Langage Evolué 2

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

Pandas Getting Started

- **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.

Pandas Getting Started

- **Import Pandas**

```
import pandas
```

– Now Pandas is imported and ready to use.

Example

```
import pandas

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

myvar = pandas.DataFrame(mydataset)

print(myvar)
```

```
    cars  passings
0    BMW         3
1  Volvo         7
2   Ford         2
```

Pandas Getting Started

- Create an alias with the as keyword while importing:

Import pandas as ps

  - Now the Pandas package can be referred to as ps instead of pandas.


```
import pandas as ps

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

myvar = ps.DataFrame(mydataset)

print(myvar)
```

```
    cars  passings
0    BMW         3
1  Volvo         7
2   Ford         2
```

Pandas Getting Started

- The version string is stored under __version__ attribute.


```
import pandas as ps

print(ps.__version__)
```

```
1.0.3
```

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.


```
import pandas as ps

a = [1, 7, 2]

myvar = ps.Series(a)

print(myvar)
```

```
0    1
1    7
2    2
dtype: int64
```

Labels

- If nothing else is specified, the values are labeled with their index number. First value has index 0, second value has index 1 etc.
- This label can be used to access a specified value.


```
import pandas as ps

a = [1, 7, 2]

myvar = ps.Series(a)

print(myvar[1])
```

```
7
```

Create Labels

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

```
import pandas as ps

a = [1, 7, 2]

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

print(myvar)
```

```
x    1
y    7
z    2
dtype: int64
```

Create Labels

- When you have created labels, you can access an item by referring to the label.

```
import pandas as ps

a = [1, 7, 2]

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

print(myvar["z"])
```


2

Key/Value Objects as Series

- You can also use a key/value object, like a dictionary, when creating a Series.
  - Create a simple Pandas Series from a dictionary:

```
import pandas as ps

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

myvar = ps.Series(calories)

print(myvar)
```

```
day1    420
day2    380
day3    390
dtype: int64
```

Key/Value Objects as Series

- To select only some of the items in the dictionary, use the index argument and specify only the items you want to include in the Series.
  - Create a Series using only data from "day1" and "day2":


```
import pandas as ps

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

myvar = ps.Series(calories, index = ["day1", "day2"])

print(myvar)
```

```
day1    420
day2    380
dtype: int64
```

DataFrames

- Data sets in Pandas are usually multi-dimensional tables, called DataFrames.
- Series is like a column, a DataFrame is the whole table.
  - Create a DataFrame from two Series:


```
import pandas as ps

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

myvar = ps.DataFrame(data)

print(myvar)
```

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

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.
  - Create a simple Pandas DataFrame:


```
import pandas as ps

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

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

print(df)
```

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

DataFrames

- **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)


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

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

```
#use a list of indexes:
print(df.loc[[0, 1]])
```

```
   calories  duration
0       420        50
1       380        40
```

DataFrames

- **Named Indexes**
  - With the index argument, you can name your own indexes.
  - Add a list of names to give each row a name:


```
import pandas as ps

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

df = ps.DataFrame(data, index = ["day1", "day2", "day3"])

print(df)
```

	calories	duration
day1	420	50
day2	380	40
day3	390	45

DataFrames

- **Locate Named Indexes**
  - Use the named index in the loc attribute to return the specified row(s).

```
#refer to the named index:
print(df.loc["day2"])
```

```
calories    380
duration     40
Name: 0, dtype: int64
```

DataFrames

- **Load Files Into a DataFrame**

```
import pandas as ps

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

print(df)
```

```
     Duration  Pulse  Maxpulse  Calories
0          60    110       130     409.1
1          60    117       145     479.0
2          60    103       135     340.0
3          45    109       175     282.4
4          45    117       148     406.0
..        ...    ...       ...       ...
164        60    105       140     290.8
165        60    110       145     300.4
166        60    115       145     310.2
167        75    120       150     320.4
168        75    125       150     330.4

[169 rows x 4 columns]
```

