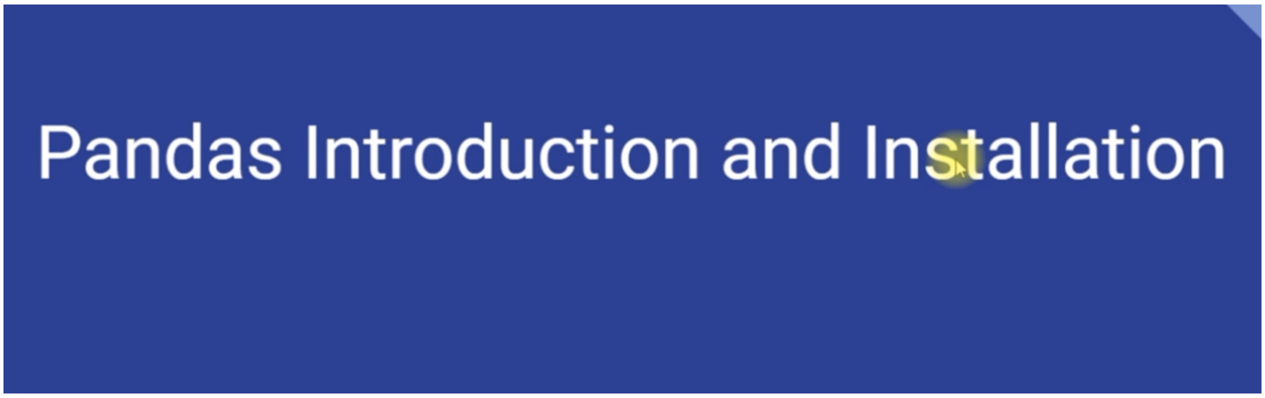
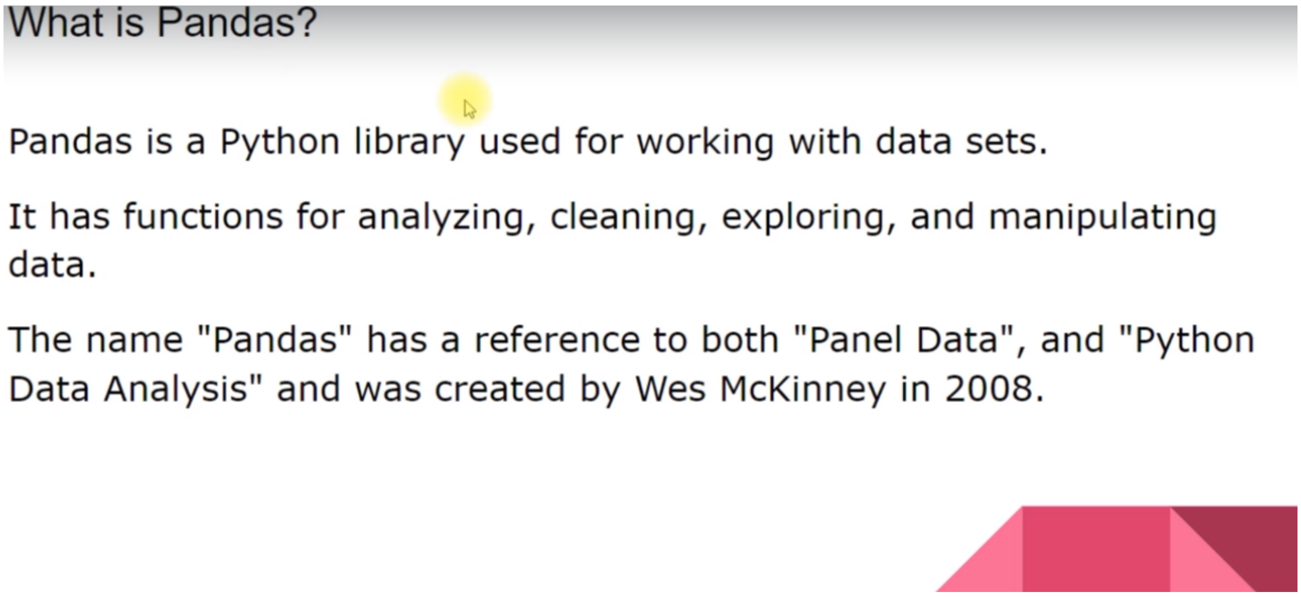


Pandas Introduction and Installation

Imagine you have a magical box called "Pandas." This box helps you organize and work with lots of information, like a super smart and friendly friend.

Pandas is like having a special toolkit for your computer that makes it easy to play with numbers and data. It's kind of like when you have a big collection of toys, and you want to arrange them neatly or see which ones you have the most of.

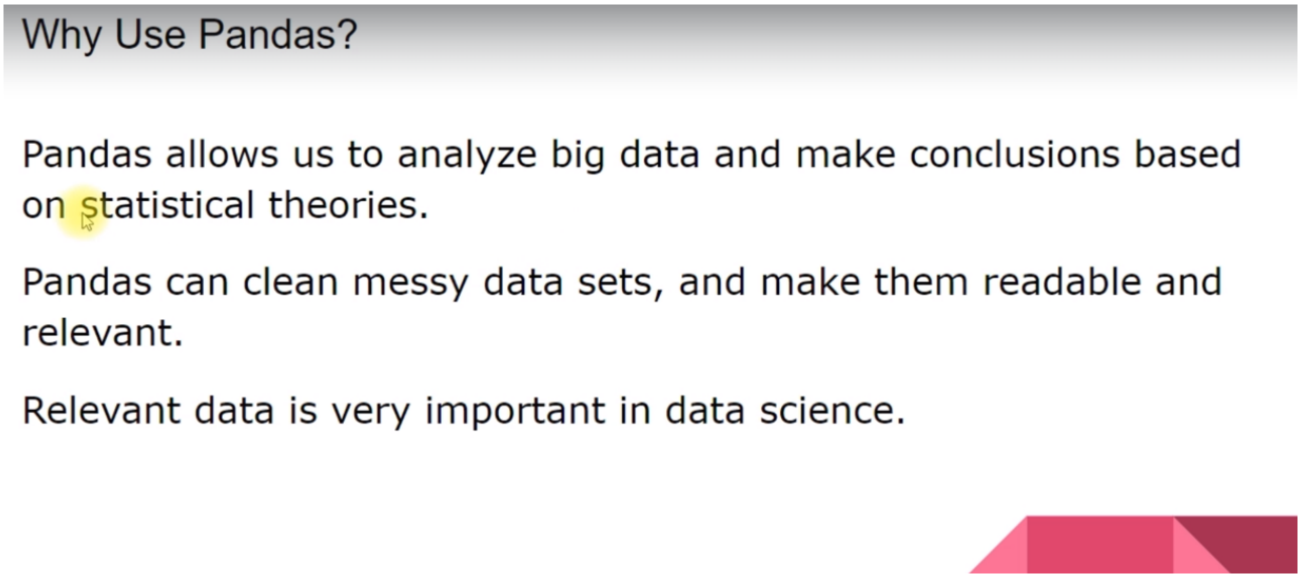


What is Pandas?

