

Study Unit

6

Introduction to Data Analysis with Pandas Outline

- Introduction to Pandas
- Series and DataFrames
- Importing and exporting dataset
- Data cleaning and preprocessing
- Exploratory data analysis
- Computing Descriptive Statistics
- Combining and Merging Datasets

Study Unit Duration

This Study Unit requires a minimum of 3 hours' formal study time. You may spend an additional 2-3 hours on revision.

INTRODUCTION TO DATA ANALYSIS WITH PANDAS



Preamble

Pandas is a popular Python package for data science. It offers powerful, expressive, and flexible data structures that make data cleaning and analysis fast and easy in Python. Pandas is an open source data analysis and manipulation tool that is often used in tandem with numerical computing tools like NumPy and SciPy, analytical libraries like statsmodels and scikit-learn, and data visualization libraries like matplotlib and Seaborn. You can think of Pandas as an extremely powerful version of spreadsheet like Microsoft Excel, with a lot more features.

Learning Outcomes of Study Unit 6

Upon completion of this study unit, you should be able to:

- 6.1 Work with Pandas Package to create series, Data frame and get information about your data
- 6.2 Manipulate Data Frames with Pandas
- 6.3 Load data in any file and export them
- 6.4 Clean and preprocess your data
- 6.5 Carry out exploratory data analysis with Pandas Profiling
- 6.6 Aggregate and do group operation with your data
- 6.7 Join, Combine and reshape data frame with Pandas



Terminologies, Acronyms and their Meaning

AI	Artificial Intelligence
ML	Machine Learning
RL	Reinforcement Learning
DL	Deep learning
EDA	Exploratory Data Analysis
NaN	Not a Number

np	Numpy
pd	Pandas
os	Operating system
AI	Artificial Intelligence
TF	TensorFlow
NULL	Missing value

6.1 Work with Pandas Package to create series, Data frame and get information about your data

If you installed the [Anaconda distribution](#) of Python - it includes **Python**, **NumPy**, and other commonly used packages for scientific computing and data science. Therefore, no further installation steps are necessary. We recommend you use the [Anaconda distribution](#) of Python as you begin your data science journey.

If you use a version of Python from python.org or a version of Python that came with your operating system, the **Anaconda Prompt** and **conda** or **pip** can be used to install NumPy.

Install Pandas with the Anaconda Prompt

To install **Pandas**, open the Anaconda Prompt and type:

conda install pandas

Type **y** for yes when prompted.

Install Pandas with pip

To install Pandas with pip, bring up a terminal window and type:

pip install pandas

This command installs Pandas in the current working Python environment.

Importing the Pandas library

To import pandas, we usually import it with its alias, **pd**.

```
import pandas as pd
```

Core components of pandas

The two primary components of pandas are the Series and DataFrame. A **Series** is essentially a column, and a **DataFrame** is a multi-dimensional table made up of a collection of Series.

Series		Series		Series		DataFrame			
	Gender		Country		Age		Gender	Country	Age
0	Male	0	Uganda	0	11	0	Male	Uganda	11
1	Female	1	Ethiopia	1	17	1	Female	Ethiopia	17
2	Female	2	Rwanda	2	15	2	Female	Rwanda	15
3	Female	3	Somalia	3	17	3	Female	Somalia	17
4	Male	4	Sudan	4	13	4	Male	Sudan	13
5	Female	5	Kenya	5	11	5	Female	Kenya	11
6	Male	6	Uganda	6	9	6	Male	Uganda	9
7	Female	7	Kenya	7	18	7	Female	Kenya	18
8	Male	8	Somalia	8	13	8	Male	Somalia	13
9	Female	9	Rwanda	9	16	9	Female	Rwanda	16

Series and Data Frame

Pandas Series

A Series is a one-dimensional array-like object containing a sequence of values and capable of holding any data type with axis labels or index. An example of a Series object is one column from a DataFrame. That is, Pandas Series is formed from only an array of data.

How to create Pandas Series

Example 1

```
Gender = pd.Series(["Male", "Female", "Female", "Female", "Male", "Female", "Male", "Female", "Male",  
                    "Female"])
```

Gender

```
0    Male
1    Female
2    Female
3    Female
4    Male
5    Female
6    Male
7    Female
8    Male
9    Female
dtype: object
```

```
type(Gender) # This is a pandas series type
```

```
<class 'pandas.core.series.Series'>
```

Example 2

```
Country = pd.Series(["Uganda", "Ethiopia", "Rwanda", "Somalia", "Sudan", "Kenya", "Uganda", "Kenya",
                    "Somalia", "Rwanda"])
```

```
Country
```

```
0    Uganda
1    Ethiopia
2    Rwanda
3    Somalia
4    Sudan
5    Kenya
6    Uganda
7    Kenya
8    Somalia
9    Rwanda
dtype: object
```

```
type(Country)
```

```
<class 'pandas.core.series.Series'>
```

Example 3

```
Age = pd.Series([11, 17, 15, 17, 13, 11, 9, 18, 13, 16])
```

```
Age
```

```
0    11
1    17
2    15
3    17
4    13
5    11
6     9
7    18
8    13
9    16
dtype: int64
```

```
type(Age)
```

```
<class 'pandas.core.series.Series'>
```

Class activity 8

Peer to Peer Interaction

Visit the LMS, locate forum activity and participate in the discussion



Consider the dataset below:

Country	Region	Total Cases
Algeria	North Africa	88252
Angola	Central Africa	15536
Benin	West Africa	3055
Botswana	Southern Africa	11531
Burkina Faso	West Africa	3156
Burundi	East Africa	694
Cabo Verde	West Africa	11036
Cameroon	Central Africa	24752
Central African Republic	Central Africa	4922
Chad	Central Africa	1725
Comoros	East Africa	616
Congo	Central Africa	5774
Ivory Coast	West Africa	21485
Djibouti	East Africa	5701

Create Pandas Series for each of the columns in the dataset.

6.1.1 Pandas DataFrame

A DataFrame represents a rectangular table of data and contains an ordered collection of columns, each of which can be a different value type (integer, float, string, boolean, etc.). We can think of a DataFrame as a bunch of Series objects put together to share the same index.

How to create Pandas DataFrame

There are many ways to create a DataFrame from scratch, but a great option is to create it from a dictionary.

Creating DataFrame from a Dictionary

Among many things that can serve as input to make a Pandas **DataFrame** is a **dict** (Dictionary). To make a data frame from a dictionary, you will need to pass it to the `DataFrame()` function in the data argument.

Example 1

```
registrations_dict = {
    "Gender": ["Male", "Female", "Female", "Female", "Male", "Female", "Male", "Female", "Male", "Female"],
    "Country": ["Uganda", "Ethiopia", "Rwanda", "Somalia", "Sudan", "Kenya", "Uganda", "Kenya", "Somalia", "Rwanda"],
    "Age": [11, 17, 15, 17, 13, 11, 9, 18, 13, 16]
}
registrations_dict
```

```
{'Gender': ['Male', 'Female', 'Female', 'Female', 'Male', 'Female', 'Male', 'Female', 'Male', 'Female'],
 'Country': ['Uganda', 'Ethiopia', 'Rwanda', 'Somalia', 'Sudan', 'Kenya', 'Uganda', 'Kenya', 'Somalia', 'Rwanda'],
 'Age': [11, 17, 15, 17, 13, 11, 9, 18, 13, 16]}
```

```
type(registrations_dict)
```

```
<class 'dict'>
```

And then pass the result to the pandas DataFrame constructor:

```
registrations_df = pd.DataFrame(registrations_dict)
registrations_df
```

	Gender	Country	Age
0	Male	Uganda	11
1	Female	Ethiopia	17
2	Female	Rwanda	15
3	Female	Somalia	17
4	Male	Sudan	13
5	Female	Kenya	11
6	Male	Uganda	9
7	Female	Kenya	18
8	Male	Somalia	13
9	Female	Rwanda	16

The resulting DataFrame will have its index assigned automatically as with Series, and the columns are placed unordered. If you specify a sequence of columns, the DataFrame's columns will be arranged in that order.

```
registrations_df = pd.DataFrame(registrations_dict, columns = ["Country", "Gender", "Age"])
```

registrations_df

	Country	Gender	Age
0	Uganda	Male	11
1	Ethiopia	Female	17
2	Rwanda	Female	15
3	Somalia	Female	17
4	Sudan	Male	13
5	Kenya	Female	11
6	Uganda	Male	9
7	Kenya	Female	18
8	Somalia	Male	13
9	Rwanda	Female	16

Example 2

This example is for corona virus (COVID-19) report as of November 25, 2020.

```
COVID19data = {
    "Country": ["Burundi", "Ethiopia", "Kenya", "Rwanda", "Somalia", "Tanzania", "Uganda", "Sudan"],
    "Total Cases": [673, 107109, 79322, 5750, 4445, 509, 18406, 16431],
    "Total Deaths": [1, 1664, 1417, 47, 113, 21, 186, 1202],
    "Total Recovered": [575, 66574, 52974, 5241, 3412, 183, 8764, 9854],
    "Active Cases" : [97, 38871, 24931, 462, 920, 305, 9456, 5375],
    "Population" : [12029114, 116082175, 54237961, 13078334, 16066789, 60399786, 46304101, 44252054]
}
COVID19data_df = pd.DataFrame(COVID19data, columns=["Country", "Total Cases", "Total Deaths", "Total Recovered", "Active Cases", "Population"])
COVID19data_df
```

	Country	Total Cases	Total Deaths	Total Recovered	Active Cases	Population
0	Burundi	673	1	575	97	12029114
1	Ethiopia	107109	1664	66574	38871	116082175
2	Kenya	79322	1417	52974	24931	54237961
3	Rwanda	5750	47	5241	462	13078334
4	Somalia	4445	113	3412	920	16066789
5	Tanzania	509	21	183	305	60399786
6	Uganda	18406	186	8764	9456	46304101
7	Sudan	16431	1202	9854	5375	44252054

We shall be using `registrations_df` and `COVID19data_df` dataframes in our various examples henceforth.

6.1.2 Viewing your data

For large DataFrames, we can select the first five observations or rows by using `.head()` attribute and the last five observation by using `.tail()` attribute.

Example 1

```
registrations_df.head()
```

	Country	Gender	Age
0	Uganda	Male	11
1	Ethiopia	Female	17
2	Rwanda	Female	15
3	Somalia	Female	17
4	Sudan	Male	13

```
registrations_df.tail()
```

	Country	Gender	Age
5	Kenya	Female	11
6	Uganda	Male	9
7	Kenya	Female	18
8	Somalia	Male	13
9	Rwanda	Female	16

Example 2

```
COVID19data_df.head()
```

	Country	Total Cases	Total Deaths	Total Recovered	Active Cases	Population
0	Burundi	673	1	575	97	12029114
1	Ethiopia	107109	1664	66574	38871	116082175
2	Kenya	79322	1417	52974	24931	54237961
3	Rwanda	5750	47	5241	462	13078334
4	Somalia	4445	113	3412	920	16066789

```
COVID19data_df.tail(3)
```

	Country	Total Cases	Total Deaths	Total Recovered	Active Cases	Population
5	Tanzania	509	21	183	305	60399786
6	Uganda	18406	186	8764	9456	46304101
7	Sudan	16431	1202	9854	5375	44252054

6.1.3 Getting information about your data

.info() method

.info() attribute provides the essential details about your dataset, such as the number of rows and columns, the number of non-null values, what type of data is in each column, and how much memory your DataFrame is using.

Example 1

```
registrations_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype  
---  -
 0   Country  10 non-null     object  
 1   Gender   10 non-null     object  
 2   Age      10 non-null     int64   
dtypes: int64(1), object(2)
memory usage: 368.0+ bytes
```

Seeing the datatype quickly is actually quite useful in data cleaning and analysis phases

Example 2

```
COVID19data_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8 entries, 0 to 7
Data columns (total 6 columns):
#   Column                Non-Null Count  Dtype
---  -
0   Country                8 non-null      object
1   Total Cases            8 non-null      int64
2   Total Deaths          8 non-null      int64
3   Total Recovered        8 non-null      int64
4   Active Cases           8 non-null      int64
5   Population              8 non-null      int64
dtypes: int64(5), object(1)
memory usage: 512.0+ bytes
```

.shape method

Another useful attribute is **.shape**, which outputs just a tuple of (rows, columns):

Example 1

```
registrations_df.shape
```

```
(10, 3)
```

The students' registration data has 10 rows and 3 columns.



Note that **.shape** has no parentheses and is a simple tuple of format (rows, columns). So, we have 10 rows and 3 columns in our students' registration DataFrame.

Example 2

```
COVID19data_df.shape
```

```
(8, 6)
```

The COVID-19 has 8 rows and 6 columns.

It is important to note that you will be going to **.shape** a lot when cleaning and transforming data. For example, you might filter some rows based on some criteria and then want to know quickly how many rows were removed or left in the dataset.

Display the index and columns in a DataFrame

.index and .columns

We can use **.index** and **.columns** attribute to see the list of observations and column names in our dataframe.

Example 1

Let's check the index and columns in the students's registration DataFrame

```
registrations_df.index
```

```
RangeIndex(start=0, stop=10, step=1)
```

The observation/index is from 0 to 9. Remember Numpy array, 0:10 means 0, 1, 2, ..., 9.

```
registrations_df.columns
```

```
Index(['Country', 'Gender', 'Age'], dtype='object')
```

The column names in the students registration dataframe are Country, Gender, and Age.

Example 2

Let's check the columns in the COVID-19 DataFrame

```
COVID19data_df.columns
```

```
Index(['Country', 'Total Cases', 'Total Deaths', 'Total Recovered', 'Active Cases', 'Population'], dtype='object')
```

The column names in the COVID-19 data are Country, Total Cases, Total Deaths, Total Recovered, Active Cases, and Population

6.2. Selection and Indexing of DataFrame

Let's learn various methods to grab data from a DataFrame.

6.2.1 Selecting a Series in a DataFrame

Remember our student's registration DataFrame:

```
Registrations_df
```

	Country	Gender	Age
0	Uganda	Male	11
1	Ethiopia	Female	17
2	Rwanda	Female	15
3	Somalia	Female	17
4	Sudan	Male	13
5	Kenya	Female	11
6	Uganda	Male	9
7	Kenya	Female	18
8	Somalia	Male	13
9	Rwanda	Female	16

A column in a DataFrame can be retrieved as a Series:

```
registrations_df['Country']
```

```
0    Uganda
1  Ethiopia
2    Rwanda
3    Somalia
4     Sudan
5     Kenya
6    Uganda
7     Kenya
8    Somalia
9     Rwanda
Name: Country, dtype: object
```

```
registrations_df.Gender
```

```
0      Male
1      Female
2      Female
3      Female
4      Male
5      Female
6      Male
7      Female
8      Male
9      Female
Name: Gender, dtype: object
```



`registrations_df['column']` works for any column name, but `registrations_df.column` only works when the column name is a valid Python variable name. That is, it follows Python naming convention (no space, no special character, not starting with number, etc.)

6.2.2 Selecting two or more columns in a dataframe

We pass a list of column names to select two or more columns in a dataframe

Example 1

```
registrations_df[["Country", "Gender"]]
```

	Country	Gender
0	Uganda	Male
1	Ethiopia	Female
2	Rwanda	Female
3	Somalia	Female
4	Sudan	Male
5	Kenya	Female
6	Uganda	Male
7	Kenya	Female
8	Somalia	Male
9	Rwanda	Female

Example 2

COVID19data_df # *Did you remember this dataframe?*

	Country	Total Cases	Total Deaths	Total Recovered	Active Cases	Population
0	Burundi	673	1	575	97	12029114
1	Ethiopia	107109	1664	66574	38871	116082175
2	Kenya	79322	1417	52974	24931	54237961
3	Rwanda	5750	47	5241	462	13078334
4	Somalia	4445	113	3412	920	16066789
5	Tanzania	509	21	183	305	60399786
6	Uganda	18406	186	8764	9456	46304101
7	Sudan	16431	1202	9854	5375	44252054

Remember we saw that `df.column` only works when the column name is a valid Python variable name.

COVID19data_df.Total Cases

We have an error because the column name **Total Cases** is not a valid variable name i.e. it has a space. An alternative to `df.column` in selecting a variable name is `df["column"]`.

