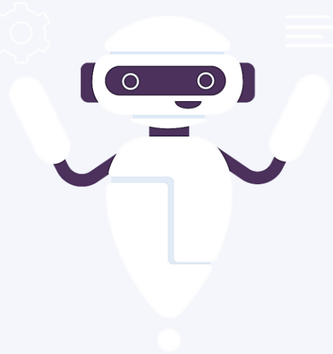
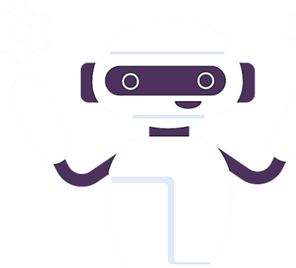
# PANDAS

# What is Pandas?



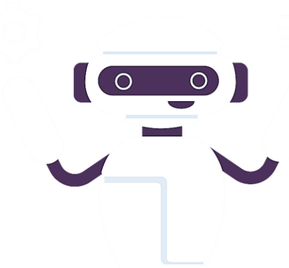
- Pandas is a Python library used for working with **data sets**.
- It has functions for analyzing, cleaning, exploring, and manipulating data.
- The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis" and was created by Wes McKinney in 2008.

# 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?

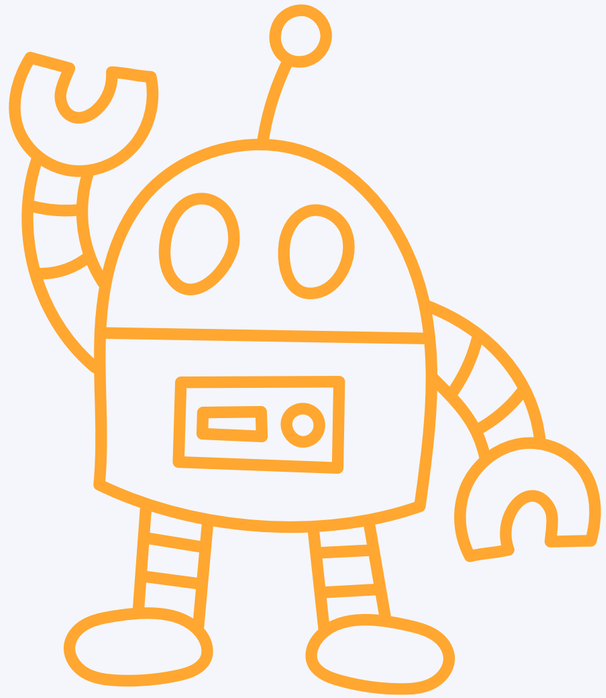
- Pandas are also able to delete rows that are not relevant, or contains wrong values, like empty or NULL values. This is called cleaning the data.

## INSTALLATION OF PANDAS

- pip install pandas

## IMPORT PANDAS

- import pandas as pd



# Pandas Series

# What is a Series?

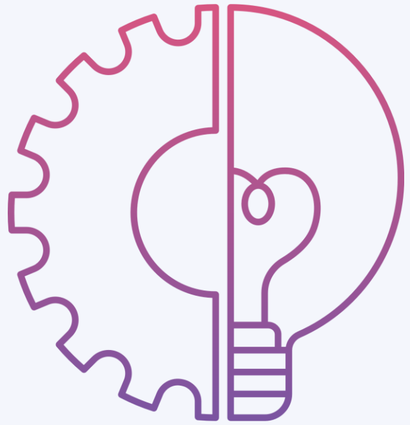
- A Pandas Series is like a column in a table.
- It is a one-dimensional array holding data of any type.

**Example:**

```
import pandas as pd

a = [1, 7, 2]

myvar = pd.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.

# 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)
```

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

# 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:**

```
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)
```

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

# Creating DataFrames from scratch

Creating DataFrames right in Python is good to know and quite useful when testing new methods and functions you find in the pandas docs.
There are many ways to create a DataFrame from scratch, but a great option is to just use a simple dict.
Let's say we have a fruit stand that sells apples and oranges. We want to have a column for each fruit and a row for each customer purchase. To organize this as a dictionary for pandas we could do something like:

