Data Wrangling

Estimated time needed: **30** minutes

Objectives

After completing this lab you will be able to:

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

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

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

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

The functions below will download the dataset into your browser:

```
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())
```

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

```
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:

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

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

```
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".

```
df = pd.read_csv(file_name, names = headers, on_bad_lines='skip')
```

> 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/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.

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

   symboling normalized-losses         make fuel-type aspiration num-of-doors  \
0        3.0                 ?  alfa-romero       gas        std          two
1        3.0                 ?  alfa-romero       gas        std          two
2        1.0                 ?  alfa-romero       gas        std          two
3        2.0               164         audi       gas        std         four
4        2.0               164         audi       gas        std         four

    body-style drive-wheels engine-location  wheel-base  ...  engine-size  \
0  convertible          rwd           front        88.6  ...        130.0
1  convertible          rwd           front        88.6  ...        130.0
```

```
2    hatchback          rwd          front        94.5  ...
152.0
3        sedan          fwd          front        99.8  ...
109.0
4        sedan          4wd          front        99.4  ...
136.0

   fuel-system  bore  stroke compression-ratio horsepower  peak-rpm
city-mpg  \
0         mpfi  3.47    2.68               9.0        111      5000
21.0
1         mpfi  3.47    2.68               9.0        111      5000
21.0
2         mpfi  2.68    3.47               9.0        154      5000
19.0
3         mpfi  3.19    3.40              10.0        102      5500
24.0
4         mpfi  3.19    3.40               8.0        115      5500
18.0

  highway-mpg  price
0        27.0  13495
1        27.0  16500
2        26.0  16500
3        30.0  13950
4        22.0  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

Identify and handle missing values

Identify missing values

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: .replace(A, B, inplace = True) to replace A by B.

```
import numpy as np

# replace "?" to NaN
```

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

```
   symboling normalized-losses         make fuel-type aspiration num-of-doors  \
0        3.0               NaN  alfa-romero       gas        std          two
1        3.0               NaN  alfa-romero       gas        std          two
2        1.0               NaN  alfa-romero       gas        std          two
3        2.0               164         audi       gas        std         four
4        2.0               164         audi       gas        std         four

    body-style drive-wheels engine-location  wheel-base  ...  engine-size  \
0  convertible          rwd           front        88.6  ...        130.0
1  convertible          rwd           front        88.6  ...        130.0
2    hatchback          rwd           front        94.5  ...        152.0
3        sedan          fwd           front        99.8  ...        109.0
4        sedan          4wd           front        99.4  ...        136.0

   fuel-system  bore  stroke compression-ratio horsepower  peak-rpm city-mpg  \
0         mpfi  3.47    2.68               9.0        111      5000     21.0
1         mpfi  3.47    2.68               9.0        111      5000     21.0
2         mpfi  2.68    3.47               9.0        154      5000     19.0
3         mpfi  3.19    3.40              10.0        102      5500     24.0
4         mpfi  3.19    3.40               8.0        115      5500     18.0

  highway-mpg  price
0        27.0  13495
1        27.0  16500
2        26.0  16500
3        30.0  13950
4        22.0  17450

[5 rows x 26 columns]
```

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.

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

   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.

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

symboling
False    206
Name: symboling, dtype: int64

normalized-losses
False    165
True      41
Name: normalized-losses, dtype: int64

make
False    206
Name: make, dtype: int64

fuel-type
False    206
Name: fuel-type, dtype: int64

aspiration
False    206
Name: aspiration, dtype: int64

num-of-doors
False    204
True       2
Name: num-of-doors, dtype: int64

body-style
False    206
Name: body-style, dtype: int64

drive-wheels
False    206
Name: drive-wheels, dtype: int64

engine-location
False    206
Name: engine-location, dtype: int64
```

```
wheel-base
False    206
Name: wheel-base, dtype: int64

length
False    206
Name: length, dtype: int64

width
False    206
Name: width, dtype: int64

height
False    206
Name: height, dtype: int64

curb-weight
False    206
Name: curb-weight, dtype: int64

engine-type
False    206
Name: engine-type, dtype: int64

num-of-cylinders
False    206
Name: num-of-cylinders, dtype: int64

engine-size
False    206
Name: engine-size, dtype: int64

fuel-system
False    206
Name: fuel-system, dtype: int64

bore
False    202
True       4
Name: bore, dtype: int64

stroke
False    202
True       4
Name: stroke, dtype: int64

compression-ratio
False    205
True       1
```

```
Name: compression-ratio, dtype: int64

horsepower
False    203
True       3
Name: horsepower, dtype: int64

peak-rpm
False    203
True       3
Name: peak-rpm, dtype: int64

city-mpg
False    205
True       1
Name: city-mpg, dtype: int64

highway-mpg
False    205
True       1
Name: highway-mpg, dtype: int64

price
False    201
True       5
Name: price, 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

Deal with missing data

How should you deal with missing data?