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:

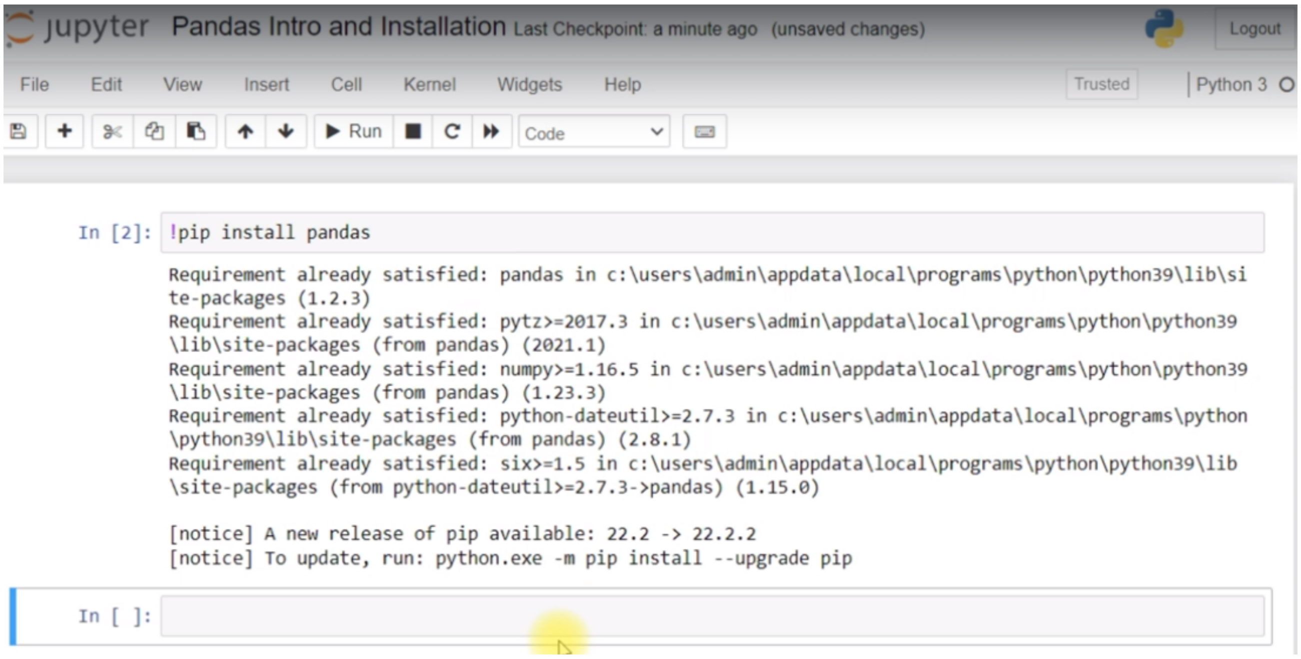
- Is there a correlation between two or more columns?
- What is average value?
- Max value?
- Min value?

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

Use the following command to install pandas on your system

pip install pandas


jupyter Pandas Intro and Installation Last Checkpoint: a minute ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

```
In [2]: !pip install pandas

Requirement already satisfied: pandas in c:\users\admin\appdata\local\programs\python\python39\lib\si
te-packages (1.2.3)
Requirement already satisfied: pytz>=2017.3 in c:\users\admin\appdata\local\programs\python\python39
\lib\site-packages (from pandas) (2021.1)
Requirement already satisfied: numpy>=1.16.5 in c:\users\admin\appdata\local\programs\python\python39
\lib\site-packages (from pandas) (1.23.3)
Requirement already satisfied: python-dateutil>=2.7.3 in c:\users\admin\appdata\local\programs\python
\python39\lib\site-packages (from pandas) (2.8.1)
Requirement already satisfied: six>=1.5 in c:\users\admin\appdata\local\programs\python\python39\lib
\site-packages (from python-dateutil>=2.7.3->pandas) (1.15.0)

[notice] A new release of pip available: 22.2 -> 22.2.2
[notice] To update, run: python.exe -m pip install --upgrade pip

In [ ]:
```

Pandas, a popular Python library for data manipulation and analysis, offers several benefits:

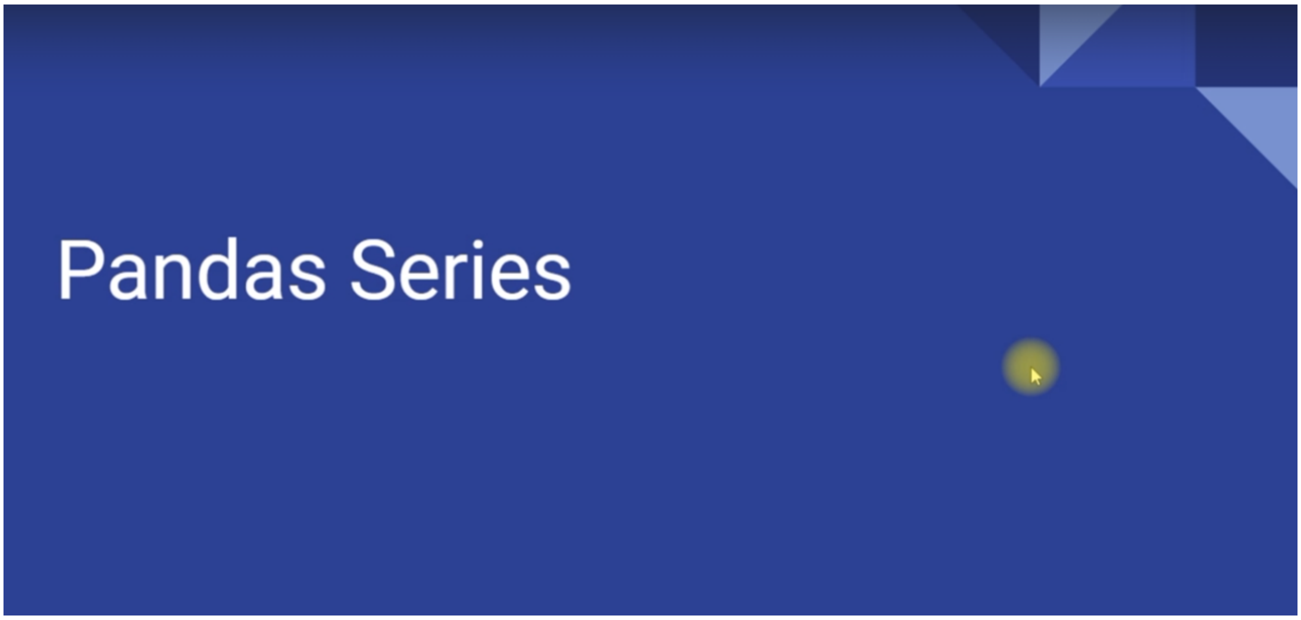
Easy Data Handling: Pandas makes it simple to load, manipulate, and analyze data from various sources like spreadsheets, databases, and CSV files.