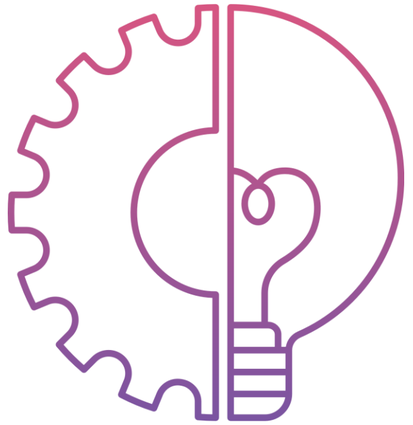
```
data = {
    'apples': [3, 2, 0, 1],
    'oranges': [0, 3, 7, 2]
}
```

```
purchases = pd.DataFrame(data)

purchases
```

**OUT:**

| | apples | oranges |
|---|---|---|
| **0** | 3 | 0 |
| **1** | 2 | 3 |
| **2** | 0 | 7 |
| **3** | 1 | 2 |

# 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)

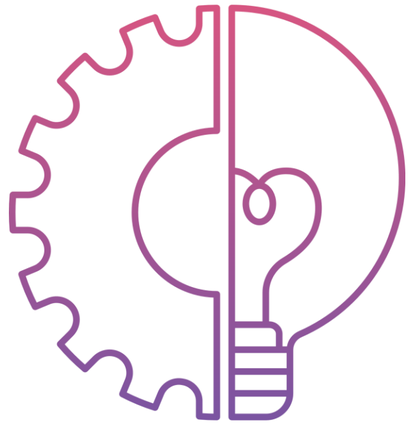
## Example:

```
import pandas as pd

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

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

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

# How to read in data

It’s quite simple to load data from various file formats into a DataFrame. In the following examples we'll keep using our apples and oranges data, but this time it's coming from various files.



# Reading data from CSVs

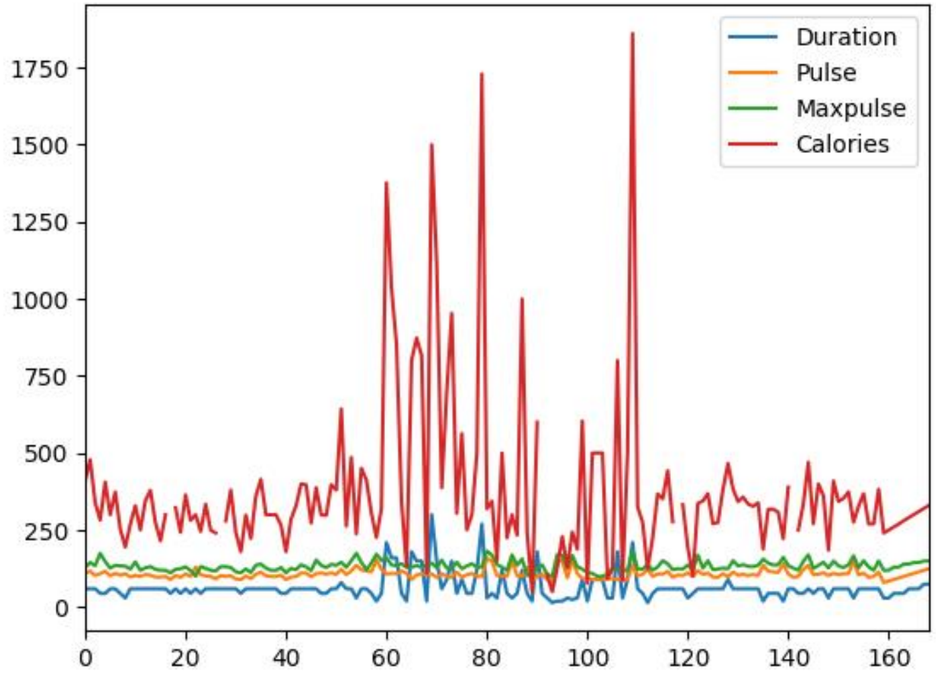
**Example:**