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.

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

```
Average of normalized-losses: 121.2661111111111
```

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

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

```
Average of bore: 3.313267326732673
```

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

```
# Write your code below and press Shift+Enter to execute
#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: 3.256903553299492
```

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

```
Average horsepower: 104.25615763546799
```

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

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

```
Average peak rpm: 5125.369458128079
```

```
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:

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

```
four    115
two      89
Name: num-of-doors, 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:

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

```
'four'
```

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

```
#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:

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

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

df.head()
```

```
   symboling normalized-losses         make fuel-type aspiration num-
of-doors  \
0        3.0        121.266111  alfa-romero       gas        std
two
1        3.0        121.266111  alfa-romero       gas        std
two
2        1.0        121.266111  alfa-romero       gas        std
two
3        2.0               164         audi       gas        std
four
4        2.0               164         audi       gas        std
four

    body-style drive-wheels engine-location  wheel-base  ...  engine-
size  \
0  convertible          rwd           front        88.6  ...
130.0
1  convertible          rwd           front        88.6  ...
130.0
2    hatchback          rwd           front        94.5  ...
152.0
3        sedan          fwd           front        99.8  ...
109.0
4        sedan          4wd           front        99.4  ...
136.0

   fuel-system  bore  stroke compression-ratio horsepower  peak-rpm
city-mpg  \
0         mpfi  3.47    2.68               9.0        111      5000
21.0
1         mpfi  3.47    2.68               9.0        111      5000
```

```
21.0
2          mpfi  2.68    3.47                 9.0         154       5000
19.0
3          mpfi  3.19    3.40                10.0         102       5500
24.0
4          mpfi  3.19    3.40                 8.0         115       5500
18.0

  highway-mpg  price
0        27.0  13495
1        27.0  16500
2        26.0  16500
3        30.0  13950
4        22.0  17450

[5 rows x 26 columns]
```

Good! Now, you have a data set 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, you use: .dtype() to check the data type .astype() to change the data type

```
df.dtypes

symboling            float64
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          float64
fuel-system           object
bore                  object
stroke                object
compression-ratio    float64
horsepower            object
```

```
peak-rpm                 object
city-mpg                float64
highway-mpg             float64
price                    object
dtype: object
```

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

```
df.dtypes
```

```
symboling               float64
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             float64
fuel-system              object
bore                    float64
stroke                  float64
compression-ratio       float64
horsepower               object
peak-rpm                float64
city-mpg                float64
highway-mpg             float64
price                   float64
dtype: object
```

Wonderful!