Data Cleaning: It helps clean messy data by handling missing values, duplicate records, and other data issues, making it easier to work with.

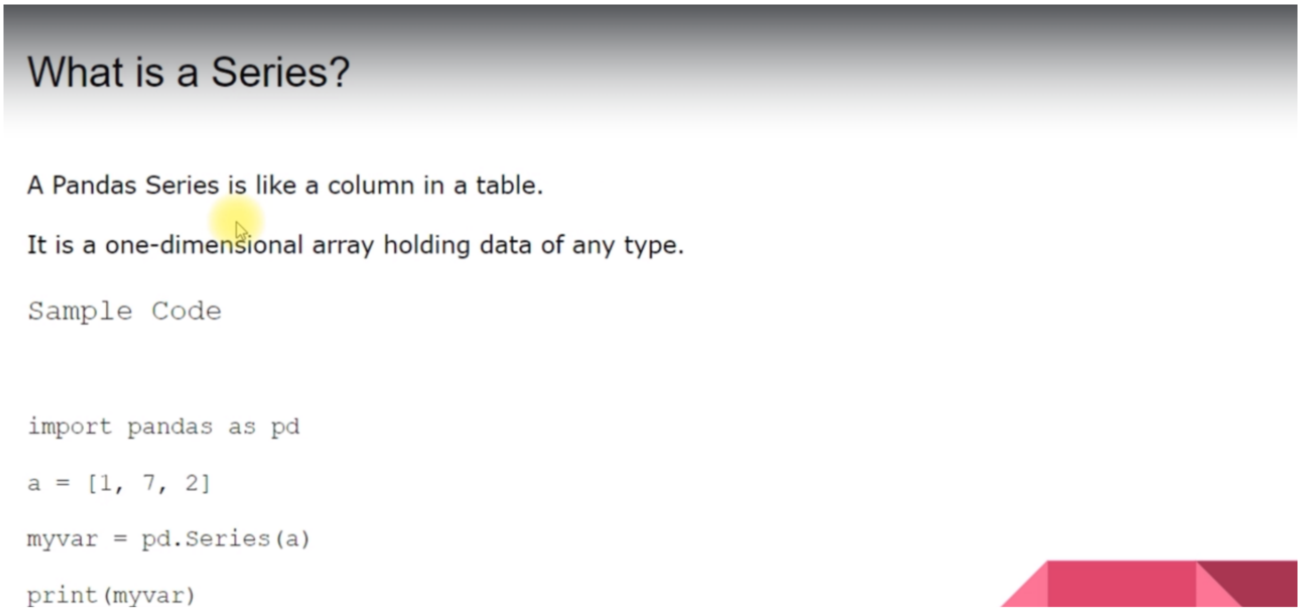
Data Exploration: Pandas provides powerful tools for exploring and summarizing data, such as calculating statistics, finding patterns, and visualizing data with charts and graphs.

Data Transformation: You can reshape and transform data easily, like pivoting tables, merging datasets, or creating new columns based on existing ones.

Time Series Analysis: It's great for working with time-based data, which is useful in finance, weather, and many other fields.



Pandas Series



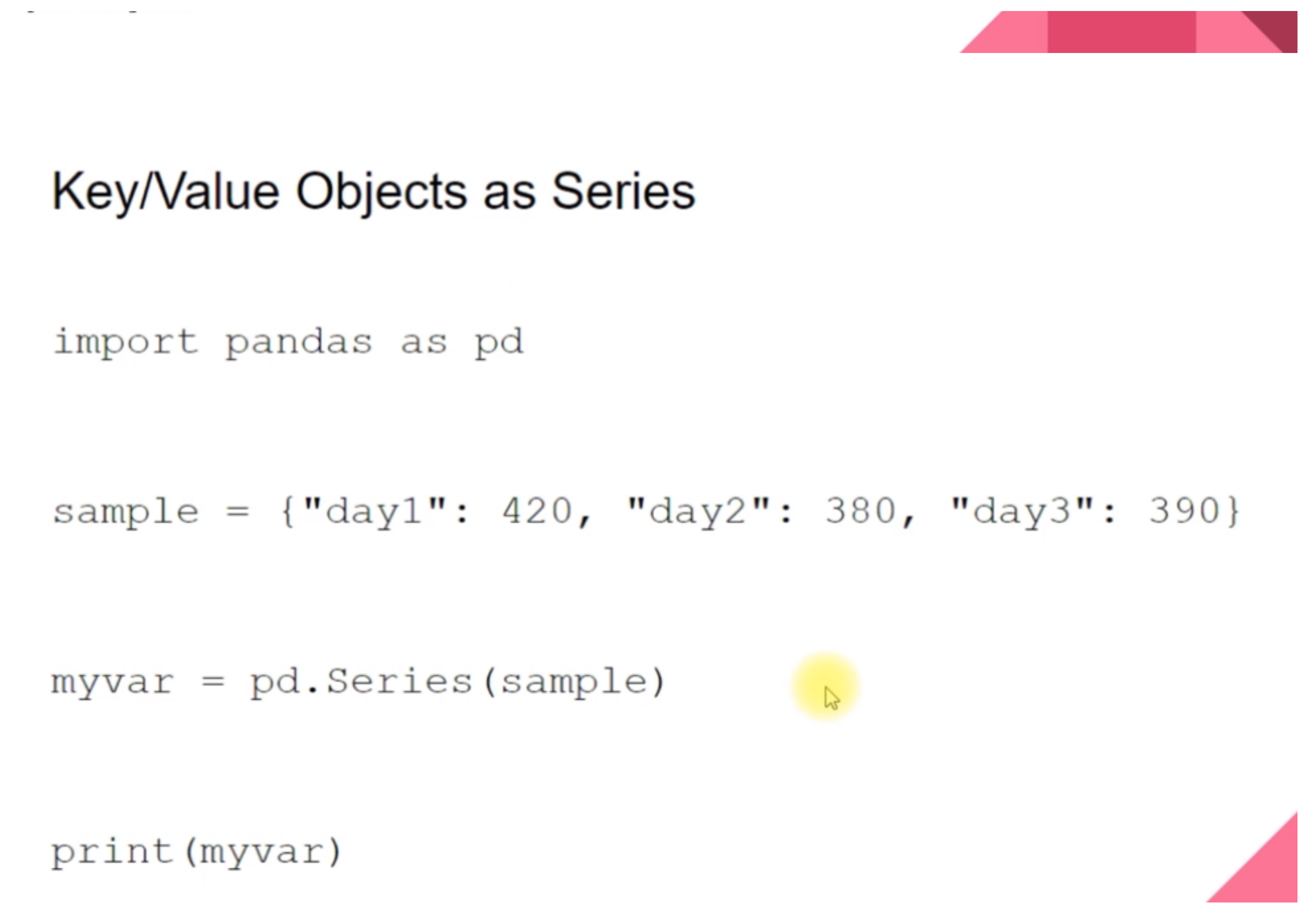
What is a Series?