```
df = pd.read_csv('purchases.csv')

df
```

**OUT:**

| | Unnamed: 0 | apples | oranges |
|---|---|---|---|
| 0 | June | 3 | 0 |
| 1 | Robert | 2 | 3 |
| 2 | Lily | 0 | 7 |
| 3 | David | 1 | 2 |

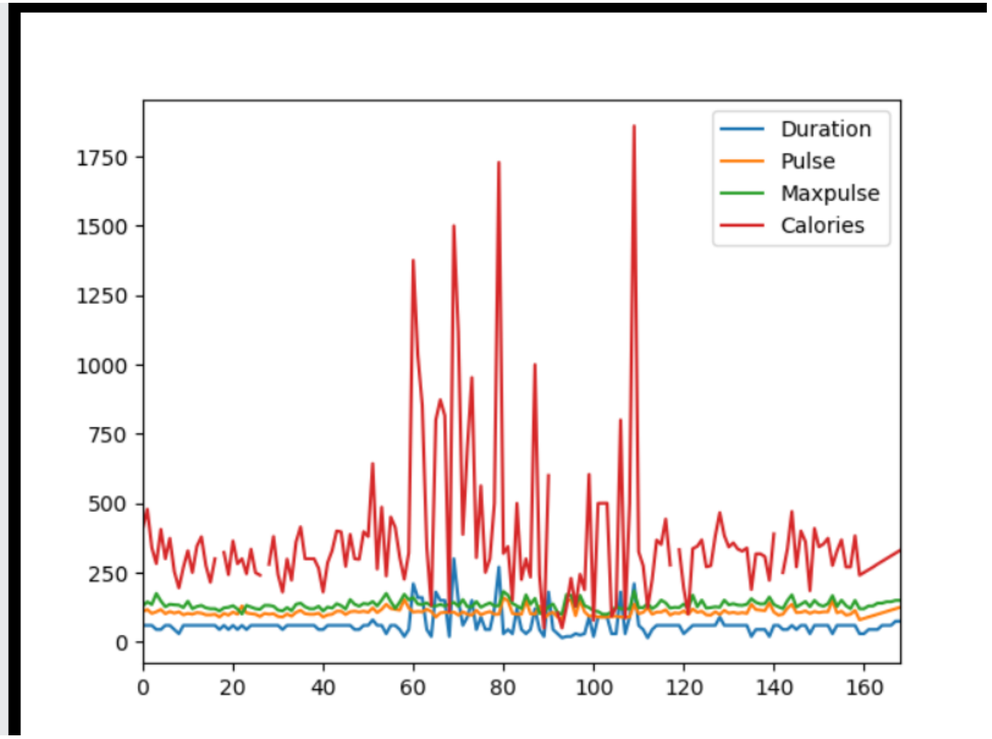
# Pandas - Plotting



# 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.
- Read more about Matplotlib in our Matplotlib Tutorial.

