Cleaning Data

What is Pandas?

Pandas is a Python library used for working with data sets.

It has functions for analyzing, cleaning, exploring, and manipulating data.

The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis" and was created by Wes McKinney in 2008.

Why Use Pandas?

Pandas allows us to analyze big data and make conclusions based on statistical theories.

Pandas can clean messy data sets, and make them readable and relevant.

Relevant data is very important in data science.

What Can Pandas Do?

Pandas gives you answers about the data. Like:

- Is there a correlation between two or more columns?
- What is average value?
- Max value?
- Min value?

Pandas are also able to delete rows that are not relevant, or contains wrong values, like empty or NULL values. This is called *cleaning* the data.

Where is the Pandas Codebase?

The source code for Pandas is located at this github repository https://github.com/pandas-dev/pandas

github: enables many people to work on the same codebase.

Installation of Pandas

If you have Python and PIP already installed on a system, then installation of Pandas is very easy.

Install it using this command:

```
C:\Users\Your Name>pip install pandas
```

If this command fails, then use a python distribution that already has Pandas installed like, Anaconda, Spyder etc.

Import Pandas

Once Pandas is installed, import it in your applications by adding the `import` keyword:

Import Pandas

Once Pandas is installed, import it in your applications by adding the `import` keyword:

```
import pandas
```

Now Pandas is imported and ready to use.

Example

Get your own Python Server

```
import pandas

mydataset = {
  'cars': ["BMW", "Volvo",
"Ford"],
  'passings': [3, 7, 2]
}

myvar =
pandas.DataFrame(mydataset)

print(myvar)
```

Pandas as pd

Pandas is usually imported under the `pd` alias.

alias: In Python alias are an alternate name for referring to the same thing.

Create an alias with the `as` keyword while importing:

```
import pandas as pd
```

Now the Pandas package can be referred to as `pd` instead of `pandas`.

Checking Pandas Version

The version string is stored under `__version__` attribute.

Example

```
import pandas as pd

print(pd.__version__)
```

Pandas Series

❮ Previous

Next ❯

What is a Series?

A Pandas Series is like a column in a table.

It is a one-dimensional array holding data of any type.

Example

Get your own Python Server

Create a simple Pandas Series from a list:

```
import pandas as pd

a = [1, 7, 2]

myvar = pd.Series(a)

print(myvar)
```

Create Labels

With the `index` argument, you can name your own labels.

Example

Create your own labels:

```
import pandas as pd

a = [1, 7, 2]

myvar = pd.Series(a, index =
["x", "y", "z"])

print(myvar)
```

Key/Value Objects as Series

You can also use a key/value object, like a dictionary, when creating a Series.

Example

Create a simple Pandas Series from a dictionary:

```
import pandas as pd

calories = {"day1": 420,
"day2": 380, "day3": 390}

myvar = pd.Series(calories)

print(myvar)
```

DataFrames

Data sets in Pandas are usually multi-dimensional tables, called DataFrames.

Series is like a column, a DataFrame is the whole table.

Example

Create a DataFrame from two Series:

```
import pandas as pd

data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}

myvar = pd.DataFrame(data)

print(myvar)
```

What is a DataFrame?

A Pandas DataFrame is a 2 dimensional data structure, like a 2 dimensional array, or a table with rows and columns.

Example

Get your own Python Server

Create a simple Pandas DataFrame:

```
import pandas as pd

data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}

#load data into a DataFrame
object:
df = pd.DataFrame(data)

print(df)
```

Result

```
   calories  duration
0       420        50
1       380        40
2       390        45
```

Locate Row

As you can see from the result above, the DataFrame is like a table with rows and columns.

Pandas use the `loc` attribute to return one or more specified row(s)

Example

Return row 0:

```
#refer to the row index:
print(df.loc[0])
```

Result

```
calories    420
duration     50
Name: 0, dtype: int64
```

Load Files Into a DataFrame

If your data sets are stored in a file, Pandas can load them into a DataFrame.

Example

Load a comma separated file (CSV file) into a DataFrame:

```
import pandas as pd

df = pd.read_csv('data.csv')

print(df)
```

Read CSV Files

A simple way to store big data sets is to use CSV files (comma separated files).

CSV files contains plain text and is a well know format that can be read by everyone including Pandas.