COVID19data_df["Total Cases"]

```
0      673
1    107109
2    79322
3     5750
4     4445
5      509
6    18406
7    16431
Name: Total Cases, dtype: int64
```

COVID19data_df[["Country", "Population", "Total Cases"]] # *Total Cases is COVID-19 cases*

	Country	Population	Total Cases
0	Burundi	12029114	673
1	Ethiopia	116082175	107109
2	Kenya	54237961	79322
3	Rwanda	13078334	5750
4	Somalia	16066789	4445
5	Tanzania	60399786	509
6	Uganda	46304101	18406
7	Sudan	44252054	16431

6.2.3 Selection with .loc and .iloc attribute

The `loc[]` and `iloc[]` DataFrame attribute enable you to select a subset of the rows and columns from a DataFrame with NumPy-like notation using either axis labels (`df.loc[]`) or integers (`df.iloc[]`). The general format for both `loc[]` and `iloc[]` are `df.loc[row, col]` and `df.iloc[row, col]` where `df` is the name of your DataFrame while the `[row, col]` specifies the row and column index of the DataFrame. Remember Python counting starts at 0, so the first row is row zero.

.loc[]

As a preliminary example, let's select first three rows and columns such as Country and Population by using `df.loc[]`.

Example 1

`df.loc[row, col]` is for selection by column label.

```
COVID19data_df.loc[0:2, ["Country", "Population"]]
```

	Country	Population
0	Burundi	12029114
1	Ethiopia	116082175
2	Kenya	54237961

In example 1, we selected the first three observations and variables Country and Population.



Please note that `.loc[]` index position is inclusive i.e. `0:5` means 0, 1, 2, 3, 4, 5. This is different from Numpy array that `0:5` is 0, 1, 2, 3, 4.

Example 2

`df.loc[row, col]` is for selection by column label.

```
COVID19data_df.loc[:, ["Country", "Total Cases"]]
```

	Country	Total Cases
0	Burundi	673
1	Ethiopia	107109
2	Kenya	79322
3	Rwanda	5750
4	Somalia	4445
5	Tanzania	509
6	Uganda	18406
7	Sudan	16431

We selected all the observations (:) and columns Country and COVID-19 Total Cases

`.iloc[]`

`df.iloc[row, col]` is for selection by column index or position. First, let us see the column index by using `.columns` attribute

```
COVID19data_df.columns
```

```
Index(['Country', 'Total Cases', 'Total Deaths', 'Total Recovered',
      'Active Cases', 'Population'],
      dtype='object')
```

Columns:	Country	Total Cases	Total Deaths	Total Recovered	Active Cases	Population
Index:	0	1	2	3	4	5

The columns Country and Population correspond to 0 and 5 indexes

```
COVID19data_df.iloc[0:3, [0, 5]]
```

	Country	Population
0	Burundi	12029114
1	Ethiopia	116082175
2	Kenya	54237961

In example 2, we selected the first three observations and variables Country and Population.



Please note that `.iloc[]` index position is exclusive i.e. 0: 3 means 0, 1, 2. This is like Numpy array.

Example 1

Select all the observations and columns Gender and Age in the student's registration list using

1. `.iloc[]` attribute
2. `.loc[]` attribute

```
registrations_df
```

	Country	Gender	Age
0	Uganda	Male	11
1	Ethiopia	Female	17
2	Rwanda	Female	15
3	Somalia	Female	17
4	Sudan	Male	13
5	Kenya	Female	11
6	Uganda	Male	9
7	Kenya	Female	18
8	Somalia	Male	13
9	Rwanda	Female	16

With `.iloc[]` attribute

`registrations_df.iloc[:, [1, 2]]` # `.iloc[]` selection is by the position.

	Gender	Age
0	Male	11
1	Female	17
2	Female	15
3	Female	17
4	Male	13
5	Female	11
6	Male	9
7	Female	18
8	Male	13
9	Female	16

With `.loc[]` attribute

`registrations_df.loc[:, ["Gender", "Age"]]` # `.loc[]` selection is by the label.

	Gender	Age
0	Male	11
1	Female	17
2	Female	15
3	Female	17
4	Male	13
5	Female	11
6	Male	9
7	Female	18
8	Male	13
9	Female	16

Example 2

Select the last five students in the student's registration dataframe

```
registrations_df
```

	Country	Gender	Age
0	Uganda	Male	11
1	Ethiopia	Female	17
2	Rwanda	Female	15
3	Somalia	Female	17
4	Sudan	Male	13
5	Kenya	Female	11
6	Uganda	Male	9
7	Kenya	Female	18
8	Somalia	Male	13
9	Rwanda	Female	16

Method 1

```
registrations_df.tail(5)
```

	Country	Gender	Age
5	Kenya	Female	11
6	Uganda	Male	9
7	Kenya	Female	18
8	Somalia	Male	13
9	Rwanda	Female	16

Method 2

```
registrations_df.loc[5:9, :] # remember .loc[] attribute is inclusive
```

	Country	Gender	Age
5	Kenya	Female	11
6	Uganda	Male	9
7	Kenya	Female	18
8	Somalia	Male	13
9	Rwanda	Female	16

Method 3

```
registrations_df.iloc[5:10, :] # remember .iloc[] attribute is exclusive
```

	Country	Gender	Age
5	Kenya	Female	11
6	Uganda	Male	9
7	Kenya	Female	18
8	Somalia	Male	13
9	Rwanda	Female	16

6.2.4 How to rename column names in a DataFrame

We can use the `.rename()` method to rename certain or all columns in a DataFrame via a `dict` (dictionary). Let's rename **Total Cases** to **COVID-19 cases**, and **Total Deaths** to **COVID-19 Death cases** in our `COVID19data_df` that is shown below:

COVID19data_df

	Country	Total Cases	Total Deaths	Total Recovered	Active Cases	Population
0	Burundi	673	1	575	97	12029114
1	Ethiopia	107109	1664	66574	38871	116082175
2	Kenya	79322	1417	52974	24931	54237961
3	Rwanda	5750	47	5241	462	13078334
4	Somalia	4445	113	3412	920	16066789
5	Tanzania	509	21	183	305	60399786
6	Uganda	18406	186	8764	9456	46304101
7	Sudan	16431	1202	9854	5375	44252054

```
COVID19data_df.rename(columns={
    "Total Cases": "COVID-19 Cases",
    "Total Deaths": "COVID-19 Death Cases",
})
```

	Country	COVID-19 Cases	COVID-19 Death Cases	Total Recovered	Active Cases	Population
0	Burundi	673	1	575	97	12029114
1	Ethiopia	107109	1664	66574	38871	116082175
2	Kenya	79322	1417	52974	24931	54237961
3	Rwanda	5750	47	5241	462	13078334
4	Somalia	4445	113	3412	920	16066789
5	Tanzania	509	21	183	305	60399786
6	Uganda	18406	186	8764	9456	46304101
7	Sudan	16431	1202	9854	5375	44252054

As you can see, the `.rename()` attribute takes the `columns` as the input in which we pass the dictionary containing the old and new column names that we want to rename in the DataFrame. The result is not saved to any variable until we save it. This happens because Pandas wants to show us whether our output or result is correct and we can amend or save automatically with the `inplace` argument as shown below:

```
COVID19data_df.rename(columns={
    "Total Cases": "COVID-19 Cases",
    "Total Deaths": "COVID-19 Death Cases"
}, inplace = True)
```

We can see the result by calling out `COVID19data_df`

COVID19data_df

	Country	COVID-19 Cases	COVID-19 Death Cases	Total Recovered	Active Cases	Population
0	Burundi	673	1	575	97	12029114
1	Ethiopia	107109	1664	66574	38871	116082175
2	Kenya	79322	1417	52974	24931	54237961
3	Rwanda	5750	47	5241	462	13078334
4	Somalia	4445	113	3412	920	16066789
5	Tanzania	509	21	183	305	60399786
6	Uganda	18406	186	8764	9456	46304101
7	Sudan	16431	1202	9854	5375	44252054

Example 1

Using students registration dataframe, rename `Gender` to `Sex`

registrations_df

	Country	Gender	Age
0	Uganda	Male	11
1	Ethiopia	Female	17
2	Rwanda	Female	15
3	Somalia	Female	17
4	Sudan	Male	13
5	Kenya	Female	11
6	Uganda	Male	9
7	Kenya	Female	18
8	Somalia	Male	13
9	Rwanda	Female	16

```
registrations_df.rename(columns={"Gender": "Sex"})
```

	Country	Sex	Age
0	Uganda	Male	11
1	Ethiopia	Female	17
2	Rwanda	Female	15
3	Somalia	Female	17
4	Sudan	Male	13
5	Kenya	Female	11
6	Uganda	Male	9
7	Kenya	Female	18
8	Somalia	Male	13
9	Rwanda	Female	16

6.2.5 Adding a Column to Your DataFrame

We can create a new column called **Fatality rate** from our COVID-19 data.

$$\text{Fatality rate} = \frac{\text{COVID-19 Death Cases}}{\text{COVID-19 Cases}}$$

Fatality rate is the proportion of deaths from a certain disease compared to the total number of people diagnosed with the disease for a particular period.

Let's see the quick overview of `COVID19data_df` DataFrame

```
COVID19data_df.head()
```

	Country	COVID-19 Cases	COVID-19 Death Cases	Total Recovered	Active Cases	Population
0	Burundi	673	1	575	97	12029114
1	Ethiopia	107109	1664	66574	38871	116082175
2	Kenya	79322	1417	52974	24931	54237961
3	Rwanda	5750	47	5241	462	13078334
4	Somalia	4445	113	3412	920	16066789

We want to calculate the fatality rate of COVID-19 for each of the countries. Do you remember how to get or extract a series from a DataFrame?



`df.column_name` or `df["column_name"]`.

```
COVID19data_df["Fatality rate"] = COVID19data_df["COVID-19 Death Cases"] / COVID19data_df["COVID-19 Cases"]
```

`COVID19data_df["Fatality rate"]` is our new series or column while `COVID19data_df["COVID-19 Death Cases"]` and `COVID19data_df["COVID-19 Cases"]` are the existing series that we want to use to get the new column. The resulting DataFrame is shown below:

```
COVID19data_df
```

	Country	COVID-19 Cases	COVID-19 Death Cases	Total Recovered	Active Cases	Population	Fatality rate
0	Burundi	673	1	575	97	12029114	0.001486
1	Ethiopia	107109	1664	66574	38871	116082175	0.015536
2	Kenya	79322	1417	52974	24931	54237961	0.017864
3	Rwanda	5750	47	5241	462	13078334	0.008174
4	Somalia	4445	113	3412	920	16066789	0.025422
5	Tanzania	509	21	183	305	60399786	0.041257
6	Uganda	18406	186	8764	9456	46304101	0.010105
7	Sudan	16431	1202	9854	5375	44252054	0.073154

Example 1

Create a new column called **Discharge rate** from the COVID-19 DataFrame.

$$\text{Discharge rate} = \frac{\text{Total Recovered}}{\text{COVID-19 Cases}}$$

Discharge rate is the proportion of people that recovered from a certain disease compared to the total number of people diagnosed with the disease for a particular period.

```
COVID19data_df["Discharge rate"] = COVID19data_df["Total Recovered"]/COVID19data_df["COVID-19 Cases"]
```

and the resulting DataFrame is:

	Country	COVID-19 Cases	COVID-19 Death Cases	Total Recovered	Active Cases	Population	Fatality rate	Discharge rate
0	Burundi	673	1	575	97	12029114	0.001486	0.854383
1	Ethiopia	107109	1664	66574	38871	116082175	0.015536	0.621554
2	Kenya	79322	1417	52974	24931	54237961	0.017864	0.667835
3	Rwanda	5750	47	5241	462	13078334	0.008174	0.911478
4	Somalia	4445	113	3412	920	16066789	0.025422	0.767604
5	Tanzania	509	21	183	305	60399786	0.041257	0.359528
6	Uganda	18406	186	8764	9456	46304101	0.010105	0.476149
7	Sudan	16431	1202	9854	5375	44252054	0.073154	0.599720

6.2.6 Dropping/Removing Columns from a DataFrame

You can drop or remove a column from an existing DataFrame by using `.drop()` attribute. In this section, we use DataFrames namely `registrations_df` and `COVID19data_df`

```
registrations_df.head()
```

	Country	Gender	Age
0	Uganda	Male	11
1	Ethiopia	Female	17
2	Rwanda	Female	15
3	Somalia	Female	17
4	Sudan	Male	13

Example 1

Let's drop the `Country` column from the `registrations_df`

```
registrations_df.drop("Country", axis = "columns")
```

	Gender	Age
0	Male	11
1	Female	17
2	Female	15
3	Female	17
4	Male	13
5	Female	11
6	Male	9
7	Female	18
8	Male	13
9	Female	16

As we can see from the resulting DataFrame that the column `Country` has been dropped. We specified axis to be “columns” to show that what we want to drop is from the columns. The `axis` argument takes two values; 1 (or “columns”) to drop columns and 0 (or “rows”) to drop index.

We can save the resulting DataFrame as a new DataFrame `registrations_df_drop` or use `inplace = True` to replace the original DataFrame

```
registrations_df_drop = registrations_df.drop("Country", axis = "columns")
```

```
registrations_df_drop
```

	Gender	Age
0	Male	11
1	Female	17
2	Female	15
3	Female	17
4	Male	13
5	Female	11
6	Male	9
7	Female	18
8	Male	13
9	Female	16

Example 2

Let's drop the `Active Cases` and `Population` columns from the `COVID19data_df` DataFrame

```
COVID19data_df
```

	Country	COVID-19 Cases	COVID-19 Death Cases	Total Recovered	Active Cases	Population	Fatality rate	Discharge rate
0	Burundi	673	1	575	97	12029114	0.001486	0.854383
1	Ethiopia	107109	1664	66574	38871	116082175	0.015536	0.621554
2	Kenya	79322	1417	52974	24931	54237961	0.017864	0.667835
3	Rwanda	5750	47	5241	462	13078334	0.008174	0.911478
4	Somalia	4445	113	3412	920	16066789	0.025422	0.767604
5	Tanzania	509	21	183	305	60399786	0.041257	0.359528
6	Uganda	18406	186	8764	9456	46304101	0.010105	0.476149
7	Sudan	16431	1202	9854	5375	44252054	0.073154	0.599720

We can pass the list containing the names of the columns to drop in the `.drop()` attribute. For example:

remember to remove column(s) we can specify axis = 1 or "columns"

```
COVID19data_df.drop(["Active Cases", "Population"], axis = 1)
```

	Country	COVID-19 Cases	COVID-19 Death Cases	Total Recovered	Fatality rate	Discharge rate
0	Burundi	673	1	575	0.001486	0.854383
1	Ethiopia	107109	1664	66574	0.015536	0.621554
2	Kenya	79322	1417	52974	0.017864	0.667835
3	Rwanda	5750	47	5241	0.008174	0.911478
4	Somalia	4445	113	3412	0.025422	0.767604
5	Tanzania	509	21	183	0.041257	0.359528
6	Uganda	18406	186	8764	0.010105	0.476149
7	Sudan	16431	1202	9854	0.073154	0.599720

Note: It is important to save the resulting DataFrame.

```
COVID19data_df_drop = COVID19data_df.drop(["Active Cases", "Population"], axis = 1)
```

and if we use `COVID19data_df.drop(["Active Cases", "Population"], axis = 1, inplace = True)`, that will replace the full DataFrame to a DataFrame where columns “Active Cases” and “Population” has been removed or dropped. That may be dangerous because you have destroyed the original DataFrame with the `inplace = True`.

6.2.7 Conditional filtering of a DataFrame

You can filter a DataFrame based on a condition of one or more Series in the DataFrame.