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

Data Standardization

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.

```
df.head()

   symboling  normalized-losses         make fuel-type aspiration  \
0        3.0                121  alfa-romero       gas        std
1        3.0                121  alfa-romero       gas        std
2        1.0                121  alfa-romero       gas        std
3        2.0                164         audi       gas        std
4        2.0                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        130.0         mpfi  3.47    2.68               9.0        111

1        130.0         mpfi  3.47    2.68               9.0        111

2        152.0         mpfi  2.68    3.47               9.0        154

3        109.0         mpfi  3.19    3.40              10.0        102

4        136.0         mpfi  3.19    3.40               8.0        115


   peak-rpm city-mpg  highway-mpg    price
0    5000.0     21.0         27.0  13495.0
1    5000.0     21.0         27.0  16500.0
2    5000.0     19.0         26.0  16500.0
3    5500.0     24.0         30.0  13950.0
4    5500.0     18.0         22.0  17450.0

[5 rows x 26 columns]
```

```
# 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()
```

```
   symboling  normalized-losses         make fuel-type aspiration  \
0        3.0                121  alfa-romero       gas        std
1        3.0                121  alfa-romero       gas        std
2        1.0                121  alfa-romero       gas        std
3        2.0                164         audi       gas        std
4        2.0                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
21.0
1         mpfi  3.47    2.68                9.0        111   5000.0
21.0
2         mpfi  2.68    3.47                9.0        154   5000.0
19.0
3         mpfi  3.19    3.40               10.0        102   5500.0
24.0
4         mpfi  3.19    3.40                8.0        115   5500.0
18.0

  highway-mpg    price  city-L/100km
0        27.0  13495.0     11.190476
1        27.0  16500.0     11.190476
2        26.0  16500.0     12.368421
3        30.0  13950.0      9.791667
4        22.0  17450.0     13.055556

[5 rows x 27 columns]
```

```python
# Write your code below and press Shift+Enter to execute
# 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()
```

```
   symboling  normalized-losses         make aspiration num-of-
doors  \
0        3.0                121  alfa-romero        std          two

1        3.0                121  alfa-romero        std          two

2        1.0                121  alfa-romero        std          two

3        2.0                164         audi        std         four

4        2.0                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     21.0    8.703704
13495.0
1                9.0         111    5000.0     21.0    8.703704
16500.0
2                9.0         154    5000.0     19.0    9.038462
16500.0
3               10.0         102    5500.0     24.0    7.833333
13950.0
4                8.0         115    5500.0     18.0   10.681818
17450.0

  city-L/100km  horsepower-binned  fuel-type-diesel  fuel-type-gas
```

```
0    11.190476                Low                 0               1
1    11.190476                Low                 0               1
2    12.368421             Medium                 0               1
3     9.791667                Low                 0               1
4    13.055556                Low                 0               1

[5 rows x 29 columns]
```

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)

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

# Write your code below and press Shift+Enter to execute
df['height'] = df['height']/df['height'].max()

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

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

Here you've normalized "length", "width" and "height" to fall 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 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.

Convert data to correct format:

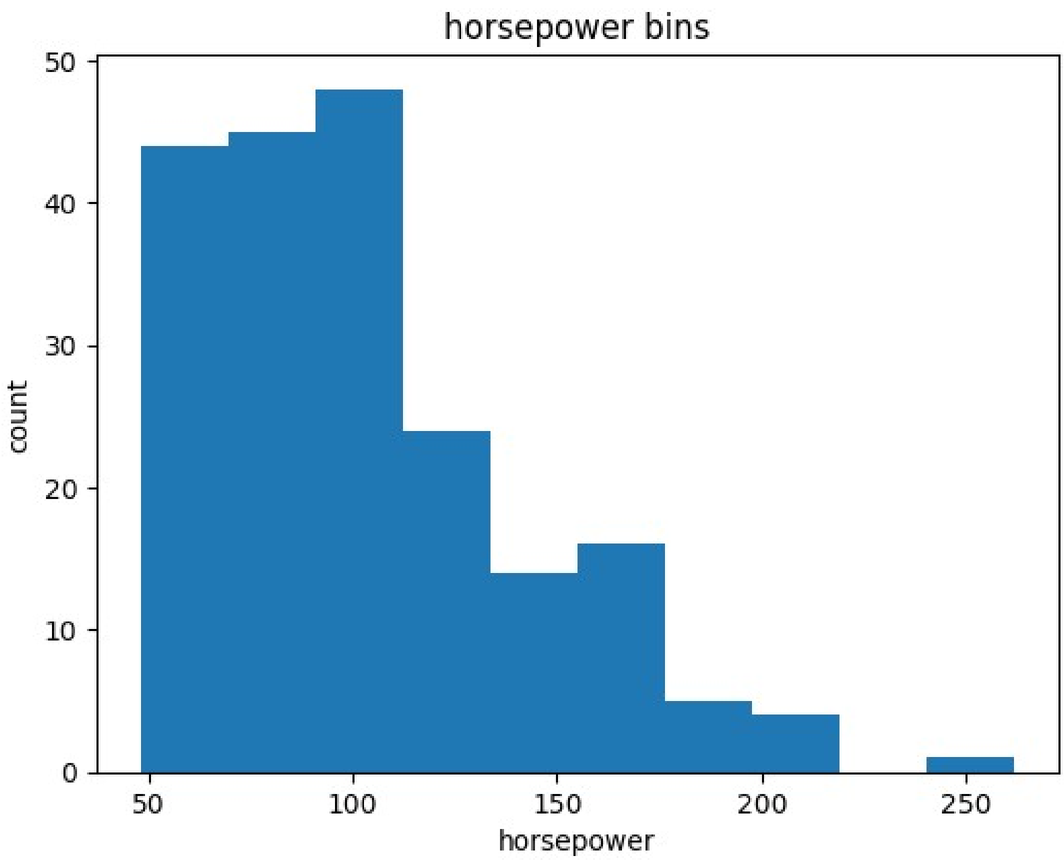
```
df["horsepower"]=df["horsepower"].astype(int, copy=True)
```

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

```
%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")
```

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



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.

