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.

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

```
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`.

Example

```
import pandas as pd

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

myvar = pd.DataFrame(mydataset)

print(myvar)
```

Checking Pandas Version

The version string is stored under `__version__` attribute.

Example

```
import pandas as pd

print(pd.__version__)
```

Pandas Series

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

Create a simple Pandas Series from a list:

```
import pandas as pd

a = [1, 7, 2]

myvar = pd.Series(a)

print(myvar)
```

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.

Example

Return the first value of the Series:

```
print(myvar[0])
```

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)
```

Try it Yourself »

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

Example

Return the value of "y":

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

Try it Yourself »

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)
```

Note: The keys of the dictionary become the labels.

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.

Example

Create a Series using only data from "day1" and "day2":

```
import pandas as pd

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

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

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)
```

Pandas DataFrames

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

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
```

Try it Yourself »

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
```

Note: This example returns a Pandas **Series**.

Example

Return row 0 and 1:

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

Result

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

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

Named Indexes

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

Example

Add a list of names to give each row a name:

```
import pandas as pd

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

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

print(df)
```

Result

```
      calories  duration
day1       420        50
day2       380        40
day3       390        45
```

Locate Named Indexes

Use the named index in the `loc` attribute to return the specified row(s).

Example

Return "day2":

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

Result

```
calories    380
duration     40
Name: day2, dtype: int64
```

Try it Yourself »

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)
```

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'.

Download data.csv. or Open data.csv

Example

Load the CSV into a DataFrame:

```
import pandas as pd

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

print(df.to_string())
```

Tip: use `to_string()` to print the entire DataFrame.

If you have a large DataFrame with many rows, Pandas will only return the first 5 rows, and the last 5 rows:

Example

Print the DataFrame without the `to_string()` method:

```
import pandas as pd

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

print(df)
```

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)
```

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.

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.

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())
```

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:

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)
```

Mode = the value that appears most frequently.

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.

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  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      20201226     100        120      250.0
  27         60  '2020/12/27'      92        118      241.0
  28         60  '2020/12/28'     103        132        NaN
  29         60  '2020/12/29'     100        132      280.0
  30         60  '2020/12/30'     102        129      380.3
  31         60  '2020/12/31'      92        115      243.0
```

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

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

Example

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  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           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
  28         60  '2020/12/28'    103        132        NaN
  29         60  '2020/12/29'    100        132      280.0
  30         60  '2020/12/30'    102        129      380.3
  31         60  '2020/12/31'     92        115      243.0
```

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.

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

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  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      20201226    100       120     250.0
  27      60  '2020/12/27'     92       118     241.0
  28      60  '2020/12/28'    103       132       NaN
  29      60  '2020/12/29'    100       132     280.0
  30          60  '2020/12/30'     102         129       380.3
  31          60  '2020/12/31'      92         115       243.0
```

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

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)
```

Discovering Duplicates

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

```
     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      20201226     100        120      250.0
  27         60  '2020/12/27'      92        118      241.0
  28         60  '2020/12/28'     103        132        NaN
  29         60  '2020/12/29'     100        132      280.0
  30         60  '2020/12/30'     102        129      380.3
  31         60  '2020/12/31'      92        115      243.0
```

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)
```

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.