In our examples we will be using a CSV file called 'data.csv'.

Download data.csv. or Open data.csv

Example

Get your own Python Server

Load the CSV into a DataFrame:

```
import pandas as pd

df = pd.read_csv('data.csv')

print(df.to_string())
```

max_rows

The number of rows returned is defined in Pandas option settings.

You can check your system's maximum rows with the `pd.options.display.max_rows` statement.

Example

Check the number of maximum returned rows:

```
import pandas as pd

print(pd.options.display.max_rows)
```

In my system the number is 60, which means that if the DataFrame contains more than 60 rows, the `print(df)` statement will return only the headers and the first and last 5 rows.

You can change the maximum rows number with the same statement.

Example

Increase the maximum number of rows to display the entire DataFrame:

```
import pandas as pd

pd.options.display.max_rows =
9999

df = pd.read_csv('data.csv')

print(df)
```

Read JSON

Big data sets are often stored, or extracted as JSON.

JSON is plain text, but has the format of an object, and is well known in the world of programming, including Pandas.

In our examples we will be using a JSON file called 'data.json'.

Open data.json.

Example

Get your own Python Server

Load the JSON file into a DataFrame:

```
import pandas as pd

df = pd.read_json('data.json')

print(df.to_string())
```

Dictionary as JSON

JSON = Python Dictionary

JSON objects have the same format as Python dictionaries.

If your JSON code is not in a file, but in a Python Dictionary, you can load it into a DataFrame directly:

Example

Load a Python Dictionary into a DataFrame:

```
import pandas as pd

data = {
  "Duration":{
    "0":60,
    "1":60,
    "2":60,
    "3":45,
    "4":45,
    "5":60
  },
  "Pulse":{
    "0":110,
    "1":117,
    "2":103,
    "3":109,
    "4":117,
    "5":102
  },
```

```
  "Maxpulse":{
    "0":130,
    "1":145,
    "2":135,
    "3":175,
    "4":148,
    "5":127
  },
  "Calories":{
    "0":409,
    "1":479,
    "2":340,
    "3":282,
    "4":406,
    "5":300
  }
}

df = pd.DataFrame(data)

print(df)
```

Viewing the Data

One of the most used method for getting a quick overview of the DataFrame, is the `head()` method.

The `head()` method returns the headers and a specified number of rows, starting from the top.

Example

Get your own Python Server

Get a quick overview by printing the first 10 rows of the DataFrame:

```
import pandas as pd

df = pd.read_csv('data.csv')

print(df.head(10))
```

Example

Print the first 5 rows of the DataFrame:

```
import pandas as pd

df = pd.read_csv('data.csv')

print(df.head())
```

Try it Yourself »

There is also a `tail()` method for viewing the *last* rows of the DataFrame.

The `tail()` method returns the headers and a specified number of rows, starting from the bottom.

Example

Print the last 5 rows of the DataFrame:

```
print(df.tail())
```

Info About the Data

The DataFrames object has a method called `info()`, that gives you more information about the data set.

Example

Print information about the data:

```
print(df.info())
```

Result

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 169 entries, 0 to 168
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Duration  169 non-null    int64
 1   Pulse     169 non-null    int64
 2   Maxpulse  169 non-null    int64
 3   Calories  164 non-null    float64
dtypes: float64(1), int64(3)
memory usage: 5.4 KB
None
```

Result Explained

The result tells us there are 169 rows and 4 columns:

```
RangeIndex: 169 entries, 0 to 168
Data columns (total 4 columns):
```

And the name of each column, with the data type:

```
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Duration  169 non-null    int64
 1   Pulse     169 non-null    int64
 2   Maxpulse  169 non-null    int64
 3   Calories  164 non-null    float64
```

Null Values

The `info()` method also tells us how many Non-Null values there are present in each column, and in our data set it seems like there are 164 of 169 Non-Null values in the "Calories" column.

column, and in our data set it seems like there are 164 of 169 Non-Null values in the "Calories" column.

Which means that there are 5 rows with no value at all, in the "Calories" column, for whatever reason.

Empty values, or Null values, can be bad when analyzing data, and you should consider removing rows with empty values. This is a step towards what is called *cleaning data*, and you will learn more about that in the next chapters.

Data Cleaning

Data cleaning means fixing bad data in your data set.

Bad data could be:

- Empty cells
- Data in wrong format
- Wrong data
- Duplicates

In this tutorial you will learn how to deal with all of them.

Our Data Set

In the next chapters we will use this data set:

