Pandas

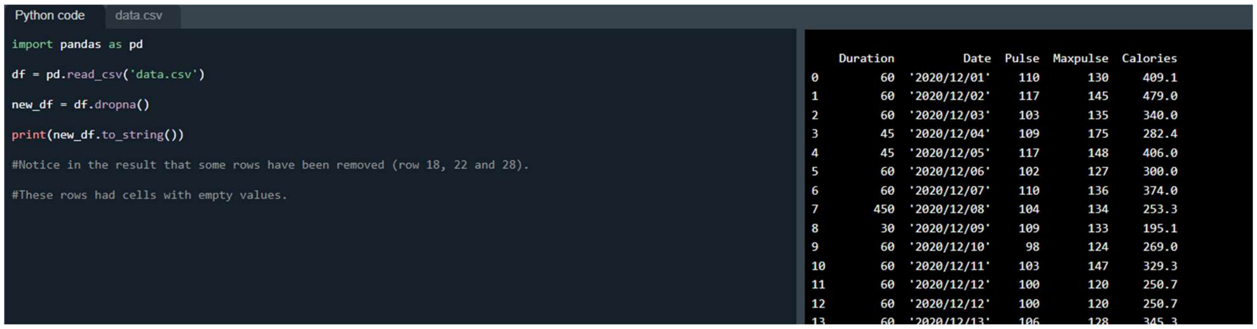
Data Cleaning

Data cleaning means fixing bad data in your data set.

Bad data could be:

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

- 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.



```python
import pandas as pd

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

new_df = df.dropna()

print(new_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.
```

```
    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
```

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:

```python
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.

Pandas

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

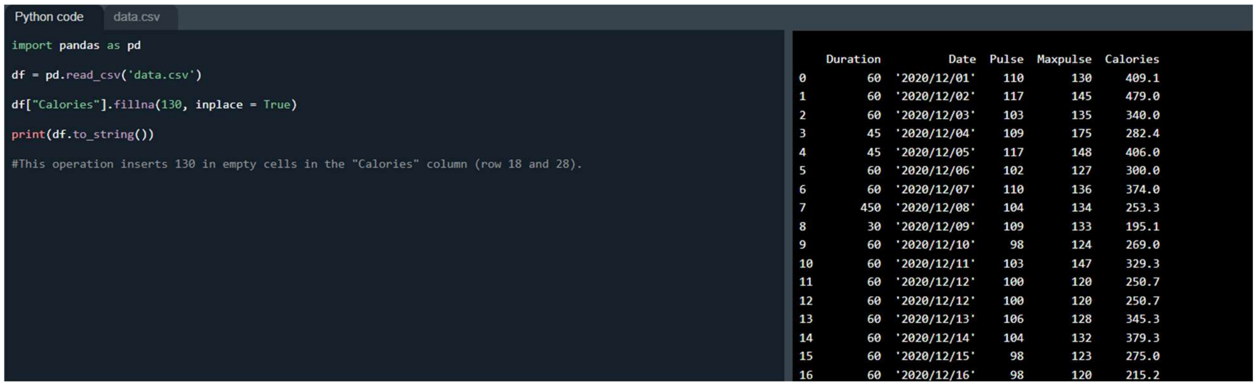
Replace NULL values with the number 130:

import pandas as pd

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

df.fillna(130, inplace = True)

Replace Only For Specified Columns



Python code data.csv

```
import pandas as pd

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

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

print(df.to_string())

#This operation inserts 130 in empty cells in the "Calories" column (row 18 and 28).
```

