|  |
| --- |
| **W2 – Basic Data Processing and Plotting with Python** |

Save your W2 notelbook with the following naming conventions.

ID\_Name\_SecNo\_W2.ipynb,

for example

**6113333\_JohnWick\_541\_W2.ipynb**

**Line plot**

With matplotlib, you can create a bunch of different plots in Python. The most basic plot is the line plot. A general recipe is given here.

import matplotlib.pyplot as plt

plt.plot(x,y)

plt.show()

Let’s create sample data and plot a graph. Try

|  |  |
| --- | --- |
| import matplotlib.pyplot as plt  %matplotlib inline  year = [1950, 1970, 1990, 2010]  pop = [2.519, 3.629, 5.263, 6.972 ]  plt.plot(year, pop)  plt.show() |  |

Now let’s try with more data. Open the companion Word file, namely W2\_dataset.docx, and copy the belowed three lists into your Jupyter notebook.

Now that we’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 (i.e., per person) for each country expressed in US Dollars.
* pop, the populations are in millions of people.

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

When you have a time scale along the horizontal axis, the line plot is your friend. But in many other cases, when you're trying to assess if there's a correlation between two variables, for example, the scatter plot is the better choice. Below is an example of how to build a scatter plot.

**Scatter Plot**

Try again with scatter plot. Change plt.plot to plt.scatter. Expected plot: You may also add plt.xscale(‘log’) to display the GDP per capita on a log scale.

|  |  |
| --- | --- |
| A picture containing screenshot, line, diagram, plot  Description automatically generated | with log scale |

From there we aim to later visulaize **A picture containing text, screenshot, diagram, plot

Description automatically generated**

Exercises:

1. **Build a scatter plot, where pop is mapped on the horizontal axis, and life\_exp is mapped on the vertical axis.**

# Histogram Plot (To explore data set and to get idea about distribution)

**A close-up of a graph

Description automatically generated with low confidence**

life\_exp, the list containing data on the life expectancy for different countries in 2007. To see how life expectancy in different countries is distributed, let’s create a histogram of life\_exp. Try

# Create histogram of life\_exp data

plt.hist(life\_exp)

# Display histogram

plt.show()

|  |  |
| --- | --- |
| We didn’t specify the number of bins. By default, Python sets the number of bins to 10 in that case. The number of bins is pretty important. Too few bins will oversimplify reality and won’t show you the details. Too many bins will overcomplicate reality and won’t show the bigger picture.  You can specify the number of bins by adding an argument bin = x, where x is the number of bins. |  |

1. **Build a histogram of life\_exp, with 5 bins. Can you tell which bin contains the most observations?**

**Build another histogram of life\_exp, this time with 20 bins. Is this better?**

1. **Now, copy the list, life\_exp1950, to the Jupyter Notebook. Compare the life expectancy data for different countries in the years 1950 and 2007. Set the bins to 15. What do you observe? Which bin has the highest frequency? Answer in Jupyter Notebook.**
2. **You're a professor teaching Data Science with Python, and you want to visually assess if the grades on your exam follow a particular distribution. Which plot do you use? (Line Plot, Scatter Plot or Histogram). Answer in Jupyter Notebook.**
3. **You're a professor in Data Analytics with Python, and you want to visually assess if longer answers on exam questions lead to higher grades. Which plot do you use? (Line Plot, Scatter Plot or Histogram). Answer in Jupyter Notebook.**

|  |
| --- |
| Tips : Data Visualization   * Many options – different plot types, many customizations * Choices depend on – data, what you want to tell |

# Labels

It’s time to customize our own plot. This is the fun part, we will see your plot come to life! We’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. As a first step, let’s add axis labels and a title to the plot. We can do this with the xlabel(), ylabel() and title() functions, available in matplotlib.pyplot.

