# Lab Plan - 2: Python Libraries for Data Science

***Outline:***

* Data visualization with Matplotlib
* Control flow
* Pandas

## Data Visualizations with Matplotlib

Matplotlib can create most kinds of charts, like line graphs, scatter plots, bar charts, pie charts, stack plots, 3D graphs, and geographic map graphs.

First, to use Matplotlib, we're going to need it so first statement would be ‘import’ statement. Next, we invoke the .plot method of pyplot to plot some coordinates (data).

|  |
| --- |
| Example 2.1: |
| import matplotlib.pyplot as plt  year= [1994,1995,1998,2000]# data points  population=[2.59,3.69,5.33,6.77]# data points  plt.plot(year,population) #plots the data on specified co-ordinates in background  plt.show() #visualises the graph  plt.scatter(year,population)  plt.show() |

The above code creates the following plots.

|  |
| --- |
| Figure 2.1: Results of matplotlib functions |

### Legends, Titles & labels

A lot of times, graphs can be self-explanatory, but having a title to the graph, labels on the axis, and a legend that explains what each line is can be necessary.

|  |
| --- |
| Example 2.2: |
| import matplotlib.pyplot as plt  year= [1994,1995,1998,2000]# data points  population=[2.59,3.69,5.33,6.77]# data points  plt.plot(year,population,label='population 1') #plots the data on specified co-ordinates in background  pop2=[4.44,3.22,5.55,6.88]  plt.plot(year,pop2, label='population 2') # plots the data and assign label to it  plt.xlabel('Independent var') # specifies xlabel  plt.ylabel('Dependent var') #specifies ylabel  plt.title('Interesting Graph') #specifies title  plt.legend() #visualises labels specified to the data  plt.show() |

The above code creates the following plot.

|  |
| --- |
| Figure 2.2: Display facilities |

Matplotlib provides alot of methods for customization of graphs. Another example of this is-

|  |
| --- |
| Example 2.3: |
| import matplotlib.pyplot as plt  year= [1994,1995,1998,2000]# data points  population=[2.59,3.69,5.33,6.77]# data points  plt.plot(year,population,'r',label='Population 1', linewidth=5) #plots the data on specified co-ordinates in background  pop2=[4.44,3.22,5.55,6.88]  plt.plot(year,pop2, 'c',label='Population 2',linewidth=5) # plots the data and assign label to it  plt.xlabel('Independent var') # specifies xlabel  plt.ylabel('Dependent var') #specifies ylabel  plt.title('Interesting Graph') #specifies title  plt.legend() #visualises labels specified to the data  plt.grid(True,color='k')  plt.fill\_between(year,population,0,color='green') #fills the specified color below the data points  plt.show() |

The above code displays following graph

|  |
| --- |
| Figure 2.3: Customization of graph |

### Histograms

To explore more about dataset one can also use histograms to get the idea about distributions.

Consider an example where we have 12 values in range of 0 and 10. To build a histogram of these values divide them into equal chunks. Suppose we go for 3 bins, finally draw bar of each line height of bar corresponds to number of data points falling in particular bin.

Python code for the above scenario-

|  |
| --- |
| Example 2.4: |
| import matplotlib.pyplot as plt  help (plt.hist)  #list with 12 values  values=[1.2,1.3,2.2,3.3,2.4,6.5,6.6,7.7,8.8,9.9,4.2,5.3]  plt.hist(values,bins=3)  plt.show() |

The above code displays following graph.

|  |
| --- |
| Figure 2.4: Histogram |

## Control flow & Pandas

### Boolean Logic&Control flow

#### Relational Operators

These operators compare the values on either sides of them and decide the relation among them.

The following table lists relational operators:

|  |  |
| --- | --- |
| **Operator** | **Description** |
| == | If the values of two operands are equal, then the condition becomes true. |
| != | If values of two operands are not equal, then condition becomes true. |
| > | If the value of left operand is greater than the value of right operand, then condition becomes true. |
| < | If the value of left operand is less than the value of right operand, then condition becomes true. |
| >= | If the value of left operand is greater than or equal to the value of right operand, then condition becomes true. |
| <= | If the value of left operand is less than or equal to the value of right operand, then condition becomes true. |

Table 2.1: Relational operators

#### Logical Operators

A boolean expression (or logicalexpression) evaluates to one of two states true or false. Python provides the boolean type that can be either set to False or True.

|  |  |
| --- | --- |
| **Operator** | **Description** |
| & AND | Returns true if both the compared expressions are true |
| | OR | Returns true even if both the compared expressions are not true |
| ~ | It is unary and has the effect of 'flipping' the result. |

#### Control flow

A program’s control flow is the order in which the program’s code executes. The control flow of a Python program is regulated by conditional statements, loops, and function calls.

