.nan, avg_3, inplace = True)
```

Calculate the mean value for the 'horsepower' column:

```
In [14]: avg_4=df['horsepower'].astype('float').mean(axis=0)
```

Replace "NaN" by mean value :

```
In [15]: df['horsepower'].replace(np.nan, avg_4, inplace= True)
```

Calculate the mean value for 'peak-rpm' column:

```
In [16]: avg_5=df['peak-rpm'].astype('float').mean(axis=0)
```

Replace NaN by mean value:

```
In [17]: df['peak-rpm'].replace(np.nan, avg_5, inplace= True)
```

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

```
In [18]: df['num-of-doors'].value_counts()
```

```
Out[18]: four    114
         two      89
         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 automatically calculate the most common type:

```
In [19]: df['num-of-doors'].value_counts().idxmax()
```

```
Out[19]: 'four'
```

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

```
In [20]: #replace the missing 'num-of-doors' values by the most frequent
         df["num-of-doors"].replace(np.nan, "four", inplace = True)
```

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

```
In [21]: # 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)
```

```
In [22]: df.head()
```

```
Out[22]:    symboling normalized-losses         make fuel-type aspiration num-of-doors  \
         0          3               122  alfa-romero       gas        std          two
         1          3               122  alfa-romero       gas        std          two
         2          1               122  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]
```

Good! Now, we obtain the dataset with no missing values.

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, we use

.dtype() to check the data type

.astype() to change the data type

Let's list the data types for each column:

```
In [23]: df.dtypes
```

```
Out[23]: 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 we 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, 'bore' and 'stroke' variables are numerical values that describe the engines, so we should expect them to be of the type 'float' or 'int', however, they are shown as type 'object'. We have to convert data types into a proper format for each column using the "astype()" method.

Convert data types to proper format:

```
In [24]: df[["bore", "stroke", "price"]] = df[["bore", "stroke", "price"]].astype("float")
         df[["normalized-losses"]] = df[["normalized-losses"]].astype("int")
         #df[["price"]] = df[["price"]].astype("float")
         df[["peak-rpm"]] = df[["peak-rpm"]].astype("float")
         print("Done")
```

```
Done
```

Let us list the columns after the conversion:

```
In [25]: df.dtypes
```

```
Out[25]: 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
```

Wonderful!

Now, we finally obtain the cleaned dataset with no missing values and all data in its proper format.

Data Standardization Data is usually collected from different agencies with different formats. (Data Standardization is also a term for a particular type of data normalization, where we subtract the mean and divide by the standard deviation.

What is Standardization?

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

Example

Transform mpg to L/100km:

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

We will need to apply **data transformation** to transform mpg into L/100km

The formula for unit conversion is L/100km = 235 / mpg

We can do many mathematical operations directly in Pandas.

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

Out[26]:
```
   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]
```

In [27]:
```
# transform 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()
```

```
Out[27]:    symboling  normalized-losses         make fuel-type aspiration  \
        0          3                122  alfa-romero       gas        std
        1          3                122  alfa-romero       gas        std
        2          1                122  alfa-romero       gas        std
        3          2                164         audi       gas        std
        4          2                164         audi       gas        std

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

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

          peak-rpm  city-mpg highway-mpg    price  city-L/100km
        0   5000.0        21          27  13495.0     11.190476
        1   5000.0        21          27  16500.0     11.190476
        2   5000.0        19          26  16500.0     12.368421
        3   5500.0        24          30  13950.0      9.791667
        4   5500.0        18          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":

```
In [28]: df['highway-L/100km'] = 235/df["highway-mpg"]
         df.head()
```

```
Out[28]:    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

         ...         bore   stroke  compression-ratio  horsepower  peak-rpm  \
0        ...         3.47     2.68                9.0         111    5000.0
1        ...         3.47     2.68                9.0         111    5000.0
2        ...         2.68     3.47                9.0         154    5000.0
3        ...         3.19     3.40               10.0         102    5500.0
4        ...         3.19     3.40                8.0         115    5500.0

  city-mpg  highway-mpg    price  city-L/100km  highway-L/100km
0       21           27  13495.0     11.190476         8.703704
1       21           27  16500.0     11.190476         8.703704
2       19           26  16500.0     12.368421         9.038462
3       24           30  13950.0      9.791667         7.833333
4       18           22  17450.0     13.055556        10.681818

[5 rows x 28 columns]
```

Question #2 Answer:
Run the code below! Did you get the right code?
Click here for the solution

```
# 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()
```

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 variable variance is 1, or scaling the variable so the variable values range from 0 to 1.

Example

To demonstrate normalization, let's say we want to scale the columns "length", "width" and "height"

Target: We would like to Normalize those variables so their value ranges from 0 to 1.