A Pandas Series is like a column in a table.

It is a one-dimensional array holding data of any type.

Sample Code

```
import pandas as pd

a = [1, 7, 2]

myvar = pd.Series(a)

print(myvar)
```



Key/Value Objects as Series

```
import pandas as pd

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

myvar = pd.Series(sample)

print(myvar)
```

why you would want to use Series in Pandas:

Using Series in Pandas is essential because it provides a convenient and powerful way to work with one-dimensional labeled data.

Labeling Data: Series allows you to associate labels (or indices) with each data point. This makes it easier to identify and access specific elements within the data.

Homogeneous Data: Series enforces that the data within it is of the same data type, which is important for many data analysis tasks where consistent data types are required.

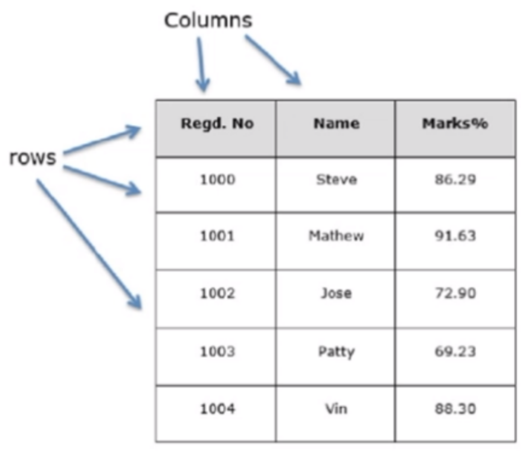
Integration with Other Pandas Structures: Series are the building blocks of more complex Pandas structures like DataFrames. Many operations that you perform on DataFrames involve Series, so understanding how to use them is fundamental.

Mathematical and Statistical Operations: You can perform various mathematical and statistical operations on Series, like summing, averaging, finding maximum and minimum values, and more.

Pandas DataFrame

What is a DataFrame?

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



Regd. No	Name	Marks%
1000	Steve	86.29
1001	Mathew	91.63
1002	Jose	72.90
1003	Patty	69.23
1004	Vin	88.30

why DataFrames are used in Pandas:

Tabular Data Representation: DataFrames organize data in a tabular format with rows and columns, making it easy to view, manipulate, and analyze structured data.

Column Labels: DataFrames allow you to assign meaningful labels to each column, which helps in identifying and understanding the data.

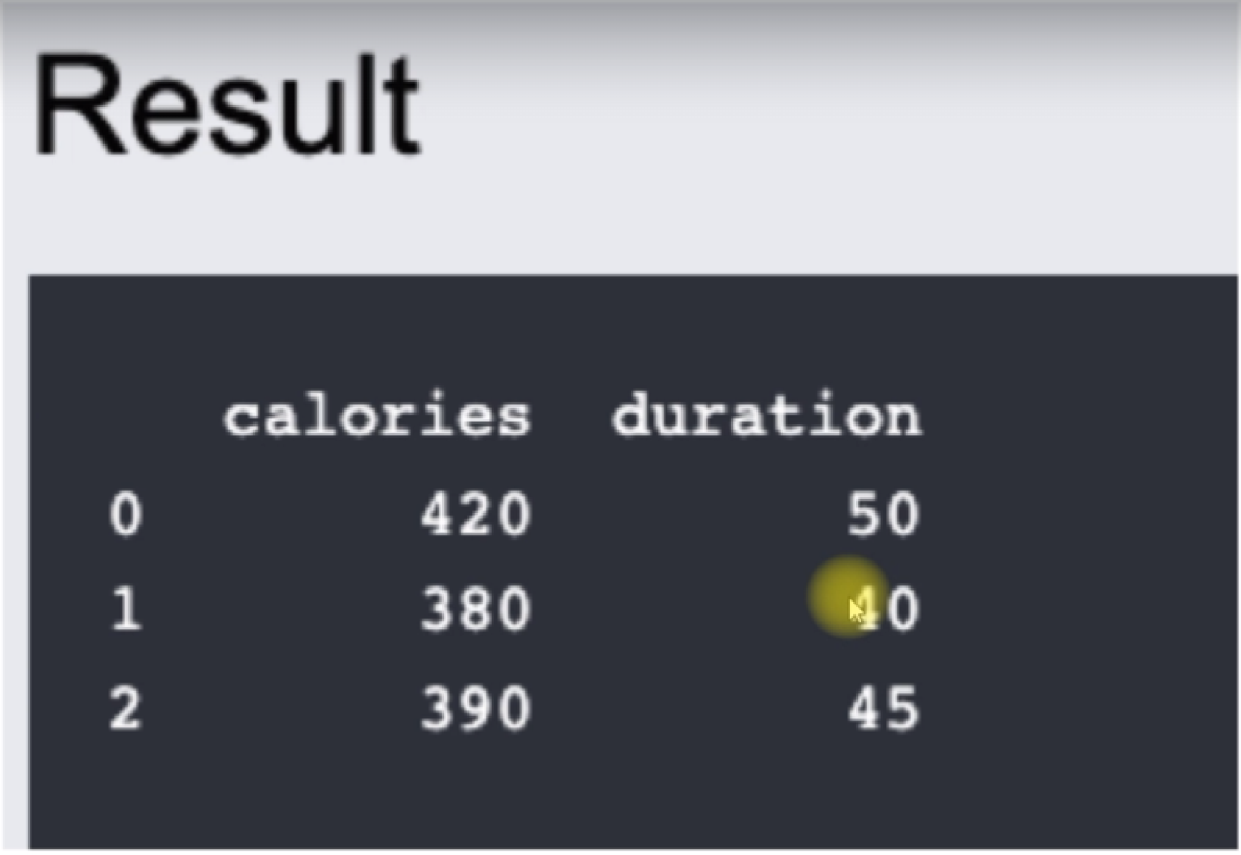
Efficient Data Handling: They are optimized for performance, enabling efficient data retrieval, filtering, and manipulation, even with large datasets.

