

Lecture 07: Functions

March 9, 2023



1 Data Analysis

Ahmed Moustafa

2 Data Analysis

We will be discussing three important Python libraries that are commonly used for data analysis: **pandas**, **matplotlib**, and **numpy**.

3 Importing Libraries in Python

- In Python, libraries or packages are collections of pre-written code that can be imported and used in your own programs.
- The most common way to import a library is using the `import` statement followed by the name of the library.
- For example, to import the `math` library, you would use the following statement:

```
[2]: import math  
  
math.sqrt(25)
```

```
[2]: 5.0
```

4 Importing Libraries with Aliases

- You can also import a library with an alias using the `as` keyword.
- This can be useful when you want to use a shorter name for a library in your code.
- For example, to import the `math` library with the alias `m`, you would use the following statement:

```
[3]: import math as m

m.sqrt(25)
```

```
[3]: 5.0
```

5 Pandas

- Pandas is a Python library that is used for data manipulation and analysis.
- It provides data structures for efficiently storing and manipulating large datasets.
- Pandas is built on top of NumPy and is often used in conjunction with it.
- Let's start with an example of how to use Pandas to load a CSV file.

6 Titanic Dataset

- The Titanic dataset contains data on the passengers of the Titanic, including their survival status, age, gender, class, and other attributes.
- The data was collected by Kaggle and is often used for predictive modeling and machine learning tasks.
- The table has 891 rows and 12 columns.
- Here's a glimpse of the table:

PassengerId	Survived	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	0	Braund, male Mr. Owen Harris		22	1	0	A/5			

7 Loading a CSV file with Pandas

```
[4]: import pandas as pd

# Load Titanic dataset
url = 'https://web.stanford.edu/class/archive/cs/cs109/cs109.1166/stuff/titanic.csv'
df = pd.read_csv(url)

# Print the first few rows of the DataFrame
df.head()
```

```
[4]:   Survived  Pclass                                Name \
0         0        3                        Mr. Owen Harris Braund
1         1        1  Mrs. John Bradley (Florence Briggs Thayer) Cum...
2         1        3                        Miss. Laina Heikkinen
3         1        1  Mrs. Jacques Heath (Lily May Peel) Futrelle
4         0        3                        Mr. William Henry Allen
```

	Sex	Age	Siblings/Spouses Aboard	Parents/Children Aboard	Fare
0	male	22.0	1	0	7.2500
1	female	38.0	1	0	71.2833
2	female	26.0	0	0	7.9250
3	female	35.0	1	0	53.1000
4	male	35.0	0	0	8.0500

8 Viewing Data

- Once you've loaded data into a `DataFrame`, you can start exploring it using various Pandas functions.
- The `head()` and `tail()` functions are useful functions for quickly viewing the first and last few rows of a `DataFrame`.

```
[5]: df.tail()
```

```
[5]:
```

	Survived	Pclass	Name	Sex	Age	\
882	0	2	Rev. Juozas Montvila	male	27.0	
883	1	1	Miss. Margaret Edith Graham	female	19.0	
884	0	3	Miss. Catherine Helen Johnston	female	7.0	
885	1	1	Mr. Karl Howell Behr	male	26.0	
886	0	3	Mr. Patrick Dooley	male	32.0	

	Siblings/Spouses Aboard	Parents/Children Aboard	Fare
882	0	0	13.00
883	0	0	30.00
884	1	2	23.45
885	0	0	30.00
886	0	0	7.75

9 Basic Statistics

- You can use Pandas to calculate basic statistics on your data, such as mean, median, and standard deviation.
- The `describe()` function provides a summary of the basic statistics of each column in the `DataFrame`.

```
[6]: df.describe()
```

```
[6]:
```

	Survived	Pclass	Age	Siblings/Spouses Aboard	\
count	887.000000	887.000000	887.000000	887.000000	
mean	0.385569	2.305524	29.471443	0.525366	
std	0.487004	0.836662	14.121908	1.104669	
min	0.000000	1.000000	0.420000	0.000000	
25%	0.000000	2.000000	20.250000	0.000000	

50%	0.000000	3.000000	28.000000	0.000000
75%	1.000000	3.000000	38.000000	1.000000
max	1.000000	3.000000	80.000000	8.000000

	Parents/Children Aboard	Fare
count	887.000000	887.000000
mean	0.383315	32.30542
std	0.807466	49.78204
min	0.000000	0.000000
25%	0.000000	7.92500
50%	0.000000	14.45420
75%	0.000000	31.13750
max	6.000000	512.32920

