

# Week 2 - Data Wrangling

## Learning Objectives

- Describe how to handle missing values
- Describe data formatting techniques
- Describe data normalization
- Demonstrate the use of binning
- Demonstrate the use of categorical variables

## 1. Pre-processing Data in Python

### Data Pre-processing

Also known as:

Data Cleaning, Data Wrangling

The process of converting or mapping data from the initial “raw” form into another format, in order to prepare the data for further analysis.

In this video, we'll be going through some data preprocessing techniques. If you're unfamiliar with the term, data preprocessing is a necessary step in data analysis. It is the process of converting or mapping data from one raw form into another format to make it ready for further analysis. Data preprocessing is often called data cleaning or data wrangling, and there are likely other terms.

Here are the topics that we'll be covering in this module.

- We'll show you how to identify and handle missing values. A missing value condition occurs whenever a data entry is left empty.
- Then we'll cover data formats. Data from different sources maybe in various formats, in different units, or in various conventions. We will introduce some methods in Python Pandas that can standardize the values into the same format, or unit, or convention.
- After that, we'll cover data normalization. Different columns of numerical data may have very different ranges and direct comparison is often not meaningful. Normalization is a way to bring all data into a similar range for more useful comparison. Specifically, we'll focus on the techniques of centering and scaling
- Then, we'll introduce data binning. Binning creates bigger categories from a set of numerical values. It is particularly useful for comparison between groups of data.
- Lastly, we'll talk about categorical variables and show you how to convert categorical values into numeric variables to make statistical modeling easier.

### Simple Dataframe Operations

`df["symboling"]`      `df["body-style"]`

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	engine-size	fuel-system	bore	stroke	compression-ratio
0	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	130	mpfi	3.47	2.68	9.0
1	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	130	mpfi	3.47	2.68	9.0
2	1	?	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	152	mpfi	2.68	3.47	9.0
3	2	164	audi	gas	std	four	sedan	fwd	front	99.8	109	mpfi	3.19	3.40	10.0
4	2	164	audi	gas	std	four	sedan	4wd	front	99.4	136	mpfi	3.19	3.40	8.0
5	2	?	audi	gas	std	two	sedan	fwd	front	99.8	136	mpfi	3.19	3.40	8.5
6	1	158	audi	gas	std	four	sedan	fwd	front	105.8	136	mpfi	3.19	3.40	8.5
7	1	?	audi	gas	std	four	wagon	fwd	front	105.8	136	mpfi	3.19	3.40	8.5
8	1	158	audi	gas	turbo	four	sedan	fwd	front	105.8	131	mpfi	3.13	3.40	8.3
9	0	?	audi	gas	turbo	two	hatchback	4wd	front	99.5	131	mpfi	3.13	3.40	7.0

### Simple Dataframe Operations

`df["symboling"]=df["symboling"]+1`

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	engine-size	fuel-system	bore	stroke	compression-ratio
0	4	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	130	mpfi	3.47	2.68	9.0
1	4	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	130	mpfi	3.47	2.68	9.0
2	2	?	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	152	mpfi	2.68	3.47	9.0
3	3	164	audi	gas	std	four	sedan	fwd	front	99.8	109	mpfi	3.19	3.40	10.0
4	3	164	audi	gas	std	four	sedan	4wd	front	99.4	136	mpfi	3.19	3.40	8.0
5	3	?	audi	gas	std	two	sedan	fwd	front	99.8	136	mpfi	3.19	3.40	8.5
6	2	158	audi	gas	std	four	sedan	fwd	front	105.8	136	mpfi	3.19	3.40	8.5
7	2	?	audi	gas	std	four	wagon	fwd	front	105.8	136	mpfi	3.19	3.40	8.5
8	2	158	audi	gas	turbo	four	sedan	fwd	front	105.8	131	mpfi	3.13	3.40	8.3
9	1	?	audi	gas	turbo	two	hatchback	4wd	front	99.5	131	mpfi	3.13	3.40	7.0

In Python, we usually perform operations along columns.

Each row of the column represents a sample, I.e, a different used car in the database.

You access a column by specifying the name of the column. For example, you can access symboling and body style. Each of these columns is a Panda series.