```
    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
```

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:

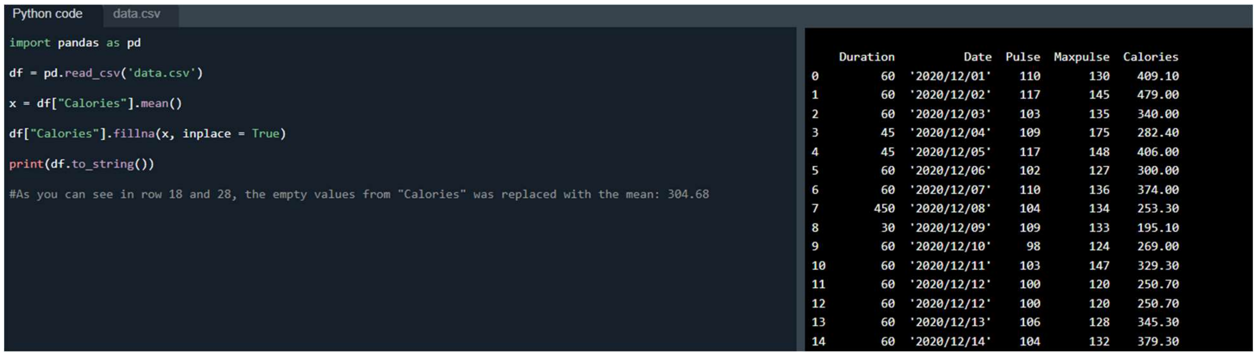
import pandas as pd

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

Pandas

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

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


Python code | data.csv

```python
import pandas as pd

df = pd.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
```

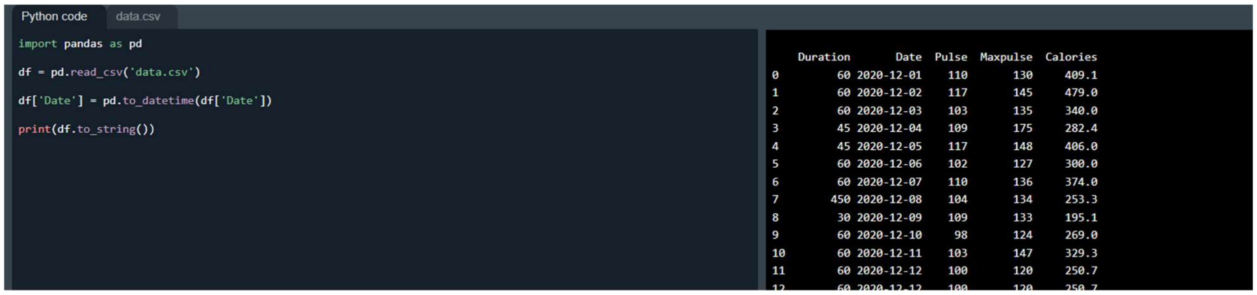
```
    Duration          Date  Pulse  Maxpulse  Calories
0         60  '2020/12/01'    110       130    409.10
1         60  '2020/12/02'    117       145    479.00
2         60  '2020/12/03'    103       135    340.00
3         45  '2020/12/04'    109       175    282.40
4         45  '2020/12/05'    117       148    406.00
5         60  '2020/12/06'    102       127    300.00
6         60  '2020/12/07'    110       136    374.00
7        450  '2020/12/08'    104       134    253.30
8         30  '2020/12/09'    109       133    195.10
9         60  '2020/12/10'     98       124    269.00
10        60  '2020/12/11'    103       147    329.30
11        60  '2020/12/12'    100       120    250.70
12        60  '2020/12/12'    100       120    250.70
13        60  '2020/12/13'    106       128    345.30
14        60  '2020/12/14'    104       132    379.30
```

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 has a to_datetime() method for this:


Python code | data.csv