# 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

plt.xlabel(xlab)

1. **Add ylabel(), title() and show the graph. You will see that it is much clearer and easier to understand.**

# Ticks

We can control the y-ticks by specifying two arguments: plt.yticks([0,1,2], [“one”,”two”,”three”]). The ticks corresponding to the numbers 0, 1 and 2 will be replaced by one, two and three, respectively. Let’s do a similar thing for the x-axis of our world development chart, with the xticks() function. The tick values 1000, 10000 and 100000 should be replaced by 1k, 10k and 100k

# Scatter plot

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

# Previous customizations

plt.xscale('log')

plt.xlabel('GDP per Capita [in USD]')

plt.ylabel('Life Expectancy [in years]')

plt.title('World Development in 2007')

# Definition of tick\_val and tick\_lab

tick\_val = [1000,10000,100000]

tick\_lab = ['1k','10k','100k']

# Adapt the ticks on the x-axis

plt.xticks(tick\_val,tick\_lab)

# After customizing, display the plot

plt.show()

1. **Let’s set yticks to 30, 50, 70, 90. Plot the graph.**

# Sizes

Right now, the scatter plot is just a cloud of blue dots, indistinguishable from each other. Let’s change this. Wouldn’t it be nice if the size of the dots corresponds to the population? Now we need to use numpy package. Let’s import numpy as np and try to following code.

# Import numpy as np

import numpy as np

# Store pop as a numpy array: np\_pop

np\_pop = np.array(pop)

# Double np\_pop

np\_pop = np\_pop \* 2

# Update: set s argument to np\_pop

plt.scatter(gdp\_cap, life\_exp, s = np\_pop)

# Previous customizations

plt.xscale('log')

plt.xlabel('GDP per Capita [in USD]')

plt.ylabel('Life Expectancy [in years]')

plt.title('World Development in 2007')

plt.xticks([1000, 10000, 100000],['1k', '10k', '100k'])

# Display the plot

plt.show()

**A picture containing text, screenshot, diagram, font

Description automatically generated**

# Colors

The next step is making the plot more colorful! To do this, a list col has been created for you. It’s a list with a color for each corresponding country, depending on the continent the country is part of. If we have another look at the script, under # Additional Customizations, we’ll see that there are two plt.text() functions now. They add the words “India” and “China” in the plot.

Copy the color list, col, from the file and run the following code. Also try to adjust an argument alpha to observe the difference in color.

# Scatter plot

plt.scatter(x = gdp\_cap, y = life\_exp, s = np.array(pop) \* 2, c = col, alpha = 0.8)

# Previous customizations

plt.xscale('log')

plt.xlabel('GDP per Capita [in USD]')

plt.ylabel('Life Expectancy [in years]')

plt.title('World Development in 2007')

plt.xticks([1000,10000,100000], ['1k','10k','100k'])

# Additional customizations

plt.text(1550, 71, 'India')

plt.text(5700, 80, 'China')

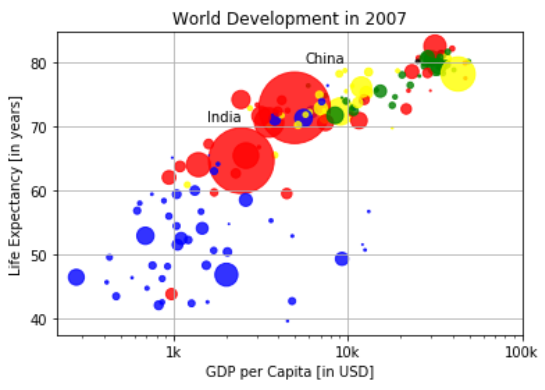
# Add grid() call

plt.grid(1)

# Show the plot

plt.show()

**A picture containing screenshot, line, diagram, plot

Description automatically generated **

# Interpretation

If you have a look at your colorful plot, it's clear that people live longer in countries with a higher GDP per capita. No high-income countries have really short life expectancy, and no low-income countries have very long-life expectancy. Still, there is a huge difference in life expectancy between countries on the same income level. Most people live in middle income countries where the difference in lifespan is huge between countries; depending on how income is distributed and how it is used.