```
#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()
```

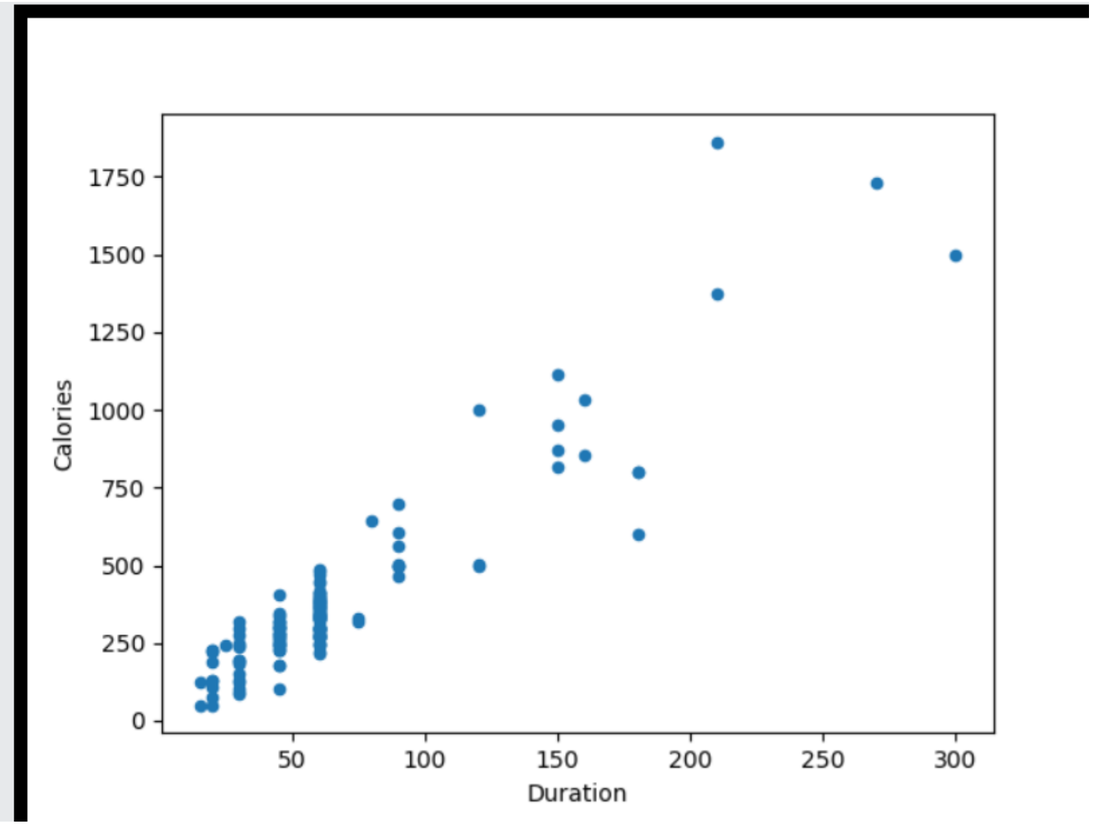


# Scatter Plot

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

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

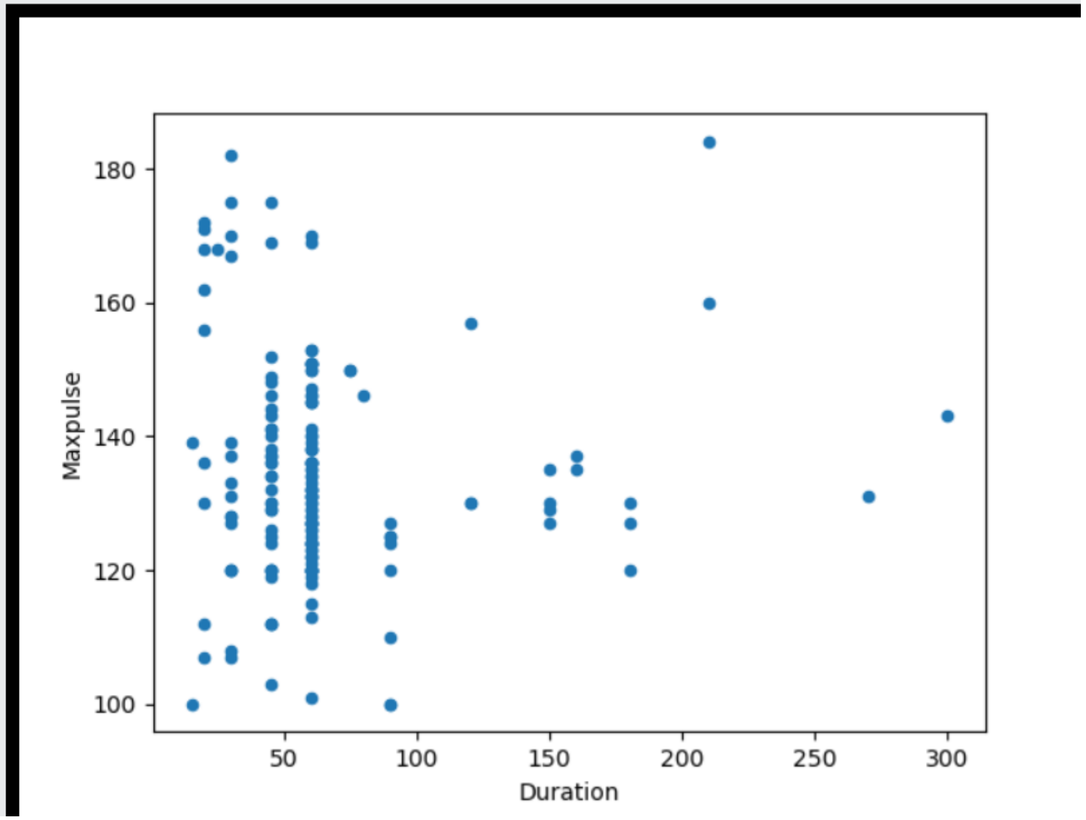
```
#Three lines to make our compiler able to draw:
import sys
import matplotlib
matplotlib.use('Agg')

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()
```



# Scatter Plot

A scatterplot where there are no relationship between the columns:

