Business Intelligence-relaterade programspråk

Pandas

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.
- 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.

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.

Installation of Pandas

```
pip install pandas
```

- Or, if you’re running Windows and having problems with pip:

```
py -m pip install pandas
```

Pandas 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 pd

a = [1, 7, 2]

myvar = pd.Series(a)

print(myvar)
```

Pandas Series

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 pd

a = [1, 7, 2]

myvar = pd.Series(a)

# Return the first value of the Series
print(myvar[0])
```

Pandas Series

Create Labels

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

```
import pandas as pd

a = [1, 7, 2]

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

print(myvar)

# Print the value of "y":
print(myvar["y"])
```

Pandas Series

Key/Value Objects as Series

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

```
import pandas as pd

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

myvar = pd.Series(calories)

print(myvar)
```

Pandas Series

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.

```
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.

```
import pandas as pd

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

myvar = pd.DataFrame(data)

print(myvar)
```

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)

```
import pandas as pd

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

myvar = pd.DataFrame(data)

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

# Return row 0 and 1:
print(df.loc[[0, 1]])
```

DataFrames

Named Indexes

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

```
import pandas as pd

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

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

print(df)

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

DataFrames

Load Files Into a DataFrame

- If your data sets are stored in a file, Pandas can load them into a DataFrame.
- Test set: https://github.com/emmio-micke/bi20python/tree/main/code/15%20-%20pandas

```
import pandas as pd

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

print(df)
```

DataFrames

Load Files Into a DataFrame

- By default, when you print a DataFrame, you will only get the first 5 rows, and the last 5 rows.
- Use `to_string()` to print the entire DataFrame.

```
import pandas as pd

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

print(df.to_string())
```

DataFrames

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.
- https://github.com/emmio-micke/bi20python/tree/main/code/15%20-%20pandas

```
import pandas as pd

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

print(df.to_string())
```

DataFrames

Dictionary

- If your data is in a Python Dictionary, you can load it into a DataFrame directly.

```
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)
```

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.

```
import pandas as pd

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

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

Analyzing DataFrames

Viewing the Data

- 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.

```
import pandas as pd

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

print(df.tail(10))

# No argument = 5 rows
print(df.tail())
```

Analyzing DataFrames

Result Explained

- The result tells us there are 169 rows and 4 columns with the name of each column, with the data type.

```
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
```

Analyzing DataFrames

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

Data Cleaning

- We will use a data set for cleaning, called `clean.csv` in the repo.
- https://github.com/emmio-micke/bi20python/tree/main/code/15%20-%20pandas

```
import pandas as pd

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

print(df.tail(10))

# No argument = 5 rows
print(df.tail())
```

Data Cleaning

- We will use a data set for cleaning, called `clean.csv` in the repo.
- https://github.com/emmio-micke/bi20python/tree/main/code/15%20-%20pandas
  - 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).

Data Cleaning

Cleaning Empty Cells

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

```
import pandas as pd

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

new_df = df.dropna()

print(new_df.to_string())
```

Data Cleaning

Cleaning Empty Cells

- Empty cells can potentially give you a wrong result when you analyze data.
- 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.
- If you want to change the original DataFrame, use the `inplace = True` argument:

```
import pandas as pd

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

df.dropna(inplace = True)

print(df.to_string())
```

Data Cleaning

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 pd

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

df.fillna(130, inplace=True)

print(df.to_string())
```

Data Cleaning

Replace Only For a 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.

```
import pandas as pd

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

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

print(df.to_string())
```

Data Cleaning

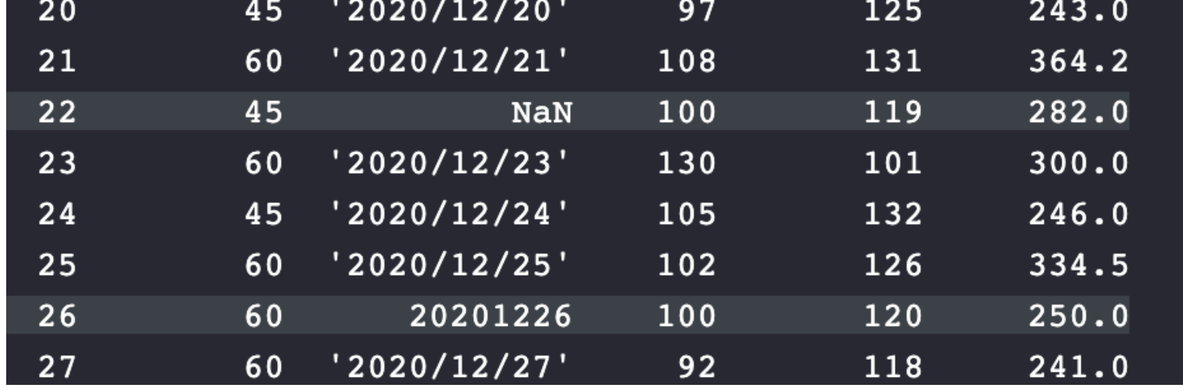
Replace Using Mean, Median, or Mode

- 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.
- **Median:** the value in the middle, after you have sorted all values ascending.
- **Mode:** the value that appears most frequently.

```
import pandas as pd

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

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

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

print(df.to_string())
```