There are many ways to manipulate Dataframes in Python. For example, you can add a value to each entry off a column. To add one to each symboling entry, use this command. This changes each value of the Data frame column by adding one to the current value.

总结:

## 2. Dealing with Missing Values in Python

### Missing Values

- What is missing value?
- Missing values occur when no data value is stored for a variable (feature) in an observation.
- Could be represented as "?", "N/A", 0 or just a blank cell.

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location
0	3	NaN	alfa-romero	gas	std	two	convertible	rwd	front

### How to deal with missing data?

Check with the data collection source

Drop the missing values

- drop the variable
- drop the data entry

Replace the missing values

- replace it with an average (of similar datapoints)
- replace it by frequency
- replace it based on other functions

Leave it as missing data

In this video, we will introduce the pervasive problem of missing values as well as strategies on what to do when you encounter missing values in your data. When no data value is stored for feature for a particular observation, we say this feature has a missing value. Usually missing value in data set appears as question mark and a zero or just a blank cell. In the example here, the normalized losses feature has a missing value which is represented with NaN. But how can you deal with missing data? There are many ways to deal with missing values and this is regardless of Python, R or whatever tool you use. Of course, each situation is different and should be judged differently. However, these are the typical options you can consider.

- The first is to check if the person or group that collected the data can go back and find what the actual value should be.
- Another possibility is just to remove the data where that missing value is found. When you drop data, you could either drop the whole variable or just the single data entry with the missing value. If you don't have a lot of observations with missing data, usually dropping the particular entry is the best. If you're removing data, you want to look to do something that has the least amount of impact.
- Replacing data is better since no data is wasted. However, it is less accurate since we need to replace missing data with a guess of what the data should be. One standard for placement technique is to replace missing values by the average value of the entire variable. As an example, suppose we have some entries that have missing values for the normalized losses column and the column average for entries with data is 4500. While there is no way for us to get an accurate guess of what the missing value is under the normalized losses column should have been, you can approximate their values using the average value of the column 4500.
- But what if the values cannot be averaged as with categorical variables? For a variable like fuel type, there isn't an average fuel type since the variable values are not numbers. In this case, one possibility is to try using the mode, the most common like gasoline.
- Finally, sometimes we may find another way to guess the missing data. This is usually because the data gathered knows something additional about the missing data. For example, he may know that the missing values tend to be old cars and the normalized losses of old cars are significantly higher than the average vehicle.
- And of course, finally, in some cases you may simply want to leave the missing data as missing data.

### How to drop missing values in Python

- Use `dataframes.dropna()` :

highway-mpg	price
...	...
20	23875
22	NaN
29	16430
...	...

axis=0 drops the entire row  
axis=1 drops the entire column

```
df.dropna(subset=["price"], axis=0, inplace = True)  
df = df.dropna(subset=["price"], axis=0)
```

Use `dataframe.replace(missing_value, new_value)` :

normalized-losses	make
...	...
164	audi
164	audi
NaN	audi
158	audi
...	...

→

normalized-losses	make
...	...
164	audi
164	audi
162	audi
158	audi
...	...

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

So, we've gone through two ways in Python to deal with missing data. We learnt to drop problematic rows or columns containing missing values and then we learnt how to replace missing values with other values. But don't forget the other ways to deal with missing data. You can always check for a higher quality data set or source or in some cases you may want to leave the missing data as missing data.

how would you deal with missing values for categorical data

☒ replace the missing value with the mode of the particular column

✓ 正确

correct, the mode is the value that appears most often

☐ replace the missing value with the mean of the particular column

☒ replace the missing value with the value that appears most often of the particular column

✓ 正确

correct, this is called the mode

what does the following line of code do to the dataframe **df**:

```
1 df.dropna(axis=0)
```

☐ replaces all values **nan** with the mean

☒ drops all rows that contain a **nan**

☐ drops all columns that contain a **nan**

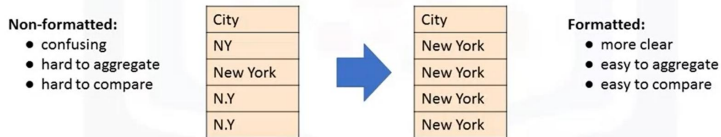
✓ 正确

correct

### 3. Data Formatting in Python

#### Data Formatting

- Data are usually collected from different places and stored in different formats.
- Bringing data into a common standard of expression allows users to make meaningful comparison.