Example 1

In our `registrations_df` DataFrame, we can show a DataFrame where only **Female** are available:

```
registrations_df
```

	Country	Gender	Age
0	Uganda	Male	11
1	Ethiopia	Female	17
2	Rwanda	Female	15
3	Somalia	Female	17
4	Sudan	Male	13
5	Kenya	Female	11
6	Uganda	Male	9
7	Kenya	Female	18
8	Somalia	Male	13
9	Rwanda	Female	16

Step 1:

Select **Gender** Series from the DataFrame

```
step1 = registrations_df["Gender"]
step1
```

```
0    Male
1  Female
2  Female
3  Female
4    Male
5  Female
6    Male
7  Female
8    Male
9  Female
Name: Gender, dtype: object
```

Step 2:

Use comparison `==` operator to distinguish whether they are “**Female**” or not.

```
step2 = registrations_df["Gender"] == "Female"
step2
```

```
0    False
1     True
2     True
3     True
4    False
5     True
6    False
7     True
8    False
9     True
Name: Gender, dtype: bool
```

Step 3:

Use the result of step 2 to select/filter the `registrations_df` DataFrame

```
registrations_df[step2]
```

	Country	Gender	Age
1	Ethiopia	Female	17
2	Rwanda	Female	15
3	Somalia	Female	17
5	Kenya	Female	11
7	Kenya	Female	18
9	Rwanda	Female	16

We do step 1 to 3 in one line of code as shown below:

```
registrations_df[registrations_df["Gender"] == "Female"]
```

	Country	Gender	Age
1	Ethiopia	Female	17
2	Rwanda	Female	15
3	Somalia	Female	17
5	Kenya	Female	11
7	Kenya	Female	18
9	Rwanda	Female	16

Example 2

In the `COVID19data_df` DataFrame, show a DataFrame where:

COVID-19 Cases is less than 3000

- The resulting DataFrame has how many rows?

COVID-19 Death Cases is less than 50 and Total Recovered is 200

- The resulting DataFrame has how many rows?

`COVID19data_df`

	Country	COVID-19 Cases	COVID-19 Death Cases	Total Recovered	Active Cases	Population	Fatality rate	Discharge rate
0	Burundi	673	1	575	97	12029114	0.001486	0.854383
1	Ethiopia	107109	1664	66574	38871	116082175	0.015536	0.621554
2	Kenya	79322	1417	52974	24931	54237961	0.017864	0.667835
3	Rwanda	5750	47	5241	462	13078334	0.008174	0.911478
4	Somalia	4445	113	3412	920	16066789	0.025422	0.767604
5	Tanzania	509	21	183	305	60399786	0.041257	0.359528
6	Uganda	18406	186	8764	9456	46304101	0.010105	0.476149
7	Sudan	16431	1202	9854	5375	44252054	0.073154	0.599720

Solution 1

COVID-19 Cases is less than 3000

- The resulting DataFrame has how many rows?

`COVID19data_df_sol1 = COVID19data_df[COVID19data_df["COVID-19 Cases"] < 3000]`

`COVID19data_df_sol1`

	Country	COVID-19 Cases	COVID-19 Death Cases	Total Recovered	Active Cases	Population	Fatality rate	Discharge rate
0	Burundi	673	1	575	97	12029114	0.001486	0.854383
5	Tanzania	509	21	183	305	60399786	0.041257	0.359528

`COVID19data_df_sol1.shape`

(2, 8)

There are two rows in this DataFrame

Solution 2

COVID-19 Death Cases is less than 50 and Total Recovered is less than 200

- The resulting DataFrame has how many rows?

As we can see from the question that the condition is based on two Series/columns:

- COVID-19 Death Cases is less than 50
- Total Recovered is less than 200

```
COVID19data_df_sol2 = COVID19data_df[(COVID19data_df["COVID-19 Death Cases"] < 50) & (COVID19data_df["Total Recovered"] < 200)]
COVID19data_df_sol2
```

	Country	COVID-19 Cases	COVID-19 Death Cases	Total Recovered	Active Cases	Population	Fatality rate	Discharge rate
5	Tanzania	509	21	183	305	60399786	0.041257	0.359528

```
COVID19data_df_sol2.shape
```

(1, 8)

The resulting DataFrame has just one row.

Review of DataFrame

We have learnt Pandas Series and DataFrame so far and based on what you have seen and understood about Series, DataFrame, and their various attribute or methods.



Source: International Trade Center (ITR)

In the figure above, a girl was carrying many flowers from the National Flower of Ethiopia. Prof. Francisca Oladipo measured the length and width of different species of flowers and dataset shown below were generated.

sepal_length	sepal_width	petal_length	petal_width	species
5.1	3.5	1.4	0.2	Rose
5	3.6	1.4	0.2	Rose
5.4	3.9	1.7	0.4	Rose
5.5	2.3	4	1.3	Lily
6.5	2.8	4.6	1.5	Lily
6.7	3.3	5.7	2.1	Carnation
7.2	3.2	6	1.8	Carnation
6.2	2.8	4.8	1.8	Carnation

1. Create a DataFrame using the sample of the dataset from the National Flower of Ethiopia (This was adapted from Iris data)
2. How many rows and columns are in the dataset?
3. How many series are in the DataFrame?

4. Rename **species** variable to **flower_categories**
5. Add **sepal_length** and **sepal_width** together and call the resulting series **sepal_length_width**
6. Drop the column **petal_width** from the DataFrame
7. From 6, select a DataFrame where **sepal_length** is greater than 5.5 and name it **perfect_data**

Solution 1

Create a DataFrame using the sample of dataset from the National Flower of Ethiopia

First, we need to import necessary libraries such as numpy, pandas, etc.

```
import numpy as np
import pandas as pd
```

To create a DataFrame, we need to first create a **dict** (dictionary) and then pass the result to DataFrame constructor

```
flower_data_dict = {
    "sepal_length": [5.1, 5, 5.4, 5.5, 6.5, 6.7, 7.2, 6.2],
    "sepal_width": [3.5, 3.6, 3.9, 2.3, 2.8, 3.3, 3.2, 2.8],
    "petal_length": [1.4, 1.4, 1.7, 4, 4.6, 5.7, 6, 4.8],
    "petal_width": [0.2, 0.2, 0.4, 1.3, 1.5, 2.1, 1.8, 1.8],
    "species": ["Rose", "Rose", "Rose", "Lily", "Lily", "Carnation", "Carnation", "Carnation"]
}
flower_data_dict
```

```
{'sepal_length': [5.1, 5, 5.4, 5.5, 6.5, 6.7, 7.2, 6.2], 'sepal_width': [3.5, 3.6, 3.9, 2.3, 2.8, 3.3, 3.2, 2.8], 'petal_length': [1.4, 1.4, 1.7, 4, 4.6, 5.7, 6, 4.8], 'petal_width': [0.2, 0.2, 0.4, 1.3, 1.5, 2.1, 1.8, 1.8], 'species': ['Rose', 'Rose', 'Rose', 'Lily', 'Lily', 'Carnation', 'Carnation', 'Carnation']}
```

As you know, this is a type **dict**

```
type(flower_data_dict)
```

```
<class 'dict'>
```

We now pass the result to **pd.DataFrame()**

```
flower_data_df = pd.DataFrame(flower_data_dict)
```

```
flower_data_df
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Rose
1	5.0	3.6	1.4	0.2	Rose
2	5.4	3.9	1.7	0.4	Rose
3	5.5	2.3	4.0	1.3	Lily
4	6.5	2.8	4.6	1.5	Lily
5	6.7	3.3	5.7	2.1	Carnation
6	7.2	3.2	6.0	1.8	Carnation
7	6.2	2.8	4.8	1.8	Carnation

As you can see, the steps to creating a DataFrame are just two:

1. Create a dictionary for the data
2. Pass the result to `pd.DataFrame()` function

Solution 2

How many rows and columns are in the dataset?

You can get the number of rows and columns in your DataFrame by using `.shape` attribute

```
flower_data_df.shape()
```

This throws an error **TypeError: 'tuple' object is not callable** because, `.shape` attribute does not have a parenthesis. Please take note of that!

```
flower_data_df.shape
```

(8, 5)

The DataFrame/dataset has 8 rows and 5 columns

Solution 3

How many series are in the DataFrame?

Series of a DataFrame is the same thing as columns or variable in that DataFrame. So therefore, we have 5 series in this DataFrame

Solution 4

rename **species** variable to **flower_categories**

To rename **species** variable to **flower_categories** we shall use **.rename()** attribute and then pass **dict** (dictionary) to it.

```
flower_data_df.rename(columns={
    "species": "flower_categories"
})
```

	sepal_length	sepal_width	petal_length	petal_width	flower_categories
0	5.1	3.5	1.4	0.2	Rose
1	5.0	3.6	1.4	0.2	Rose
2	5.4	3.9	1.7	0.4	Rose
3	5.5	2.3	4.0	1.3	Lily
4	6.5	2.8	4.6	1.5	Lily
5	6.7	3.3	5.7	2.1	Carnation
6	7.2	3.2	6.0	1.8	Carnation
7	6.2	2.8	4.8	1.8	Carnation

Solution 5

Add **sepal_length** and **sepal_width** together and call the resulting series **sepal_length_width**

To create **sepal_length_width** series, add the series of **sepal_length** and **sepal_width** together.

```
flower_data_df["sepal_length_width"] = flower_data_df.sepal_length + flower_data_df.sepal_width
flower_data_df
```

	sepal_length	sepal_width	petal_length	petal_width	species	sepal_length_width
0	5.1	3.5	1.4	0.2	Rose	8.6
1	5.0	3.6	1.4	0.2	Rose	8.6
2	5.4	3.9	1.7	0.4	Rose	9.3
3	5.5	2.3	4.0	1.3	Lily	7.8
4	6.5	2.8	4.6	1.5	Lily	9.3
5	6.7	3.3	5.7	2.1	Carnation	10.0
6	7.2	3.2	6.0	1.8	Carnation	10.4
7	6.2	2.8	4.8	1.8	Carnation	9.0

Solution 6

Drop the column **petal_width** from the DataFrame

```
flower_data_df.drop(["petal_width"])
```

```

KeyError                                Traceback (most recent call last)
<ipython-input-15-00097ae73ae6> in <module>
----> 1 flower_data_df.drop(["petal_width"])

C:\Users\defaultuser0\Anaconda3\lib\site-packages\pandas\core\frame.py in drop(self, labels, axis, index, columns, level, inplace, errors)
   4172         level=level,
   4173         inplace=inplace,
-> 4174         errors=errors,
   4175     )
   4176

C:\Users\defaultuser0\Anaconda3\lib\site-packages\pandas\core\generic.py in drop(self, labels, axis, index, columns, level, inplace, errors)
   3885         for axis, labels in axes.items():
   3886             if labels is not None:
-> 3887                 obj = obj._drop_axis(labels, axis, level=level, errors=errors)
   3888
   3889             if inplace:

C:\Users\defaultuser0\Anaconda3\lib\site-packages\pandas\core\generic.py in _drop_axis(self, labels, axis, level, errors)
   3919         new_axis = axis.drop(labels, level=level, errors=errors)
   3920     else:
-> 3921         new_axis = axis.drop(labels, errors=errors)
   3922         result = self.reindex(**{axis.name: new_axis})
   3923

C:\Users\defaultuser0\Anaconda3\lib\site-packages\pandas\core\indexes\base.py in drop(self, labels, errors)
   5282         if mask.any():
   5283             if errors != "ignore":
-> 5284                 raise KeyError(f"{labels[mask]} not found in axis")
   5285             indexer = indexer[~mask]
C:\Users\defaultuser0\Anaconda3\lib\site-packages\pandas\core\indexes\base.py in drop(self, labels, errors)
   5282         if mask.any():
   5283             if errors != "ignore":
-> 5284                 raise KeyError(f"{labels[mask]} not found in axis")
   5285             indexer = indexer[~mask]
   5286         return self.delete(indexer)

KeyError: "[petal_width] not found in axis"
```

This threw an error because axis argument has not been specified.

```
flower_data_df.drop(["petal_width"], axis = "columns")
```

	sepal_length	sepal_width	petal_length	species	sepal_length_width
0	5.1	3.5	1.4	Rose	8.6
1	5.0	3.6	1.4	Rose	8.6
2	5.4	3.9	1.7	Rose	9.3
3	5.5	2.3	4.0	Lily	7.8
4	6.5	2.8	4.6	Lily	9.3
5	6.7	3.3	5.7	Carnation	10.0
6	7.2	3.2	6.0	Carnation	10.4
7	6.2	2.8	4.8	Carnation	9.0

`petal_width` has been removed from the columns. Let's save the result as `flower_data_df_drop`

```
flower_data_df_drop = flower_data_df.drop(["petal_width"], axis = "columns")
```

```
flower_data_df_drop
```

	sepal_length	sepal_width	petal_length	species	sepal_length_width
0	5.1	3.5	1.4	Rose	8.6
1	5.0	3.6	1.4	Rose	8.6
2	5.4	3.9	1.7	Rose	9.3
3	5.5	2.3	4.0	Lily	7.8
4	6.5	2.8	4.6	Lily	9.3
5	6.7	3.3	5.7	Carnation	10.0
6	7.2	3.2	6.0	Carnation	10.4
7	6.2	2.8	4.8	Carnation	9.0

Solution 7

From solution 6, select a DataFrame where `sepal_length` is greater than 5.5 and name it `perfect_data`

We shall use our new DataFrame `flower_data_df_drop` to filter where `sepal_length` is greater than 5.5

```
perfect_data = flower_data_df_drop[flower_data_df_drop["sepal_length"] > 5.5]
```

perfect_data

	sepal_length	sepal_width	petal_length	species	sepal_length_width
4	6.5	2.8	4.6	Lily	9.3
5	6.7	3.3	5.7	Carnation	10.0
6	7.2	3.2	6.0	Carnation	10.4
7	6.2	2.8	4.8	Carnation	9.0

6.3 Import data with Pandas

As you can see from various examples, it is so tedious to start creating DataFrame from scratch. In the data science world, data will be available for you on a spreadsheet such as **MS-Excel**. Our job as a data scientist is to import those datasets using **Pandas**. Pandas can import different format of data such as **.xlsx**(Excel data), **.csv**(comma separated value), **.sav**(SPSS data), **.dat**(STATA data), or any other data online (the web.)

How to read in data

It's quite simple to load data from various file formats into a DataFrame. In this section, you will learn how to import different file format with **Pandas**.

File format	Pandas Function	Description
.csv	.read_csv()	Load delimited data from a file, URL, or file-like object; use comma as default delimiter
.xlsx or .xls (old Excel file format)	.read_excel()	Read tabular data from an Excel XLSX or XLS file
.json	.read_json()	Read data from a JSON (JavaScript Object Notation) string representation
.html	.read_html()	Read all tables found in the given HTML document
.sas	read_sas	Read a Statistical Analysis Software (SAS) dataset stored in one of the SAS system's custom storage formats
.dta	read_stata	Read a dataset from Stata file format
.sav	read_sav	Read a dataset from Statistical Package for Social Sciences (SPSS) file format

Current working directory

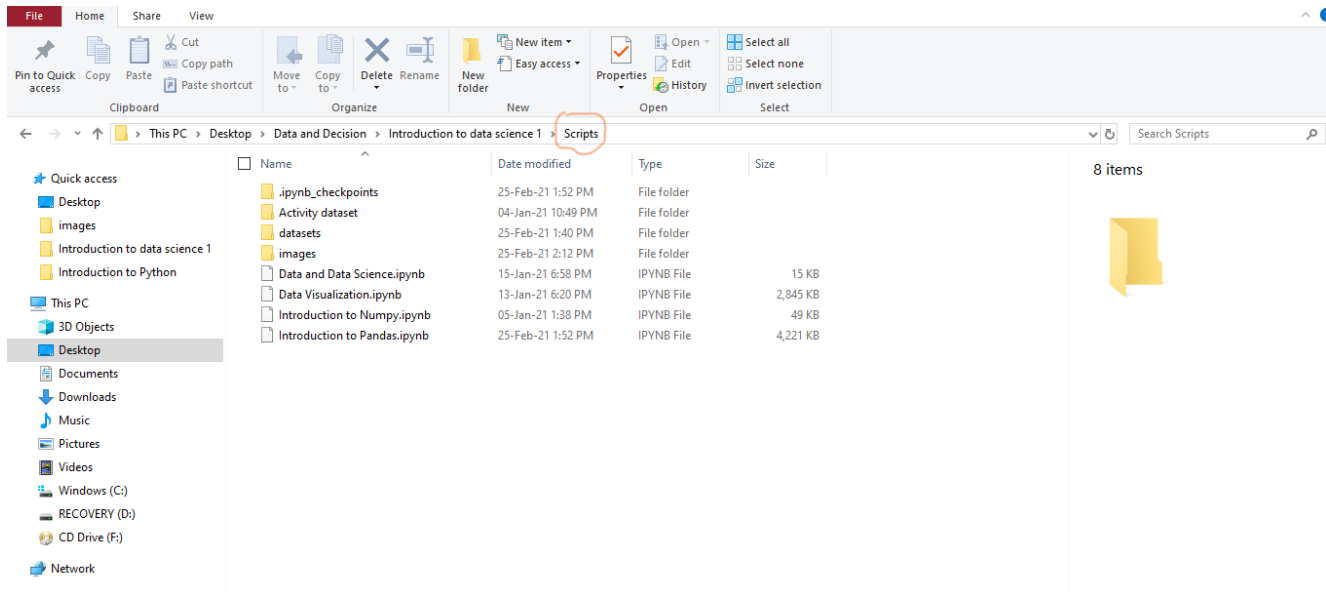
Your data must be in the same working directory you are currently working in. Your current working directory is where Python is running. For example, **pwd** (print working directory) shows you your working directory.