1. **What can you say about the plot? Which one is True?**
   * **The countries in blue, corresponding to Africa, have both low life expectancy and a low GDP per capita.**
   * **There is a negative correlation between GDP per capita and life expectancy.**
   * **China has both a lower GDP per capita and lower life expectancy compared to India.**

# First start with Pandas.

We will start with loading structured data from .csv file. You need to import Pandas and use pd.read\_csv() command. Let’s load the data from brics.csv to brics which is a dataframe variable (2D structure data). Brics.head() will show the first five records.

import pandas as pd

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

brics.head()

A picture containing text, screenshot, font, number

Description automatically generated

Assume that we want to select countries with area over 8 sqr.km. Three simple steps are needed.

1. Select the are column.
   1. Different ways – try
      1. brics[‘area’]
      2. brics.loc[:,’area’]
      3. brics.iloc[:,2]
2. Do comparison on area column.
   1. Use comparison operator and apply to selected column. Try
      1. brics[‘area’] > 8, this will give you Boolean results.
3. Use result to select countries.
   1. Apply the comparison results to dataframe.
      1. Either assign the comparison result to a new variable first and apply it to the dataframe, i.e., result = brics[‘area’] > 8 then brics[result]
      2. Apply the comparison directly to the dataframe i.e., brics[brics[‘area]>8]

A picture containing text, screenshot, font, number

Description automatically generated

Only countries that area is > 8 are shown. Note that all columns will be displayed. To display some columns, we can do it by putting column’s name in the [ ] right after the above command. One good thing about Pandas is that commands are linear (in most cases). You can put one after the others. Try

A screenshot of a computer

Description automatically generated with medium confidence

1. **Show countries where the population is 200 or higher.**

What if we want to choose a value in between. For example, area > 8 and area < 10. In Pandas, we cannot simply use brics[‘area’] > 8 and brics[‘area’] < 10. We need a help from Numpy package. Let’s load numpy as np and try the following logical\_and() command.

A screenshot of a computer

Description automatically generated with medium confidence

Just like earlier, brics[np.logical\_and(brics['area']>8,brics['area']<10)] will also give you the same result.

1. **Show country’s capital (only) that the population is over 1000 or area is less than 3. Expected output is New Delhi, Beijing, Pretoria**

Let’s try with cars.csv.

1. **Complete the following instructions.**
   * **Load cars.csv**
   * **Select the cars\_per\_cap column from cars as a Pandas Series and store it as cpc.**
   * **Use cpc in combination with a comparison operator and 500. You want to end up with a boolean Series that's True if the corresponding country has a cars\_per\_cap of more than 500 and False otherwise. Store this boolean Series as many\_cars.**
   * **Use many\_cars to subset cars, similar to what you did before. Store the result as car\_maniac.**
   * **Print out car\_maniac to see if you got it right.**
2. **Create a DataFrame cpc100500, that includes all the observations of cars that have a cars\_per\_cap between 100 and 500. Then print cpc100500.**

**Loop over DataFrame.**

Iterating over a Pandas DataFrame is typically done with the iterrows() method. Used in a for loop, every observation is iterated over and on every iteration the row label and actual row contents. In this and the following exercises we will be working on the cars DataFrame. It contains information on the cars per capita and whether people drive right or left for seven countries in the world.

# Import cars data

import pandas as pd

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

# Iterate over rows of cars

for key in cars:

print(key)

Observe what the result is. Just column name, isn’t it?

Try again with

# Import cars data

import pandas as pd

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

# Iterate over rows of cars

for key,value in cars.iterrows():

print(key)

print(value)

The row data that’s generated by iterrows() on every run is a Pandas Series. This format is not very convenient to print out. Luckily, we can easily select variables from the Pandas Series using square brackets. Try

# Import cars data

import pandas as pd

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

# Adapt for loop

for lab,row in cars.iterrows():

print(lab +": "+ str(row['cars\_per\_cap']))

**Add Column to DataFrame**

After loading the data, we can manipulate the data in many ways. Let’s start with adding a new column to cars.