In this video, we'll look at the problem of data with different formats, units and conventions and the pandas methods that help us deal with these issues. Data is usually collected from different places by different people which may be stored in different formats. **Data formatting** means bringing data into a common standard of expression that allows users to make meaningful comparisons. As a part of dataset cleaning, data formatting ensures the data is consistent and easily understandable. For example, people may use different expressions to represent New York City, such as uppercase N uppercase Y, uppercase N lowercase y, uppercase N uppercase Y and New York. Sometimes, this unclean data is a good thing to see. For example, if you're looking at the different ways people tend to write New York, then this is exactly the data that you want. Or if you're looking for ways to spot fraud, perhaps writing N.Y. is more likely to predict an anomaly than if someone wrote out New York in full. But perhaps more often than not, we just simply want to treat them all as the same entity or format to make statistical analyses easier down the road.

Referring to our used car dataset, there's a feature named city-miles per gallon in the dataset, which refers to a car fuel consumption in miles per gallon unit. However, you may be someone who lives in a country that uses metric units. So, you would want to convert those values to liters per 100 kilometers, the metric version. To transform miles per gallon to liters per 100 kilometers, we need to divide 235 by each value in the city-miles per gallon column. In Python, this can easily be done in one line of code. You take the column and set it to equal to 235, divide it by the entire column. In the second line of code, rename column name from city-miles per gallon to city-liters per 100 kilometers using the dataframe rename method.

#### Applying calculations to an entire column

- Convert "mpg" to "L/100km" in Car dataset.



#### Incorrect data types

- Sometimes the wrong data type is assigned to a feature.

```
df["price"].tail(5)
200    16845
201    19045
202    21485
203    22470
204    22625
Name: price, dtype: object
```

#### Data Types in Python and Pandas

- There are many data types in pandas
- Objects : "A", "Hello"..
- Int64 : 1,3,5
- Float64 : 2.123, 632.31,0.12

For a number of reasons, including when you import a dataset into Python, the data type may be incorrectly established. For example, here we noticed the assigned data type to the price feature is object. Although the expected data type should really be an integer or float type.

It is important for later analysis to explore the features data type and convert them to the correct data types. Otherwise, the developed models later on may behave strangely, and totally valid data may end up being treated like missing data.

#### Correcting data types

To identify data types:

- Use `dataframe.dtypes()` to identify data type.

To convert data types:

- Use `dataframe.astype()` to convert data type.

Example: convert data type to integer in column "price"

```
df["price"] = df["price"].astype("int")
```

There are many data types in pandas.

- Objects can be letters or words.
- Int64 are integers.
- Floats are real numbers.

There are many others that we will not discuss.

To identify features data type, in Python we can use the `dataframe.dtypes` method and check the data type of each variable in a data frame. In the case of wrong data types, the method `dataframe.astype` can be used to convert a data type from one format to another. For example, using `astype int` for the price column, you can convert the object column into an integer type variable.



## 4. Data Normalization in Python

### Data Normalization

- Uniform the features value with different range.

length	width	height
168.8	64.1	48.8
168.8	64.1	48.8
171.2	65.5	52.4
176.6	66.2	54.3
176.6	66.4	54.3
177.3	66.3	53.1
192.7	71.4	55.7
192.7	71.4	55.7
192.7	71.4	55.9

scale	[150,250]	[50,100]	[50,100]
impact	large	small	small

### Data Normalization

age	income
20	100000
30	20000
40	500000



age	income
0.2	0.2
0.3	0.04
0.4	1

#### Not-normalized

- "age" and "income" are in different range.
- hard to compare
- "income" will influence the result more

#### Normalized

- similar value range.
- similar intrinsic influence on analytical model.