```python
import pandas as pd

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

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

print(df.to_string())
```

```
    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
```

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)

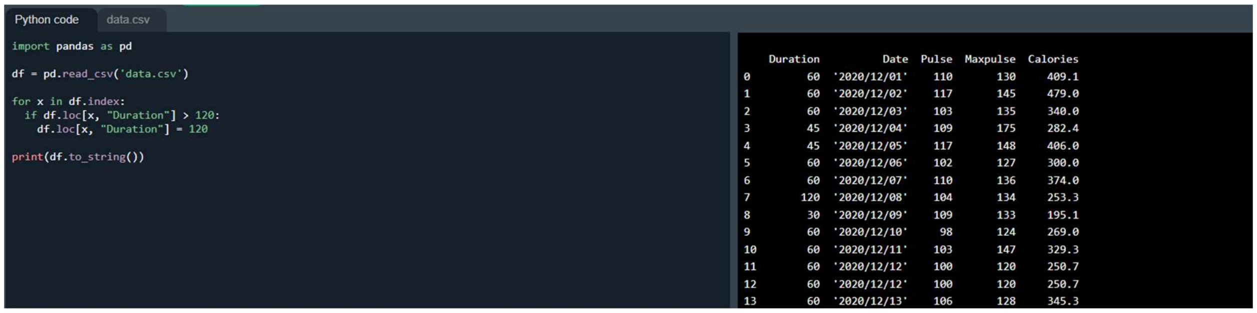
Wrong Data

"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".

Replace Values:

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

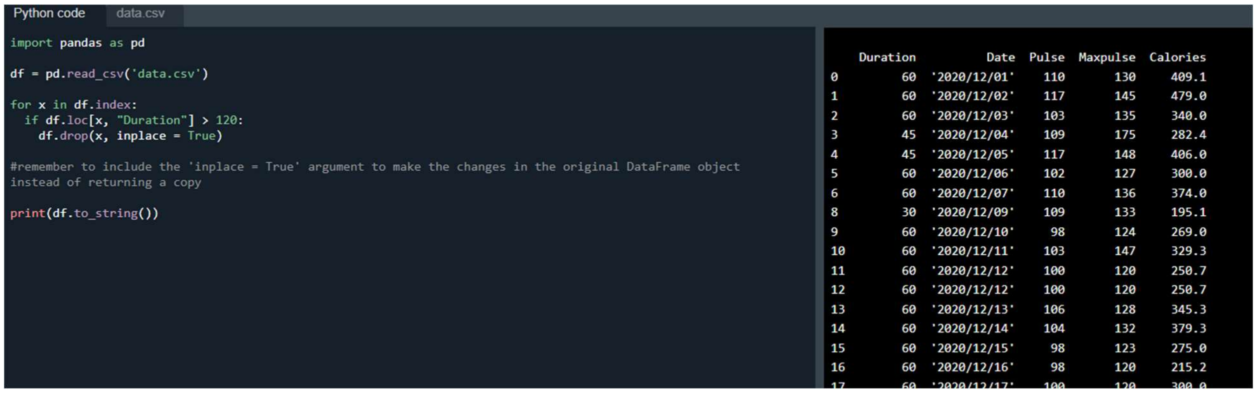
Pandas



```python
import pandas as pd

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

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

print(df.to_string())
```

```
    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        120  '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
```

Removing Rows



```python
import pandas as pd

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

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

#remember to include the 'inplace = True' argument to make the changes in the original DataFrame object
instead of returning a copy

print(df.to_string())
```

```
    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
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
```

Removing Duplicates:

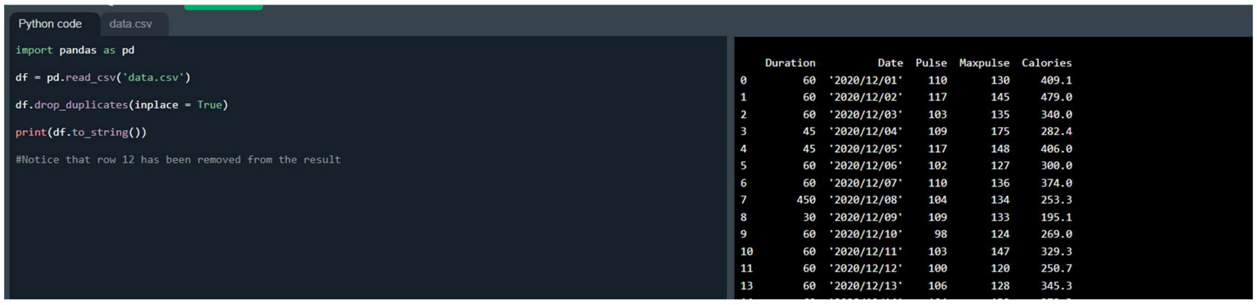
To discover duplicates, we can use the duplicated() method.

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())