Data Alignment: DataFrames automatically align data based on their row and column labels. This feature simplifies operations involving multiple DataFrames.

Data Cleaning and Transformation: DataFrames provide powerful tools for cleaning, transforming, and reshaping data, making it suitable for various analysis tasks.

Sample Code

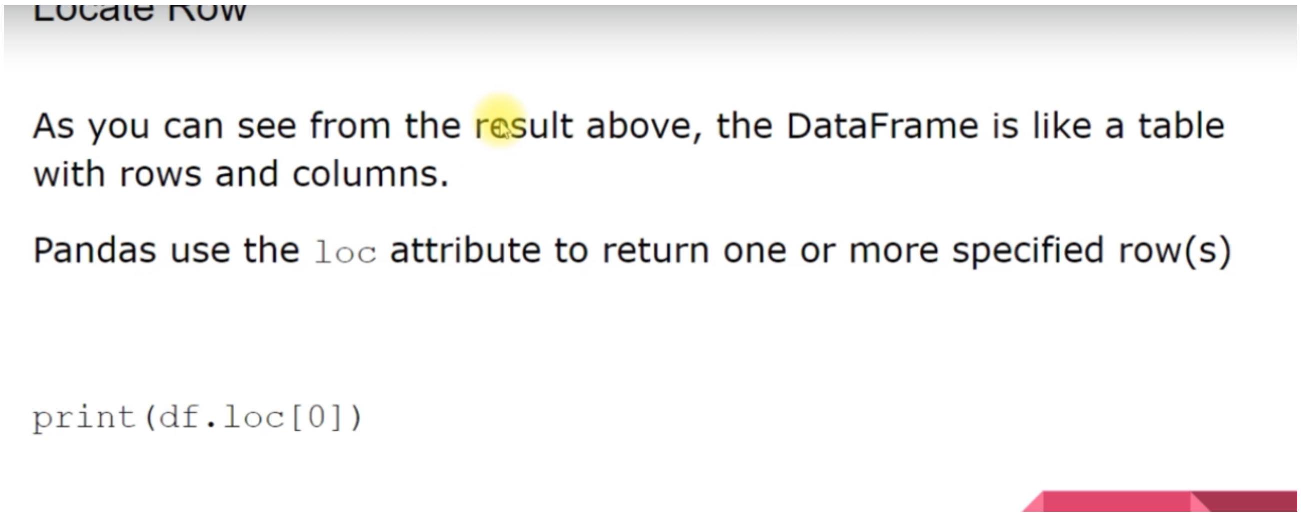
```
import pandas as pd


data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}
df = pd.DataFrame(data)
print(df)
```



Result

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

Locate Row



Locate Row

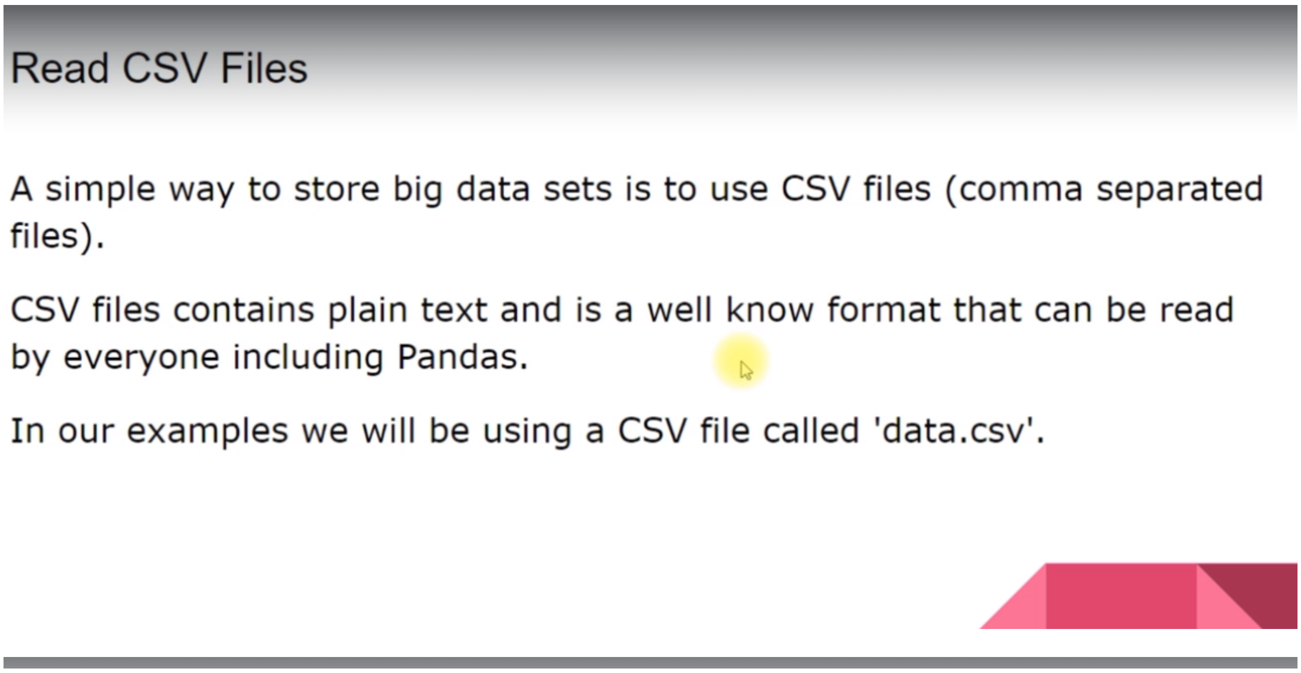
As you can see from the result above, the DataFrame is like a table with rows and columns.