```
pwd
```

```
C:\\Users\\OGUNDEPO EZEKIEL .A\\Desktop\\Data and Decision\\Introduction to data science
1\\Scripts
```



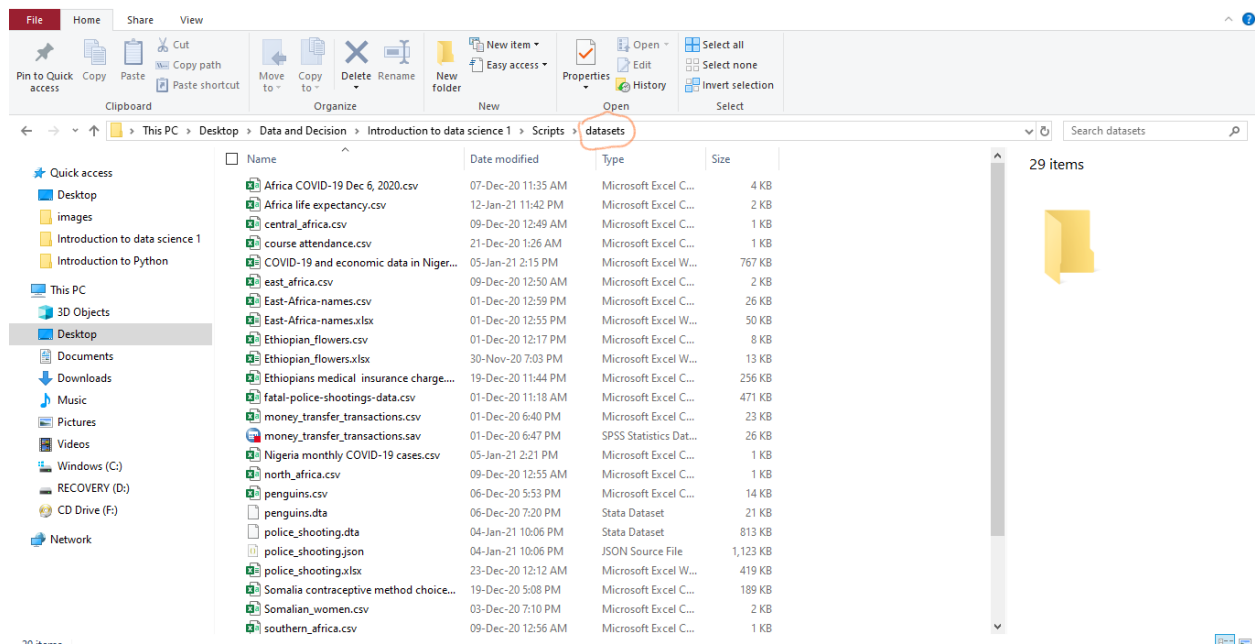
Note that my working directory will be different from your working directory. In fact, individual working directory will always be different.



As you can see, my current working directory is `C:\\Users\\OGUNDEPO EZEKIEL\\.A\\Desktop\\Data and Decision\\Introduction to data science 1\\Scripts`

You can also change the current working directory by using `cd` (change directory) command by specify the destination path. It can be absolute or relative path. For example, let's change the working directory to the folder that hosts our datasets by using:

```
cd datasets
```



Let's confirm our current working directory now with **pwd** command

```
pwd
```

As you can see, I have changed my working directory from `C:\Users\OGUNDEPO EZEKIEL .A\Desktop\Introduction to Data Science` to `C:\Users\OGUNDEPO EZEKIEL .A\Desktop\Introduction to Data Science\datasets`

As you can see, I have changed my working directory from `C:\\Users\\OGUNDEPO EZEKIEL .A\\Desktop\\Data and Decision\\Introduction to data science 1\\Scripts`` to ``C:\\Users\\OGUNDEPO EZEKIEL .A\\Desktop\\Data and Decision\\Introduction to data science 1\\Scripts\\datasets`

Relative and absolute path

Alternatively, we can work with a relative path. For example, using ``datasets/`` instead of absolute path or full path like ``C:/Users/OGUNDEPO EZEKIEL .A/Desktop/Data and Decision/Introduction to data science 1/Scripts/datasets``.

An absolute path specifies the location of a file or directory from the root directory. In other words, an absolute path is a complete path from start of actual file system. By contrast, a relative path

starts from some given working directory, avoiding the need to provide the full absolute path. For example, ``datasets/Ethiopian_flowers.csv`` is a relative path.



`datasets/Ethiopian_flowers.csv` is a relative path while `C:/Users/OGUNDEPO EZEKIEL.A/Desktop/Data and Decision/Introduction to data science 1/Scripts/datasets/ Ethiopian_flowers.csv` is an absolute path

Import data from CSV

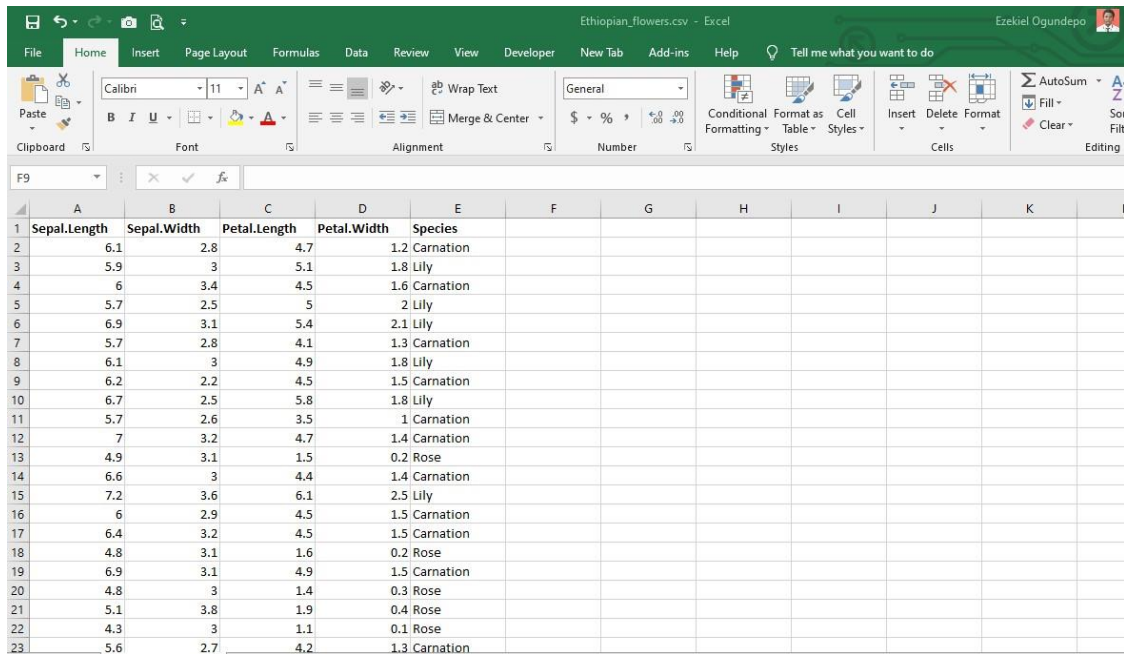
With `.csv` (comma-separated value) format all you need is `pd.read_csv()` command to load in the data as a DataFrame.

Example 1

Load the Ethiopian's flowers dataset (`Ethiopian_flowers.csv`) that was adapted from the well-known Iris dataset. Remember the first thing is to import all the necessary libraries such as `numpy`, `pandas`, etc.



The data is in the datasets folder or directory.



	A	B	C	D	E	F	G	H	I	J	K	L
	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species							
1												
2	6.1	2.8	4.7	1.2	Carnation							
3	5.9	3	5.1	1.8	Lily							
4	6	3.4	4.5	1.6	Carnation							
5	5.7	2.5	5	2	Lily							
6	6.9	3.1	5.4	2.1	Lily							
7	5.7	2.8	4.1	1.3	Carnation							
8	6.1	3	4.9	1.8	Lily							
9	6.2	2.2	4.5	1.5	Carnation							
10	6.7	2.5	5.8	1.8	Lily							
11	5.7	2.6	3.5	1	Carnation							
12	7	3.2	4.7	1.4	Carnation							
13	4.9	3.1	1.5	0.2	Rose							
14	6.6	3	4.4	1.4	Carnation							
15	7.2	3.6	6.1	2.5	Lily							
16	6	2.9	4.5	1.5	Carnation							
17	6.4	3.2	4.5	1.5	Carnation							
18	4.8	3.1	1.6	0.2	Rose							
19	6.9	3.1	4.9	1.5	Carnation							
20	4.8	3	1.4	0.3	Rose							
21	5.1	3.8	1.9	0.4	Rose							
22	4.3	3	1.1	0.1	Rose							
23	5.6	2.7	4.2	1.3	Carnation							

```
import numpy as np
import pandas as pd

ethiopian_flower = pd.read_csv("datasets/Ethiopian_flowers.csv")
```

You have imported Ethiopian flowers successfully! Bravo! Henceforth, when you import a dataset, the result is always a **DataFrame**.

```
type(ethiopian_flower)

<class 'pandas.core.frame.DataFrame'>
```

So therefore, all the pandas attributes that have been discussed so far can be applied.

Let's see what is in `ethiopian_flower` that we just imported. Functions such as `.head()`, `.tail()`, `.shape`, etc. are useful in exploring our dataset.

```
ethiopian_flower.head(10) # To see the first ten observations
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
0	6.1	2.8	4.7	1.2	Carnation
1	5.9	3.0	5.1	1.8	Lily
2	6.0	3.4	4.5	1.6	Carnation
3	5.7	2.5	5.0	2.0	Lily
4	6.9	3.1	5.4	2.1	Lily
5	5.7	2.8	4.1	1.3	Carnation
6	6.1	3.0	4.9	1.8	Lily
7	6.2	2.2	4.5	1.5	Carnation
8	6.7	2.5	5.8	1.8	Lily
9	5.7	2.6	3.5	1.0	Carnation

```
ethiopian_flower.tail(5) # To see the last five observations
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
345	7.2	3.0	5.8	1.6	Lily
346	6.7	3.1	4.7	1.5	Carnation
347	7.0	3.2	4.7	1.4	Carnation
348	5.6	3.0	4.5	1.5	Carnation
349	4.8	3.0	1.4	0.3	Rose

```
ethiopian_flower.shape
```

```
(350, 5)
```

Ethiopian flower dataset has 350 observations/rows with 5 variables/columns

```
ethiopian_flower.columns # list all the column names in the dataset
```

```
Index(['Sepal.Length', 'Sepal.Width', 'Petal.Length', 'Petal.Width', 'Species'], dtype='object')
```

The variables in the dataset are [Sepal.Length](#), [Sepal.Width](#), [Petal.Length](#), [Petal.Width](#), and [Species](#).

```
ethiopian_flower.info()
```

`.info()` attribute shows us that `ethiopian_flower` dataset has 4 variables are float datatypes and 1 variable that is object (or string) datatype.

With **.xlsx** or **xls** format, all you need is **pd.read_excel()** command to load in the data

Load the **East-Africa-names.xlsx** which was scraped from <https://www.behindthename.com/submit/names/usage/eastern-african>. Since you have imported all the necessary libraries such as numpy and pandas, you don't need to import them here again.

East-Africa-names.xlsx - Excel

Ezekiel Ogundepo

File Home Insert Page Layout Formulas Data Review View Developer New Tab Add-ins Help Design Tell me what you want to do

Clipboard Paste Font Alignment Number Conditional Formatting Styles Cell Styles Insert Delete Format Cells AutoSum Fill Clear Editing

A20 X ✓ fx Abikanile

	A	B	C	D	E	F	G	H	I	J	K	L
1	Name	Gender	Origin.1	Origin.2	Origin.3	Origin.4						
2	Aadaan	M	Somali									
3	Aadam	M	Urdu	Somali	Estonian							
4	Aaden	M	Somali									
5	Aamiina	F	Somali									
6	Abadir	M	Arabic	Bohairic	Sahidic							
7	Abaynesh	F	Amharic									
8	Abdi	M	Somali	Indonesian	Turkish							
9	Abdikadir	M	Somali									
10	Abdirahim	M	Somali									
11	Abdirahman	M	Somali									
12	Abdirizak	M	Somali									
13	Abdulkadir	M	Somali									
14	Abdullahi	M	Somali	Nigerian								
15	Abduwali	M	Uyghur	Somali								
16	Abebe	M	Amharic	Ethiopian								
17	Abebech	F	Amharic	Ethiopian								
18	Abera	M	Ethiopian									
19	Abigaili	F	Biblical	Swahili								
20	Abikanile	F	Yao									
21	Abiy	M	Amharic									
22	Aboyo	F	Luo									
23	Abreham	M	Ethiopian									

Sheet1 Sheet2 Sheet3

Excel Workbook can have many worksheets and each can host different data. While importing your data, you need to specify the index of the sheet that you want to import. Remember Python index starts from 0.

```
east_africa_names = pd.read_excel("datasets/East-Africa-names.xlsx", sheet_name = 0)
```

You have imported East Africa names successfully! You are now an expert in importing file!

Let's see various names in the `east_africa_names` data that we just imported.

```
east_africa_names.head(20)
```

	Name	Gender	Origin.1	Origin.2	Origin.3	Origin.4
0	Aadaan	M	Somali	NaN	NaN	NaN
1	Aadam	M	Urdu	Somali	Estonian	NaN
2	Aaden	M	Somali	NaN	NaN	NaN
3	Aamiina	F	Somali	NaN	NaN	NaN
4	Abadir	M	Arabic	Bohairic	Sahidic	NaN
5	Abaynesh	F	Amharic	NaN	NaN	NaN
6	Abdi	M	Somali	Indonesian	Turkish	NaN
7	Abdikadir	M	Somali	NaN	NaN	NaN
8	Abdirahim	M	Somali	NaN	NaN	NaN
9	Abdirahman	M	Somali	NaN	NaN	NaN
10	Abdirizak	M	Somali	NaN	NaN	NaN
11	Abdulkadir	M	Somali	NaN	NaN	NaN
12	Abdullahi	M	Somali	Nigerian	NaN	NaN
13	Abduwali	M	Uyghur	Somali	NaN	NaN
14	Abebe	M	Amharic	Ethiopian	NaN	NaN
15	Abebech	F	Amharic	Ethiopian	NaN	NaN
16	Abera	M	Ethiopian	NaN	NaN	NaN
17	Abigaili	F	Biblical	Swahili	NaN	NaN
18	Abikanile	F	Yao	NaN	NaN	NaN
19	Abiy	M	Amharic	NaN	NaN	NaN

Importing data from STATA

With `.dta` format, all you need is `pd.read_dta()` command to load in the data

Example 1

Load the [penguins.dta](#) data. The palmer penguins data contains size measurements for three penguin species observed on three islands in the Palmer Archipelago, Antarctica.



Data Editor (Edit) - [penguins.dta]

File Edit View Data Tools

species[11] Adelie

	species	island	bill_lengt~m	bill_depth~m	flipper_le~m	body_mass_g	sex	
1	Adelie	Torgersen	39.1	18.7	181	3750	male	
2	Adelie	Torgersen	39.5	17.4	186	3800	female	
3	Adelie	Torgersen	40.3	18	195	3250	female	
4	Adelie	Torgersen	36.7	19.3	193	3450	female	
5	Adelie	Torgersen	39.3	20.6	190	3650	male	
6	Adelie	Torgersen	38.9	17.8	181	3625	female	
7	Adelie	Torgersen	39.2	19.6	195	4675	male	
8	Adelie	Torgersen	34.1	18.1	193	3475	female	
9	Adelie	Torgersen	42	20.2	190	4250	male	
10	Adelie	Torgersen	37.8	17.1	186	3300	female	
11	Adelie	Torgersen	37.8	17.3	180	3700	male	
12	Adelie	Torgersen	41.1	17.6	182	3200	female	
13	Adelie	Torgersen	38.6	21.2	191	3800	male	
14	Adelie	Torgersen	34.6	21.1	198	4400	male	
15	Adelie	Torgersen	36.6	17.8	185	3700	female	
16	Adelie	Torgersen	38.7	19	195	3450	female	
17	Adelie	Torgersen	42.5	20.7	197	4500	male	
18	Adelie	Torgersen	34.4	18.4	184	3325	female	
19	Adelie	Torgersen	46	21.5	194	4200	male	
20	Adelie	Biscoe	37.8	18.3	174	3400	female	
21	Adelie	Biscoe	37.7	18.7	180	3600	male	
22	Adelie	Biscoe	35.9	19.2	189	3800	female	
23	Adelie	Biscoe	38.2	18.1	185	3950	male	
24	Adelie	Biscoe	38.8	17.2	180	3800	male	

Penguins from STATA

```
penguis_data = pd.read_stata("datasets/penguins.dta")
```

```
penguis_data.head()
```

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
0	Adelie	Torgersen	39.1	18.7	181	3750	male
1	Adelie	Torgersen	39.5	17.4	186	3800	female
2	Adelie	Torgersen	40.3	18.0	195	3250	female
3	Adelie	Torgersen	36.7	19.3	193	3450	female
4	Adelie	Torgersen	39.3	20.6	190	3650	male

```
penguis_data.tail()
```

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
332	Chinstrap	Dream	55.8	19.8	207	4000	male
333	Chinstrap	Dream	43.5	18.1	202	3400	female
334	Chinstrap	Dream	49.6	18.2	193	3775	male
335	Chinstrap	Dream	50.8	19.0	210	4100	male
336	Chinstrap	Dream	50.2	18.7	198	3775	female

```
penguins_data.shape
```

```
(337, 7)
```

Penguins data has 343 rows and 7 columns.

Import data from JSON

JSON stands for JavaScript Object Notation. With `.json` format, all you need is `pd.read_json()` command to load in the data

Example 1

Load the `Ugandan_women.json` data. This data set gives the average heights and weights for Ugandan women aged 30–39.

