# **Python basic for data analyst**

* Each line in a dataset is a row, and commas separate the values.
* To understand the data, you must analyze the attributes for each column of data.
* Python libraries are collections of functions and methods that facilitate various functionalities without writing code from scratch and are categorized into Scientific Computing, Data Visualization, and Machine Learning Algorithms.
* Many data science libraries are interconnected; for instance, Scikit-learn is built on top of NumPy, SciPy, and Matplotlib.
* The data format and the file path are two key factors for reading data with Pandas.
* The **read\_CSV** method in Pandas can read files in CSV format into a Pandas DataFrame.
* Pandas has unique data types like object, float, Int, and datetime.
* Use the **dtype** method to check each column’s data type; misclassified data types might need manual correction.
* Knowing the correct data types helps apply appropriate Python functions to specific columns.
* Using **Statistical Summary** with **describe()** provides count, mean, standard deviation, min, max, and quartile ranges for numerical columns.
* You can also use **include='all'** as an argument to get summaries for object-type columns.
* The statistical summary helps identify potential issues like outliers needing further attention.
* Using the **info() Method** gives an overview of the top and bottom 30 rows of the DataFrame, useful for quick visual inspection.
* Some statistical metrics may return "NaN," indicating missing values, and the program can’t calculate statistics for that specific data type.
* Python can connect to databases through specialized code, often written in Jupyter notebooks.
* SQL Application Programming Interfaces (APIs) and Python DB APIs (most often used) facilitate the interaction between Python and the DBMS.
* **SQL APIs** connect to DBMS with one or more API calls, build SQL statements as a text string, and use API calls to send SQL statements to the DBMS and retrieve results and statuses.
* **DB-API**,Python's standard for interacting with relational databases,uses **connection objects** to establish and manage database connections and **cursor objects** to run queries and scroll through the results.
* Connection Object methods include the cursor(), commit(), rollback(), and close() commands.
* You can import the database module, use the **Connect API** to open a connection, and then create a cursor object to run queries and fetch results.
* Remember to close the database connection to free up resources.

How to download and handle dataset from url link

import requests  
def dowload\_file(url, save\_as):  
 response = requests.get(url, stream=True)  
 if response.status\_code == 200:  
 with open(save\_as, 'wb') as file:  
 for chunk in response.iter\_content(chunk\_size=8192):  
 file.write(chunk)  
 print(f"File downloaded and saved as {save\_as}")  
 else:  
 print(f"Failed to download file. Status code: {response.status\_code}")

# Data Analysis with Python

## Cheat Sheet: Importing Data Sets

|  |  |  |
| --- | --- | --- |
| **Package/Method** | **Description** | **Code Example** |
| Read CSV data set | Read the CSV file containing a data set to a pandas data frame | 1. 1   df = pd.read\_csv(<CSV\_path>, header = None)  # load without header  df = pd.read\_csv(<CSV\_path>, header = 0)  # load using first row as header  Copied!  Note: The labs in this course run in JupyterLite environment. In JupyterLite environment, you'll need to download the required file to the local environment and then use the local path to the file as the CSV\_path. However, in case you are using JupyterLabs, or any other Python compiler on your local machine, you can use the URL of the required file directly as the CSV\_path. |
| Print first few entries | Print the first few entries (default 5) of the pandas data frame | df.head(n) #n=number of entries; default 5 |
| Print last few entries | Print the last few entries (default 5) of the pandas data frame | df.tail(n) #n=number of entries; default 5  Copied! |
| Assign header names | Assign appropriate header names to the data frame | df.columns = headers  Copied! |
| Replace "?" with NaN | Replace the entries "?" with NaN entry from Numpy library | df = df.replace("?", np.nan) |
| Retrieve data types | Retrieve the data types of the data frame columns | df.dtypes |
| Retrieve statistical description | Retrieve the statistical description of the data set. Defaults use is for only numerical data types. Use include="all" to create summary for all variables | df.describe() #default use df.describe(include="all") |
| Retrieve data set summary | Retrieve the summary of the data set being used, from the data frame | df.info() |
| Save data frame to CSV | Save the processed data frame to a CSV file with a specified path | df.to\_csv(<output CSV path>) |

Module 2 : Preprocessing Data in Python

Data pre-processing in Python

The process of converting or mapping data from the initial “raw” form another format, in order to prepare the data for further analysis.

Also known as: Data Cleaning, Data wrangling.

How to drop missing values in Python

Use dataframe.dropna() – axis = 0 row, axis = 1 column

Ex: df.dropna(subset = [“price”], axis = 0, inplace = True)

How to replace?

Use dataframe.replace(missing\_value, new\_value)

Replace by mean value: mean = df[“normalize-losses”].mean()

Replace it in dataframe : df[“nomalize-losses”].replace(np.nan, mean)