Approach: Replace origianl value by (original value)/(maximum value)

```
In [29]: # replace (origianl value) by (original value)/(maximum value)
         df['length'] = df['length']/df['length'].max()
         df['width'] = df['width']/df['width'].max()
```

Questiont #3:

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

```
In [30]: df['height']=df['height']/df['height'].max()
         df[["length","width","height"]].head()
```

```
Out[30]:      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
```

Question #3 Answer:
Run the code below! Did you get the right code?
Click here for the solution

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

Here we can see we've normalized "length", "width", and "height" in the range of [0,1].
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 57 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.

1.4 Example of Binning Data In Pandas

Convert data to correct format:

```
In [31]: df["horsepower"]=df["horsepower"].astype(float, copy=True)
```

We would like four bins of equal size bandwidth. The fourth is because the function "cut" includes the rightmost value:

```
In [32]: binwidth = (max(df["horsepower"])-min(df["horsepower"]))/4
```

We build a bin array with a minimum value to a maximum value, with bandwidth calculated above. The bins will be values used to determine when one bin ends and another begins:

```
In [33]: bins = np.arange(min(df["horsepower"]), max(df["horsepower"]), binwidth)
         bins
```

```
Out[33]: array([ 48. , 101.5, 155. , 208.5])
```

We set group names:

In [34]:
```
group_names = ['Low', 'Medium', 'High']
```

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

In [35]:
```
df['horsepower-binned'] = pd.cut(df['horsepower'], bins, labels=group_names,include_low
df[['horsepower','horsepower-binned']].head(20)
```

Out[35]:
```
    horsepower horsepower-binned
0        111.0            Medium
1        111.0            Medium
2        154.0            Medium
3        102.0            Medium
4        115.0            Medium
5        110.0            Medium
6        110.0            Medium
7        110.0            Medium
8        140.0            Medium
9        101.0               Low
10       101.0               Low
11       121.0            Medium
12       121.0            Medium
13       121.0            Medium
14       182.0              High
15       182.0              High
16       182.0              High
17        48.0               Low
18        70.0               Low
19        70.0               Low
```

Check the dataframe above carefully, you will find the last column provides the bins for "horsepower" with 3 categories ("Low","Medium" and "High").

We successfully narrow the intervals from 57 to 3!

1.5 Bins visualization

Normally, a histogram is used to visualize the distribution of bins we created above.

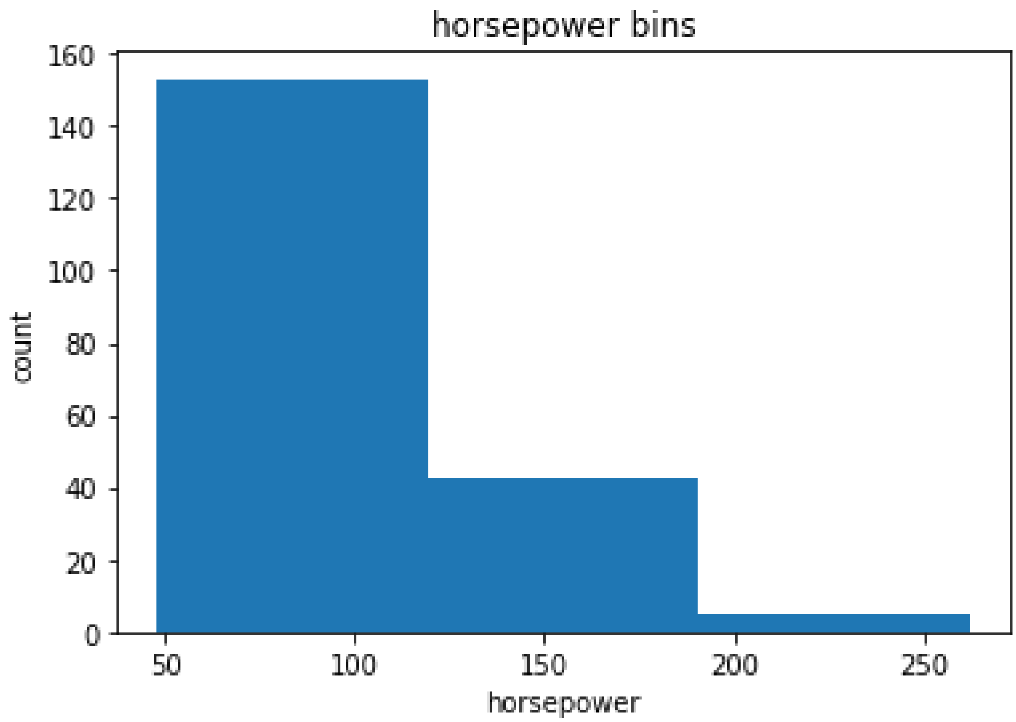
In [36]:
```
%matplotlib inline
import matplotlib as plt
from matplotlib import pyplot

a = (0,1,2)

# 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")
```

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



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

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 themselves don't have inherent meaning.