```
#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()
```



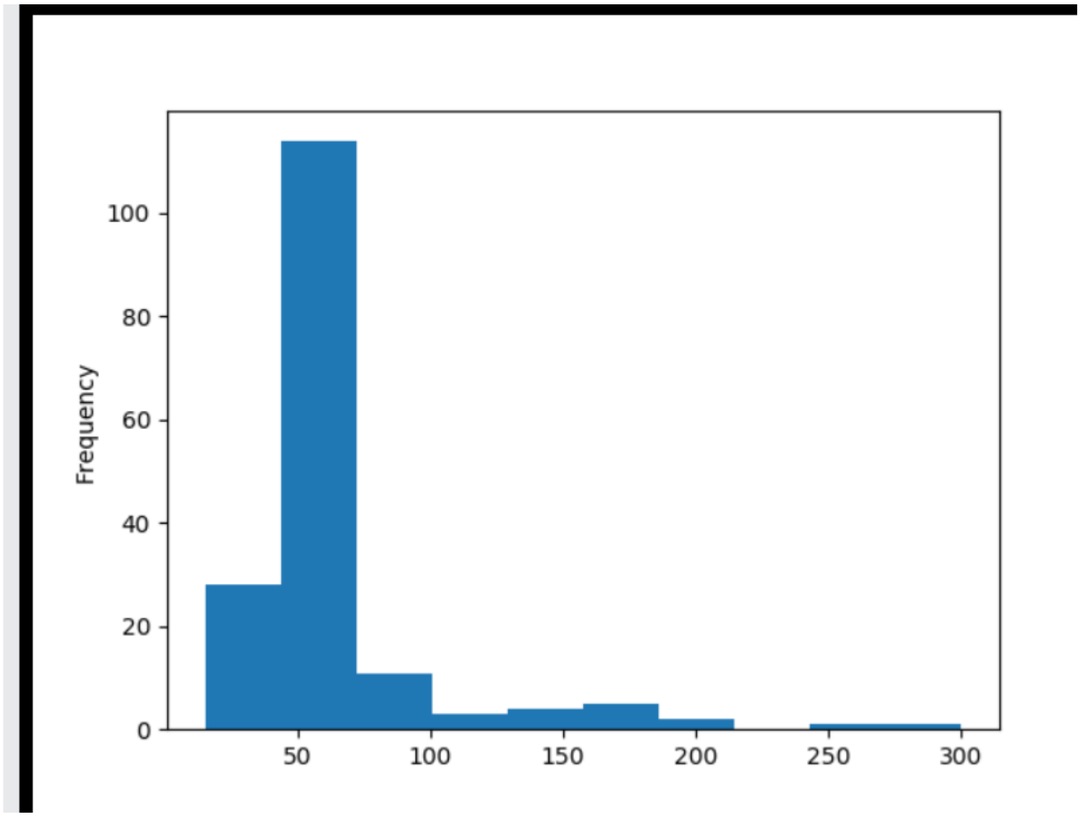
# Histogram

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:

```
#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()
```

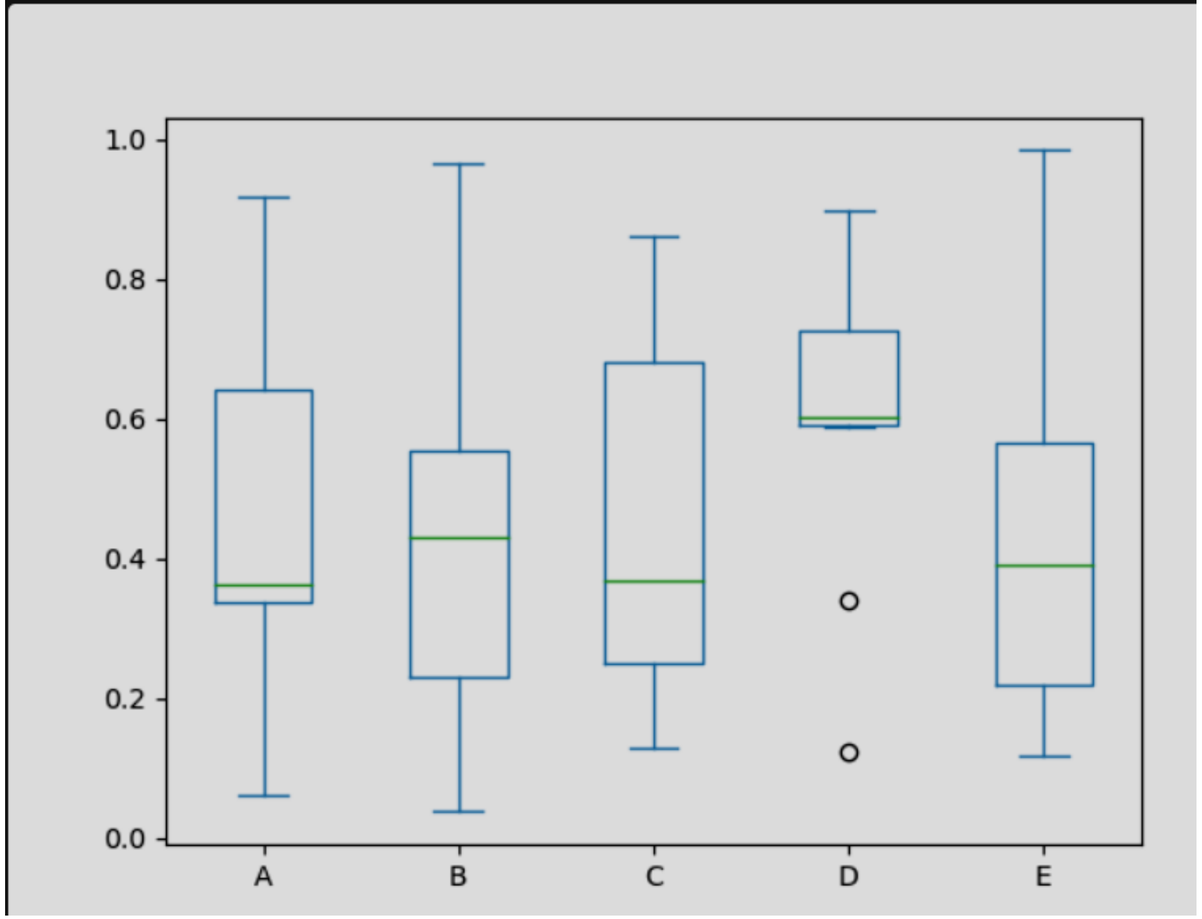


# Box plot

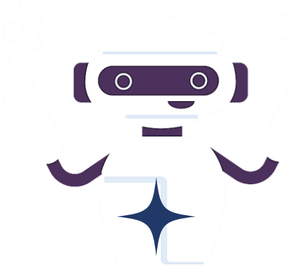
Boxplot can be drawn calling Series.plot.box() and DataFrame.plot.box(), or DataFrame.boxplot() to visualize the distribution of values within each column.

For instance, here is a boxplot representing five trials of 10 observations of a uniform random variable on [0,1).

```
df = pd.DataFrame(np.random.rand(10, 5), columns=["A", "B", "C", "D", "E"])

df.plot.box();
```



# COUNTS

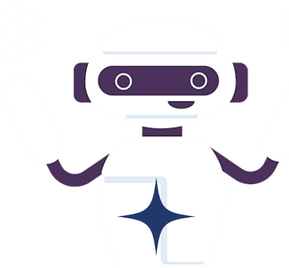


count: number of entries

**Example:**

```
In [2]: iris['sepal_length'].count() # Applied to Series
Out[2]: 150

In [3]: iris['sepal_width'].count() # Applied to Series
Out[3]: 150

In [4]: iris[['petal_length', 'petal_width']].count() # Applied
   ...: to DataFrame
Out[4]:
petal_length    150
petal_width     150
dtype: int64

In [5]: type(iris[['petal_length', 'petal_width']].count()) #
   ...: returns Series
Out[5]: pandas.core.series.Series
```

# MEAN



mean: average of entries

**Example:**

```
In [6]: iris['sepal_length'].mean() # Applied to Series
Out[6]: 5.843333333333335

In [7]: iris.mean() # Applied to entire DataFrame
Out[7]:
sepal_length    5.843333
sepal_width     3.057333
petal_length    3.758000
petal_width     1.199333
dtype: float64
```

# STANDARD DEVIATION

std: standard deviation

**Example:**

```
In [8]: iris.std()
Out[8]:
sepal_length    0.828066
sepal_width     0.435866
petal_length    1.765298
petal_width     0.762238
dtype: float64
```

# RANGES

**Example:**

```
In [15]: iris.min()
Out[15]:
sepal_length        4.3
sepal_width           2
petal_length          1
petal_width         0.1
species          setosa
dtype: object

In [16]: iris.max()
Out[16]:
sepal_length          7.9
sepal_width           4.4
petal_length          6.9
petal_width           2.5
species          virginica
dtype: object
```

# Thank You