In this video, we'll be talking about data normalization. An important technique to understand in data pre-processing. When we take a look at the used car data set, we notice in the data that the feature length ranges from 150-250, while feature width and height ranges from 50-100. **We may want to normalize these variables so that the range of the values is consistent. This normalization can make some statistical analyses easier down the road. By making the ranges consistent between variables, normalization enables a fair comparison between the different features, making sure they have the same impact. It is also important for computational reasons.**

Here is another example that will help you understand why normalization is important. Consider a data set containing two features, age and income. Where age ranges from 0-100, while income ranges from 0-20,000 and higher. Income is about 1,000 times larger than age and ranges from 20,000-500,000. So, these two features are in very different ranges. When we do further analysis, like linear regression for example, the attribute income will intrinsically influence the result more due to its larger value. But this doesn't necessarily mean it is more important as a predictor. The nature of the data biases the linear regression model to weigh income more heavily than age. To avoid this, we can normalize these two variables into values that range from 0 to 1. Compare the two tables at the right. After normalization, both variables now have a similar influence on the models we will build later.

## Methods of normalizing data

Several approaches for normalization:

①

$$x_{new} = \frac{x_{old}}{x_{max}}$$

Simple Feature scaling

②

$$x_{new} = \frac{x_{old} - x_{min}}{x_{max} - x_{min}}$$

Min-Max

③

$$x_{new} = \frac{x_{old} - \mu}{\sigma}$$

Z-score


There are several ways to normalize data. I will just outline three techniques.

- The first method called **simple feature scaling** just divides each value by the maximum value for that feature. This makes the new values **range between zero and one**.
- The second method called **min-max** takes each value  $x_{old}$  subtract it from the minimum value of that feature, then divides by the range of that feature. Again, the resulting new values **range between zero and one**.
- The third method is called **z-score** or standard score. In this formula for each value you subtract the  $\mu$  which is the average of the feature, and then divide by the standard deviation  $\sigma$ . The resulting values **hover around zero, and typically range between negative three and positive three but can be higher or lower**.

# Simple Feature Scaling in Python

With Pandas:

length	width	height
168.8	64.1	48.8
168.8	64.1	48.8
180.0	65.5	52.4
...	...	...




length	width	height
0.81	64.1	48.8
0.81	64.1	48.8
0.87	65.5	52.4
...	...	...

```
df["length"] = df["length"]/df["length"].max()
```

# Min-max in Python

With Pandas:

length	width	height
168.8	64.1	48.8
168.8	64.1	48.8
180.0	65.5	52.4
...	...	...




length	width	height
0.41	64.1	48.8
0.41	64.1	48.8
0.58	65.5	52.4
...	...	...

```
df["length"] = (df["length"]-df["length"].min()) /  
(df["length"].max()-df["length"].min())
```

# Z-score in Python

With Pandas:

length	width	height
168.8	64.1	48.8
168.8	64.1	48.8
180.0	65.5	52.4
...	...	...



length	width	height
-0.034	64.1	48.8
-0.034	64.1	48.8
0.039	65.5	52.4
...	...	...

```
df["length"] = (df["length"]-df["length"].mean())/df["length"].std()
```

Following our earlier example, we can apply the normalization method on the length feature.

- First, we use the simple feature scaling method, where we divide it by the maximum value in the feature. Using the pandas method max, this can be done in just one line of code.
- Here's the min-max method on the length feature. We subtract each value by the minimum of that column, then divide it by the range of that column. The max minus the min.
- Finally, we apply the z-score method on length feature to normalize the values. Here we apply the mean and STD method on the length feature. Mean method will return the average value of the feature in the data set, and STD method will return the standard deviation of the features in the data set.

## 5. Binning in Python

### Binning

- Binning: Grouping of values into "bins"
- Converts numeric into categorical variables
- Group a set of numerical values into a set of "bins"
- "price" is a feature range from 5,000 to 45,500 (in order to have a **better representation** of price)

price: 5000, 10000, 12000, 12000, 30000, 31000, 39000, 44000, 44500

bins: low Mid High

In this video, we'll talk about binning as a method of data pre-processing. **Binning** is when you group values together into bins. For example, you can bin "age" into [0 to 5], [6 to 10], [11 to 15] and so on.

Sometimes, binning can improve accuracy of the predictive models. In addition, sometimes we use data binning to group a set of numerical values into a smaller number of bins to have a better understanding of the data distribution.