Data Cleaning

Cleaning Data of Wrong Format



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

- 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.
- 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.
- Let's try to convert all cells in the 'Date' column into dates with Pandas `to_datetime()` method.

```
import pandas as pd

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

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

print(df.to_string())
```

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

Data Cleaning

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.

Data Cleaning

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.

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

Data Cleaning

Wrong Data - Replacing Values

- 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.

```
import pandas as pd

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

# 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

print(df.to_string())
```

Data Cleaning

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.

```
import pandas as pd

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

# Delete rows where "Duration" is higher than 120:
for x in df.index:
  if df.loc[x, "Duration"] > 120:
    df.drop(x, inplace = True)

print(df.to_string())
```

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

Data Cleaning

Duplicates

- 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.

```
import pandas as pd

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

print(df.duplicated())
```

Data Cleaning

Removing Duplicates

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

```
import pandas as pd

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

df.drop_duplicates(inplace = True)

print(df.to_string())
```

Data Correlations

Finding Relationships

- The `corr()` method calculates the relationship between each column in your data set.
- These examples uses the file `data.csv`.
- *Note:* The `corr()` method ignores "not numeric" columns.

```
import pandas as pd

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

print(df.corr())
```

Data Correlations

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.

Data Correlations

Result Explained

- **Perfect Correlation**: We can see that ”Duration” and ”Duration” got the number 1.000000, which makes sense, each column always has a perfect relationship with itself.

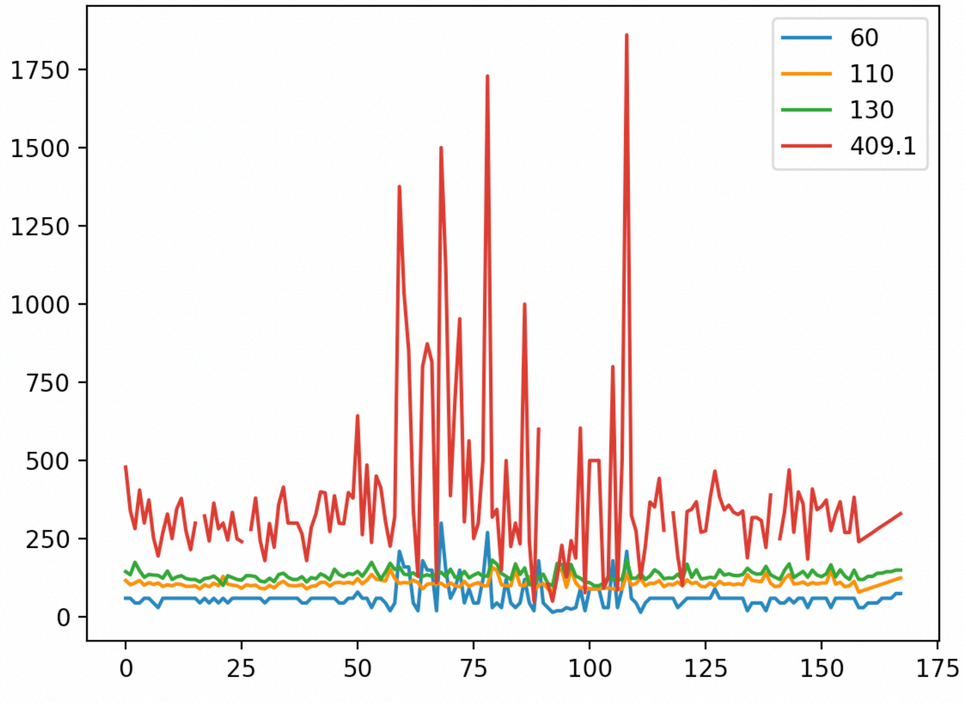
- **Good Correlation**: ”Duration” and ”Calories” got a 0.922721 correlation, which is a very good correlation, and we can predict that the longer you work out, the more calories you burn, and the other way around: if you burned a lot of calories, you probably had a long work out.

- **Bad Correlation**: ”Duration” and ”Maxpulse” got a 0.009403 correlation, which is a very bad correlation, meaning that we can not predict the max pulse by just looking at the duration of the work out, and vice versa.

Plotting

- Pandas uses the `plot()` method to create diagrams.
- Pythons uses Pyplot, that we looked at earlier.