```
    Duration           Date  Pulse  Max
0         60  '2020/12/01'    110
1         60  '2020/12/02'    117
2         60  '2020/12/03'    103
3         45  '2020/12/04'    109
4         45  '2020/12/05'    117
5         60  '2020/12/06'    102
6         60  '2020/12/07'    110
7        450  '2020/12/08'    104
8         30  '2020/12/09'    109
9         60  '2020/12/10'     98
10        60  '2020/12/11'    103
11        60  '2020/12/12'    100
12        60  '2020/12/12'    100
13        60  '2020/12/13'    106
14        60  '2020/12/14'    104
15        60  '2020/12/15'     98
16        60  '2020/12/16'     98
17        60  '2020/12/17'    100
18        45  '2020/12/18'     90
19        60  '2020/12/19'    103
20        45  '2020/12/20'     97
21        60  '2020/12/21'    108
22        45           NaN    100
23        60  '2020/12/23'    130
24        45  '2020/12/24'    105
25        60  '2020/12/25'    102
26        60    2020/12/26    100
27        60  '2020/12/27'     92
28        60  '2020/12/28'    103
29        60  '2020/12/29'    100
30        60  '2020/12/30'    102
31        60  '2020/12/31'     92
```

The data set contains some empty cells ("Date" in row 22, and "Calories" in row 18 and 28).

The data set contains wrong format ("Date" in row 26).

The data set contains wrong data ("Duration" in row 7).

The data set contains duplicates (row 11 and 12).

Empty Cells

Empty cells can potentially give you a wrong result when you analyze data.

Remove Rows

One way to deal with empty cells is to remove rows that contain empty cells.

This is usually OK, since data sets can be very big, and removing a few rows will not have a big impact on the result.

Example

Return a new Data Frame with no empty cells:

```
import pandas as pd

df = pd.read_csv('data.csv')

new_df = df.dropna()

print(new_df.to_string())
```

Note: By default, the `dropna()` method returns a *new* DataFrame, and will not change the original.

If you want to change the original DataFrame, use the `inplace = True` argument:

Example

Remove all rows with NULL values:

```
import pandas as pd

df = pd.read_csv('data.csv')

df.dropna(inplace = True)

print(df.to_string())
```

Replace Empty Values

Another way of dealing with empty cells is to insert a *new* value instead.

This way you do not have to delete entire rows just because of some empty cells.

The `fillna()` method allows us to replace empty cells with a value:

Example

Replace NULL values with the number 130:

```
import pandas as pd

df = pd.read_csv('data.csv')

df.fillna(130, inplace = True)
```

Replace Only For Specified Columns

The example above replaces all empty cells in the whole Data Frame.

To only replace empty values for one column, specify the *column name* for the DataFrame:

Example

Replace NULL values in the "Calories" columns with the number 130:

```
import pandas as pd

df = pd.read_csv('data.csv')

df["Calories"].fillna(130,
inplace = True)
```

Replace Using Mean, Median, or Mode

A common way to replace empty cells, is to calculate the mean, median or mode value of the column.

Pandas uses the `mean()` `median()` and `mode()` methods to calculate the respective values for a specified column:

Example

Calculate the MEAN, and replace any empty values with it:

```
import pandas as pd

df = pd.read_csv('data.csv')

x = df["Calories"].mean()

df["Calories"].fillna(x,
inplace = True)
```

Mean = the average value (the sum of all values divided by number of values).

Example

Calculate the MEDIAN, and replace any empty values with it:

```
import pandas as pd

df = pd.read_csv('data.csv')

x = df["Calories"].median()

df["Calories"].fillna(x,
inplace = True)
```

Median = the value in the middle, after you have sorted all values ascending.

Example

Calculate the MODE, and replace any empty values with it:

```
import pandas as pd

df = pd.read_csv('data.csv')

x = df["Calories"].mode()[0]

df["Calories"].fillna(x,
inplace = True)
```

Example

Calculate the MODE, and replace any empty values with it:

```
import pandas as pd

df = pd.read_csv('data.csv')

x = df["Calories"].mode()[0]

df["Calories"].fillna(x,
inplace = True)
```

Try it Yourself »

Mode = the value that appears most frequently.

Data of Wrong Format

Cells with data of wrong format can make it difficult, or even impossible, to analyze data.

To fix it, you have two options: remove the rows, or convert all cells in the columns into the same format.

Convert Into a Correct Format

In our Data Frame, we have two cells with the wrong format. Check out row 22 and 26, the 'Date' column should be a string that represents a date:

	Duration	Date	Pulse	Max
0	60	'2020/12/01'	110	
1	60	'2020/12/02'	117	
2	60	'2020/12/03'	103	
3	45	'2020/12/04'	109	
4	45	'2020/12/05'	117	
5	60	'2020/12/06'	102	
6	60	'2020/12/07'	110	
7	450	'2020/12/08'	104	
8	30	'2020/12/09'	109	
9	60	'2020/12/10'	98	
10	60	'2020/12/11'	103	
11	60	'2020/12/12'	100	
12	60	'2020/12/12'	100	
13	60	'2020/12/13'	106	
14	60	'2020/12/14'	104	
15	60	'2020/12/15'	98	
16	60	'2020/12/16'	98	
17	60	'2020/12/17'	100	
18	45	'2020/12/18'	90	
19	60	'2020/12/19'	103	
20	45	'2020/12/20'	97	
21	60	'2020/12/21'	108	
22	45	NaN	100	
23	60	'2020/12/23'	130	
24	45	'2020/12/24'	105	
25	60	'2020/12/25'	102	
26	60	20201226	100	
27	60	'2020/12/27'	92	
28	60	'2020/12/28'	103	
29	60	'2020/12/29'	100	
30	60	'2020/12/30'	102	
31	60	'2020/12/31'	92	

Let's try to convert all cells in the 'Date' column into dates.

Pandas has a `to_datetime()` method for this:

Example

Get your own Python Server

Convert to date:

```
import pandas as pd

df = pd.read_csv('data.csv')

df['Date'] =
pd.to_datetime(df['Date'])

print(df.to_string())
```

Result:

```
    Duration          Date  Pulse  Maxpu
0         60  '2020/12/01'    110
1         60  '2020/12/02'    117
2         60  '2020/12/03'    103
3         45  '2020/12/04'    109
4         45  '2020/12/05'    117
5         60  '2020/12/06'    102
6         60  '2020/12/07'    110
7        450  '2020/12/08'    104
8         30  '2020/12/09'    109
9         60  '2020/12/10'     98
10        60  '2020/12/11'    103
11        60  '2020/12/12'    100
12        60  '2020/12/12'    100
13        60  '2020/12/13'    106
14        60  '2020/12/14'    104
15        60  '2020/12/15'     98
16        60  '2020/12/16'     98
17        60  '2020/12/17'    100
18        45  '2020/12/18'     90
19        60  '2020/12/19'    103
20        45  '2020/12/20'     97
21        60  '2020/12/21'    108
22        45           NaT    100
23        60  '2020/12/23'    130
24        45  '2020/12/24'    105
25        60  '2020/12/25'    102
26        60  '2020/12/26'    100
27        60  '2020/12/27'     92
28        60  '2020/12/28'    103
29        60  '2020/12/29'    100
30        60  '2020/12/30'    102
31        60  '2020/12/31'     92
```

Removing Rows

The result from the converting in the example above gave us a NaT value, which can be handled as a NULL value, and we can remove the row by using the `dropna()` method.

Example

Remove rows with a NULL value in the "Date" column:

```
df.dropna(subset=['Date'],
inplace = True)
```

Pandas - Fixing Wrong Data

❮ Previous

Next ❯

Wrong Data

"Wrong data" does not have to be "empty cells" or "wrong format", it can just be wrong, like if someone registered "199" instead of "1.99".

Sometimes you can spot wrong data by looking at the data set, because you have an expectation of what it should be.

If you take a look at our data set, you can see that in row 7, the duration is 450, but for all the other rows the duration is between 30 and 60.

It doesn't have to be wrong, but taking in consideration that this is the data set of someone's workout sessions, we conclude with the fact that this person did not work out in 450 minutes.

	Duration	Date	Pulse	Max
0	60	'2020/12/01'	110	
1	60	'2020/12/02'	117	
2	60	'2020/12/03'	103	
3	45	'2020/12/04'	109	
4	45	'2020/12/05'	117	
5	60	'2020/12/06'	102	
6	60	'2020/12/07'	110	
7	450	'2020/12/08'	104	
8	30	'2020/12/09'	109	
9	60	'2020/12/10'	98	
10	60	'2020/12/11'	103	
11	60	'2020/12/12'	100	
12	60	'2020/12/12'	100	
13	60	'2020/12/13'	106	
14	60	'2020/12/14'	104	
15	60	'2020/12/15'	98	
16	60	'2020/12/16'	98	
17	60	'2020/12/17'	100	
18	45	'2020/12/18'	90	
19	60	'2020/12/19'	103	
20	45	'2020/12/20'	97	
21	60	'2020/12/21'	108	
22	45	NaN	100	
23	60	'2020/12/23'	130	
24	45	'2020/12/24'	105	
25	60	'2020/12/25'	102	
26	60	20201226	100	
27	60	'2020/12/27'	92	
28	60	'2020/12/28'	103	
29	60	'2020/12/29'	100	
30	60	'2020/12/30'	102	
31	60	'2020/12/31'	92	

How can we fix wrong values, like the one for "Duration" in row 7?

Replacing Values

One way to fix wrong values is to replace them with something else.

In our example, it is most likely a typo, and the value should be "45" instead of "450", and we could just insert "45" in row 7:

Example

Get your own Python Server

Set "Duration" = 45 in row 7:

```
df.loc[7, 'Duration'] = 45
```

Try it Yourself »

For small data sets you might be able to replace the wrong data one by one, but not for big data sets.

To replace wrong data for larger data sets you can create some rules, e.g. set some boundaries for legal values, and replace any values that are outside of the boundaries.

Example

Loop through all values in the "Duration" column.

If the value is higher than 120, set it to 120:

```
for x in df.index:
  if df.loc[x, "Duration"] >
120:
    df.loc[x, "Duration"] = 120
```

Try it Yourself »

Removing Rows

Another way of handling wrong data is to remove the rows that contains wrong data.

This way you do not have to find out what to replace them with, and there is a good chance you do not need them to do your analyses.

Example

Delete rows where "Duration" is higher than 120:

```
for x in df.index:
  if df.loc[x, "Duration"] >
120:
    df.drop(x, inplace = True)
```

Discovering Duplicates

Duplicate rows are rows that have been registered more than one time.

```
    Duration          Date  Pulse  Max
0         60  '2020/12/01'    110
1         60  '2020/12/02'    117
2         60  '2020/12/03'    103
3         45  '2020/12/04'    109
4         45  '2020/12/05'    117
5         60  '2020/12/06'    102
6         60  '2020/12/07'    110
7        450  '2020/12/08'    104
8         30  '2020/12/09'    109
9         60  '2020/12/10'     98
10        60  '2020/12/11'    103
11        60  '2020/12/12'    100
12        60  '2020/12/12'    100
13        60  '2020/12/13'    106
14        60  '2020/12/14'    104
15        60  '2020/12/15'     98
16        60  '2020/12/16'     98
17        60  '2020/12/17'    100
18        45  '2020/12/18'     90
19        60  '2020/12/19'    103
20        45  '2020/12/20'     97
21        60  '2020/12/21'    108
22        45           NaN    100
23        60  '2020/12/23'    130
24        45  '2020/12/24'    105
25        60  '2020/12/25'    102
26        60      20201226    100
27        60  '2020/12/27'     92
28        60  '2020/12/28'    103
29        60  '2020/12/29'    100
30        60  '2020/12/30'    102
31        60  '2020/12/31'     92
```

By taking a look at our test data set, we can assume that row 11 and 12 are duplicates.

To discover duplicates, we can use the `duplicated()` method.

The `duplicated()` method returns a Boolean values for each row:

Example

Get your own Python Server

Returns `True` for every row that is a duplicate, otherwise `False`:

```
print(df.duplicated())
```

Removing Duplicates

To remove duplicates, use the `drop_duplicates()` method.

Example

Remove all duplicates:

```
df.drop_duplicates(inplace = True)
```

Try it Yourself »

Remember: The `(inplace = True)` will make sure that the method does NOT return a *new* DataFrame, but it will remove all duplicates from the *original* DataFrame.