Lab 05 – Data Visualization I

Submission: `lab05.ipynb`, pushed to your Git repo on `master`.
Points: 10
Due: Friday, January 31, 2pm

Objectives

- The basics of `matplotlib` and `seaborn`

A Reminder…

This is a reminder of what you are in the midst of embarking on – a journey into the depths of data science, with a lot of emphasis on machine learning and data mining. You are learning about one of the hottest, most sought after skillsets in computer science today. It does not come easy, and the depth of knowledge required to become a solid, highly sought-after candidate is immense. The amount you are able to take away from this course will be entirely up to you. This is an active-learning, hands-on course, designed for you to learn through experience. Applying what you learn, rather than sitting and taking notes. And, the labs get progressively more complex. As an upper level elective, you are expected to delve in deeply into this material, independently.

Introduction

Before you begin these exercises, if you are not familiar with matplotlib, I strongly recommend that you work through the short, concise, but highly effective beginner's tutorial on the **pyplot** interface of **matplotlib**: https://matplotlib.org/tutorials/introductory/pyplot.html . It completely worth your time if you know nothing about matplotlib.

Then, check out https://matplotlib.org/tutorials/index.html. Page through some of these tutorials and example code, and remember this page. It has a wealth of example code you'll be needing for the rest of the semester.

Bookmark this API reference, because you are going to be referring to this API perhaps more than any other page this semester. (OK, probably not, but it will feel like it at times.) https://matplotlib.org/api/index.html

Oh, and this Cheat Sheet for Matplotlib from the DataCamp community is marvelous:
https://s3.amazonaws.com/assets.datacamp.com/blog_assets/Python_Matplotlib_Cheat_Sheet.pdf

Before you begin

Matplotlib is the quintessential, core plotting and data viz library for Python data scientists. However, there are others continuing to rise up, most of which are built on Matplotlib, and some that are trying to become completely new frameworks.

We're going to introduce you to TWO visualization frameworks in this lab:

- **Matplotlib** – the core, most essential framework you should learn
- **Seaborn** – built on top of Matplotlib, much easier and more intuitive, with better results most of the time

Soon, we're also going to introduce you to a third, **Plotly**, which is a completely new framework designed for interactive data analysis and visualization. This will be introduced in a separate lab.

Let's update your conda environment to be sure you have the correct packages installed. As usual, open a terminal and be sure that you have activated the correct conda environment for the course. Then, issue the following commands at the terminal:

```
$ conda install seaborn
$ conda install -c plotly plotly
```

From Data to Visualizations

There are a lot of ways we can visualize our data, but we must understand the type of our data first. This lab is going to step you through a large number of exercises to have you understand many of the most common visualizations of your data, and for what type of data they are good for.

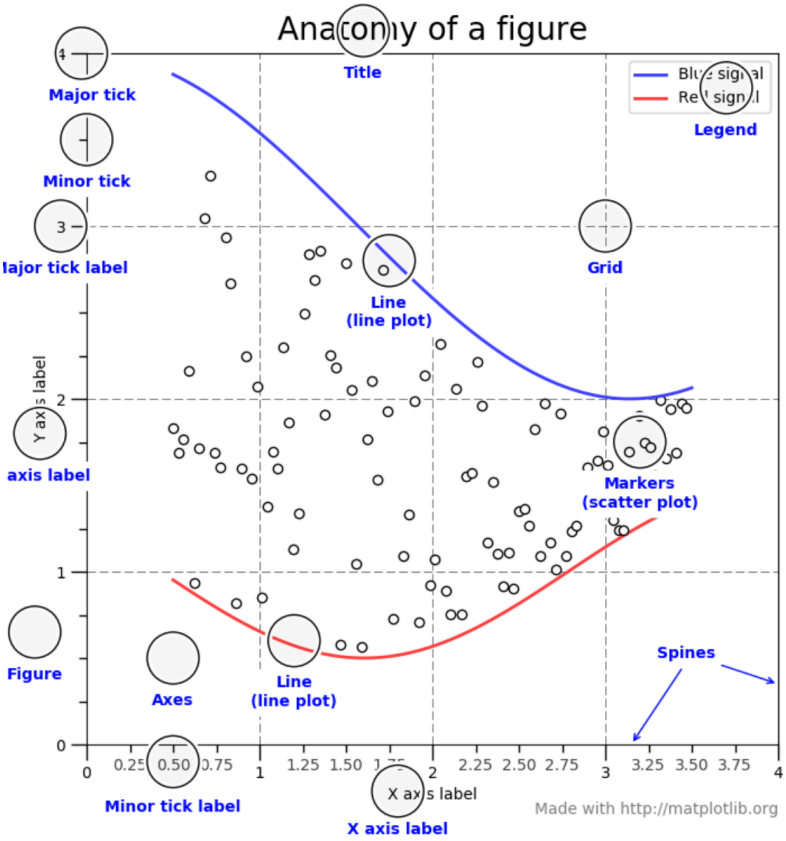
I STRONGLY encourage you to explore the following web site:

https://www.data-to-viz.com/

This is one of my most favorite sites to help students make sense of all of the different visualizations out there! I highly recommend you to take a moment to go through it. It will have answers to a lot of the questions you are going to be asked.

Part I - matplotlib

This diagram is your sanity as you begin to wrap your head around this visualization framework. The names that are used to label the important entities on the plot below matches the names used in the API:



Captured from https://matplotlib.org/tutorials/introductory/usage.html

Start by creating your **`lab05.ipynb`** file. Then, create your header cell, then your first cell to set up ALL of the imports we'll be using throughout this lab:

```
import numpy as np
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.graph_objects as go

print("np version:",np.__version__)
print("pd version:",pd.__version__)
print("mpl version:",mpl.__version__)
print("sns version:",sns.__version__)
```

1) [M] Read https://matplotlib.org/tutorials/introductory/usage.html only up to the section titled **Backends**. (NOTE: at the end, you will read about a Jupyter widget library called `ipympl`. Do NOT install this! Skip these instructions!) Then, summarize the following important parts of a plot:

    a. Figure
    b. Axes
    c. Axis

d. Artist

Click on the following: https://matplotlib.org/tutorials/index.html#introductory . Then, click on **Pyplot tutorial**. This page has good, basic examples to help you get going for the rest of the material

2) [P] Set the seed of numpy's RNG to 10 with the following:

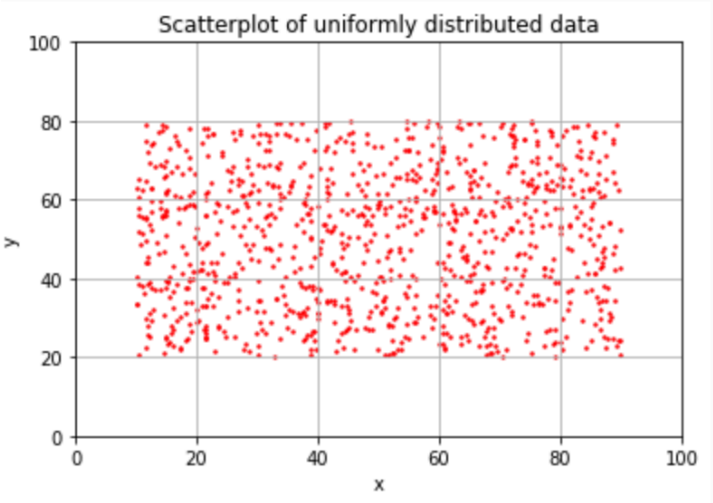
```
np.random.seed(10)
```

3) [P] Create a dataframe named `df_uniform` that contains 1000 observations. It should have two variables, named `x` and `y`. For each observation, `x` should be generated from a uniform distribution between 10 and 90, and y should be generated from a uniform between 20 and 80. Show the `head()` of the dataframe.

4) [M] What is a **scatterplot**? What does it show? What type of data is it used for? Is it good for one variable, or more?