Pandas use the `loc` attribute to return one or more specified row(s)

```
print(df.loc[0])
```

http://localhost:8888/notebooks/Pandas%2BDataFrame%2BPractical.ipynb

Pandas Read CSV

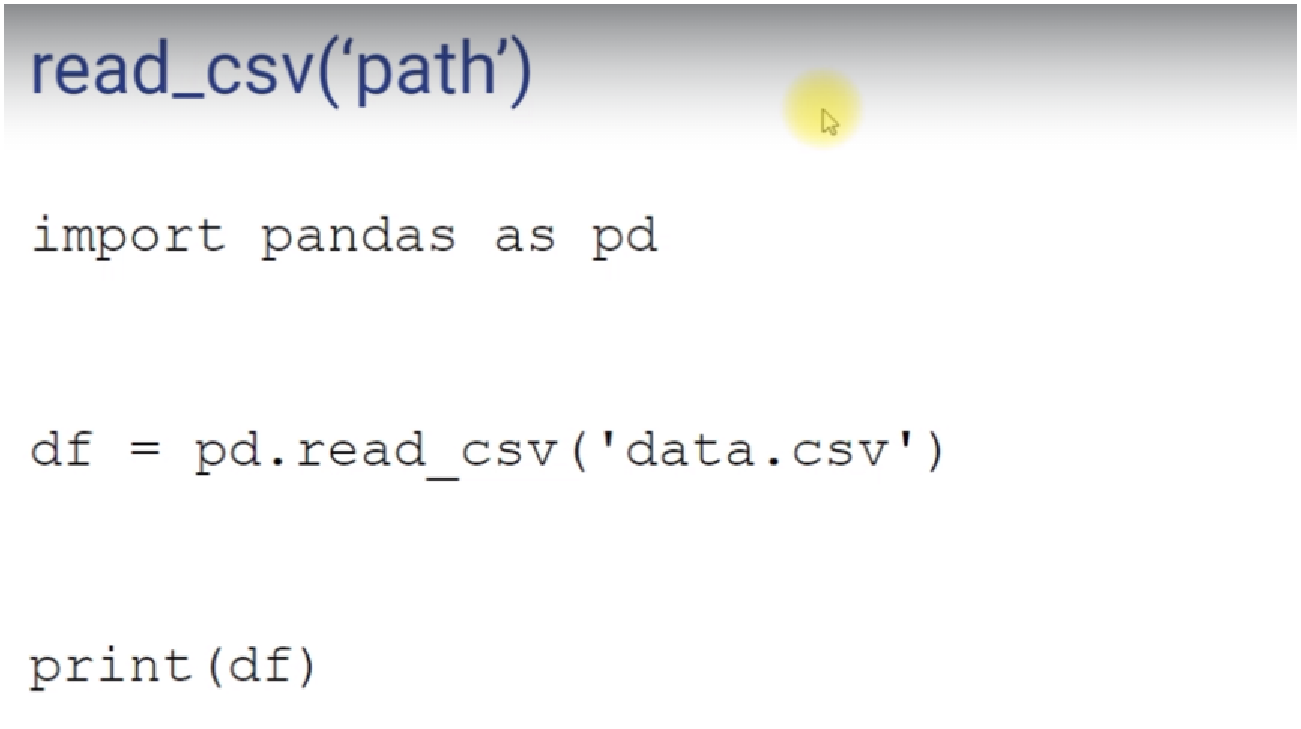


Read CSV Files

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'.



read_csv('path')

```
import pandas as pd

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

print(df)
```

Import Pandas: First, make sure you have Pandas installed. You can import Pandas at the beginning of your Python script or Jupyter Notebook:

import pandas as pd

Specify the CSV File Path: Provide the path to the CSV file you want to read. You can use either an absolute path or a relative path from your current working directory.

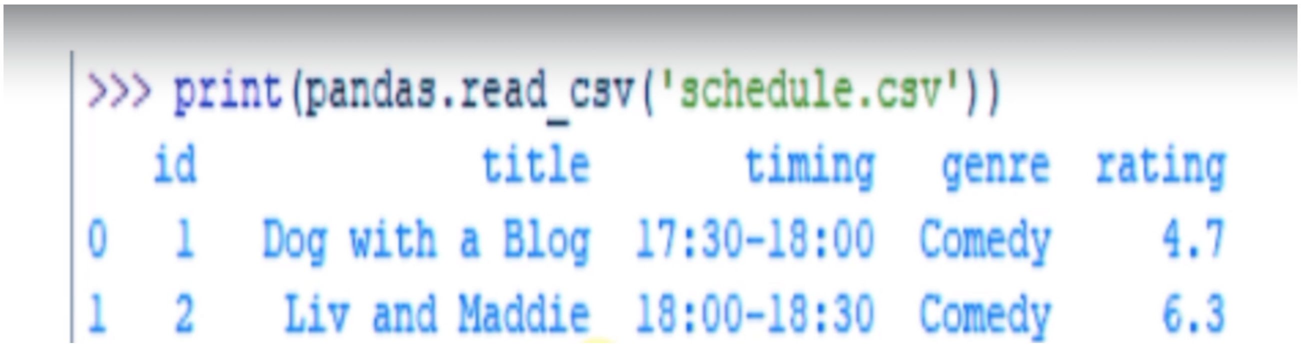
Read the CSV File: Use the read_csv function to read the CSV file into a Pandas DataFrame:

Explore the Data: You can now work with your data stored in the DataFrame (df). Here are some common operations:

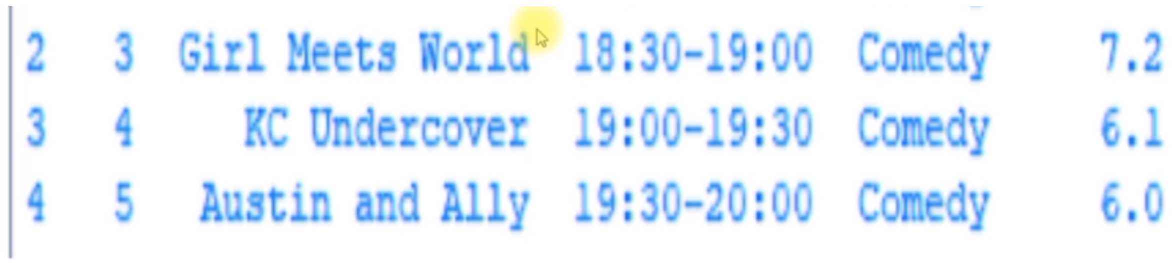
- **Viewing Data**: To quickly inspect the first few rows of the DataFrame, use head()
- df.head()

Summary Statistics: Get summary statistics for numerical columns with describe():df.describe()