```python
import pandas as pd

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

df.drop_duplicates(inplace = True)

print(df.to_string())

#Notice that row 12 has been removed from the 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
13        60  '2020/12/13'    106       128     345.3
```

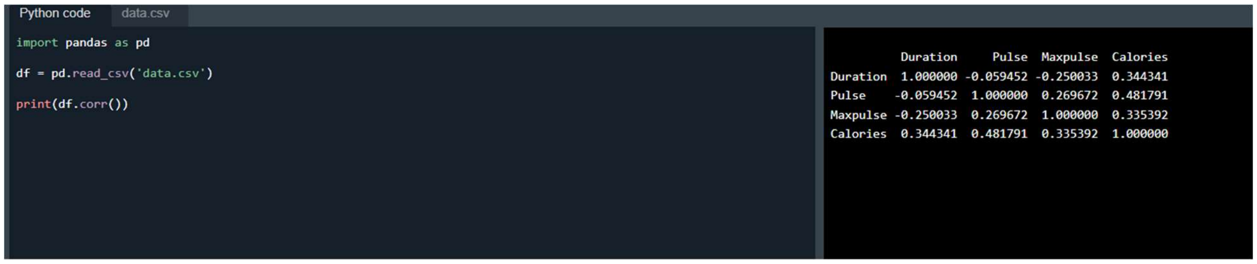
Data Correlations:

Finding Relationships

Pandas

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

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



```
import pandas as pd

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

print(df.corr())
```

```
          Duration     Pulse  Maxpulse  Calories
Duration  1.000000 -0.059452 -0.250033  0.344341
Pulse    -0.059452  1.000000  0.269672  0.481791
Maxpulse -0.250033  0.269672  1.000000  0.335392
Calories  0.344341  0.481791  0.335392  1.000000
```

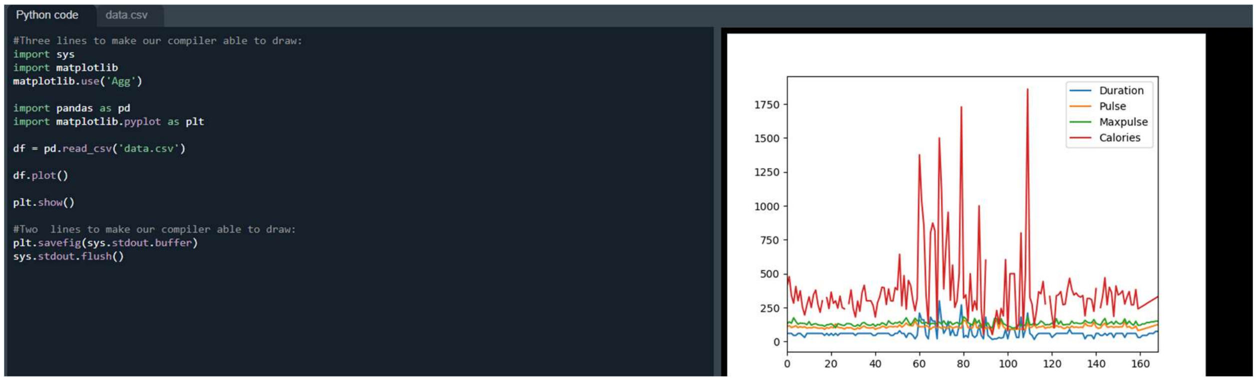
Plotting

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.

Import pyplot from Matplotlib and visualize our DataFrame:

```
import pandas as pd
import matplotlib.pyplot as plt

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

df.plot()

plt.show()
```



```
#Three lines to make our compiler able to draw:
import sys
import matplotlib
matplotlib.use('Agg')

import pandas as pd
import matplotlib.pyplot as plt

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

df.plot()

plt.show()