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

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

df.plot()

plt.show()
```



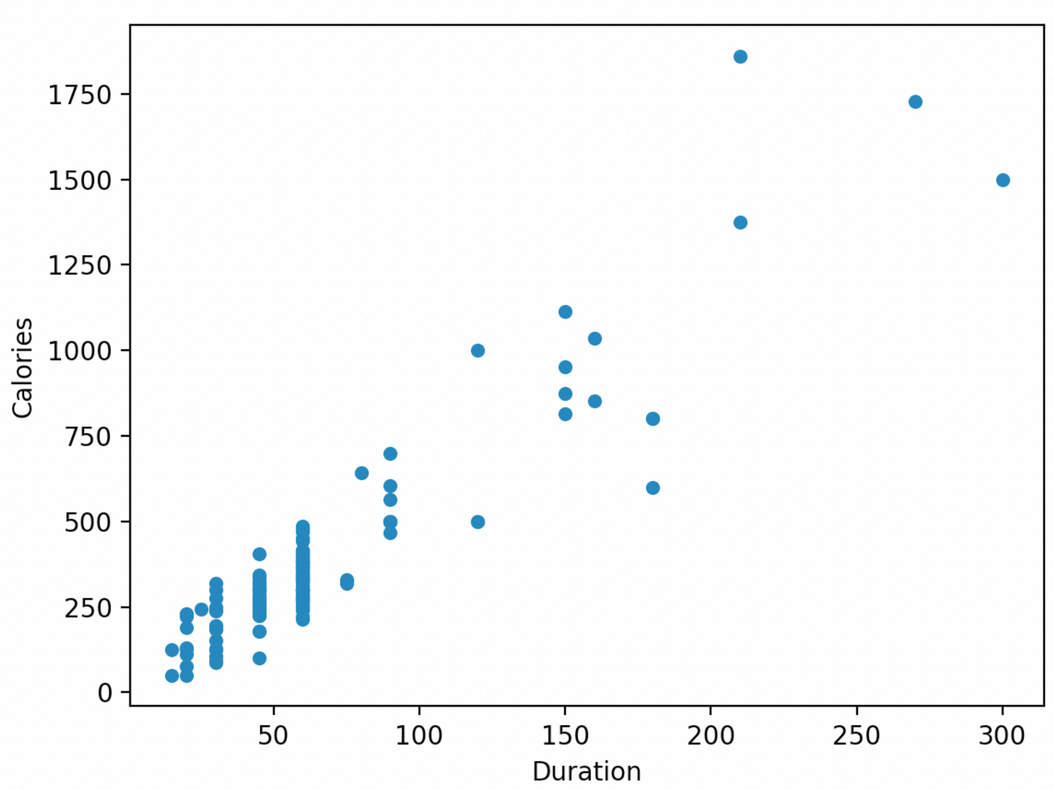
Plotting
Scatter Plot

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

```
kind = 'scatter'
```

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

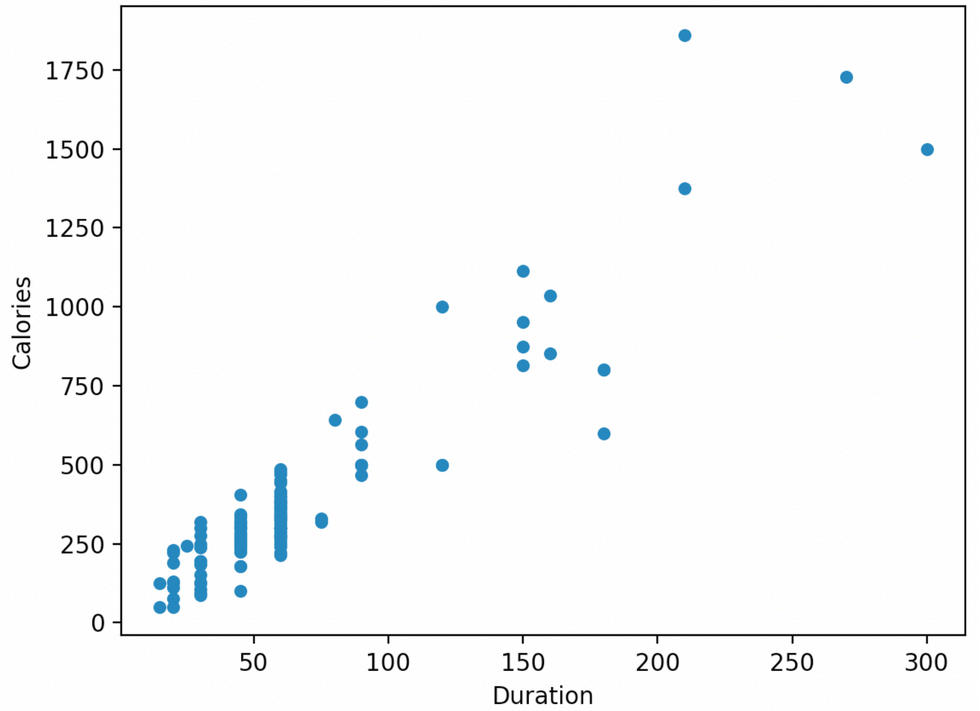
```
import pandas as pd
import matplotlib.pyplot as plt