#### if Statement

Often, you need to execute some statements only if some condition holds, or choose statements to execute depending on several mutually exclusive conditions. The Python compound statement if, which uses if, elif, andelse clauses, lets you conditionally execute blocks of statements. Here’s the syntax for the if statement:

ifexpression:

statement(s)

elifexpression:

statement(s)

elifexpression:

statement(s)

...

else:

statement(s)

The elif and else clauses are optional. Note that unlike some languages, Python does not have a switch statement, so you must use if, elif, and elsefor all conditional processing.

|  |
| --- |
| Example 2.5: |
| x=2  if x < 0:  print ("x is negative")  elif x % 2:  print ("x is positive and odd")  else:  print( "x is even and non-negative") |

The above code prints ‘x is even and non-negative’.

### Pandas

**Pandas** is a Python package providing fast, flexible, and expressive data structures designed to make working with “relational” or “labeled” data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, **real world** data analysis in Python. Additionally, it has the broader goal of becoming **the most powerful and flexible open source data analysis / manipulation tool available in any language**. It is already well on its way toward this goal.

Pandas is well suited for many different kinds of data:

* Tabular data with heterogeneously-typed columns, as in an SQL table or Excel spreadsheet
* Ordered and unordered (not necessarily fixed-frequency) time series data.
* Arbitrary matrix data (homogeneously typed or heterogeneous) with row and column labels
* Any other form of observational / statistical data sets. The data actually need not be labeled at all to be placed into a pandas data structure

The two primary data structures of pandas, **Series** (1-dimensional) and **DataFrame** (2-dimensional), handle the vast majority of typical use cases in finance, statistics, social science, and many areas of engineering.

A common way to start databases is to use CSV (Comma Separated Values) files pandas efficiently handles these files. CSV files are used to store a large number of variables – or data. They are incredibly simplified spreadsheets – think Excel – only the content is stored in plaintext.The text inside a CSV file is laid out in rows, and each of those has columns, all separated by commas. Every line in the file is a row in the spreadsheet, while the commas are used to define and separate cells.

Let’s take a look at one such example which retrieves data from CSV file. Use the given csv file.



|  |
| --- |
| Example 2.6: |
| import pandas as pd  data=pd.read\_csv('C:/Users/Mehak/Downloads/Import\_User\_Sample\_en.csv',index\_col=0)#skips the 0th column  print(data) |

The above code prints the following data

|  |
| --- |
|  |

Using pandas some of the basics operation like manipulation, addition, deletion of data from CSV file can be easily done.

|  |
| --- |
| Example 2.7: |
| import pandas as pd  data=pd.read\_csv('C:/Users/Mehak/Downloads/Import\_User\_Sample\_en.csv')#skips the 0th column  #Original File  print("Original file")  print(data)  #Retrieval of particular Column  print("Retrieval of particular column")  print(data['User Name']) #retrieves user name column  #Adding Data  data["Marital status"]=['Married','Single','Divorced','Single','Single'] #adds column to CSV file  print("Retrieval of newly added column")  print(data["Marital status"])  #Manipulation of Columns  data["New Number"]=data['Office Number']/data['ZIP or Postal Code']  print("Retrieval of newly added column through manipulation")  print(data['New Number'])  #Row Access  data.set\_index("Last Name", inplace=True)  print(data)  print(data.loc["Andrews"])  #Element Access  print("Element Access")  print(data.loc["Andrews"],["Address"])  #Row Access  print('Row access via iloc',data.iloc[0:2, :])  #Column Access  print('Column access via iloc',data.iloc[:, [1]]) |

Execute the above code and find out the alterations made in the CSV file data.

#### Boolean Indexing

Your dataframes and series can also be indexed with a conditional operation—a dataframe or series with the same dimensions as the one you are selecting from, but with every value either being set to True or False. This is known as 'Boolean Indexing'. You can create a new boolean series either by manually specifying the values, or by using a conditional.To index with your boolean series, simply feed it back into your regular series with using the [] bracket-selection syntax. The result is a new series that once again has the same dimensions, however only values corresponding to True in the boolean series be returned.If you need even finer grain control of what gets selected, you can further combine multiple boolean indexing conditionals together using the bit-wise logical operators | and &.

Something handy that you can do with a dataframe or series is write into a slice. Take precaution while doing this, as you may encounter issues with non-homogeneous dataframes. It is far safer, and generally makes more sense, to do this sort of operation on a per column basis rather than across your entire dataframe.

