**Pandas**

**Q. 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.**

**Q. 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.**

**Q. 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.**

**Q. 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.**

**Q. 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.**

**Q. DataFrames**

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

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

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

**Q. 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)**

**Note: When using [], the result is a Pandas DataFrame.**

**Q. Load Files Into a DataFrame**

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

**Q. Truncation**

**-In Pandas, when you print a DataFrame directly (e.g., using print(df)), the output may be truncated if the DataFrame is too large. Truncation occurs when Pandas limits the number of rows and columns displayed to make the output more concise and readable. You might see something like this:**

**column1 column2 column3**

**0 value value value**

**1 value value value**

**... ... ... ...**

**99 value value value**

**-The ... indicates that some rows or columns are not shown. This happens because Pandas has default display settings to handle large datasets.**

**Why Use .to\_string()?**

**-The .to\_string() method is used to output the entire DataFrame without truncation. It overrides the default behavior and displays all rows and columns, regardless of their size.**

**Tip: use to\_string() to print the entire DataFrame.**

**Q. 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.**

**Q. Viewing the Data**

**head()**

**-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.**

**Note: if the number of rows is not specified, the head() method will return the top 5 rows.**

**tail()**

**-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.**

**info()**

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

**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.**

**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***

**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**

**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.**

**Note: Now, the dropna(inplace = True) will NOT return a new DataFrame, but it will remove all rows containing NULL values from the original DataFrame.**

**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**

**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**

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

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

**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.**

**1)** **Convert Into a Correct Format**

**Using - to\_datetime()**

**ex-**

**import pandas as pd  
  
df = pd.read\_csv('data.csv')  
  
df['Date'] = pd.to\_datetime(df['Date'])  
  
print(df.to\_string())**

**2)** **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**](https://www.w3schools.com/python/pandas/pandas_cleaning_wrong_format.asp)[**Next ❯**](https://www.w3schools.com/python/pandas/pandas_cleaning_duplicates.asp)

**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.**

**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:**

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

**df.loc[7, 'Duration'] = 45**

**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**

**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)**

**Pandas - Removing DuplicatesDiscovering Duplicates**

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

**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, otherwise False:**

**print(df.duplicated())**

**To remove duplicates, use the drop\_duplicates() method.**

**Example**

**Remove all duplicates:**

**df.drop\_duplicates(inplace = True)**

**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.**

**Pandas - Data Correlations**

**Finding Relationships**

**A great aspect of the Pandas module is the corr() method.**

**The corr() method calculates the relationship between each column in your data set.**

**The examples in this page uses a CSV file called: 'data.csv'.**

**Note: The corr() method ignores "not numeric" columns.**

**Result Explained**

**The Result of the corr() method is a table with a lot of numbers that represents how well the relationship is between two columns.**

**The number varies from -1 to 1.**

**1 means that there is a 1 to 1 relationship (a perfect correlation), and for this data set, each time a value went up in the first column, the other one went up as well.**

**0.9 is also a good relationship, and if you increase one value, the other will probably increase as well.**

**-0.9 would be just as good relationship as 0.9, but if you increase one value, the other will probably go down.**

**0.2 means NOT a good relationship, meaning that if one value goes up does not mean that the other will.**

**What is a good correlation?**

**It depends on the use, but I think it is safe to say you have to have at least 0.6 (or -0.6) to call it a good correlation.**

**Plotting**

**Pandas uses the plot() method to create diagrams.**

**We can use Pyplot, a submodule of the Matplotlib library to visualize the diagram on the screen.**

* **Pandas Summary:**

**Basic Pandas Skills**

1. **Data Structures**
   * **Understanding Pandas Series and DataFrame.**
   * **Creating and manipulating Series and DataFrames.**
2. **Reading and Writing Data**
   * **Reading data from various file formats: CSV, Excel, JSON, SQL, HTML, and more.**
   * **Saving DataFrames to different formats.**
3. **Basic Data Inspection**
   * **.head(), .tail(), .info(), .describe()**
   * **Checking data types: .dtypes, .astype()**
   * **Accessing dimensions: .shape**

**Data Cleaning**

1. **Handling Missing Data**
   * **Detecting missing values: .isnull(), .notnull()**
   * **Filling missing values: .fillna()**
   * **Dropping missing values: .dropna()**
2. **Renaming and Reordering**
   * **Renaming columns and indexes: .rename()**
   * **Changing column order.**
3. **Duplicates**
   * **Detecting and removing duplicates: .duplicated(), .drop\_duplicates()**
4. **Data Type Conversion**
   * **Converting data types using .astype()**
   * **Handling categorical data.**

**Data Manipulation**

1. **Indexing and Selecting Data**
   * **Using .loc[], .iloc[], and .at[].**
   * **Boolean indexing.**
   * **Filtering rows and columns.**
2. **Sorting**
   * **Sorting by index or values: .sort\_index(), .sort\_values()**
3. **Aggregation and Grouping**
   * **Grouping data: .groupby()**
   * **Aggregations like sum(), mean(), count(), etc.**
4. **Merging and Joining**
   * **Combining DataFrames: .merge(), .concat(), .join()**
   * **Aligning datasets.**

**Advanced Pandas Skills**

1. **Pivot Tables and Crosstabs**
   * **Creating pivot tables: .pivot\_table()**
   * **Summarizing data using .crosstab()**
2. **Time Series Analysis**
   * **Working with dates: .to\_datetime()**
   * **Resampling and time-based grouping.**
3. **Data Transformation**
   * **Applying functions to columns or rows: .apply(), .map(), .applymap()**
   * **Vectorized operations.**
4. **Handling Large Datasets**
   * **Chunking for large file processing: chunksize parameter in read\_csv.**
   * **Optimizing memory usage by changing data types.**
5. **MultiIndex**
   * **Working with hierarchical indexing for complex datasets.**
6. **Window Functions**
   * **Rolling and expanding calculations: .rolling(), .expanding()**

**Visualization with Pandas**

1. **Basic Plotting**
   * **Plotting using .plot()**
   * **Customizing plots: labels, titles, and legends.**
2. **Integration with Libraries**
   * **Using Matplotlib and Seaborn alongside Pandas.**