Pandas Read CSV

- **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'.


```
import pandas as ps

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

print(df)
```

	Duration	Pulse	Maxpulse	Calories
0	60	110	130	409.1
1	60	117	145	479.0
2	60	103	135	340.0
3	45	109	175	282.4
4	45	117	148	406.0
5	60	102	127	300.5
6	60	110	136	374.0
7	45	104	134	253.3
8	30	109	133	195.1
9	60	98	124	269.0
10	60	103	147	329.3
11	60	100	120	250.7

Pandas Read CSV

- **max_rows**
  - The number of rows returned is defined in Pandas option settings.
  - You can check your system's maximum rows with the statement.

```
import pandas as ps

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

```
     Duration  Pulse  Maxpulse  Calories
0          60    110       130     409.1
1          60    117       145     479.0
2          60    103       135     340.0
3          45    109       175     282.4
4          45    117       148     406.0
..        ...    ...       ...       ...
164        60    105       140     290.8
165        60    110       145     300.4
166        60    115       145     310.2
167        75    120       150     320.4
168        75    125       150     330.4

[169 rows x 4 columns]
```


  - 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.

Pandas Read CSV

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

```
import pandas as ps

ps.options.display.max_rows = 90

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

print(df)
```

Pandas Read JSON

- **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'.

Pandas - Analyzing DataFrames

- **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.


```
Get a quick overview by printing the first 10 rou

import pandas as ps

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

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

	Duration	Pulse	Maxpulse	Calories
0	60	110	130	409.1
1	60	117	145	479.0
2	60	103	135	340.0
3	45	109	175	282.4
4	45	117	148	406.0
5	60	102	127	300.5
6	60	110	136	374.0
7	45	104	134	253.3
8	30	109	133	195.1
9	60	98	124	269.0

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 course you will learn how to deal with all of them.

Data Cleaning

- **Our Data Set**
  - In the next chapters we will use this data set:

	Duration	Date	Pulse	Maxpulse	Calories
0	60	'2020/12/01'	110	130	409.1
1	60	'2020/12/02'	117	145	479.0
2	60	'2020/12/03'	103	135	340.0
3	45	'2020/12/04'	109	175	282.4
4	45	'2020/12/05'	117	148	406.0
5	60	'2020/12/06'	102	127	300.0
6	60	'2020/12/07'	110	136	374.0
7	450	'2020/12/08'	104	134	253.3
8	30	'2020/12/09'	109	133	195.1
9	60	'2020/12/10'	98	124	269.0
10	60	'2020/12/11'	103	147	329.3
11	60	'2020/12/12'	100	120	250.7
12	60	'2020/12/12'	100	120	250.7
13	60	'2020/12/13'	106	128	345.3
14	60	'2020/12/14'	104	132	379.3
15	60	'2020/12/15'	98	123	275.0
16	60	'2020/12/16'	98	120	215.2
17	60	'2020/12/17'	100	120	300.0
18	45	'2020/12/18'	90	112	NaN
19	60	'2020/12/19'	103	123	323.0
20	45	'2020/12/20'	97	125	243.0
21	60	'2020/12/21'	108	131	364.2
22	45	NaN	100	119	282.0
23	60	'2020/12/23'	130	101	300.0
24	45	'2020/12/24'	105	132	246.0
25	60	'2020/12/25'	102	126	334.5
26	60	2020/12/26	100	120	250.0

Data Cleaning

- **In the next chapters we will use this data set:**
  - 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)

Pandas - Cleaning Empty Cells

- **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.

Pandas - Cleaning Empty Cells

```
import pandas as ps

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

new_df = df.dropna()

print(new_df.to_string())
```


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

- Notice in the result that some rows have been removed (row 18, 22 and 28).
- These rows had cells with empty values.

Pandas - Cleaning Empty Cells

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

```
import pandas as ps

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

df.dropna(inplace = True)

print(df.to_string())
```


  - Notice in the result that some rows have been removed (row 18, 22 and 28).
  - These rows had cells with empty values.
  - Now, the dropna(inplace = True) will NOT return a new DataFrame, but it will remove all rows containing NULL values from the original DataFrame.