```
ugandan_women = pd.read_json("datasets/Ugandan_women.json")
```

```
ugandan_women.head()
```

	height	weight
0	63	129
1	69	150
2	71	159
3	68	146
4	58	115

```
ugandan_women.tail()
```

	height	weight
295	65	135
296	62	126
297	66	139
298	58	115
299	58	115

```
ugandan_women.shape
```

```
(300, 2)
```

The dataset consists of 300 rows and 2 columns

Import data from SPSS

We can also import data from the Statistical Package for the Social Sciences (SPSS) software with Pandas. Note, however, we need to install the **pyreadstat** first.

How to install Pyreadstat

Open up your Anaconda Prompt or Terminal and run

pip install pyreadstat

```
Anaconda Prompt
(base) C:\Users\OGUNDEPO EZEKIEL .A>pip install pyreadstat
```

Pandas depends on **pyreadstat** for reading **.sav** files. As always, we need to **import pandas as pd**.

Example 1

Load [money_transfer_transactions.sav](#) data. This data set is about money transaction in Uganda, Somalia, Kenya, Ethiopia South Sudan, Tanzania, and Rwanda.

money_transfer_transactions.sav [DataSet1] - IBM SPSS Statistics Data Editor

	customer_id	gender	age	direction	transfer_amount	currency	is_smartphone	is_urban	transaction_duration	dow
1	118	Female	35	Inflow	98	ETB	True	False	45.05	Friday
2	90	Female	27	Outflow	50	TZS	False	False	3.81	Friday
3	30	Female	52	Outflow	13	KES	True	True	44.51	Thursday
4	64	Male	20	Outflow	49	TZS	True	False	21.29	Sunday
5	28	Female	49	Outflow	52	RWF	True	True	11.51	Friday
6	115	Female	35	Outflow	99	RWF	True	True	84.56	Friday
7	68	Male	32	Outflow	73	SSP	False	False	1.45	Sunday
8	58	Male	38	Outflow	55	TZS	False	True	26.27	Thursday
9	43	Female	47	Outflow	16	UGX	True	True	77.11	Saturday
10	9	Male	20	Inflow	5	KES	True	False	83.68	Saturday
11	84	Male	19	Inflow	46	TZS	False	True	33.49	Saturday
12	27	Male	67	Inflow	41	ETB	False	True	13.85	Friday
13	36	Female	23	Inflow	29	ETB	False	False	18.38	Friday
14	11	Male	47	Outflow	41	SOS	False	True	43.56	Sunday
15	113	Female	54	Inflow	14	KES	False	False	75.46	Saturday
16	85	Female	50	Outflow	45	RWF	False	False	93.22	Saturday
17	78	Female	50	Outflow	57	TZS	True	False	31.09	Friday
18	3	Male	53	Outflow	4	RWF	False	False	47.94	Sunday
19	33	Male	68	Outflow	43	TZS	False	True	50.95	Sunday
20	80	Female	19	Inflow	54	ETB	False	False	6.60	Friday
21	79	Female	22	Outflow	55	ETB	False	True	28.42	Sunday
22	76	Male	34	Inflow	90	UGX	True	False	97.50	Sunday
23	117	Female	31	Inflow	54	SOS	True	True	12.56	Friday

Data View Variable View

```
import pandas as pd
```

```
money_transfer = pd.read_spss("datasets/money_transfer_transactions.sav")
```

```
money_transfer.head()
```

	customer_id	gender	age	direction	transfer_amount	currency	is_smartphone	is_urban	transaction_duration	dow
0	118.0	Female	35.0	Inflow	98.0	ETB	True	False	45.05	Friday
1	90.0	Female	27.0	Outflow	50.0	TZS	False	False	3.81	Friday
2	30.0	Female	52.0	Outflow	13.0	KES	True	True	44.51	Thursday
3	64.0	Male	20.0	Outflow	49.0	TZS	True	False	21.29	Sunday
4	28.0	Female	49.0	Outflow	52.0	RWF	True	True	11.51	Friday

```
money_transfer.tail()
```

	customer_id	gender	age	direction	transfer_amount	currency	is_smartphone	is_urban	transaction_duration	dow
495	80.0	Male	49.0	Inflow	56.0	SSP	True	False	33.02	Sunday
496	60.0	Female	40.0	Outflow	31.0	ETB	True	False	36.75	Sunday
497	32.0	Female	32.0	Outflow	74.0	KES	True	True	73.69	Thursday
498	28.0	Female	63.0	Outflow	43.0	SSP	True	True	24.02	Sunday
499	22.0	Male	48.0	Inflow	36.0	SOS	True	False	96.20	Saturday

```
money_transfer.shape
```

```
(500, 10)
```

Money transaction dataset has 500 observations/rows with 10 variables/columns.

6.3.1 Export data with Pandas

You can save DataFrame in Python to any file of your choice. Similar to the ways we read in data, pandas provide intuitive commands to save it:

```
df.to_csv('new_file.csv') # to save as csv file
```

```
df.to_excel('new_file.xlsx') # to save as Excel file
```

```
df.to_stata('new_file.dta') # to save as STATA file
```

```
df.to_spss('new_file.sav') # to save as SPSS file
```

```
df.to_json('new_file.json') # to save as JSON file
```

When we save as **.csv**, **.xlsx**, or any file formats, all we have to input into those functions is our desired filename with the appropriate file extension.

Example

Import **fatal-police-shootings-data.csv** data as **police_shooting** and export the resulting DataFrame as:

3. **.dta**

4. **.xlsx**

5. **.json**

Solution

```
import pandas as pd
```

```
police_shooting = pd.read_csv("datasets/fatal-police-shootings-data.csv")
```

```
police_shooting.head() # To see the first 5 observation
```

	date	manner_of_death	armed	age	gender	race	city	state	signs_of_mental_illness	threat_level	flee	body_camera	longitude	latitude
0	02-01-15	shot	gun	53.0	M	A	Shelton	WA	True	attack	Not fleeing	False	-123.122	47.247
1	02-01-15	shot	gun	47.0	M	W	Aloha	OR	False	attack	Not fleeing	False	-122.892	45.487
2	03-01-15	shot and Tasered	unarmed	23.0	M	H	Wichita	KS	False	other	Not fleeing	False	-97.281	37.695
3	04-01-15	shot	toy weapon	32.0	M	W	San Francisco	CA	True	attack	Not fleeing	False	-122.422	37.763
4	04-01-15	shot	nail gun	39.0	M	H	Evans	CO	False	attack	Not fleeing	False	-104.692	40.384

```
police_shooting.shape # To check the dimension of the dataset
```

```
(5793, 14)
```

```
police_shooting.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5793 entries, 0 to 5792
Data columns (total 14 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   date                                  5793 non-null   object
1   manner_of_death                      5793 non-null   object
2   armed                                5580 non-null   object
3   age                                  5547 non-null   float64
4   gender                              5792 non-null   object
5   race                                 5256 non-null   object
6   city                                 5793 non-null   object
7   state                                5793 non-null   object
8   signs_of_mental_illness              5793 non-null   bool
9   threat_level                        5793 non-null   object
10  flee                                 5486 non-null   object
11  body_camera                         5793 non-null   bool
12  longitude                           5510 non-null   float64
13  latitude                           5510 non-null   float64
dtypes: bool(2), float64(3), object(9)
memory usage: 554.5+ KB
```

To export as .dta (STATA file), we use:

```
police_shooting.to_stata("datasets/police_shooting.dta")
```

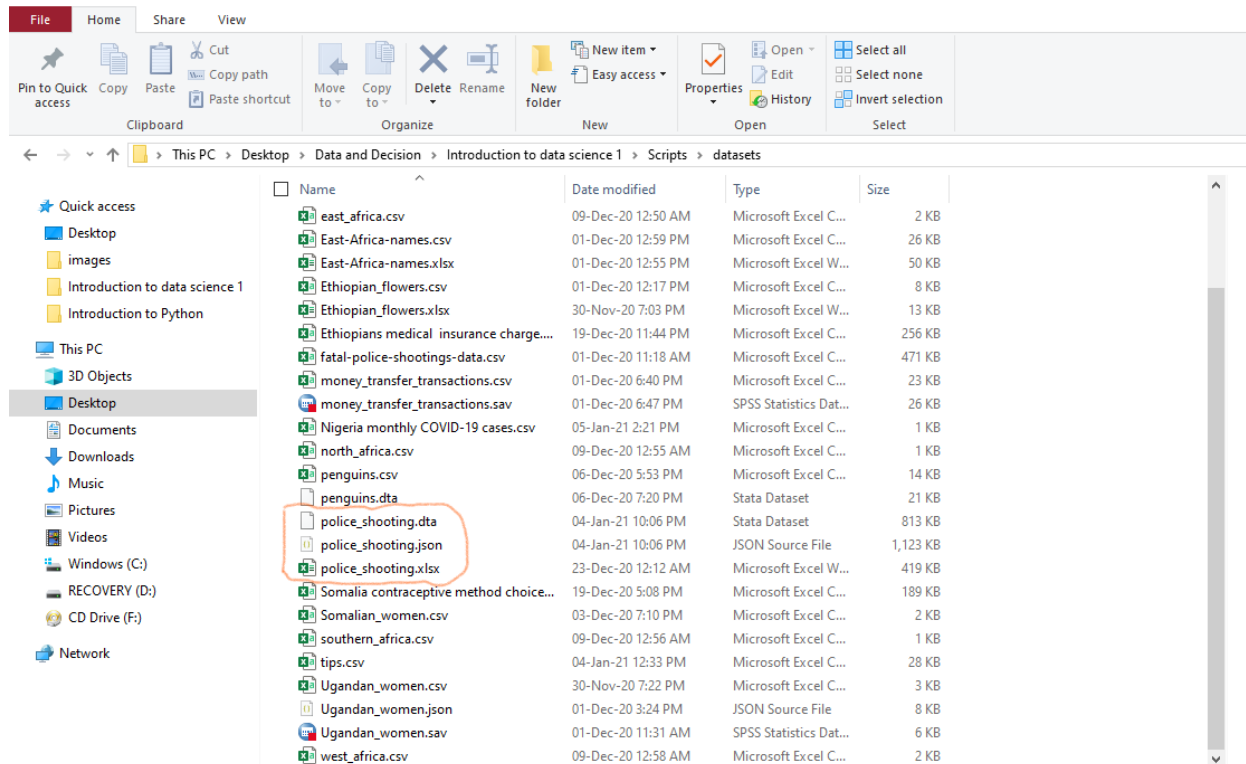
To export as .xlsx (Excel file), we use:

```
police_shooting.to_excel("datasets/police_shooting.xlsx")
```

To export as .json (JSON file), we use:

```
police_shooting.to_json("datasets/police_shooting.json")
```

We then check our working directory for all the data that we just exported.



6.4 Data cleaning and preprocessing

6.4.1 Working with missing data

In this section, we will discuss missing values also referred to as **NA**, **NaN** or **NULL** in pandas. When exploring data, you'll most likely encounter missing or null values, which symbolically represent non-existent values.

We can check whether there are missing values in one or more columns of a DataFrame by using `.isna()` or `.isnull()` while `.isna().sum()` or `.isnull().sum()` can be used to get the number of missing values in each of the columns of a DataFrame.

Example 1

Import `somalia_women.csv` dataset and count the number of missing values in its columns.

```
import numpy as np
import pandas as pd
```



```
somalia_women = pd.read_csv("datasets/Somalian_women.csv")
```

```
somalia_women.head()
```

	height	weight
0	NaN	64.0
1	63.0	58.0
2	70.0	83.0
3	61.0	NaN
4	64.0	70.0

Let's check whether there is a missing value by using `.isna()` or `.isnull()`.

```
somalia_women.isna()
```

	height	weight
0	True	False
1	False	False
2	False	False
3	False	True
4	False	False
...
245	False	False
246	False	False
247	False	True
248	False	False
249	False	False

250 rows × 2 columns

As you can see, there are some values that are missing in **height** and **weight** of `somalia_women` DataFrame. To get the number of missing values we use `.isna().sum()` or `.isnull().sum()`.

```
somalia_women.isna().sum()
```

```
height      4
```

```
weight      5
```

dtype: int64

height has 4 missing values while weight has 5 missing values.

Example 2

Import **police_shooting.xlsx** which is in the **datasets** directory and examine its missing values.

Since we have imported all the necessary libraries in Example 1, we just use it directly to import **policeshooting.xlsx** file.

```
police_shooting = pd.read_excel("datasets/police_shooting.xlsx")
```

We can then use **.isna().sum()** to check its missing values.

```
police_shooting.isna().sum()
```

```
Unnamed: 0          0
date               0
manner_of_death    0
armed             213
age              246
gender            1
race             537
city              0
state             0
signs_of_mental_illness  0
threat_level       0
flee             307
body_camera        0
longitude         283
latitude          283
dtype: int64
```

As you can see, there are missing values in **armed**, **age**, **gender**, **race**, **flee**, **longitude** and **latitude** columns

6.4.2 Data imputation

There are two options in dealing with missing values:

1. Get rid of rows or columns with nulls
2. Replace nulls with non-null values which is known as data imputation.

1. Get rid of rows or columns with nulls

Every data scientist or analyst regularly face the problem of whether to remove (drop) the missing values (NA, NULL, NAN) or impute them. This is a decision that requires intimate knowledge of your data and its context. Overall, removing null data is only suggested if you have a small amount of missing data. Remove nulls is pretty simple by using `.dropna()`.

Example 1

For example, let's remove the missing values in the Somalia women dataset. Remember in our **somalia_women** DataFrame, there are some missing values in both height and weight columns.

```
somalia_women.head()
```

	height	weight
0	NaN	64.0
1	63.0	58.0
2	70.0	83.0
3	61.0	NaN
4	64.0	70.0

Let's use `.shape` to check the number of rows and columns before removing any missing values.

```
somalia_women.shape
```

```
(250, 2)
```

```
somalia_women.isna().sum()
```

```
height    4
weight    5
dtype: int64
```

Now let's drop those missing values with `.dropna()` attribute. This action will delete any row with at least a single **NA**, **Null**, or **NAN** value but it will return a new DataFrame without altering the original one. You could specify `inplace=True` in this method as well.