Use the given csv file:



|  |
| --- |
| Example 2.8: |
| import pandas as pd  data1=pd.read\_csv('C:/Users/Mehak/Desktop/DataScience/check/direct\_marketing.csv',index\_col=0)#skips the 0th column  print(data1)  #creates a sequence having boolean values for each of the record  print(data1.DM\_category < 7)  # creates sequence which qualifies the given condition  print(data1[ data1.DM\_category< 7 ])  # checks the given condition and creates sequence with qualified records  print(data1[ (data1.DM\_category < 7) & (data1.newbie == 0) ])  #writing to a slice  data1[data1.DM\_category < 7] = -100  print(data1) |

Execute the above code and find out the alterations made in the CSV file data.

## 

## Exercise

**Basic plots with matplotlib**

Question 1:

* print() the last item from both the year and the poplist to see what the predicted population for the year 2100 is.
* Before you can start, you should import matplotlib.pyplotas plt. pyplot is a sub-package of matplotlib, hence the dot.
* Use plt.plot() to build a line plot. year should be mapped on the horizontal axis, pop on the vertical axis. Don't forget to finish off with the show() function to actually display the plot.

**# Print the last item from year and pop**

**# Import matplotlib.pyplot as plt**

**# Make a line plot: year on the x-axis, pop on the y-axis**

Question 2:

* Change the line plot that's coded in the script to a scatter plot.
* Finish off your script with plt.show() to display the plot.

year= [1994,1995,1998,2000]# data points

pop=[2.59,3.69,5.33,6.77]# data points

**# Import matplotlib.pyplot as plt**

**# Create scatter plot using scatter function**

**# Show plot**

Now that you've built your first line plot, let's start working on the data that professor Hans Rosling used to build his beautiful bubble chart. It was collected in 2007. Two lists are available for you:

life\_exp which contains the life expectancy for each country and

gdp\_cap, which contains the GDP per capita, for each country expressed in US Dollar.

GDP stands for Gross Domestic Product. It basically represents the size of the economy of a country. Divide this by the population and you get the GDP per capita.

Question3:

* Print the last item from both the list gdp\_cap, and the list life\_exp; it is information about Zimbabwe.
* Build a line chart, with gdp\_cap on the x-axis, and life\_exp on the y-axis. Does it make sense to plot this data on a line plot?
* Don't forget to finish off with a plt.show() command, to actually display the plot.

import matplotlib.pyplot as plt; import importlib; importlib.reload(plt)

import pandas as pd

plt.clf()

df = pd.read\_csv('http://assets.datacamp.com/course/intermediate\_python/gapminder.csv', index\_col = 0)

gdp\_cap = list(df.gdp\_cap)

life\_exp = list(df.life\_exp)

**# Print the last item of gdp\_cap and life\_exp**

**# Import matplotlib.pyplot as plt**

**# Make a line plot, gdp\_cap on the x-axis, life\_exp on the y-axis**

**# Display the plot**

**Histograms**

Question 4:

* Use plt.hist() to create a histogram of the values in prices(a list containing atleast 20 different values). Do not specify the number of bins; Python will set the number of bins to 10 by default for you.
* Add plt.show() to actually display the histogram. Can you tell which bin contains the most observations?

**# Import matplotlib.pyplot as plt**

**#Create a list named as prices**

**# Create histogram of prices data**

**# Display histogram**

Question 5:

* Build a histogram of prices, with 5 bins. Can you tell which bin contains the most observations?
* Build another histogram of prices, this time with 20bins. Is this better?

**# Build histogram with 5 bins**

**# Show and clean up plot**

plt.show()

plt.clf()

**# Build histogram with 20 bins**

**# Show and clean up again**

plt.show()

plt.clf()

**Customizations**

Question 6: Repeat question 2 and 3 and add valid xlabel, ylabel and title which should be your roll number.

**Boolean logic & Controlflow**

Question 7:

* Examine the if statement that prints out "Looking around in the kitchen." if room equals "kit".
* Write another if statement that prints out "big place!" if area is greater than 15.

**# Define variables**

room = "kit"

area = 14.0

**# if statement for room**

if room == "kit" :

print("looking around in the kitchen.")

**# if statement for area**

Question 8:

* Add an else statement to the second control structure so that "pretty small." is printed out if area > 15 evaluates to False.

**# Define variables**

room = "kit"

area = 14.0

**# if-else construct for room**

if room == "kit" :

print("looking around in the kitchen.")

else :

print("looking around elsewhere.")

**# if-else construct for area**

if area > 15 :

print("big place!")

Question 9:

* Add an elif to the second control structure such that "medium size, nice!" is printed out if area is greater than 10.