Pandas - Cleaning Empty Cells

- **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:


```
import pandas as ps

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

df.fillna(130, inplace = True)

print(df.to_string())
```

Pandas - Cleaning Empty Cells

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


```
import pandas as ps

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

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

print(df.to_string())
```

Pandas - Cleaning Empty Cells

- **Replace Using Mean, Median, or Mode**
  - **Mean** = the average value (the sum of all values divided by number of values).
  - 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:
    - Calculate the MEAN, and replace any empty values with it:


```
import pandas as ps

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

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

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

print(df.to_string())
```

As you can see in row 18 and 28, the empty values from "Calories" was replaced with the mean: 304.68

Pandas - Cleaning Empty Cells

- **Replace Using Mean, Median, or Mode**
  - **Median** = the value in the middle, after you have sorted all values ascending.

```
import pandas as ps

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

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

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

print(df.to_string())
```


  - As you can see in row 18 and 28, the empty values from "Calories" was replaced with the median: 291.2

Pandas - Cleaning Empty Cells

- **Replace Using Mean, Median, or Mode**
  - **Mode** = the value that appears most frequently.
  - Calculate the MODE, and replace any empty values with it:


```
import pandas as ps

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

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

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

print(df.to_string())
```

Pandas - Cleaning Data of Wrong Format

- **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.

Pandas - Cleaning Data of Wrong 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:

18	45	'2020/12/18'	90	112	NaN
19	60	'2020/12/19'	103	123	323.0
20	45	'2020/12/20'	97	125	243.0
21	60	'2020/12/21'	108	131	364.2
22	45	NaN	100	119	282.0
23	60	'2020/12/23'	130	101	300.0
24	45	'2020/12/24'	105	132	246.0
25	60	'2020/12/25'	102	126	334.5
26	60	20201226	100	120	250.0

Pandas - Cleaning Data of Wrong Format

- Let's try to convert all cells in the 'Date' column into dates.
- Pandas has a to_datetime() method for this:

```
import pandas as ps

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

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

print(df.to_string())
```

21	60	'2020/12/21'	108	131	364.2
22	45	NaT	100	119	282.0
23	60	'2020/12/23'	130	101	300.0
24	45	'2020/12/24'	105	132	246.0
25	60	'2020/12/25'	102	126	334.5
26	60	'2020/12/26'	100	120	250.0
27	60	'2020/12/27'	92	118	241.0

- As you can see from the result, the date in row 26 was fixed, but the empty date in row 22 got a NaT (Not a Time) value, in other words an empty value. One way to deal with empty values is simply removing the entire row.

Pandas - Cleaning Data of Wrong Format

- **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.


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

Pandas - Fixing Wrong Data

- **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.

Pandas - Fixing Wrong Data

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

	Duration	Date	Pulse	Maxpulse	Calories
0	60	'2020/12/01'	110	130	409.1
1	60	'2020/12/02'	117	145	479.0
2	60	'2020/12/03'	103	135	340.0
3	45	'2020/12/04'	109	175	282.4
4	45	'2020/12/05'	117	148	406.0
5	60	'2020/12/06'	102	127	300.0
6	60	'2020/12/07'	110	136	374.0
7	450	'2020/12/08'	104	134	253.3
8	30	'2020/12/09'	109	133	195.1
9	60	'2020/12/10'	98	124	269.0
10	60	'2020/12/11'	103	147	329.3
11	60	'2020/12/12'	100	120	250.7
12	60	'2020/12/12'	100	120	250.7
13	60	'2020/12/13'	106	128	345.3

Pandas - Fixing Wrong Data

- **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:
  - Exemple
    - Set "Duration" = 45 in row 7:

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

Pandas - Fixing Wrong Data

– Exemple

- Set "Duration" = 45 in row 7

5	60	'2020/12/06'	102	127	300.0
6	60	'2020/12/07'	110	136	374.0
7	450	'2020/12/08'	104	134	253.3
8	30	'2020/12/09'	109	133	195.1
9	60	'2020/12/10'	98	124	269.0

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

5	60	'2020/12/06'	102	127	300.0
6	60	'2020/12/07'	110	136	374.0
7	45	'2020/12/08'	104	134	253.3
8	30	'2020/12/09'	109	133	195.1
9	60	'2020/12/10'	98	124	269.0

Pandas - Fixing Wrong Data

- 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.