```
somalia_women_dropna = somalia_women.dropna()
somalia_women_dropna
```

	height	weight
1	63.0	58.0
2	70.0	83.0
4	64.0	70.0
5	72.0	67.0
6	59.0	46.0
...
244	63.0	58.0
245	61.0	52.0
246	63.0	58.0
248	65.0	64.0
249	71.0	88.0

241 rows × 2 columns

```
somalia_women_dropna.shape
```

```
(241, 2)
```

```
somalia_women.shape
```

```
(250, 2)
```

You will notice that `dropna()` has reduced the number of rows from 250 to 241.

Other than just dropping rows, you can also drop columns with null values by setting `axis="columns"` or 1

Example 2

Import `East-Africa-names.csv` as `east_africa` and remove any column with missing values.

```
east_africa = pd.read_csv("datasets/East-Africa-names.csv", encoding = "ISO-8859-1")
```

Important note

Sometimes there can be `UnicodeDecodeError` when reading CSV file in Pandas with Python. You will notice that I used `encoding = "ISO-8859-1"` option to stop this error.

Class activity 8 (Peer to peer review activity)



Peer to Peer Interaction

Visit the LMS, locate forum activity and participate in the discussion

Try the code below and notice whether it throws an error or not

```
east_africa = pd.read_csv("datasets/East-Africa-names.csv")
```

Correct the error by using `encoding = "ISO-8859-1"` in the `pd.read_csv()` option.

Okay, let's continue our work!

```
east_africa.head()
```

	Name	Gender	Origin.1	Origin.2	Origin.3	Origin.4
0	Aadaan	M	Somali	NaN	NaN	NaN
1	Aadam	M	Urdu	Somali	Estonian	NaN
2	Aaden	M	Somali	NaN	NaN	NaN
3	Aamiina	F	Somali	NaN	NaN	NaN
4	Abadir	M	Arabic	Bohairic	Sahidic	NaN

```
east_africa.isna().sum()
```

```
Name      0
Gender     0
Origin.1   0
Origin.2   970
Origin.3  1093
Origin.4  1116
dtype: int64
```

There are some missing values in `Origin.2`, `Origin.3`, `Origin.4` of `east_Africa` DataFrame

Let's check the number of rows and columns before using `.dropna(axis = "columns")` on the `east_africa` DataFrame.

```
east_africa.shape
```

```
(1128, 6)
```

There are 1128 rows and 6 columns. Now let us remove columns with some missing values with `.dropna(axis = "columns")`.

```
east_africa_dropna = east_africa.dropna(axis = "columns")
```

```
east_africa_dropna
```

	Name	Gender	Origin.1
0	Aadaan	M	Somali
1	Aadam	M	Urdu
2	Aaden	M	Somali
3	Aamiina	F	Somali
4	Abadir	M	Arabic
...
1123	Tufani	M	Swahili
1124	Yordanos	M	Ethiopian
1125	Zaitun	M	Swahili
1126	Zamzam	M	Arabic
1127	Zula	F	Tigrinya

1128 rows × 3 columns

```
east_africa_dropna.shape
```

```
(1128, 3)
```

As you can see, `.dropna(axis = "columns")` has removed all columns with missing values.

2. Replace nulls with non-null values which is known as data imputation

Imputation is a conventional technique used to keep valuable data that have missing values. There may be instances where dropping every row with a missing value removes too big a chunk from your dataset, so instead we can impute the missing value with another value, usually the mean or the median of that column.

Example 1

let's fill the missing values in **height** and **weight** of **somalia_women** DataFrame with their means. The average or mean in each column will be used to replace any missing value found in that column.

```
somalia_women.head()
```

	height	weight
0	NaN	64.0
1	63.0	58.0
2	70.0	83.0
3	61.0	NaN
4	64.0	70.0

```
somalia_women.isna().sum()
```

```
height    4
weight    5
dtype: int64
```

Here's the mean value of **somalia_women** height:

```
somalia_women["height"].mean() # measured in kg
```

```
64.71544715447155
```

and here is the mean value of **somalia_women's** weight

```
somalia_women["weight"].mean() # measured in lbs
```

```
64.13469387755102
```

We could also use the code below to achieve the same thing:

```
somalia_women.mean()
```

```
height    64.715447
weight    64.134694
dtype: float64
```

The mean value of 64.71 shall be used to replace any **NaN** in height Series of `somalia_women` DataFrame while the mean value of 64.13 will also be used to replace any **NaN** in the weight Series.

We can achieve this by using `.fillna()` attribute

```
somalia_women_imput = somalia_women.fillna(somalia_women.mean())
```

```
somalia_women_imput
```

	height	weight
0	64.715447	64.000000
1	63.000000	58.000000
2	70.000000	83.000000
3	61.000000	64.134694
4	64.000000	70.000000
...
245	61.000000	52.000000
246	63.000000	58.000000
247	67.000000	64.134694
248	65.000000	64.000000
249	71.000000	88.000000

250 rows × 2 columns

```
somalia_women_imput.isna().sum()
```

```
height    0
weight    0
dtype: int64
```


As you can see, all the missing values have been replaced by their mean values.

6.4.3 Checking for duplicates in a DataFrame

In this discussion, you will learn how to find duplicate rows in a DataFrame based on all or a list of columns. For this we will use `.duplicated()` method on the DataFrame.

Syntax :

`DataFrame.duplicated(subset = None, keep = "first")`.

This results to Boolean Series denoting duplicate rows.

Parameters:

subset: This takes a column or list of column label. Its default value is None. After passing columns, it will consider them only for duplicates.

keep: This controls how to consider duplicate value. It has only three distinct value and default is **first**.

If **first**, this considers first value as unique and rest of the same values as duplicate.

If **last**, this considers last value as unique and rest of the same values as duplicate.

If **False**, this considers all of the same values as duplicates.

Let's import course attendance DataFrame from our datasets directory to see how we can remove duplicates from the DataFrame.

Load the Ethiopian's flowers dataset (.csv) that was adapted from the well-known Iris dataset. Remember the first thing is to import all the necessary libraries such as `numpy`, `pandas`, etc.

```
import numpy as np
import pandas as pd

course_attendance = pd.read_csv("datasets/course attendance.csv")
```

It is a small dataset so we can easily print it out

course_attendance

	Name	Gender	Age	Country	Course taken
0	Aadaan	M	16	Somalia	CS 1 2
1	Francis	M	18	Kenya	CS 1 3
2	Aamiina	F	20	Somalia	CS 1 5
3	Ibrahim	M	20	South Sudan	CS 1 3
4	Getu	M	16	Ethiopia	CS 2 2
5	Francisca	F	16	Uganda	CS 1 6
6	Simon	M	19	South Sudan	CS 2 2
7	Abiy	M	19	Ethiopia	CS 2 3
8	Abreham	M	20	Ethiopian	CS 1 5
9	Abshir	M	20	Somalia	CS 2 4
10	Jamal	M	18	South Sudan	CS 1 2
11	Allan	M	19	Kenya	CS 2 1
12	Mariam	F	20	Uganda	CS 3 1
13	Francisca	F	16	Uganda	CS 1 6
14	Aliya	F	18	Uganda	CS 3 1
15	Francis	M	18	Kenya	CS 1 3
16	Simon	M	19	South Sudan	CS 2 2

Example 1

Select duplicate rows based on all columns.

Here, we do not pass any argument therefore it takes default values for both the arguments i.e. **subset = None** and **keep = "first"**. Selecting duplicate rows except first occurrence based on all columns.

The code is `course_attendance[course_attendance.duplicated()]`. Let's break it one by one for your understanding.

Step 1: Check for duplicate rows using `.duplicated()` method. This results to Boolean Series (`True/False`) denoting duplicate rows.

```
course_attendance.duplicated()
```

```
0    False
1    False
2    False
3    False
4    False
5    False
6    False
7    False
8    False
9    False
10   False
11   False
12   False
13    True
14   False
15    True
16    True
dtype: bool
```

As you can see, we have some missing values in row 14, 16, and 17 respectively. Remember Python index starts with 0. To check for the number of duplicates use:

```
course_attendance.duplicated().sum()
```

3

Step 2: Use the result of step 1 to select the duplicate rows in the DataFrame i.e. `DataFrame[step1]`.

```
course_attendance[course_attendance.duplicated()]
```

	Name	Gender	Age	Country	Course taken
13	Francisca	F	16	Uganda	CS 1 6
15	Francis	M	18	Kenya	CS 1 3
16	Simon	M	19	South Sudan	CS 2 2

Example 2

Select duplicate rows based on all columns

If you want to consider all duplicates except the last one, then, pass `keep = "last"` as an argument.

	Name	Gender	Age	Country	Course taken
1	Francis	M	18	Kenya	CS 1 3
5	Francisca	F	16	Uganda	CS 1 6
6	Simon	M	19	South Sudan	CS 2 2

Example 3

Select duplicate rows based on some columns

If you want to select duplicate rows based only on some selected columns then pass the list of column names in subset as an argument.

```
course_attendance[course_attendance.duplicated("Name")]
```

	Name	Gender	Age	Country	Course taken
13	Francisca	F	16	Uganda	CS 1 6
15	Francis	M	18	Kenya	CS 1 3
16	Simon	M	19	South Sudan	CS 2 2

```
course_attendance[course_attendance.duplicated("Course taken")]
```

	Name	Gender	Age	Country	Course taken
3	Ibrahim	M	20	South Sudan	CS 1 3
6	Simon	M	19	South Sudan	CS 2 2
8	Abreham	M	20	Ethiopian	CS 1 5
10	Jamal	M	18	South Sudan	CS 1 2
13	Francisca	F	16	Uganda	CS 1 6
14	Aliya	F	18	Uganda	CS 3 1
15	Francis	M	18	Kenya	CS 1 3
16	Simon	M	19	South Sudan	CS 2 2

Example 4

Select duplicate rows based on some columns

Select duplicate rows based on more than one column names

```
course_attendance[course_attendance.duplicated(["Name", "Country"])]
```

	Name	Gender	Age	Country	Course taken
13	Francisca	F	16	Uganda	CS 1 6
15	Francis	M	18	Kenya	CS 1 3
16	Simon	M	19	South Sudan	CS 2 2

Class activity 9 (Peer to peer review activity)



Peer to Peer Interaction

Visit the LMS, locate forum activity and participate in the discussion

Import `penguins.csv` dataset as `penguins` and examine duplicate rows in the DataFrame

Note: Penguins dataset is in the `activity_datasets` directory or folder.

6.5 Exploratory Data Analysis (EDA)

Exploratory data analysis (EDA) is an approach used by data scientists to analyze and investigate data sets and summarize their main characteristics by using data visualization methods. In a simple form, the main purpose of EDA is to help look at data and see what it can tell us before making any assumptions. It can help identify obvious errors, as well as to understand patterns within the data, detect outliers (extreme values), and find interesting relations among the variables. For more information about Pandas Profiling Report check this [source](#).

6.5.1 Pandas Profiling

`pandas_profiling` is a Python module that extends the Pandas DataFrame with `df.profile_report()` for quick exploratory data analysis with just a few lines of code. In addition, it also generates interactive reports in web format with all the information easily available in the data. In summary,

what pandas profiling does is save us all the work of visualizing and understanding the distribution of each variable.

Pandas Profiling installation

Open up your Anaconda Prompt or Terminal and run

pip install pandas-profiling

```
bash-3.2$ pip install pandas-profiling
Collecting pandas-profiling
  Downloading pandas_profiling-2.9.0-py2.py3-none-any.whl (258 kB)
    |████████████████████████████████████████| 258 kB 111 kB/s
Collecting visions[type_image_path]==0.5.0
  Downloading visions-0.5.0-py3-none-any.whl (64 kB)
    |████████████████████████████████████████| 64 kB 183 kB/s
Requirement already satisfied: ipywidgets>=7.5.1 in ./opt/anaconda3/lib/python3.8/site-packages (from pandas-profiling) (7.5.1)
Requirement already satisfied: matplotlib>=3.2.0 in ./opt/anaconda3/lib/python3.8/site-packages (from pandas-profiling) (3.3.2)
Collecting htmlmin>=0.1.12
  Downloading htmlmin-0.1.12.tar.gz (19 kB)
Requirement already satisfied: pandas!=1.0.0,!1.0.1,!1.0.2,!1.1.0,>=0.25.3 in ./opt/anaconda3/lib/python3.8/site-packages (from pandas-profiling) (1.1.3)
Collecting missingno>=0.4.2
  Downloading missingno-0.4.2-py3-none-any.whl (9.7 kB)
Collecting phik>=0.9.10
  Downloading phik-0.10.0-py3-none-any.whl (599 kB)
    |████████████████████████████████████████| 599 kB 187 kB/s
Collecting tangled-up-in-unicode>=0.0.6
  Downloading tangled_up_in_unicode-0.0.6-py3-none-any.whl (3.1 MB)
    |████████████████████████████████████████| 2.8 MB 525 kB/s eta 0:00:01
```

Installing pandas-profiling

6.5.2 Importing Pandas Profiling

Like other libraries such as **numpy** or **pandas**, we also need to import pandas profiling after installation.

```
import numpy as np
import pandas as pd
from pandas_profiling import ProfileReport
```

6.5.3 Using Pandas Profiling

Example 1

Load the Ethiopian's flowers dataset(.csv) as **ethiopian_flower** DataFrame. The dataset is in the datasets folder.

```
ethiopian_flower = pd.read_csv("datasets/Ethiopian_flowers.csv")
```

We then use `.profile_report()` method on the DataFrame.

ethiopian_flower.profile_report()

Dataset info

Number of variables	5
Number of observations	350
Missing cells	0 (0.0%)
Duplicate rows	224 (64.0%)
Total size in memory	13.8 KiB
Average record size in memory	40.4 B

Warnings

Dataset has 224 (64.0%) duplicate rows

Petal.Width is highly correlated with Petal.Length ($\rho = 0.9644331901$)

Warning

Rejected

Variables types

Numeric	3
Categorical	1
Boolean	0
Date	0
URL	0
Text (Unique)	0
Rejected	1
Unsupported	0

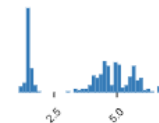
Variables

Petal.Length

Numeric

Distinct count	40
Unique (%)	11.4%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0

Mean	3.844857143
Minimum	1.1
Maximum	6.9
Zeros (%)	0.0%



Toggle details

Petal.Width

Highly-correlated

This variable is highly correlated with Petal.Length and should be ignored for analysis

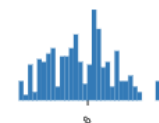
Correlation 0.9644331901

Sepal.Length

Numeric

Distinct count	32
Unique (%)	9.1%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0

Mean	5.832285714
Minimum	4.3
Maximum	7.9
Zeros (%)	0.0%



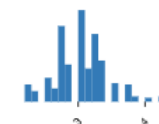
Toggle details

Sepal.Width

Numeric

Distinct count	22
Unique (%)	6.3%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0

Mean	3.042857143
Minimum	2.2
Maximum	4.4
Zeros (%)	0.0%



Toggle details

Species

Categorical

Distinct count	3
Unique (%)	0.9%
Missing (%)	0.0%
Missing (n)	0

Lily	121
Carnation	119
Rose	110

Toggle details

Example 2

Load the `money_transfer_transactions.csv` data as `money_transfer` DataFrame. This data set is about money transaction in Uganda, Somalia, Kenya, Ethiopia South Sudan, Tanzania, and Rwanda. The dataset is in the datasets folder.