5) [P] Generate a scatterplot of the data using matplotlib's `scatter` method. The tutorial and examples for matplotlib have plenty of examples that are very close to what you need. Your plot must:
   a. Have a title
   b. Label both axes with "`x`" and "`y`" respectively
   c. Change the x and y axis to display between 0 and 100
   d. Change the default point size
   e. Change the default color of the point
   f. Display a grid

   Here is one possible example:



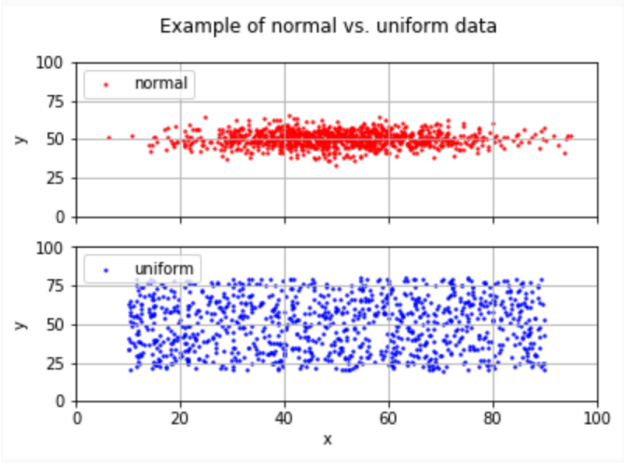
6) [P] Generate a data frame called `df_normal` with 1000 observations, two variables names x and y again. This time, x should be generated from a normal distribution with mean 50 and standard deviation 15, and y with mean 50 and standard deviation 5. Again, show the `head()` of `df_normal`.

7) [P] Repeat your scatterplot above with `df_normal` . Use a different color point, and title your plot accordingly.

8) [P] Generate a single figure that contains two axes that are adjacent to each other. You should have:
   a. at least one shared axis

b. appropriate axis labels
c. make the range of the axis on both plots the same
d. display a legend on each to be sure both are labeled correctly as "normal" or "uniform"
e. One title at the top

Here is one example:



9) [P] Display both `df_uniform` and `df_normal` on one shared plot, with an appropriate legend

As you might surmise, there are an enormous number of other types of plots you can use to understand your data. Matplotlib is quite impressive, and gives you great control for creating publication quality graphs. However, its interface is complex.

During our EDA phase, we often just want quick ways to explore our data and understand our distributions. Fortunately, pandas has a nice interface to matplotlib that makes it quite easy to generate some quick plots. We will look at those next.

10) [M] What is a **histogram**? In your answer, in addition to defining it, please clearly indicate what type of data its good for, and whether it's good for one variable or to show relationships between multiple variables.

11) [P] The pandas `DataFrame` class has a useful interface to matplotlib that will help you generate some quick plots as you explore your data. To get you started, generate a **histogram** of both the x and y variables for `df_uniform`. Use 30 bins, and set the range of both variables to be 0 – 100. Repeat this exercise on `df_normal`. (HINT:Use the `hist()` method of `DataFrame`.)

12) [M] What is a *quantile*?

13) [M] In terms of quantiles, what is a *quartile*? What about a *percentile*? What is an *Inter-quartile range (IQR)*?

14) [M, P] Read about the `quantile()` method for data frames, and use it to numerically show the 25th, median, and 75th percentiles, and compute the IQR (Inter-quartile range) for both variables, on both data frames. Compare and contrast.

15) [M] There are many ways to define what we mean by an outlier. A very common technique is known as the IQR rule for outliers. **What is a definition of an outlier in terms of IQR**? (Your answer should use 1.5 * IQR .)

16) [P] Write a function called `IQR_outlier_limits` that takes a dataframe as input, and computes the minimum and maximum outlier thresholds for each variable (numeric only assumed), stored in a data frame that is returned as a result.

    For example:

```
[100]: IQR_outlier_limits(df_normal)
```

[100]:

	x	y
min_out	8.836307	36.085185
max_out	90.278777	63.812816

    (The results will depend on the actual distribution of your random data. However, the results should make sense! Verify your results to ensure that you have some observations that are outliers. Your normal data should have some. Your uniform data should not.)

17) Use your function you defined above to determine the outliers for `df_normal` and `df_uniform`, if any. (HINT: You should get at least a few outliers for `df_normal`)

18) [M] What is a **box plot**? (A.k.a boxplot, box-and-whisker plot). In your answer, please clearly indicate what it is, what type of data its good for, and whether it's good for one variable or to show relationships between multiple variables.

19) [P] Again, use the `DataFrame` plotting interface to generate a **box plot** on both x and y variables of both `df_uniform` and `df_normal`.