Pandas - Fixing Wrong Data

- **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
```

Pandas - Fixing Wrong Data

- **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.

Pandas - Fixing Wrong Data

- **Example**
  - Delete rows where "Duration" is higher than 120:

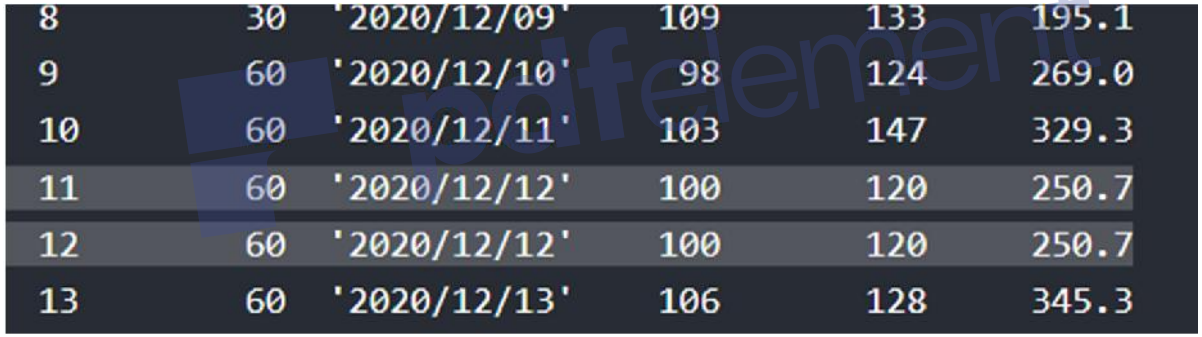
5	60	'2020/12/06'	102	127	300.0
6	60	'2020/12/07'	110	136	374.0
7	450	'2020/12/08'	104	134	253.3
8	30	'2020/12/09'	109	133	195.1
9	60	'2020/12/10'	98	124	269.0

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

5	60	'2020/12/06'	102	127	300.0
6	60	'2020/12/07'	110	136	374.0
8	30	'2020/12/09'	109	133	195.1
9	60	'2020/12/10'	98	124	269.0
10	60	'2020/12/11'	103	147	329.3
11	60	'2020/12/12'	100	120	250.7

Pandas - Removing Duplicates

- **Discovering Duplicates**
  - Duplicate rows are rows that have been registered more than one time.



8	30	'2020/12/09'	109	133	195.1
9	60	'2020/12/10'	98	124	269.0
10	60	'2020/12/11'	103	147	329.3
11	60	'2020/12/12'	100	120	250.7
12	60	'2020/12/12'	100	120	250.7
13	60	'2020/12/13'	106	128	345.3

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

Pandas - Removing Duplicates

- **Example**
  - Returns True for every row that is a duplica
    othwerwise False:


```
import pandas as ps

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

print(df.duplicated())
```

```
0      False
1      False
2      False
3      False
4      False
5      False
6      False
7      False
8      False
9      False
10     False
11     False
12      True
13     False
14     False
15     False
16     False
```

Pandas - Removing Duplicates

- To discover duplicates, we can use the duplicated() method.
- The duplicated() method returns a Boolean values for each row:

- **Example**
  - Returns True for every row that is a duplicate, othwerwise False:

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

Pandas - Removing Duplicates

- **Removing Duplicates**
  - To remove duplicates, use the drop_duplicates() method.
    - Remove all duplicates:


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