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

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

plt.show()
```



Plotting

Scatter Plot

- In the previous example, we learned that the correlation between ”Duration” and ”Calories” was 0.922721, and we conluded with the fact that higher duration means more calories burned.
- By looking at the scatterplot, would you agree?

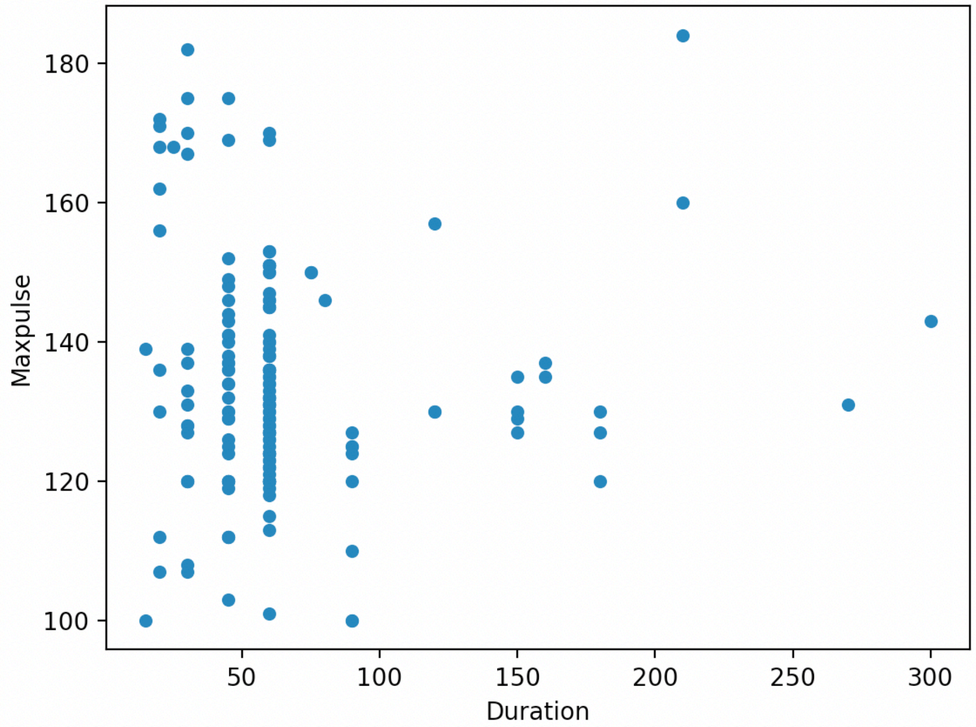


Plotting

Scatter Plot

- Let's create another scatterplot, where there is a low correlation between the columns, like ”Duration” and ”Maxpulse", with the correlation 0.009403.

```
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()
```



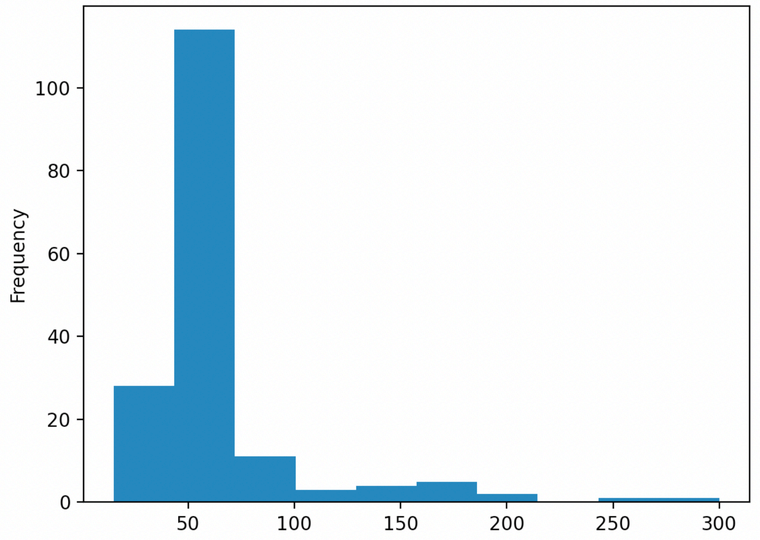
Plotting

Histogram

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

- The histogram tells us that there were over 100 workouts that lasted between 50 and 60 minutes.

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

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

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

plt.show()
```



Exercises

- https://www.w3resource.com/python-exercises/pandas/index.php