As example, "price" here is an attribute range from 5,000 to 45,500. Using binning, we categorize the price into three bins: low price, medium price, and high prices.

### Binning in Python pandas

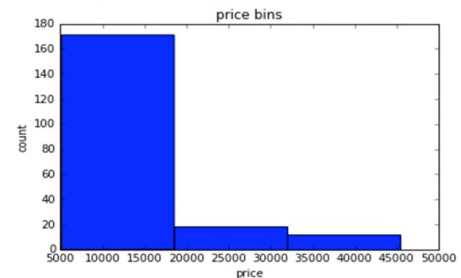
price
13495
16500
18920
41315
5151
6295
...



price	price-binned
13495	Low
16500	Low
18920	Medium
41315	High
5151	Low
6295	Low
...	...

### Visualizing binned data

- E.g., Histograms



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

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

```
df["price-binned"] = pd.cut(df["price"], bins, labels=group_names, include_lowest=True)
```

In the actual car dataset, "price" is a numerical variable ranging from 5188 to 45400, it has 201 unique values. We can categorize them into 3 bins: low, medium, and high-priced cars.

In Python we can easily implement the binning: We would like 3 bins of equal binwidth, so we need 4 numbers as dividers that are equal distance apart. First we use the numpy function "linspace" to return the array "bins" that contains 4 equally spaced numbers over the specified interval of the price. We create a list "group\_names" that contains the different bin names. We use the pandas function "cut" to segment and sort the data values into bins.

You can then use histograms to visualize the distribution of the data after they've been divided into bins. This is the histogram that we plotted based on the binning that we applied in the price feature. From the plot, it is clear that most cars have a low price, and only very few cars have high price.

## 6. Turning categorical variables into quantitative variables in Python

### Categorical Variables

#### Problem:

- Most statistical models cannot take in the objects/strings as input

Car	Fuel	...
A	gas	...
B	diesel	...
C	gas	...
D	gas	...

### Categorical → Numeric

#### Solution:

- Add dummy variables for each unique category
- Assign 0 or 1 in each category

Car	Fuel	...	gas	diesel
A	gas	...	1	0
B	diesel	...	0	1
C	gas	...	1	0
D	gas	...	1	0

“One-hot encoding”

In this video, we'll discuss how to turn categorical variables into quantitative variables in Python. **Most statistical models cannot take in objects or strings as input and for model training only take the numbers as inputs.**

In the car data set, the fuel type feature as a categorical variable has two values, gas or diesel, which are in string format. For further analysis, Jerry has to convert these variables into some form of numeric format. We encode the values by adding new features corresponding to each unique element in the original feature we would like to encode. In the case where the feature fuel has two unique values, gas and diesel, we create two new features, gas and diesel. When a value occurs in the original feature, we set the corresponding value to one in the new feature. The rest of the features are set to zero. In the fuel example, for car B, the fuel value is diesel. Therefore we set the feature diesel equal to one and the gas feature to zero. Similarly, for Car D, the fuel value is gas. Therefore we set the feature gas equal to one and the feature diesel equal to zero. This technique is often called "One-hot encoding".

### Dummy variables in Python pandas

- Use `pandas.get_dummies()` method.
- Convert categorical variables to dummy variables (0 or 1)

fuel	
gas	
diesel	
gas	
gas	

gas	diesel
1	0
0	1
1	0
1	0

```
pd.get_dummies(df['fuel'])
```

In Pandas, we can use `get_dummies` method to convert categorical variables to dummy variables.

In Python, transforming categorical variables to dummy variables is simple. Following the example, **`pd.get_dummies`** method gets the fuel type column and creates the data frame `dummy_variable_1`.

The `get_dummies` method automatically generates a list of numbers, each one corresponding to a particular category of the variable.

### Lesson Summary

**Identify and Handle Missing Values:** Drop rows with incomplete information and impute missing data using the mean values.

**Understand Data Formatting:** Wrangle features in a dataset and make them meaningful for data analysis.

**Apply normalization to a data set:** By understanding the relevance of using feature scaling on your data and how normalization and standardization have varying effects on your data analysis.