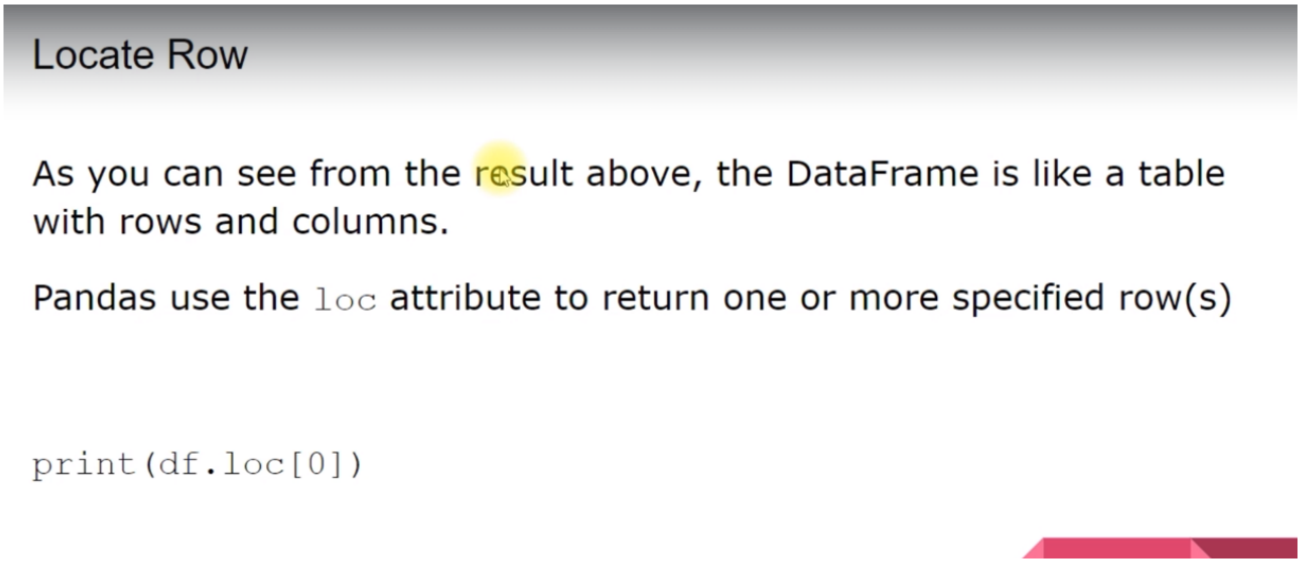
Accessing Columns: Access specific columns by their names, for example: df['column_name']



```
>>> print(pandas.read_csv('schedule.csv'))
   id             title       timing   genre  rating
0   1   Dog with a Blog  17:30-18:00  Comedy     4.7
1   2    Liv and Maddie  18:00-18:30  Comedy     6.3
```



```
2   3  Girl Meets World  18:30-19:00  Comedy    7.2
3   4      KC Undercover  19:00-19:30  Comedy    6.1
4   5   Austin and Ally  19:30-20:00  Comedy    6.0
```



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)

```
print(df.loc[0])
```

To locate a specific row in a Pandas DataFrame:

By Index Label:

- Use .loc['label'] to find by index label.

By Integer Index Position:

- Use .iloc[index] to find by position.

By Condition:

- Use boolean indexing with conditions.

By Multiple Conditions:

- Combine conditions with & or |.

http://localhost:8888/notebooks/Read%2BCSV.ipynb

Analyzing Data Frames

Analyzing a DataFrame in Pandas is like being a detective for data! 🕵️🕵️ Imagine you have a big table of information, like a super-sized spreadsheet. This table is called a DataFrame in Pandas.

Here's how we analyze it step by step:-

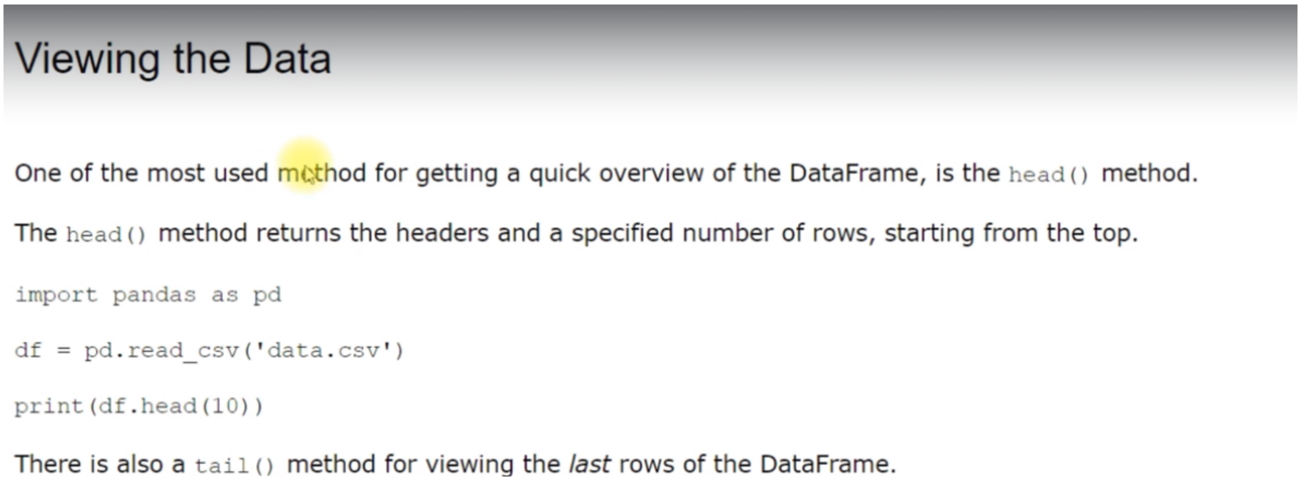
Looking at the Clues (Viewing Data): We start by looking at the data to see what's inside. Pandas lets us see the first few rows using head(). It's like peeking at the first few pages of a book.

Counting and Summing (Summary Stats): We want to know more about the data, like how many items are in a category or what the total is. Pandas helps us count, add, and find averages using things like sum() and describe(). It's like counting how many candies you have!

Finding Clues (Filtering Data): Sometimes, we want to look at only certain parts of the data. Pandas helps us do that by creating a new DataFrame with just the data we want. It's like picking only the red candies from a jar of mixed candies!

Grouping Clues (Grouping and Aggregating): We can group similar things together, like grouping all the candies by flavor. Pandas helps us do this and even calculate things like the average number of candies in each group.

Visualizing Clues (Making Charts): Sometimes, we use charts and graphs to understand data better. Pandas can create cool charts to help us see trends and patterns, just like drawing pictures of candies to see which one is the most popular!