```
money_transfer = pd.read_csv("datasets/money_transfer_transactions.csv")
```

We then use `.profile_report()` method on the DataFrame.`money_transfer.profile_report()`

Overview

Dataset info

Number of variables	10
Number of observations	500
Missing cells	0 (0.0%)
Duplicate rows	307 (61.4%)
Total size in memory	39.2 KiB
Average record size in memory	80.3 B

Warnings

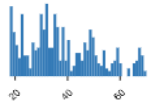
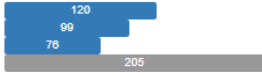
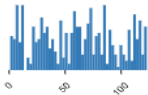
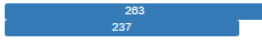
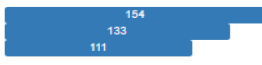
Dataset has 307 (61.4%) duplicate rows

Variables types

Numeric	4
Categorical	4
Boolean	2
Date	0
URL	0
Text (Unique)	0
Rejected	0
Unsupported	0

Warning

Variables

age Numeric	Distinct count 50 Unique (%) 10.0% Missing (%) 0.0% Missing (n) 0 Infinite (%) 0.0% Infinite (n) 0	Mean 38.084 Minimum 18 Maximum 70 Zeros (%) 0.0%		Toggle details
currency Categorical	Distinct count 7 Unique (%) 1.4% Missing (%) 0.0% Missing (n) 0			Toggle details
customer_id Numeric	Distinct count 94 Unique (%) 18.8% Missing (%) 0.0% Missing (n) 0 Infinite (%) 0.0% Infinite (n) 0	Mean 60.54 Minimum 1 Maximum 123 Zeros (%) 0.0%		Toggle details
direction Categorical	Distinct count 2 Unique (%) 0.4% Missing (%) 0.0% Missing (n) 0			Toggle details
dow Categorical	Distinct count 4 Unique (%) 0.8% Missing (%) 0.0% Missing (n) 0			Toggle details

Class activity 10 (Peer to peer review activity)



Peer to Peer Interaction

Visit the LMS, locate forum activity and participate in the discussion

Import `Somalia contraceptive method choice.csv` as `somalia_contraceptive`.

Description of the dataset:

The dataset is about the current contraceptive method choice (no use, long-term methods, or short-term methods) of Somalia woman based on her demographic and socio-economic characteristics.

Your task:

Perform exploratory data analysis using Pandas Profiling report.

Note: Somalia contraceptive method choice dataset is in the `activity_datasets` directory or folder.

6.0 Summarizing and Computing Descriptive Statistics

You can summarize your dataset with a set of common mathematical and statistical methods with Pandas attribute. For example, let's check the summary statistics in the `somalia_women` DataFrame.

`.describe()` method

Using `.describe()` on an entire DataFrame we can get a summary of the distribution of continuous variables:

```
somalia_women_dropna.describe()
```

	height	weight
count	241.000000	241.000000
mean	64.697095	64.016598
std	4.437007	14.355303
min	58.000000	44.000000
25%	61.000000	52.000000
50%	65.000000	64.000000
75%	68.000000	75.000000
max	72.000000	93.000000

Examples

Consider `east_africa_dropna` DataFrame in which we have removed all columns with any missing values.

```
east_africa_dropna.describe()
```

	Name	Gender	Origin.1
count	1128	1128	1128
unique	1047	2	41
top	Guled	M	Amharic
freq	3	614	242

`.describe()` can also be used on a categorical variable to get the count of rows, unique count of categories, top category, and frequency of top category. For example, in `registrations_df` DataFrame

```
registrations_df
```

	Country	Gender	Age
0	Uganda	Male	11
1	Ethiopia	Female	17
2	Rwanda	Female	15
3	Somalia	Female	17
4	Sudan	Male	13
5	Kenya	Female	11
6	Uganda	Male	9
7	Kenya	Female	18
8	Somalia	Male	13
9	Rwanda	Female	16

```
registrations_df["Gender"].describe()
```

```
count      10
unique       2
top      Female
freq         6
Name: Gender, dtype: object
```

Frequency distribution of Series

.value_counts() method

.value_counts() attribute can tell us the frequency distribution of all values in a column or Series.

To illustrate these, consider examples below:

Example 1

```
registrations_df["Gender"].value_counts()
```

```
Female    6
Male      4
Name: Gender, dtype: int64
```

Example 2

Consider `east_africa` names DataFrame, what is the distribution of male and female in the **Gender** column?

```
east_africa.head()
```

	Name	Gender	Origin.1	Origin.2	Origin.3	Origin.4
0	Aadaan	M	Somali	NaN	NaN	NaN
1	Aadam	M	Urdu	Somali	Estonian	NaN
2	Aaden	M	Somali	NaN	NaN	NaN
3	Aamiina	F	Somali	NaN	NaN	NaN
4	Abadir	M	Arabic	Bohairic	Sahidic	NaN

```
east_africa["Gender"].value_counts()
```

```
M    614
F    514
Name: Gender, dtype: int64
```

As you can see, there are 616 and 515 male and female names respectively.

`.value_counts()` also accepts some other arguments i.e. `sort = True/False` and `normalize = True/False`. The Series is sorted by value in descending order with `sort = True` and relative frequencies with `normalize = True`.

```
east_africa["Gender"].value_counts(sort = True, normalize = True)
```

```
M    0.544326
F    0.455674
Name: Gender, dtype: float64
```

You can get the result in percentage by multiplying by 100

```
east_africa["Gender"].value_counts(sort = True, normalize = True) * 100
```

```
M    0.544326
F    0.455674
Name: Gender, dtype: float64
```

Class activity 11 (Tutor guided question)



Peer to Peer Interaction

Visit the LMS, locate forum activity and participate in the discussion

Find and display the name of the most common species in the [penguins.dta](#) dataset.

Solution

We need to import the [penguins.dta](#) dataset. Remember this is a STATA file ([.dta](#))

First, check your current working directory by using [pwd](#). Make sure [penguins.dta](#) is in your working directory.

```
pwd
```

```
C:\\Users\\OGUNDEPO EZEKIEL .A\\Desktop\\Data and Decision\\Introduction to data science
1\\Scripts
```

But [penguins.dta](#) is in the [datasets](#) folder so I will use relative path [datasets/penguins.dta](#).

Now let's import our [penguins.dta](#) dataset with Pandas.

```
import pandas as pd
penguins = pd.read_stata("datasets/penguins.dta")
penguins.head()
```

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
0	Adelie	Torgersen	39.1	18.7	181	3750	male
1	Adelie	Torgersen	39.5	17.4	186	3800	female
2	Adelie	Torgersen	40.3	18.0	195	3250	female
3	Adelie	Torgersen	36.7	19.3	193	3450	female
4	Adelie	Torgersen	39.3	20.6	190	3650	male

Let's not forget the question!

Find and display the name of the most common species in the penguins' dataset.

Select the **species** Series from the penguins DataFrame and apply `value_counts()` or `.describe()` on it

```
penguins["species"].value_counts()
```

```
Adelie      150
Gentoo      119
Chinstrap    68
```

```
penguins["species"].describe()
```

```
count      337
unique       3
top      Adelie
freq       150
Name: species, dtype: object
```

Adelie is the most common species in the penguins' dataset

6.1 Data Aggregation and Group Operations

6.1.1 Grouping one variable

By “**group by**” we are referring to a process involving one or more of the following steps:

- Splitting the data into groups based on some criteria
- Applying a function to each group independently
- Combining the results into a data structure

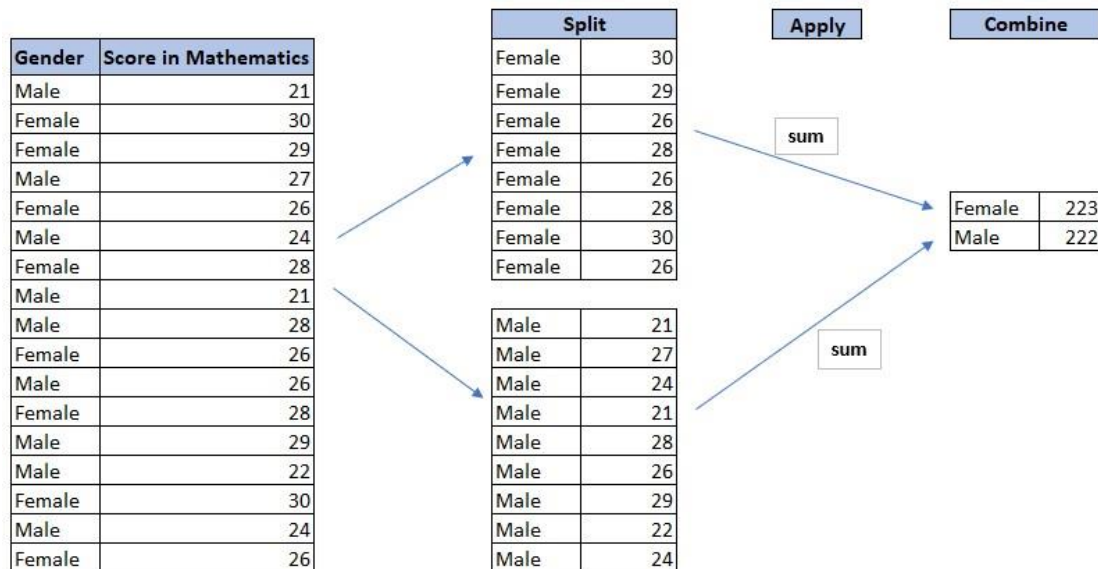


Illustration of a group aggregation

Example 1

Consider **penguins** DataFrame, what is the mean **bill_length(mm)** of each of the species.

```
penguins.head()
```

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
0	Adelie	Torgersen	39.1	18.7	181	3750	male
1	Adelie	Torgersen	39.5	17.4	186	3800	female
2	Adelie	Torgersen	40.3	18.0	195	3250	female
3	Adelie	Torgersen	36.7	19.3	193	3450	female
4	Adelie	Torgersen	39.3	20.6	190	3650	male

We call the **.groupby()** on the DataFrame and with a column (a Series) to groupby with. We then apply some operation to each of the groups. For example, to compute group means we can call the groupby's mean method:

```
penguins.groupby("species").mean()
```

	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g
species				
Adelie	38.800000	18.342667	190.026667	3705.500000
Chinstrap	48.833824	18.420588	195.823529	3733.088235
Gentoo	47.568067	14.996639	217.235294	5092.436975

We can now see the mean of each continuous variable for each of the species.

The mean `bill_length` of Adelie, Chinstrap, and Gentoo are 38.800000, 48.833824, and 47.568067 respectively.

We can actually use:

```
penguins.groupby(["species"])[["bill_length_mm"]].mean()
```

```
species
Adelie      38.800000
Chinstrap   48.833824
Gentoo      47.568067
Name: bill_length_mm, dtype: float64
```

To get the same results with only mean of the `bill_length_mm` for each specie

6.1.2 More than one grouping

Groupby also allows for more than 1 grouping. For example, what is the sum of the `bill_depth (mm)` for the combination of species and sex of Penguins?

```
penguins.groupby(["species", "sex"])[["bill_length_mm"]].sum()
```

```
species    sex
Adelie     female  2791.7
           male    3028.3
Chinstrap  female  1583.5
           male    1737.2
Gentoo     female  2642.7
           male    3017.9
Name: bill_length_mm, dtype: float64
```

6.1.3 Data Aggregation

We can use more than one aggregation such as `mean`, `count`, `min`, and `sum` for a groupby object by passing it to the `.agg()` attribute. For example, we can pass a list of aggregation functions to `agg` to evaluate independently on the data groups.

Function names	Definition
count	Number of non-NA values in the group
sum	Sum of non-NA values
mean	Mean of non-NA values
median	Arithmetic median of non-NA values
std	Standard deviation
var	Variance
min	Minimum of non-NA values
max	Maximum of non-NA values

Example 1

Find the **sum**, **mean**, and **standard deviation** of **bill_lenght** for each of the species in the **Penguins** dataset.

```
penguins.groupby(["species"])[["bill_length_mm"]].agg(["sum", "mean", "std"])
```

	sum	mean	std
species			
Adelie	5820.0	38.800000	2.670219
Chinstrap	3320.7	48.833824	3.339256
Gentoo	5660.6	47.568067	3.106116

Example 2

Using Penguins dataset, find **minimum** and **maximum** of **body_mass** for the combination of **species** and **sex** of Penguins?

```
penguins.groupby(["species", "sex"])["body_mass_g"].agg(["min", "max"])
```

		min	max
species	sex		
Adelie	female	2850	3900
	male	3325	4775
Chinstrap	female	2700	4150
	male	3250	4800
Gentoo	female	3950	5200
	male	4750	6300

For example, we can interpret body mass for Adelie specie as:

species	sex	min	max
Adelie	female	2850	3900
Adelie	male	3325	4775

The minimum and maximum body mass for female Adelie Penguins are 2850 and 3900 respectively while that of male Adelie are 3325 and 4775 respectively. It can be seen that male Adelie has more body mass than female Adelie.

6.1.4 Cross-Tabulations (Crosstab)

Contingency tables also called crosstabs or two-way tables are used in statistics to present categorical data in terms of frequency counts. In other words, it is a special type of frequency distribution table, where two variables are shown simultaneously. More precisely, an **r×c** contingency table shows the observed frequency of two variables, the observed frequencies of




which are arranged into *r* rows and *c* columns. The intersection of a row and a column of a contingency table is called a cell.

How to do crosstabulation?

We use `pd.crosstab()` attribute to do crosstabulation and `pd.crosstab()` contains the following:

```
pd.crosstab(index, columns, values=None, rownames=None, colnames=None,
aggfunc=None, margins=False, margins_name='All', dropna=True, normalize=False)
```

Some major parameters are:

-  **index**: Series- values to group by in the rows
-  **column**: Series - values to group by in the columns
-  **margins** bool (True/False), default False: Add row/column margins (subtotals)

Additional resources

For more information about `pd.crosstab()`, check this [link](#).

Example 1

```
pd.crosstab(index = penguins["species"], columns = penguins["sex"], margins = True)
```

sex	female	male	All
species			
Adelie	75	75	150
Chinstrap	34	34	68
Gentoo	58	61	119
All	167	170	337

As you can see, there are 58 female and 61 male Gentoo Penguins

Example 2

Create a crosstabulation table using index for species, Island and sex for columns

```
pd.crosstab(index = penguins["species"], columns = [penguins["island"], penguins["sex"]], margins=True)
```

island	Biscoe		Dream		Torgersen		All
sex	female	male	female	male	female	male	
species							
Adelie	22	22	27	28	26	25	150
Chinstrap	0	0	34	34	0	0	68
Gentoo	58	61	0	0	0	0	119
All	80	83	61	62	26	25	337

6.6 Combining and Merging Datasets

It's rare that a data analysis involves only a single table of data. Typically, you have many tables of data, and you must combine them to answer the questions that you're interested in. Collectively, multiple tables of data are called relational data because it is the relations, not just the individual datasets, that are important.

Relations are always defined between a pair of tables. All other relations are built up from this simple idea: the relations of three or more tables are always a property of the relations between each pair. Sometimes both elements of a pair can be the same table! This is needed if, for example, you have a table of people, and each person has a reference to their parents.

For example, consider the two DataFrames below:

```
COVID19_info_dic = {
    "Country": ["Burundi", "Ethiopia", "Kenya", "Rwanda", "Somalia", "Tanzania", "Ugan", "Sudan"],
    "Confirmed cases": [673, 107109, 79322, 5750, 4445, 509, 18406, 16431],
    "Death cases": [1, 1664, 1417, 47, 113, 21, 186, 1202],
    "Recovered cases": [575, 66574, 52974, 5241, 3412, 183, 8764, 9854],
    "Population": [12029114, 116082175, 54237961, 13078334, 16066789, 60399786, 46304101, 44252054]
}
```

```
COVID19_info_df = pd.DataFrame(COVID19data, columns= ["Country", "Total Cases", "Total Deaths", "Total Recovered", "Population"])
```

COVID19_info_df

	Country	Total Cases	Total Deaths	Total Recovered	Population
0	Burundi	673	1	575	12029114
1	Ethiopia	107109	1664	66574	116082175
2	Kenya	79322	1417	52974	54237961
3	Rwanda	5750	47	5241	13078334
4	Somalia	4445	113	3412	16066789
5	Tanzania	509	21	183	60399786
6	Uganda	18406	186	8764	46304101
7	Sudan	16431	1202	9854	44252054

```
epidemiology_info_dic = {"Country": ["Burundi", "Ethiopia", "Kenya", "Rwanda", "Somalia", "Tanzania", "Uganda", "Sudan"],
    "Fatality rate" : [0.001, 0.016, 0.018, 0.008, 0.025, 0.041, 0.010, 0.073],
    "Discharge rate" : [0.854, 0.622, 0.668, 0.911, 0.768, 0.360, 0.476, 0.600]
}
```