10 Indexing and Selection

- You can use indexing and selection to retrieve specific data from a **DataFrame**.
- The `loc[]` function is used for label-based indexing, where you can specify the row and column labels.
- The `iloc[]` function is used for integer-based indexing, where you can specify the row and column numbers.

```
[7]: df.iloc[2:5]
```

```
[7]:   Survived  Pclass                                Name  Sex \
2         1        3                Miss. Laina Heikkinen  female
3         1        1  Mrs. Jacques Heath (Lily May Peel) Futrelle  female
4         0        3                Mr. William Henry Allen   male

   Age  Siblings/Spouses Aboard  Parents/Children Aboard  Fare
2  26.0                      0                        0    7.925
3  35.0                      1                        0   53.100
4  35.0                      0                        0    8.050
```

```
[8]: df.loc[2:5, ['Survived', 'Pclass']]
```

```
[8]:   Survived  Pclass
2         1        3
3         1        1
4         0        3
5         0        3
```

11 Filtering Data

- You can use Boolean indexing to filter data in a **DataFrame** based on a certain condition.
- For example, you can filter the Titanic dataset to only show passengers who survived:

```
[9]: # Filter Titanic dataset to only show passengers who survived
survivors = df[df['Survived'] == 1]
survivors.head()
```

```
[9]:   Survived  Pclass                                Name \
1         1         1  Mrs. John Bradley (Florence Briggs Thayer) Cum...
2         1         3                                Miss. Laina Heikkinen
3         1         1      Mrs. Jacques Heath (Lily May Peel) Futrelle
8         1         3  Mrs. Oscar W (Elisabeth Vilhelmina Berg) Johnson
9         1         2      Mrs. Nicholas (Adele Achem) Nasser

      Sex  Age  Siblings/Spouses Aboard  Parents/Children Aboard   Fare
1  female  38.0                1                0  71.2833
2  female  26.0                0                0   7.9250
3  female  35.0                1                0  53.1000
8  female  27.0                0                2  11.1333
9  female  14.0                1                0  30.0708
```

12 Grouping and Aggregation

- Grouping and aggregation are powerful tools for summarizing and analyzing data in a DataFrame.
- You can group data in a DataFrame based on one or more columns, and then apply an aggregation function like `sum()`, `mean()`, or `count()`.
- For example, you can group the Titanic dataset by ticket class and calculate the average age for each class:

```
[10]: # Group Titanic dataset by ticket class and calculate the average age for each
      ↪ class
age_by_class = df.groupby('Pclass')['Age'].mean()
age_by_class
```

```
[10]: Pclass
1     38.788981
2     29.868641
3     25.188747
Name: Age, dtype: float64
```

13 Merging Data

- Sometimes you need to combine multiple datasets into a single DataFrame.
- Pandas provides the `merge()` function for merging datasets based on one or more common columns.

14 Matplotlib

- Matplotlib is a Python library used for creating data visualizations.

- It provides a wide range of tools for creating line plots, bar plots, histograms, scatterplots