#Two  lines to make our compiler able to draw:
plt.savefig(sys.stdout.buffer)
sys.stdout.flush()
```

Scatter Plot

Specify that you want a scatter plot with the kind argument:

kind = 'scatter'

A scatter plot needs an x- and a y-axis.

Pandas

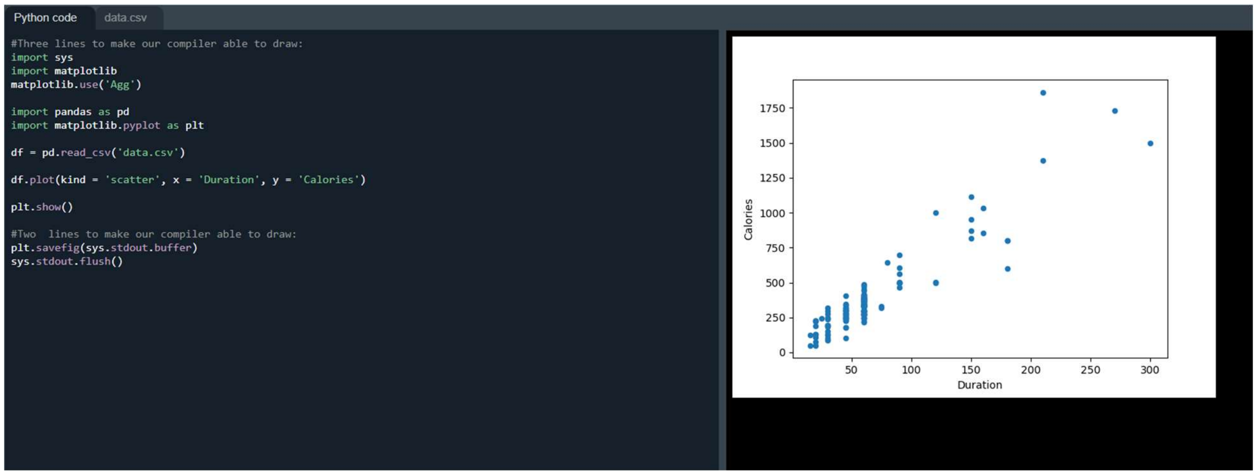
In the example below we will use "Duration" for the x-axis and "Calories" for the y-axis.

Include the x and y arguments like this:

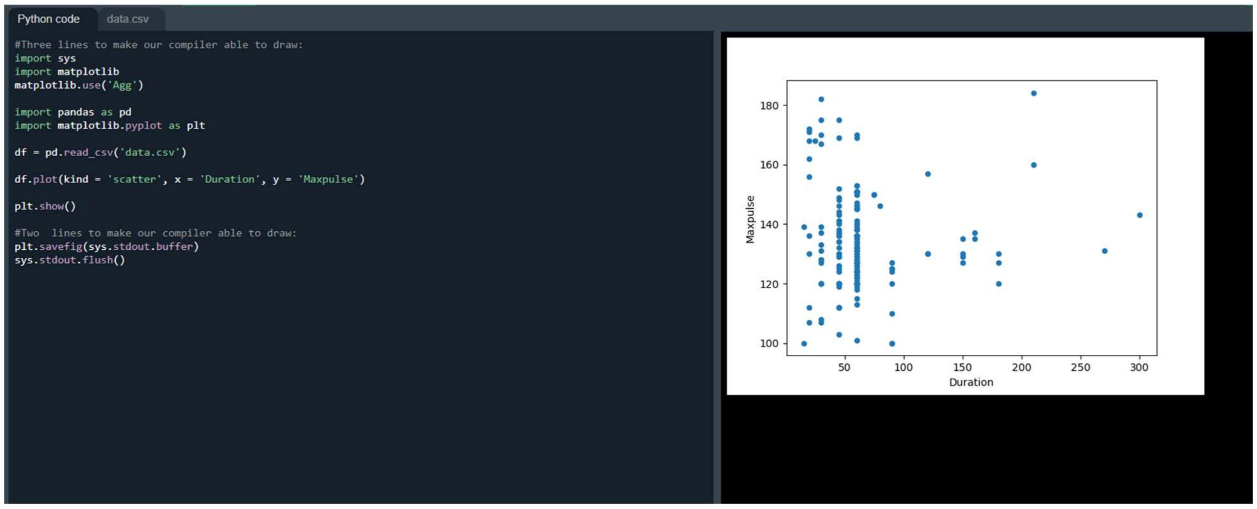
x = 'Duration', y = 'Calories'

Example