20) [M] From your understanding of a box plot, summarize the distribution of these data, comparing the uniform and the normal distributed data. You should be using the correct terminology, meaning, interpreting the box plot results in terms of quartiles, outliers, etc.

21) [M] What is a **density** plot? In your answer, please clearly indicate what it is, the type of data you use it for, and whether it's good for one variable or to show relationships between multiple variables.

22) [P] Generate a density plot for both x and y variables of both `df_uniform` and `df_normal`.

23) [M] Interpret the density plot results

24) [P] Go back to the `describe()` method you learn about in previous labs. Show the results of `describe()` for both data frames. This stills you much information about the distribution of the data.

25) [M] What is a quantile-quantile plot (Q-Q plot)? (Wikipedia provides a good answer for this one https://en.wikipedia.org/wiki/Q%E2%80%93Q_plot)

26) [P] Load the `scipy.stats` package as `stats`. Look up the API and read about the `probplot` function. (See http://www.statsmodels.org/dev/generated/statsmodels.graphics.gofplots.qqplot.html#statsmodels.graphics.gofplots.qqplot) This can generate a Q-Q plot for you quite easily. Generate a Q-Q plot for a sample of 100 points from the x-variable `df_uniform`. Do the same for `df_normal`. For illustrative purposes, assume

your distribution is normal for both plots (even though we know it is not!)

27) [M] Compare and contrast your resulting plot. Does the output suggest that one is indeed normally distributed, and the other is not?

28) [P] Repeat the experiment, showing a Q-Q plot for 100 samples of each dataset. However, now assume the distribution is uniform.

29) Again, compare the plots. Does the output suggest one is indeed normally distributed and the other is not?

Part II - seaborn

`seaborn` is a data visualization package built on top of matplotlib. (If you did not, go back to the beginning of this lab and be sure to install the packages required!) Matplotlib is a solid graphics engine that can handle a LOT of different techniques. Seaborn takes matplotlib to the next level of modernization, makes many plotting tasks easier, and generally, it adds some seriously nice attractiveness.

Before you begin, please take the time to read through https://seaborn.pydata.org/introduction.html. Then, just so you have a sense of what some of its capabilities are, browse through the various tutorials on https://seaborn.pydata.org/tutorial.html . There are a LOT of different tutorials here. Seaborn focuses on helping you plot statistical relationships in your data, which is precisely what we're after when we are doing EDA with our data!

30) [P] Show a single scatterplot of `df_normal` using `sns`. Change the default color and point type that is used in the plot.

For the remainder of these exercises, you are required to use seaborn, but select at least two aspects of your plot to make them unique. It could be the color of the point, size, background, grid, etc. etc. There are many choices. Use these exercises to learn about this wonderful visualization framework, and to tap into the artist in you!

31) [P] Show a scatterplot of both `df_uniform` and `df_normal` side by side on the same figure

32) [P] Show the distribution of only the x variable for both `df_uniform` and `df_normal`, with a density curve and a rugplot at the bottom. (Look at `sns.distplot`)

33) [M] What is a `jointplot` in Seaborn?

34) [P] Use `sns.jointplot` to show the bivariate distribution of `x` and `y` for `df_uniform` and `df_normal`

35) [P] Show a hexbin plot using `sns.jointplot` for `df_normal`

Part III – Some basic data preprocessing

36) [P] Create an additional variable in `df_uniform` called `x_fac1` that represents a factor with 3 levels, "X1", "X2", and "X3". You should *discretize* according to equal width bins over the distribution of x. (Divide the range of x into three.)

37) [P] Create an additional variable in `df_uniform` called `x_fac2` that represents a factor with 3 levels, "X1", "X2", and "X3". This time, you should discretize using equal depth bins over the distribution of x. Select your division criteria such that there are an equal number of data in each bin. Verify that the distribution of your

data each has the same number of data (within 1).

38) [P] Create a side by side scatter plot showing the distribution of `df_uniform`, using `x_fac1` as the color for one plot, and `x_fac2` as the color for your other plot.

Deliverables

Commit and push `lab05.ipynb`. Be sure you have every cell run, and output generated. Verify that your file is pushed properly on Gitlab.