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

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

Set group names:

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

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

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

```
    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:

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

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

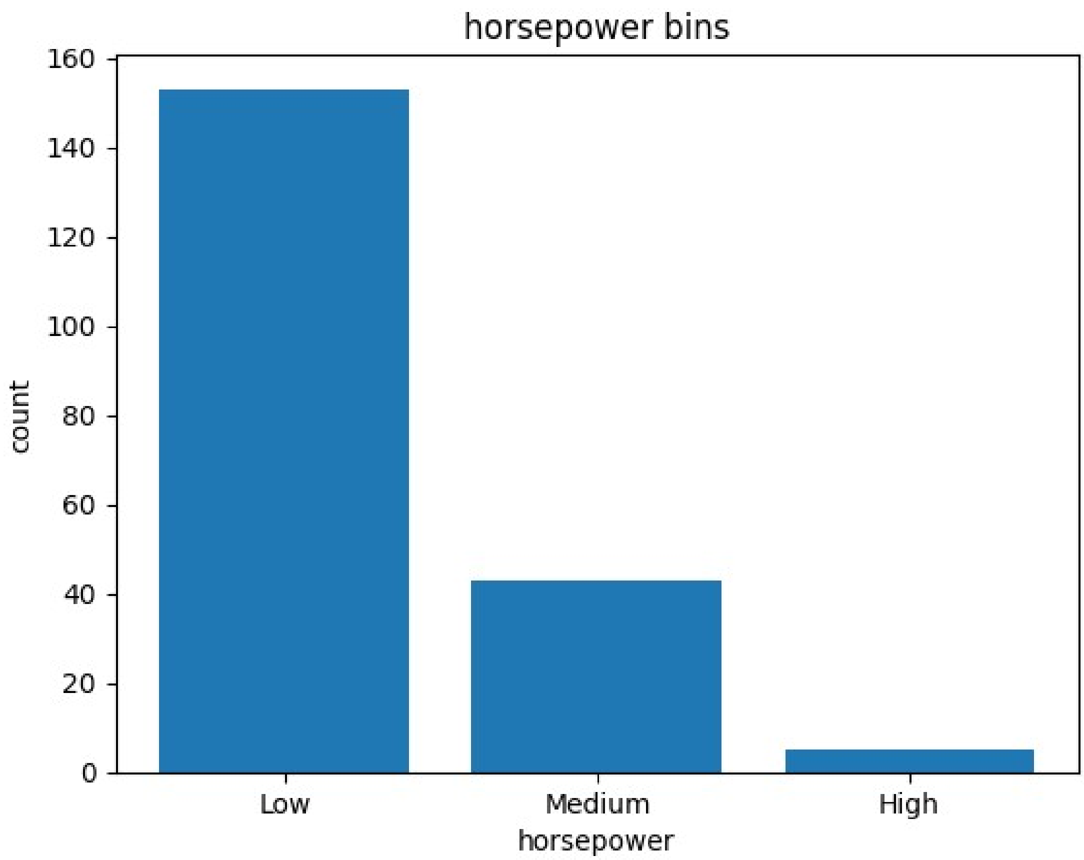
Plot the distribution of each bin:

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

# set x/y labels and plot title
```