# Import cars data

import pandas as pd

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

# Code for loop that adds COUNTRY column

for lab,row in cars.iterrows():

cars.loc[lab, "COUNTRY"] = row["country"].upper()

# Print cars

print(cars)

This adds a new column with label “COUNTRY” and we assign the country’s name (uppercase) to the corresponding field.

1. **Now let’s add a new column, namely name\_length, to brics and assign the length of each country’s name.**

A way better approach if you want to calculate an entire DataFrame column by applying a function on a particular column in an element-wise fashion, is apply().

brics["name\_length"] = brics["country"].apply(len)

1. **Let’s try .apply() to add a new column to cars. Use str.upper argument in apply().**

**Let’s investigate Netflix data.**

Assume that the following lists are known. Note that durations is a list of average Netflix’s movies lengths in the corresponding years.

*# Create the years and durations lists*

years = [2011,2012,2013,2014,2015,2016,2017,2018,2019,2020]

durations = [103,101,99,100,100,95,95,96,93,90]

*# Create a dictionary with the two lists*

movie\_dict = {"years":years,"durations":durations}

*# Print the dictionary*

movie\_dict

We are now trying to create a dataframe from the given lists.

*# Import pandas under its usual alias*

import pandas as pd

*# Create a DataFrame from the dictionary*

durations\_df = pd.DataFrame(movie\_dict)

*# Print the DataFrame*

print(durations\_df)

Let’s plot the graph.

*# Import matplotlib.pyplot under its usual alias and create a figure*

import matplotlib.pyplot as plt

fig = plt.figure()

*# Draw a line plot of release\_years and durations*

plt.plot(durations\_df['years'], durations\_df['durations'])

*# Create a title*

plt.title('Netflix Movie Durations 2011-2020')

A line graph on a white background

Description automatically generated with low confidence

Well, it looks like there is something to the idea that movie lengths have decreased over the past ten years.

Luckily the original CSV for Netflix data is available. Let's create another DataFrame, this time with all of the data. Given the length of our friend's data, printing the whole DataFrame is probably not a good idea, so we will inspect it by printing only the first five rows.

*# Read in the CSV as a DataFrame*

netflix\_df = pd.read\_csv(r'netflix\_data.csv')

*# Print the first five rows of the DataFrame*

netflix\_df.head()

Okay, we have our data! Now we can dive in and start looking at movie lengths.

Or can we? Looking at the first five rows of our new DataFrame, we notice a column type. Scanning the column, it's clear there are also TV shows in the dataset! Moreover, the duration column we planned to use seems to represent different values depending on whether the row is a movie or a show (perhaps the number of minutes versus the number of seasons)?

Let’s choose columns *title, country, genre, release\_year* and *duration.*

*# Subset the DataFrame for type "Movie"*

netflix\_df\_movies\_only = netflix\_df.query('type == "Movie"')

*# Select only the columns of interest*

netflix\_movies\_col\_subset = netflix\_df\_movies\_only[['title','country','genre','release\_year','duration']]

*# Print the first five rows of the new DataFrame*

netflix\_movies\_col\_subset.head()

Let’s do a scatter plot to see data distribution over the years.

*# Create a figure and increase the figure size*

fig = plt.figure(figsize=(12,8))

*# Create a scatter plot of duration versus year*

plt.scatter(netflix\_movies\_col\_subset.release\_year, netflix\_movies\_col\_subset.duration)

*# Create a title*

plt.title("Movie Duration by Year of Release")

*# Show the plot*

Plt.show()

A picture containing screenshot

Description automatically generated

This is already much more informative than the simple plot we created when our friend first gave us some data. We can also see that, while newer movies are overrepresented on the platform, many short movies have been released in the past two decades.

Upon further inspection, something else is going on. Some of these films are under an hour long! Let's filter our DataFrame for movies with a duration under 60 minutes and look at the genres. This might give us some insight into what is dragging down the average.