```
epidemiology_info_df = pd.DataFrame(COVID19data_2, columns= ["Country", "Fatality rate", "Discharge rate"])
```

epidemiology_info_df

	Country	Fatality rate	Discharge rate
0	Burundi	0.001	0.854
1	Ethiopia	0.016	0.622
2	Kenya	0.018	0.668
3	Rwanda	0.008	0.911
4	Somalia	0.025	0.768
5	Tanzania	0.041	0.360
6	Uganda	0.010	0.476
7	Sudan	0.073	0.600

6.6.1 Keys

The variables used to connect each pair of tables are called keys. A key is a variable (or set of variables) that uniquely identifies an observation. In simple cases, a single variable is sufficient to identify an observation. For example, each COVID-19 data is uniquely identified by their country name. In other cases, multiple variables may be needed.

There are two types of keys:

A **primary key** uniquely identifies an observation in its own table. For example, `COVID19_info_df["Country"]` is a primary key because it uniquely identifies each COVID-19 cases in the COVID-19 DataFrame.

A **foreign key** uniquely identifies an observation in another table. For example, `epidemiology_info_df["Country"]` is a foreign key because it appears in the COVID-19 DataFrame where it matches each country to a unique COVID-19 cases.





Data contained in pandas objects can be combined together in a number of ways:

1. **pd.merge** connects rows in DataFrames based on one or more keys.
2. **pd.concat** concatenates or “stacks” together objects along an axis.

6.7 Joining/merging DataFrames with `pd.merge()` function

pd.merge() combine datasets by linking rows using one or more keys. There are 4 types of joining.

Type of merge to be performed includes:

-  left: use only keys from the left DataFrame
-  right: use only keys from the right DataFrame
-  outer: use union of keys from both DataFrames
-  inner: use intersection of keys from both DataFrames


```
pd.merge(left, right, how='inner', on=None, left_on=None, right_on=None, left_index=False,
right_index=False, sort=True, suffixes=('_x', '_y'), copy=True, indicator=False, validate=None)
```

Some of the key parameters are:

left: The first DataFrame

right: The second DataFrame

on: Column to join on.

how: One of 'left', 'right', 'outer', 'inner'. Defaults to inner.

The left **left_on** and **right_on** are useful when primary and foreign keys have different names (column names)

left_on: Columns from the left DataFrame to use as keys. Can either be column names, index level names, or arrays with length equal to the length of the DataFrame or Series.

right_on: Columns or index levels from the right DataFrame or Series to use as keys. Can either be column names, index level names, or arrays with length equal to the length of the DataFrame or Series.

Example 1

Consider **COVID19_info_df** and **epidemiology_info_df** DataFrames.

```
COVID19_info_df.head()
```

	Country	Total Cases	Total Deaths	Total Recovered	Population
0	Burundi	673	1	575	12029114
1	Ethiopia	107109	1664	66574	116082175
2	Kenya	79322	1417	52974	54237961
3	Rwanda	5750	47	5241	13078334
4	Somalia	4445	113	3412	16066789

```
epidemiology_info_df.head()
```

	Country	Fatality rate	Discharge rate
0	Burundi	0.001	0.854
1	Ethiopia	0.016	0.622
2	Kenya	0.018	0.668
3	Rwanda	0.008	0.911
4	Somalia	0.025	0.768

We can perform left merging or joining by using:

```
country_info = pd.merge(COVID19_info_df, epidemiology_info_df, on = "Country", how = "left")
```

country_info

	Country	Total Cases	Total Deaths	Total Recovered	Population	Fatality rate	Discharge rate
0	Burundi	673	1	575	12029114	0.001	0.854
1	Ethiopia	107109	1664	66574	116082175	0.016	0.622
2	Kenya	79322	1417	52974	54237961	0.018	0.668
3	Rwanda	5750	47	5241	13078334	0.008	0.911
4	Somalia	4445	113	3412	16066789	0.025	0.768
5	Tanzania	509	21	183	60399786	0.041	0.360
6	Uganda	18406	186	8764	46304101	0.010	0.476
7	Sudan	16431	1202	9854	44252054	0.073	0.600

As you can see, the two datasets have been merge together.

Example 2

Consider **band_members** and **band_instruments** DataFrames below:

```
band_members = pd.DataFrame({
    "name": ["Mick", "John", "Paul"],
    "band": ["Stones", "Beatles", "Beatles"]
})
band_members
```

	name	band
0	Mick	Stones
1	John	Beatles
2	Paul	Beatles

```
band_instruments = pd.DataFrame(
{"name": ["John", "Paul", "Keith"],
"plays": ["guitar", "bass", "guitar"]}
)
band_instruments
```

	name	plays
0	John	guitar
1	Paul	bass
2	Keith	guitar

Inner join

Use inner join to combine band_members and band_instruments DataFrames

```
inner_join = pd.merge(band_members, band_instruments, on = "name", how = "inner")
inner_join
```

	name	band	plays
0	John	Beatles	guitar
1	Paul	Beatles	bass

inner join uses intersection of keys from both DataFrames and includes all rows where there is an intersection

Left join

Use left join to combine band_members and band_instruments DataFrames

```
left_join = pd.merge(band_members, band_instruments, on = "name", how = "left")
left_join
```

	name	band	plays
0	Mick	Stones	NaN
1	John	Beatles	guitar
2	Paul	Beatles	bass

left join uses only keys from the left DataFrame and includes all its rows in the merging.

Right join

Use right join to combine `band_members` and `band_instruments` DataFrames

```
right_join = pd.merge(band_members, band_instruments, on = "name", how = "right")
right_join
```

	name	band	plays
0	John	Beatles	guitar
1	Paul	Beatles	bass
2	Keith	NaN	guitar

right join uses only keys from the right DataFrame and includes all its rows for the merging.

Outer join

Use outer join to combine `band_members` and `band_instruments` DataFrames

```
outer_join = pd.merge(band_members, band_instruments, on = "name", how = "outer")
outer_join
```

	name	band	plays
0	Mick	Stones	NaN
1	John	Beatles	guitar
2	Paul	Beatles	bass
3	Keith	NaN	guitar

outer join also known as full join uses union of keys from both DataFrames and includes all rows in the two DataFrames for the merging.

Class activity 12 (Group work)



Peer to Peer Interaction

Visit the LMS, locate forum activity and participate in the discussion

COVID-19 information for the West Africa countries are stored in the **activity_datasets** directory/folder:

west_africa_COVID_cases.csv: This consists COVID-19 cases for each West Africa country

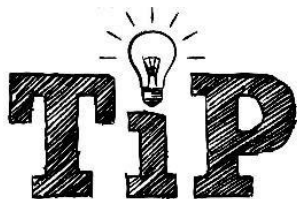
west_africa_COVID_test.csv: COVID-19 testing information for each West Africa country

Use Pandas to:

1. import **west_africa_COVID_cases.csv** as **west_africa_covid**
2. import **west_africa_COVID_test.csv** as **west_africa_test**
3. use **pd.merge()** to merge/join the two DataFrames using left join

6.7.1 Concatenating DataFrames with **pd.concat()** function

To simply concatenate or combine the DataFrames along the row you can use the **pd.concat()** function in pandas. You will have to pass the names of the DataFrames in a list as the argument to the **pd.concat()** function.



DataFrames to be concatenated/combined must have the same columns names.

Example 1

Let's import `east_africa.csv` and `central_africa.csv` COVID-19 datasets which can be found in the **datasets** directory and combine them using `pd.concat()`.

We shall import the dataset using a relative path `datasets/...`

```
east_africa_covid = pd.read_csv("datasets/east_africa.csv")
```

```
east_africa_covid.head()
```

	Country	Region	Total Cases	Total Deaths	Total Recovered	Active Cases	Serious_Critical	Population	Total Tests	Tests/1M pop
0	Burundi	East Africa	694	1.0	575	118	NaN	12039804	62215.0	5167.0
1	Comoros	East Africa	616	7.0	600	9	NaN	877463	NaN	NaN
2	Djibouti	East Africa	5701	61.0	5593	47	NaN	994138	93540.0	94092.0
3	Eritrea	East Africa	632	NaN	508	124	NaN	3567406	21655.0	6070.0
4	Ethiopia	East Africa	113295	1747.0	80831	30717	317.0	116168018	1669754.0	14374.0

```
centra_africa_covid = pd.read_csv("dataset/central_africa.csv")
```

```
centra_africa_covid.head()
```

	Country	Region	Total Cases	Total Deaths	Total Recovered	Active Cases	Serious_Critical	Population	Total Tests	Tests/1M pop
0	Angola	Central Africa	15536	354	8335	6847	14.0	33295883	171247.0	5143.0
1	Cameroon	Central Africa	24752	443	23344	965	52.0	26825311	149000.0	5554.0
2	Central African Republic	Central Africa	4922	63	1924	2935	2.0	4865546	32711.0	6723.0
3	Chad	Central Africa	1725	102	1564	59	NaN	16624422	NaN	NaN
4	Congo	Central Africa	5774	94	4988	692	NaN	5575542	NaN	NaN

We can combine the two DataFrames by passing the names of the DataFrames in a list as the argument to the `pd.concat()` function.

```
east_and_centra_africa_covid19 = pd.concat([east_africa_covid, centra_africa_covid])
```

```
east_and_centra_africa_covid19
```

	Country	Region	Total Cases	Total Deaths	Total Recovered	Active Cases	Serious_Critical	Population	Total Tests	Tests/1M pop
0	Burundi	East Africa	694	1.0	575	118	NaN	12039804	62215.0	5167.0
1	Comoros	East Africa	616	7.0	600	9	NaN	877463	NaN	NaN
2	Djibouti	East Africa	5701	61.0	5593	47	NaN	994138	93540.0	94092.0
3	Eritrea	East Africa	632	NaN	508	124	NaN	3567406	21655.0	6070.0
4	Ethiopia	East Africa	113295	1747.0	80831	30717	317.0	116168018	1669754.0	14374.0
5	Kenya	East Africa	88380	1526.0	68929	17925	76.0	54273641	927082.0	17082.0
6	Madagascar	East Africa	17513	255.0	16927	331	16.0	27991823	95472.0	3411.0
7	Malawi	East Africa	6051	185.0	5476	390	4.0	19338802	76148.0	3938.0
8	Mauritius	East Africa	508	10.0	465	33	NaN	1272680	289552.0	227514.0
9	Mayotte	East Africa	5181	49.0	2964	2168	4.0	275601	25231.0	91549.0
10	Mozambique	East Africa	16244	133.0	14416	1695	NaN	31624580	238362.0	7537.0
11	Rwanda	East Africa	6129	51.0	5696	382	NaN	13088013	644631.0	49254.0
12	Seychelles	East Africa	184	NaN	168	16	NaN	98609	5200.0	52734.0
13	Somalia	East Africa	4525	121.0	3480	924	NaN	16080166	NaN	NaN
14	South Sudan	East Africa	3166	62.0	2977	127	1.0	11249945	61964.0	5508.0

You will notice that the two DataFrames `east_africa_covid` and `centra_africa_covid` are now concatenated or combined into a single DataFrame called `east_and_centra_africa_covid19` along the row. However, the row labels seem to be wrong! If you want the row labels to adjust

automatically according to the join, you will have to set the argument `ignore_index` as `True` while calling the `pd.concat()` function:

```
east_and_centra_africa_covid19 = pd.concat([east_africa_covid, centra_africa_covid],
ignore_index = True)
east_and_centra_africa_covid19
```

	Country	Region	Total Cases	Total Deaths	Total Recovered	Active Cases	Serious_Critical	Population	Total Tests	Tests/1M pop
0	Burundi	East Africa	694	1.0	575	118	NaN	12039804	62215.0	5167.0
1	Comoros	East Africa	616	7.0	600	9	NaN	877463	NaN	NaN
2	Djibouti	East Africa	5701	61.0	5593	47	NaN	994138	93540.0	94092.0
3	Eritrea	East Africa	632	NaN	508	124	NaN	3567406	21655.0	6070.0
4	Ethiopia	East Africa	113295	1747.0	80831	30717	317.0	116168018	1669754.0	14374.0
5	Kenya	East Africa	88380	1526.0	68929	17925	76.0	54273641	927082.0	17082.0
6	Madagascar	East Africa	17513	255.0	16927	331	16.0	27991823	95472.0	3411.0
7	Malawi	East Africa	6051	185.0	5476	390	4.0	19338802	76148.0	3938.0
8	Mauritius	East Africa	508	10.0	465	33	NaN	1272680	289552.0	227514.0
9	Mayotte	East Africa	5181	49.0	2964	2168	4.0	275601	25231.0	91549.0
10	Mozambique	East Africa	16244	133.0	14416	1695	NaN	31624580	238362.0	7537.0
11	Rwanda	East Africa	6129	51.0	5696	382	NaN	13088013	644631.0	49254.0
12	Seychelles	East Africa	184	NaN	168	16	NaN	98609	5200.0	52734.0
13	Somalia	East Africa	4525	121.0	3480	924	NaN	16080166	NaN	NaN
14	South Sudan	East Africa	3166	62.0	2977	127	1.0	11249945	61964.0	5508.0

Example 2

Let's import `north_africa.csv` and concatenate it to `east_and_centra_africa_covid19` DataFrame using `pd.concat()`.

```
north_africa_covid = pd.read_csv("datasets/north_africa.csv")
north_africa_covid.head()
```

	Country	Region	Total Cases	Total Deaths	Total Recovered	Active Cases	Serious_Critical	Population	Total Tests	Tests/1M pop
0	Algeria	North Africa	88252	2516	57146	88252	46.0	44188203	NaN	NaN
1	Egypt	North Africa	118014	6750	103324	7940	46.0	103155589	1000000.0	9694.0
2	Libya	North Africa	86580	1231	56702	28647	NaN	6911249	458079.0	66280.0
3	Morocco	North Africa	379657	6245	331301	42111	940.0	37097956	4074451.0	109830.0
4	Sudan	North Africa	19196	1290	10942	6964	NaN	44282906	NaN	NaN

```
north_east_centra = pd.concat([north_africa_covid, east_and_centra_africa_covid19],
```



```
ignore_index = True)
```

```
north_east_centra.head()
```

	Country	Region	Total Cases	Total Deaths	Total Recovered	Active Cases	Serious_Critical	Population	Total Tests	Tests/1M pop
0	Algeria	North Africa	88252	2516.0	57146	88252	46.0	44188203	NaN	NaN
1	Egypt	North Africa	118014	6750.0	103324	7940	46.0	103155589	1000000.0	9694.0
2	Libya	North Africa	86580	1231.0	56702	28647	NaN	6911249	458079.0	66280.0
3	Morocco	North Africa	379657	6245.0	331301	42111	940.0	37097956	4074451.0	109830.0
4	Sudan	North Africa	19196	1290.0	10942	6964	NaN	44282906	NaN	NaN

Additional resources

For more resources on merging and concatenating data, check the following resources:

https://pandas.pydata.org/pandas-docs/stable/user_guide/merging.html

<https://bit.ly/datacamp-joining-dataframes-pandas>

Class activity 13 (Peer to peer review activity)

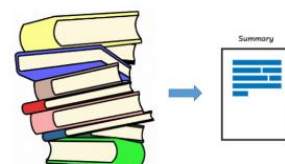


Peer to Peer Interaction

Visit the LMS, locate forum activity and participate in the discussion

import `west_africa.csv` and `southern_africa.csv` datasets in the `activities/datasets` directory and combined them using `pd.concat()` and represent the resulting DataFrame as `west_south`.

1. What is the number of columns and rows in `west_south` DataFrame?
2. Examine the number of missing values in each of the columns if there is any.
3. How many countries are in the `west_south` DataFrame?
4. Use `west_south.describe()` to get the number of total COVID-19 cases in the DataFrame






Summary of Study Unit 6

In this study unit, you have learnt that:

1. Pandas is a popular Python package for data science
2. Both Series and DataFrame can be used to create or generate a dataset.
3. With Pandas you can import different file format such as .xlsx, .csv, etc.
4. You can get information about your dataset using `.info()`, `.shape`, `.head()`, `.tail()` or `.columns` attributes
5. Pandas Profiling can be used to generate a quick exploratory data analysis of your dataset.

Additional resources

For more additional resources on Pandas, check the following resources:

-  <http://bit.ly/data-school-25-pandas-tricks>
-  https://pandas.pydata.org/pandas-docs/stable/user_guide/10min.html
-  <https://bit.ly/python-pandas-tutorial-complete-introduction-for-beginners>