

Table of content

- Identify and handle missing values
 - Identify missing values
 - Deal with missing values
 - Correct data format
- Data standardization
 - Data Normalization (centering/scaling)
- Binning
 - Indicator variable

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.

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

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
import matplotlib.pyplot as plt
```

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

URL of the dataset

This dataset was hosted on IBM Cloud object click [HERE](#) for free storage

```
In [2]: filename = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDevPython list headers containing name of headers
```

```
In [3]: headers = ["symboling", "normalized-losses", "make", "fuel-type", "aspiration", "num-of-doors", "body-style", "drive-wheels", "engine-location", "wheel-base", "engine-size"]

# Write your code below and press Shift+Enter to execute
df = pd.read_csv(filename, names = headers)
```

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)
```

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()
```

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size
0	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130
1	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130
2	1	?	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152
3	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	105
4	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136

5 rows × 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:

- Identify missing data
- Deal with missing data
- Correct data format

Identify and handle missing values

Identify 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:

```
.replace(A, B, inplace = True)
```

to replace A by B

```
In [6]: 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	body-style	drive-wheels	engine-location	wheel-base	...	engine-size
0	3	NaN	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130
1	3	NaN	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130
2	1	NaN	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152
3	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	105
4	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136

5 rows × 26 columns

Identify missing values

Evaluating for Missing Data

The missing values are converted to default. We use the following functions to identify these missing values. There are 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.

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

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

5 rows × 26 columns

"True" stands for missing value, while "False" stands for not missing value.

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_counts()" counts the number of "True" values.

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

symboling	False	205
Name: symboling, dtype: int64		
normalized-losses	False	164
True	41	2
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	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	4
Name: bore, dtype: int64		
stroke	False	201
True	4	4
Name: stroke, dtype: int64		
compression-ratio	False	205
Name: compression-ratio, dtype: int64		
horsepower	False	203
True	2	2
Name: horsepower, dtype: int64		
peak-rpm	False	203
True	2	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	4
Name: price, dtype: int64		

Based on the summary above, each column has 205 rows of data, seven 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 to deal with missing data?

- drop data
 - drop the whole row
 - drop the whole column
- replace data
 - replace it by mean
 - replace it by frequency
 - 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 occur

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 for prediction; therefore any row now without price data is not useful to us

Calculate the average of the column

```
In [9]: 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" by mean value in "normalized-losses" column

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

Calculate the mean value for "bore" column

```
In [11]: avg_bore=df["bore"].astype('float').mean(axis=0)
print("Average of bore:", avg_bore)
```

Average of bore: 3.3297512437810943

Replace NaN by mean value

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

Calculate the mean value for "stroke" column by mean.

```
In [13]: # Write your code below and press Shift+Enter to execute
avg_stroke=df["stroke"].astype('float').mean(axis=0)
print("Average of stroke is ", avg_stroke)
df["stroke"].replace(np.nan, avg_stroke, inplace=True)
```

Average of stroke is 3.25542288572139

Click here for the solution

Calculate the mean value for the "horsepower" column:

```
In [14]: avg_horsepower = df["horsepower"].astype('float').mean(axis=0)
print("Average horsepower:", avg_horsepower)
```

Average horsepower: 104.25615763546799

Replace "NaN" by mean value:

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

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

```
In [16]: avg_peakrpm=df["peak-rpm"].astype('float').mean(axis=0)
print("Average peak rpm:", avg_peakrpm)
```

Average peak rpm: 5125.369458128079

Replace NaN by mean value:

```
In [17]: 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:

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

19	70	Low
----	----	-----

Lets see the number of vehicles in each bin.

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

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

"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 dropped two rows
df.reset_index(drop=True, inplace=True)
```

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

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size
0	3	122	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130
1	3	122	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130
2	1	122	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152
3	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	105
4	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136

5 rows × 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

Lets list the data types for each column

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

symboling	int64
normalized-losses	object
make	object
fuel-type	object
aspiration	object
num-of-doors	object
body-style	object
drive-wheels	object
engine-location	object
wheel-base	float64
length	float64
width	float64
height	float64
curb-weight	int64
engine-type	int64
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.

```
In [24]: 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")
```

Lets us list the columns after the conversion

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

symboling	int64
normalized-losses	int64
make	object
fuel-type	object
aspiration	object
num-of-doors	object
body-style	object
drive-wheels	object
engine-location	object
wheel-base	float64
length	float64
width	float64
height	float64
curb-weight	int64
engine-type	object
num-of-cylinders	object
engine-size	int64
fuel-system	object
bore	float64
stroke	float64
compression-ratio	float64
horsepower	object
peak-rpm	float64
city-mpg	int64
highway-mpg	int64
price	float64
dtype:	object

Wonderfull!

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 accept 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()
```

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size
0	3	122	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130
1	3	122	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130
2	1	122	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152
3	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	105
4	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136

5 rows × 26 columns

```
In [27]: # 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	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	fuel-system
0	3	122	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	mpfi
1	3	122	alfa-romero	gas	std	two	convertible	rwd	front			


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