*# Filter for durations shorter than 60 minutes*

short\_movies = netflix\_movies\_col\_subset.query('duration < 60')

*# Print the first 20 rows of short\_movies*

short\_movies.head(20)

Interesting! It looks as though many of the films that are under 60 minutes fall into genres such as "Children", "Stand-Up", and "Documentaries". This is a logical result, as these types of films are probably often shorter than 90 minute Hollywood blockbuster.

We could eliminate these rows from our DataFrame and plot the values again. But another interesting way to explore the effect of these genres on our data would be to plot them, but mark them with a different color.

In Python, there are many ways to do this, but one fun way might be to use a loop to generate a list of colors based on the contents of the genre column. Much as we did in Intermediate Python, we can then pass this list to our plotting function in a later step to color all non-typical genres in a different color!

Note: Although we are using the basic colors of red, blue, green, and black, *matplotlib* has many named colors you can use when creating plots. For more information, you can refer to the documentation [*here*](https://matplotlib.org/stable/gallery/color/named_colors.html)!

*# Define an empty list*

colors = []

*# Iterate over rows of netflix\_movies\_col\_subset*

for lab, row **in** netflix\_movies\_col\_subset.iterrows() :

if row['genre'] == "Children" :

colors.append("red")

elif row['genre'] == "Documentaries" :

colors.append("blue")

elif row['genre'] == "Stand-Up" :

colors.append("green")

else:

colors.append("black")

*# Inspect the first 10 values in your list*

colors[:10]

Lovely looping! We now have a colors list that we can pass to our scatter plot, which should allow us to visually inspect whether these genres might be responsible for the decline in the average duration of movies.

This time, we'll also spruce up our plot with some additional axis labels and a new theme with plt.style.use(). The latter isn't taught in Intermediate Python, but can be a fun way to add some visual flair to a basic matplotlib plot. You can find more information on customizing the style of your plot [here](https://matplotlib.org/stable/tutorials/introductory/customizing.html)!

*# Set the figure style and initalize a new figure*

plt.style.use('fivethirtyeight')

fig = plt.figure(figsize=(12,8))

*# Create a scatter plot of duration versus release\_year*

plt.scatter(netflix\_movies\_col\_subset.release\_year, netflix\_movies\_col\_subset.duration, color = colors)

*# Create a title and axis labels*

plt.title("Movie duration by year of release")

plt.xlabel("Release year")

plt.ylabel("Duration (min)")

*# Show the plot*

A picture containing text, screenshot, diagram, plot

Description automatically generated

Complete the following tasks.

1. Show the first five movies from USA with columns *title, country, genre, release\_year, duration.*
   1. (Subset the dataframe for type Movie, then subset for type United States

A screen shot of a list

Description automatically generated with low confidence

1. Find average movie lengths (duration) of each genre. To get this to group the data by genre (use .groupby(‘genre’)) and find mean values. After that

A screenshot of a movie

Description automatically generated with low confidence

1. Plot horizontal bar chart (.barh) where x is genre and y is average duration. Set title to “US Movies Only” and x-axis title to “Average Minutes”.

A picture containing text, screenshot, font, number

Description automatically generated

1. Try also with the scatter plot.

A picture containing text, screenshot, font, number

Description automatically generated

1. Now find all Neftlix movies/TV show/etc. released between 2016 and 2020 with columns *title, country, genre, release\_year, duration.*

A screenshot of a computer

Description automatically generated with low confidence

1. Find out how many Moview/TV show/etc. do we have from each country. You need .groupby(‘country’).count()

A picture containing text, screenshot, font, number

Description automatically generated

1. Sort the dataframe by title and plot the following horizontal bar chart.

A picture containing text, screenshot, line, font

Description automatically generated

1. Select only top 2, 3 and 4 countries and plot the pie chart.

A picture containing screenshot, font, graphics, diagram

Description automatically generated

1. Plot the following donut. Google how to do the donut plot. Also, Google how to show actual values and percentage in the chart.