15 Plotting a Simple Line Graph with Matplotlib

- Let's start with an example of how to use Matplotlib to create a simple line graph.
- For this example, we will use a dataset of global CO2 emissions from 1960 to 2014.
- Here's the code to load the dataset:

```
[11]: import pandas as pd
import matplotlib.pyplot as plt

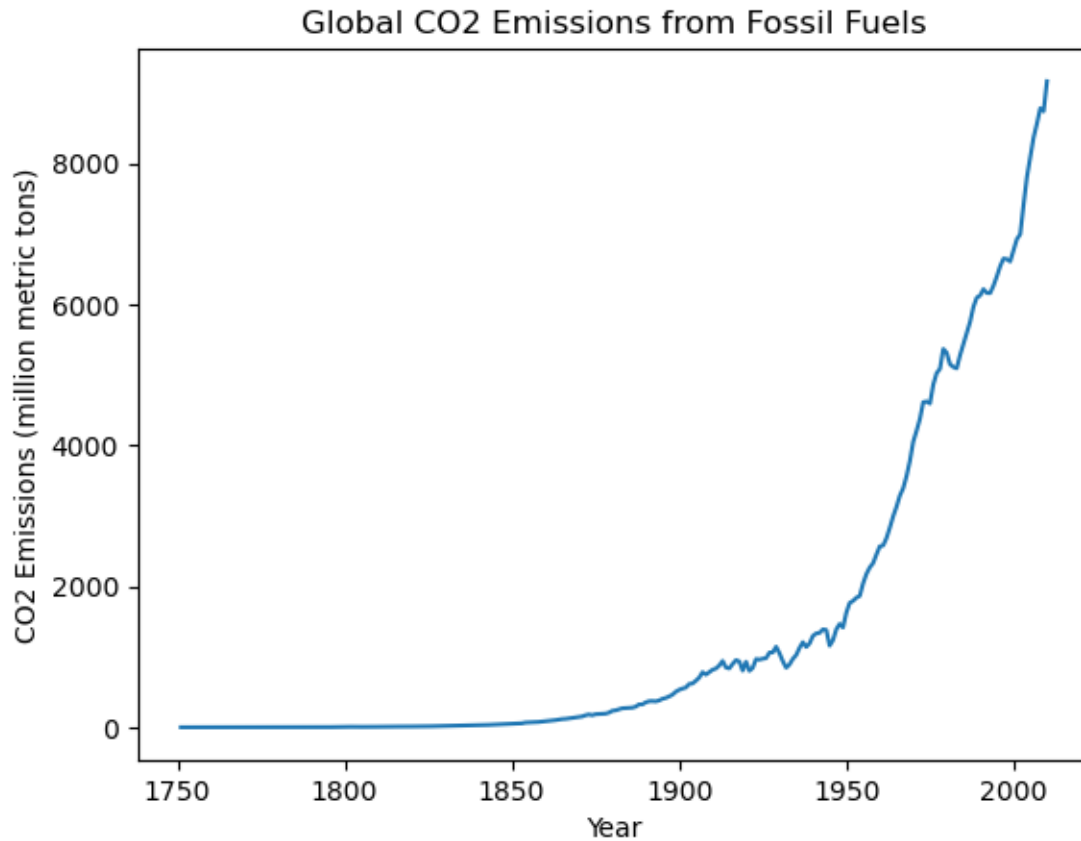
url = 'https://raw.githubusercontent.com/datasets/co2-fossil-global/master/
      ↪global.csv'
co2 = pd.read_csv(url)

co2.head()
```

```
[11]:   Year  Total  Gas Fuel  Liquid Fuel  Solid Fuel  Cement  Gas Flaring  \
0  1751      3         0         0         3         0         0
1  1752      3         0         0         3         0         0
2  1753      3         0         0         3         0         0
3  1754      3         0         0         3         0         0
4  1755      3         0         0         3         0         0

      Per Capita
0         NaN
1         NaN
2         NaN
3         NaN
4         NaN
```

```
[12]: plt.plot(co2['Year'], co2['Total'])
plt.xlabel('Year')
plt.ylabel('CO2 Emissions (million metric tons)')
plt.title('Global CO2 Emissions from Fossil Fuels')
plt.show()
```