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

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



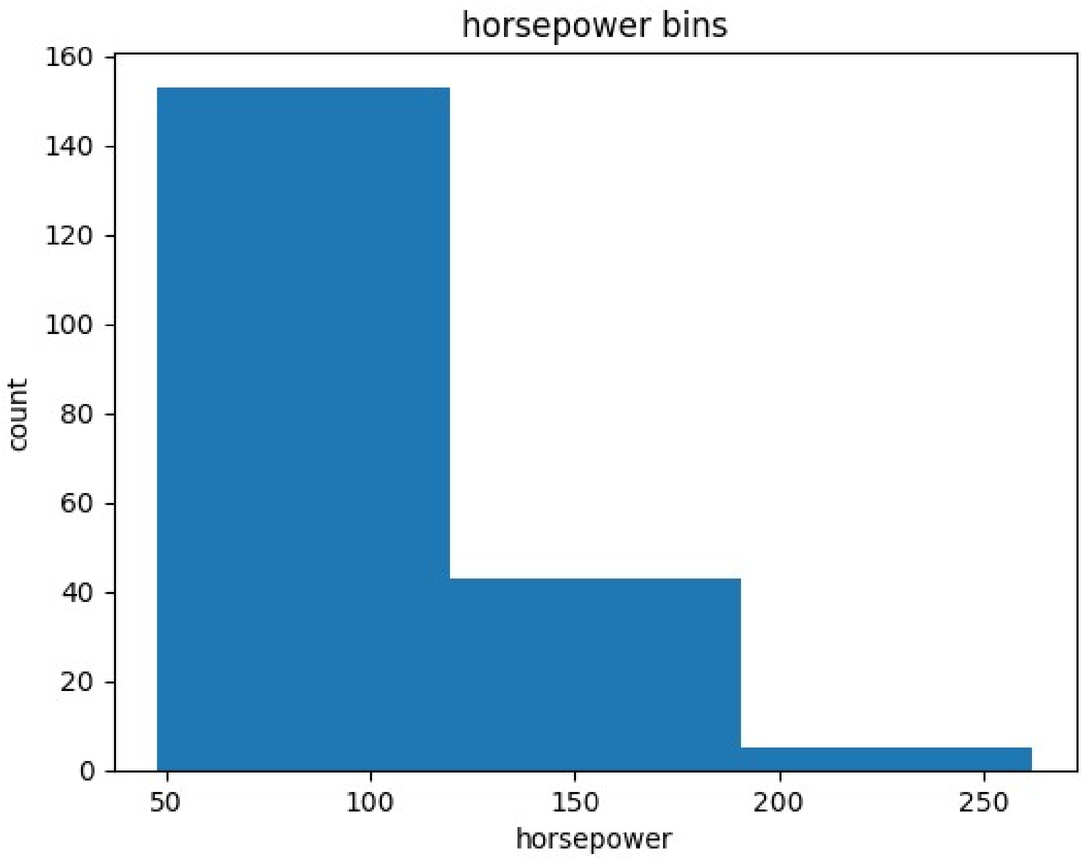
Normally, you use a histogram to visualize the distribution of bins we created above.

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


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

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

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



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

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.

```
df.columns

Index(['symboling', 'normalized-losses', 'make', 'fuel-type',
'aspiration',
       'num-of-doors', 'body-style', 'drive-wheels', 'engine-
location',
       'wheel-base', 'length', 'width', 'height', 'curb-weight',
'engine-type',
       'num-of-cylinders', 'engine-size', 'fuel-system', 'bore',
'stroke',
```

```
       'compression-ratio', 'horsepower', 'peak-rpm', 'city-mpg',
       'highway-mpg', 'price', 'city-L/100km', 'horsepower-binned'],
      dtype='object')
```

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

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

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

Change the column names for clarity:

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

   fuel-type-diesel  fuel-type-gas
0                 0              1
1                 0              1
2                 0              1
3                 0              1
4                 0              1
```

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

```
# 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)

df.head()

   symboling  normalized-losses         make aspiration num-of-
doors  \
0        3.0                121  alfa-romero        std          two

1        3.0                121  alfa-romero        std          two

2        1.0                121  alfa-romero        std          two

3        2.0                164         audi        std         four

4        2.0                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     21.0        27.0
13495.0
1                9.0         111    5000.0     21.0        27.0
16500.0
2                9.0         154    5000.0     19.0        26.0
16500.0
3               10.0         102    5500.0     24.0        30.0
13950.0
4                8.0         115    5500.0     18.0        22.0
17450.0

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

[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.

```
# Write your code below and press Shift+Enter to execute
# 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)
```

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

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

Question #5:

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

```python
# Write your code below and press Shift+Enter to execute
# 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:

```python
df.to_csv('clean_df.csv')
```

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	2.3	Abhishek Gagneja	Instructional Update
2020-10-30	2.2	Lakshmi	Changed URL of csv
2020-09-09	2.1	Lakshmi	Updated Indicator Variables section
2020-08-27	2.0	Lavanya	Moved lab to course repo in GitLab