**# Define variables**

room = "bed"

area = 14.0

**# if-elif-else construct for room**

if room == "kit" :

print("looking around in the kitchen.")

elif room == "bed":

print("looking around in the bedroom.")

else :

print("looking around elsewhere.")

**# if-elif-else construct for area**

if area > 15 :

print("big place!")

else :

print("pretty small.")

**Pandas**

In the exercises that follow, you will be working with vehicle data in different countries. Each observation corresponds to a country, and the columns give information about the number of vehicles per capita, whether people drive left or right, and so on. This data is available in a CSV file, named cars.csv.

Question 10:

* To import CSV files, you still need the pandas package: import it as pd.
* Use pd.read\_csv() to import cars.csv data as a DataFrame. Store this dataframe as cars.
* Print out cars. Does everything look OK?



**# Import pandas as pd**

**# Import the cars.csv data: cars**

**# Print out cars**

Question 11:

Remember index\_col, an argument of read\_csv() that you can use to specify which column in the CSV file should be used as a row label? Well, that's exactly what you need here!

* Specify the index\_col argument inside pd.read\_csv(): set it to 0, so that the first column is used as row labels.
* Has the printout of cars improved now?

Question 12:

In the sample code, the same cars data is imported from a CSV files as a Pandas DataFrame. To select only the cars\_per\_cap column from cars, you can use:

cars['cars\_per\_cap']

cars[['cars\_per\_cap']]

The single bracket version gives a Pandas Series, the double bracket version gives a Pandas DataFrame.

* Use single square brackets to print out the country column of cars as a Pandas Series.
* Use double square brackets to print out the country column of cars as a Pandas DataFrame. Do this by putting country in two square brackets this time.

**# Import cars data**

import pandas as pd

cars = pd.read\_csv('cars.csv', index\_col = 0)

**# Print out country column as Pandas Series**

**# Print out country column as Pandas DataFrame**

Question 13:

With loc you can do practically any data selection operation on DataFrames you can think of. loc is label-based, which means that you have to specify rows and columns based on their row and column labels.

Try out the following commands to experiment with loc to select observations:

cars.loc['RU']

cars.loc[['RU']]

cars.loc[['RU', 'AUS']]

As before, code is included that imports the cars data as a Pandas DataFrame.

* Use loc to select the observation corresponding to Japan as a Series. The label of this row is JAP. Make sure to print the resulting Series.
* Use loc to select the observations for Australia and Egypt as a DataFrame. You can find out about the labels of these rows by inspecting cars  Make sure to print the resulting DataFrame.

**# Import cars data**

import pandas as pd

cars = pd.read\_csv('cars.csv', index\_col = 0)

**# Print out observation for Japan**

**# Print out observations for Australia and Egypt**

Question 14:

loc also allows you to select both rows and columns from a DataFrame. To experiment, try out the following commands in the IPython Shell.

cars.loc['IN', 'cars\_per\_cap']

cars.loc[['IN', 'RU'], 'cars\_per\_cap']

cars.loc[['IN', 'RU'], ['cars\_per\_cap', 'country']]

* Print out the drives\_right value of the row corresponding to Morocco (its row label is MOR)
* Print out a sub-DataFrame, containing the observations for Russia and Morocco and the columns country and drives\_right.

**# Import cars data**

import pandas as pd

cars = pd.read\_csv('cars.csv', index\_col = 0)

**# Print out drives\_right value of Morocco**

**# Print sub-DataFrame**

### Think Tank

It's time to customize your own plot. This is the fun part, you will see your plot come to life!

You're going to work on the scatter plot with world development data: GDP per capita on the x-axis (logarithmic scale), life expectancy on the y-axis. The code for this plot is available in the script.

Question 15:

* The strings xlab and ylab are already set for you. Use these variables to set the label of the x- and y-axis.
* The string title is also coded for you. Use it to add a title to the plot.
* After these customizations, finish the script with plt.show()to actually display the plot.

import matplotlib.pyplot as plt; import importlib; importlib.reload(plt)

import pandas as pd

plt.clf()

df = pd.read\_csv('http://assets.datacamp.com/course/intermediate\_python/gapminder.csv', index\_col = 0)

gdp\_cap = list(df.gdp\_cap)

life\_exp = list(df.life\_exp)

**# Basic scatter plot, log scale**

plt.scatter(gdp\_cap, life\_exp)

plt.xscale('log')

**# Strings**

xlab = 'GDP per Capita [in USD]'

ylab = 'Life Expectancy [in years]'

title = 'World Development in 2007'

**# Add axis labels**

**# Add title**

**# After customizing, display the plot**