```
import pandas as pd
import matplotlib.pyplot as plt

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

df.plot(kind = 'scatter', x = 'Duration', y = 'Calories')

plt.show()
```



```
#Three lines to make our compiler able to draw:
import sys
import matplotlib
matplotlib.use('Agg')

import pandas as pd
import matplotlib.pyplot as plt

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

df.plot(kind = 'scatter', x = 'Duration', y = 'Calories')

plt.show()

#Two  lines to make our compiler able to draw:
plt.savefig(sys.stdout.buffer)
sys.stdout.flush()
```



```
#Three lines to make our compiler able to draw:
import sys
import matplotlib
matplotlib.use('Agg')

import pandas as pd
import matplotlib.pyplot as plt

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

df.plot(kind = 'scatter', x = 'Duration', y = 'Maxpulse')

plt.show()

#Two  lines to make our compiler able to draw:
plt.savefig(sys.stdout.buffer)
sys.stdout.flush()
```

Histogram

Pandas

Use the kind argument to specify that you want a histogram:

kind = 'hist'

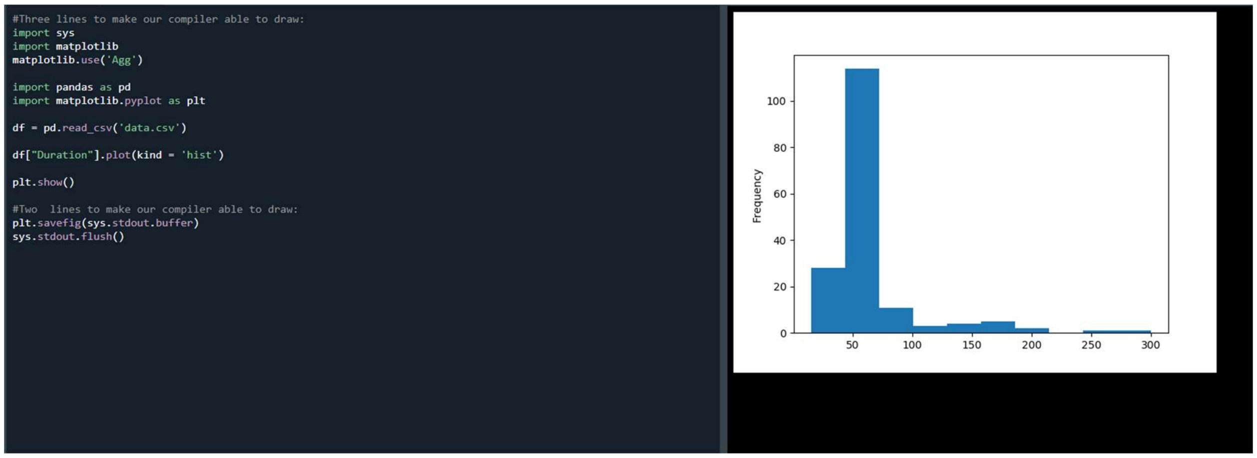
A histogram needs only one column.

A histogram shows us the frequency of each interval, e.g. how many workouts lasted between 50 and 60 minutes?

In the example below we will use the "Duration" column to create the histogram:

Example

df["Duration"].plot(kind = 'hist')



```
#Three lines to make our compiler able to draw:
import sys
import matplotlib
matplotlib.use('Agg')

import pandas as pd
import matplotlib.pyplot as plt

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

df["Duration"].plot(kind = 'hist')

plt.show()

#Two  lines to make our compiler able to draw:
plt.savefig(sys.stdout.buffer)
sys.stdout.flush()
```

Thank you