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))
```

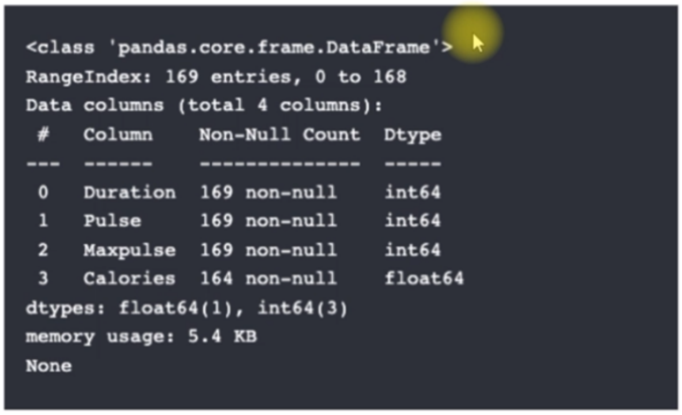
There is also a `tail()` method for viewing the *last* rows of the DataFrame.

The `tail()` method returns the headers and a specified number of rows, starting from the bottom.

Info About the Data

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

```
print(df.info())
```



```
<class 'pandas.core.frame.DataFrame'>
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
dtypes: float64(1), int64(3)
memory usage: 5.4 KB
None
```

Describe Method

The `describe()` method returns description of the data in the DataFrame.If the DataFrame contains numerical data, the description contains these information for each column:

count - The number of not-empty values.

mean - The average (mean) value.

std - The standard deviation.

min - the minimum value.

25% - The 25% percentile*.

50% - The 50% percentile*.

75% - The 75% percentile*.

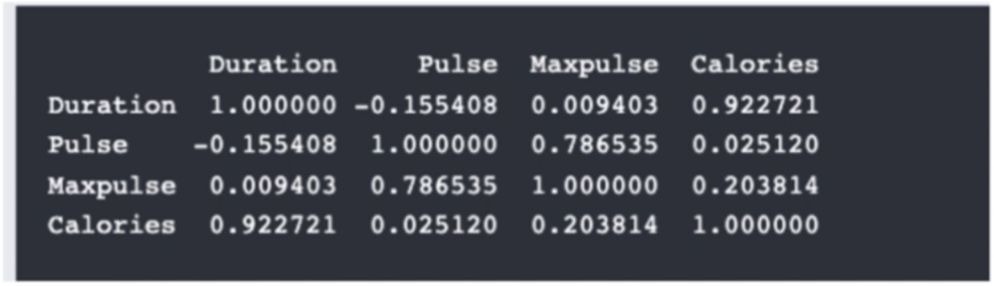
max - the maximum value.

Data Correlations

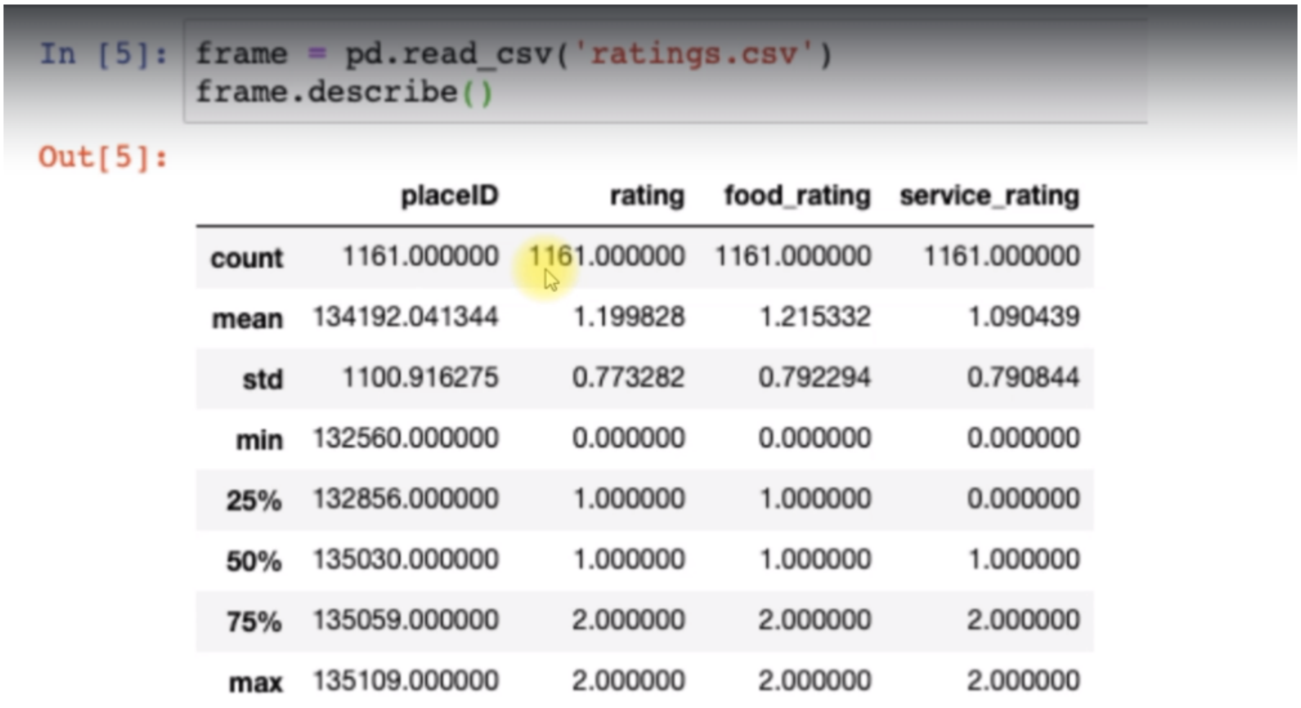
A great aspect of the Pandas module is the `corr()` method.

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

```
df.corr()
```



```
          Duration     Pulse  Maxpulse  Calories
Duration  1.000000 -0.155408  0.009403  0.922721
Pulse    -0.155408  1.000000  0.786535  0.025120
Maxpulse  0.009403  0.786535  1.000000  0.203814
Calories  0.922721  0.025120  0.203814  1.000000
```



```
In [5]: frame = pd.read_csv('ratings.csv')
        frame.describe()
Out[5]:
```

	placeID	rating	food_rating	service_rating
count	1161.000000	1161.000000	1161.000000	1161.000000
mean	134192.041344	1.199828	1.215332	1.090439
std	1100.916275	0.773282	0.792294	0.790844
min	132560.000000	0.000000	0.000000	0.000000
25%	132856.000000	1.000000	1.000000	0.000000
50%	135030.000000	1.000000	1.000000	1.000000
75%	135059.000000	2.000000	2.000000	2.000000
max	135109.000000	2.000000	2.000000	2.000000

http://localhost:8888/notebooks/Analyzing%2BDataFrames.ipynb