Why do we use indicator variables?

So that 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 attribute in regression analysis, we convert "fuel-type" into indicator variables.

We will use the panda's method 'get_dummies' to assign numerical values to different categories of fuel type.

In [37]:
```
df.columns
```

Out[37]:
```
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')
```

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

In [38]:
```
dummy_variable_1 = pd.get_dummies(df["fuel-type"])
dummy_variable_1.head()
```

Out[38]:
```
   diesel  gas
0       0    1
1       0    1
2       0    1
3       0    1
4       0    1
```

Change column names for clarity:

In [39]:
```
dummy_variable_1.rename(columns={'fuel-type-diesel':'gas', 'fuel-type-diesel':'diesel'}
dummy_variable_1.head()
```

Out[39]:
```
   diesel  gas
0       0    1
1       0    1
2       0    1
3       0    1
4       0    1
```

We now have the value 0 to represent "gas" and 1 to represent "diesel" in the column "fuel-type". We will now insert this column back into our original dataset.

In [40]:
```
# 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)
```

In [41]:
```
df.head()
```

Out[41]:
```
   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 ...
```

```
   horsepower  peak-rpm  city-mpg highway-mpg    price  city-L/100km  \
0       111.0    5000.0        21          27  13495.0     11.190476
1       111.0    5000.0        21          27  16500.0     11.190476
2       154.0    5000.0        19          26  16500.0     12.368421
3       102.0    5500.0        24          30  13950.0      9.791667
4       115.0    5500.0        18          22  17450.0     13.055556

  highway-L/100km  horsepower-binned  diesel  gas
0        8.703704             Medium       0    1
1        8.703704             Medium       0    1
2        9.038462             Medium       0    1
3        7.833333             Medium       0    1
4       10.681818             Medium       0    1

[5 rows x 30 columns]
```

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

Question #4:

As above, create indicator variable to the column of "aspiration": "std" to 0, while "turbo" to 1:

In [42]:
```
dummy_variable_2 = pd.get_dummies(df["aspiration"])
dummy_variable_2.head()
```

Out[42]:
```
   std  turbo
0    1      0
1    1      0
2    1      0
3    1      0
4    1      0
```

In [43]:
```
dummy_variable_2.rename(columns={'aspiration':'std', 'aspiration':'turbo'}, inplace=Tru
dummy_variable_2.head()
```

Out[43]:
```
   std  turbo
0    1      0
1    1      0
2    1      0
3    1      0
4    1      0
```

In [44]:
```
# merge data frame "df" and "dummy_variable_2"
df = pd.concat([df, dummy_variable_2], axis=1)

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

df.head()
```

```
Out[44]:    symboling  normalized-losses         make num-of-doors   body-style  \
        0          3                122  alfa-romero          two  convertible
        1          3                122  alfa-romero          two  convertible
        2          1                122  alfa-romero          two    hatchback
        3          2                164         audi         four        sedan
        4          2                164         audi         four        sedan

          drive-wheels engine-location  wheel-base    length     width  ...    \
        0          rwd           front        88.6  0.811148  0.890278  ...
        1          rwd           front        88.6  0.811148  0.890278  ...
        2          rwd           front        94.5  0.822681  0.909722  ...
        3          fwd           front        99.8  0.848630  0.919444  ...
        4          4wd           front        99.4  0.848630  0.922222  ...

           city-mpg  highway-mpg    price  city-L/100km  highway-L/100km  \
        0        21           27  13495.0     11.190476         8.703704
        1        21           27  16500.0     11.190476         8.703704
        2        19           26  16500.0     12.368421         9.038462
        3        24           30  13950.0      9.791667         7.833333
        4        18           22  17450.0     13.055556        10.681818

          horsepower-binned  diesel  gas  std  turbo
        0            Medium       0    1    1      0
        1            Medium       0    1    1      0
        2            Medium       0    1    1      0
        3            Medium       0    1    1      0
        4            Medium       0    1    1      0

        [5 rows x 31 columns]
```

Question #4 Answer:
Run the code below! Did you get the right code?
Click here for the solution

```
# 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=T

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

Question #5:
Merge the new dataframe to the original dataframe then drop the column 'aspiration':

```
In [ ]:
```

Question #5 Answer:

Run the code below! Did you get the right code?
Click here for the solution

```
#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

```
In [45]: df.to_csv('clean_df.csv')
```

2 About the Authors:

This notebook written by Mahdi Noorian PhD ,Joseph Santarcangelo PhD, Bahare Talayian, Eric Xiao, Steven Dong, Parizad , Hima Vsudevan and Fiorella Wenver.
Link