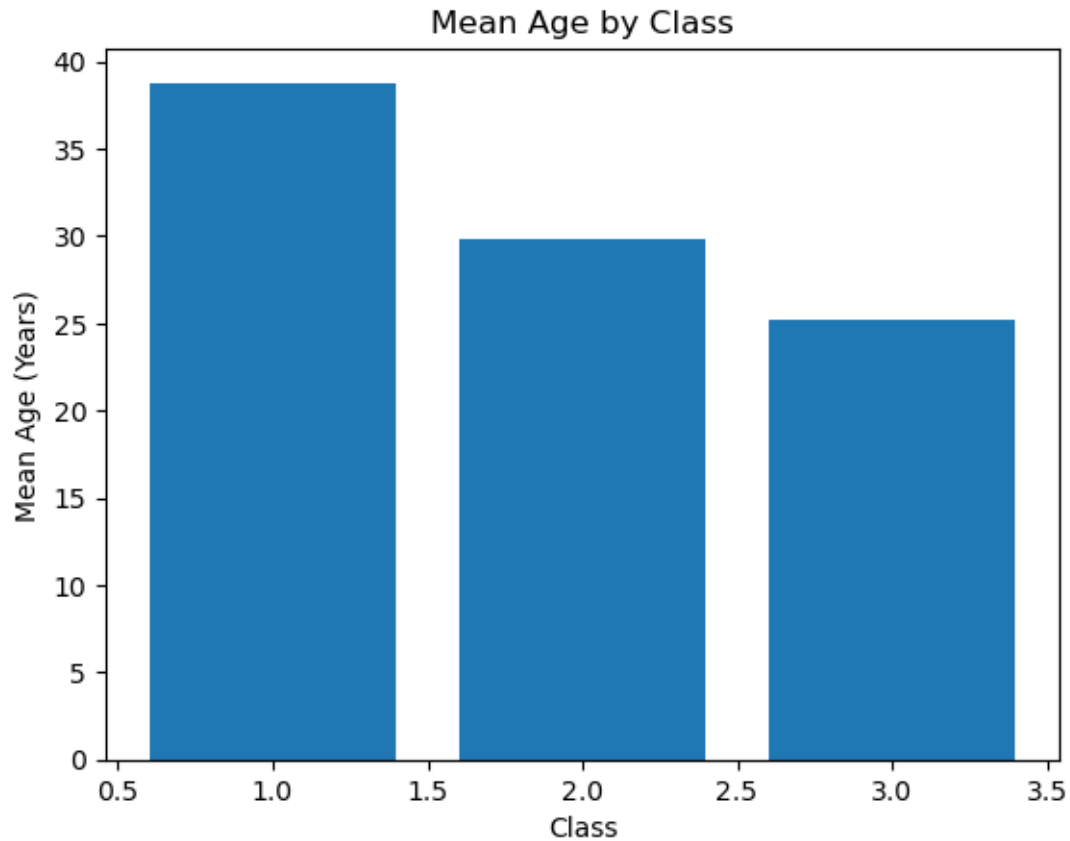
16 Creating a Bar Plot with Matplotlib

- You can also use Matplotlib to create a bar plot.
- For this example, we will use a dataset of the number of medals won by different countries in the 2016 Summer Olympics.
- Here's the code to load the dataset and create a bar plot:

```
[13]: plt.bar(age_by_class.index, age_by_class.values)

plt.title('Mean Age by Class')
plt.xlabel('Class')
plt.ylabel('Mean Age (Years)')

plt.show()
```



17 Numpy

- NumPy is a Python library used for numerical computing.
- It provides a wide range of tools for working with arrays and matrices.
- NumPy is used in many scientific computing applications.
- Let's start with an example of how to use NumPy to create an array.

18 Creating a NumPy Array

- To create a NumPy array, you can use the `numpy.array()` function.
- Here's the code to create a NumPy array:

```
[14]: import numpy as np

data = [1, 2, 3, 4, 5]
arr = np.array(data)
arr
```

```
[14]: array([1, 2, 3, 4, 5])
```


19 NumPy Array Operations

- You can perform various operations on NumPy arrays.
- For example, you can add, subtract, multiply, and divide arrays.
- Here's the code to add two arrays:

```
[15]: import numpy as np

arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
arr3 = arr1 + arr2
arr3
```

```
[15]: array([5, 7, 9])
```

20 NumPy Array Indexing and Slicing

- You can also index and slice NumPy arrays.
- Here's the code to create a NumPy array and slice it:

```
[16]: import numpy as np

arr = np.array([1, 2, 3, 4, 5])
arr[2]
```

```
[16]: 3
```

```
[17]: arr[1:4]
```

```
[17]: array([2, 3, 4])
```

21 NumPy Broadcasting

- Broadcasting is a powerful NumPy feature that allows you to perform operations on arrays of different shapes.
- Here's an example:

```
[18]: import numpy as np

# Create a 2D array of shape (3, 4)
arr1 = np.array([[1, 2, 3, 4],
                 [5, 6, 7, 8],
                 [9, 10, 11, 12]])

# Create a 1D array of shape (4,)
arr2 = np.array([2, 2, 2, 2])

# Add the 1D array to each row of the 2D array using broadcasting
```

```
result = arr1 + arr2  
  
print(result)
```

```
[[ 3  4  5  6]  
 [ 7  8  9 10]  
[11 12 13 14]]
```

22 NumPy Broadcasting 2/3

In this example, we have a 2D NumPy array `arr1` with shape (3, 4) and a 1D NumPy array `arr2` with shape (4,). We want to add the values in `arr2` to each row of `arr1`. Normally, this operation would not be possible because the two arrays have different shapes. However, NumPy broadcasting allows us to perform this operation by “stretching” or “broadcasting” the 1D array to match the shape of the 2D array.

In this case, NumPy broadcasts the 1D array `arr2` to a 2D array of shape (3, 4) by duplicating its values along the first dimension. This allows us to perform element-wise addition between the two arrays.

23 NumPy Broadcasting 3/3

Note that broadcasting is not always possible or desirable, and certain conditions must be met for it to work correctly. For example, the trailing dimensions of the two arrays must either match or be equal to 1, among other rules. It’s important to understand these rules and use broadcasting judiciously to avoid errors and unexpected results.