```
Out[36]: Low      153
        Medium  43
        High    5
        Name: horsepower-binned, dtype: int64
```

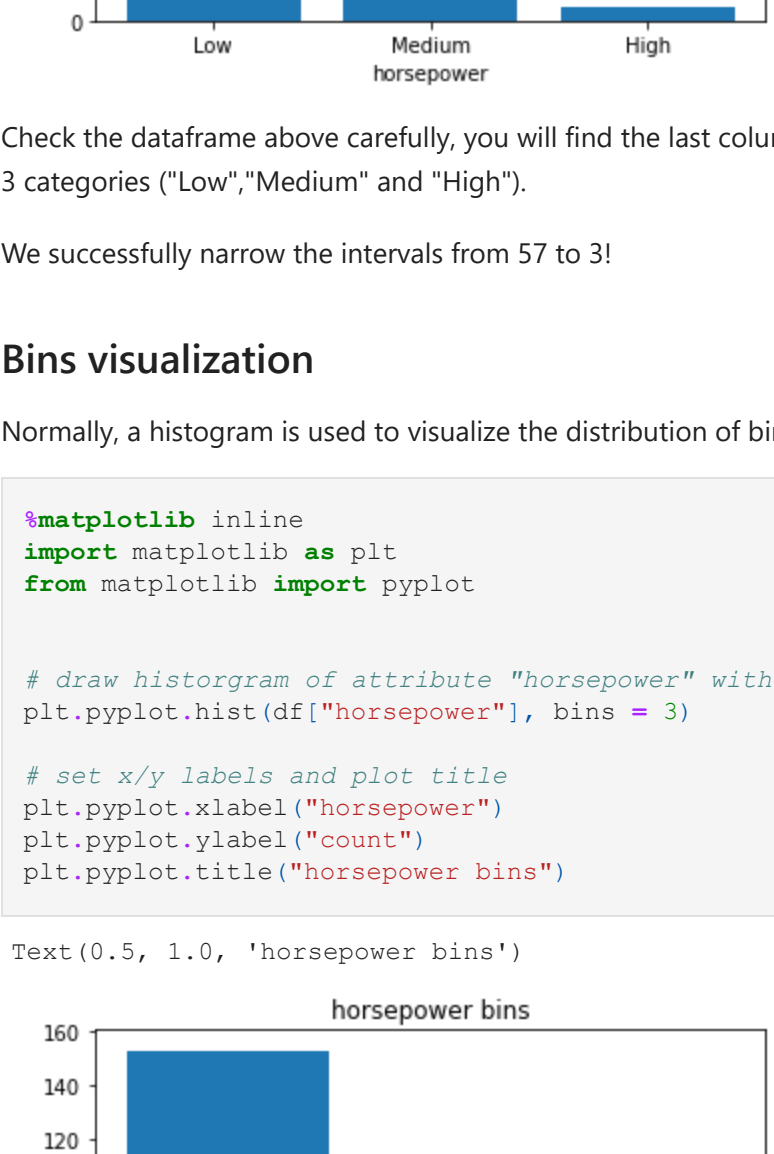
Lets plot the distribution of each bin.

```
In [37]: %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")
```

Out[37]: Text(0.5, 1.0, 'horsepower bins')



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!

Bins visualization

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

```
In [38]: %matplotlib inline
import matplotlib as plt
from matplotlib import pyplot

# draw histogram 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[38]: 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 we use indicator variables?

So we can use categorical variables for regression analysis in the later modules.

Example

We see the column "fuel-type" has two unique values, "gas" or "diesel". Regression doesn't understand words, only numbers. To use this 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 [39]: df.columns

Out[39]: 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/100km', 'price', 'city-l/100km', 'horsepower-binned'], dtype='object')

get indicator variables and assign it to data frame "dummy_variable_1"

```
In [40]: dummy_variable_1 = pd.get_dummies(df["fuel-type"])
dummy_variable_1.head()
```

Out[40]:

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

change column names for clarity

```
In [41]: dummy_variable_1.rename(columns={'gas':'fuel-type-gas', 'diesel':'fuel-type-diesel'},
dummy_variable_1.head()
```

Out[41]:

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

In the dataframe, column fuel-type has a value for 'gas' and 'diesel' as 0s and 1s now.

```
In [42]: # 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 [43]: df.head()

Out[43]:

	symboling	normalized-losses	make	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	length	...	con
0	3	122	alfa-romero	std	two	convertible	rwd	front	88.6	0.811148
1	3	122	alfa-romero	std	two	convertible	rwd	front	88.6	0.811148
2	1	122	alfa-romero	std	two	hatchback	rwd	front	94.5	0.822681
3	2	164	audi	std	four	sedan	fwd	front	99.8	0.848630
4	2	164	audi	std	four	sedan	4wd	front	99.4	0.848630

5 rows x 29 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"

```
In [44]: # Write your code below and press Shift+Enter to execute
dummy_variable_2=pd.get_dummies(df['aspiration'])
dummy_variable_2.rename(columns={'std':'aspiration-std','turbo':'aspiration-turbo'},
dummy_variable_2.head()
```

Out[44]:

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

► Click here for the solution

Question #5:

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

```
In [45]: df = pd.concat([df, dummy_variable_2], axis=1)
df.drop('aspiration', axis = 1, inplace=True)
df.head()
```

Out[45]:

	symboling	normalized-losses	make	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	length	width	...	peak rpm
0	3	122	alfa-romero	two	convertible	rwd	front	88.6	0.811148	0.890278	...	5000
1	3	122	alfa-romero	two	convertible	rwd	front	88.6	0.811148	0.890278	...	5000
2	1	122	alfa-romero	two	hatchback	rwd	front	94.5	0.822681	0.909722	...	5000
3	2	164	audi	four	sedan	fwd	front	99.8	0.848630	0.919444	...	5500
4	2	164	audi	four	sedan	4wd	front	99.4	0.848630	0.922222	...	5500

5 rows x 30 columns

► Click here for the solution

Save the new csv

```
In [46]: df.to_csv('clean_df.csv')
```