KEY_Lesson18_LineGraph_ScatterPlot

February 4, 2020

1 Matplotlib: line Graph and Scatter Plot

1.0.1 Line Graph

In the earlier lesson we learned that line graphs are used to simply show changes over time. In this lesson, we want to use line graphs to compare multiple variables over time.

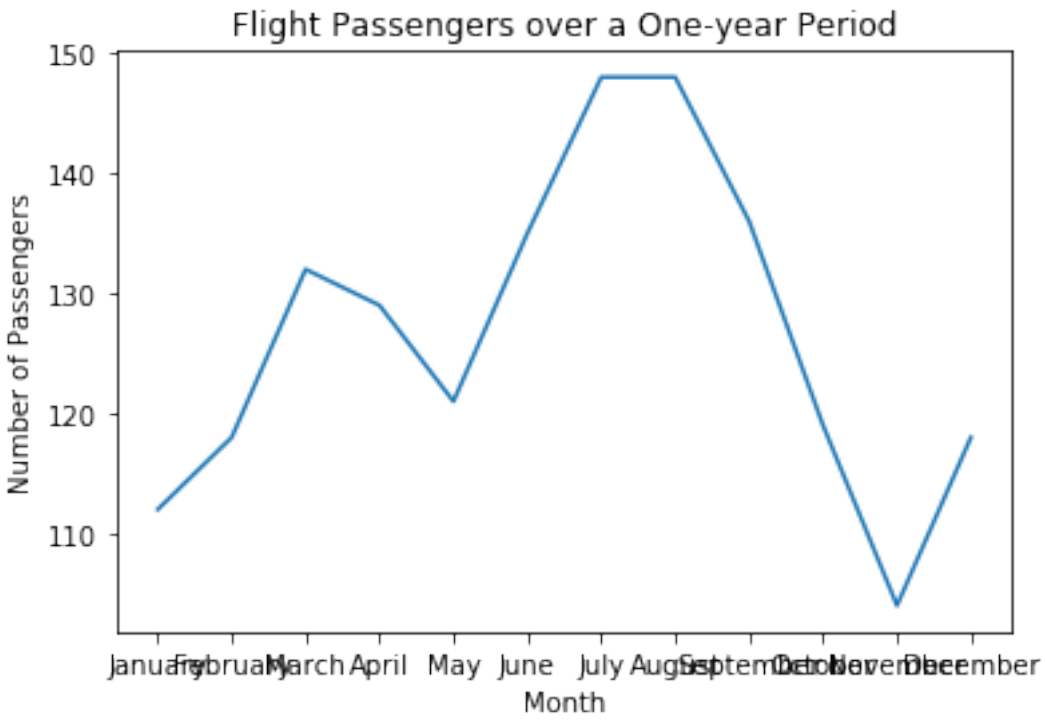
First, let's import `matplotlib.pyplot` and `seaborn`and load `flights` data as we did earlier today.

[0]:
```
# import packages
import matplotlib.pyplot as plt
import seaborn as sns
flights = sns.load_dataset("flights")
```

In this example, rather than looking at the trend over years, we are interested in how the number of passengers changes over a one-year period.

[2]:
```
# select all 12 observations for year 1949 and assign this subset to a new␣
 ↪dataframe named "flights_1949"
# use plt to plot the number of passengers for each month of year 1949.
# add labels/title to the plot
flights_1949 = flights[flights['year'] == 1949]
plt.plot(list(flights_1949["month"]), flights_1949["passengers"])
plt.title("Flight Passengers over a One-year Period")
plt.xlabel("Month")
plt.ylabel("Number of Passengers")
# HINT: X-axis correspond to a string variable for month names, e.g. January,␣
 ↪February, ...
# HINT: To get the X-axis values, you can use a list variable
```

[2]:
```
Text(0, 0.5, 'Number of Passengers')
```

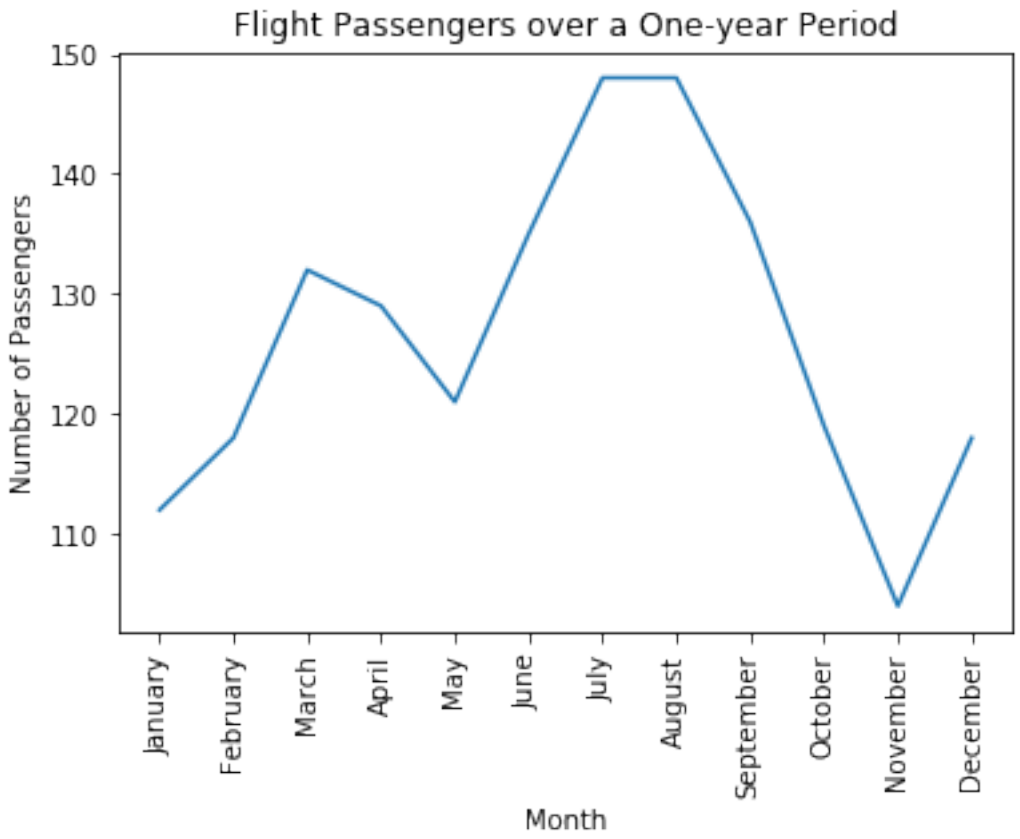


How do you think we can untangle values on x-axis?

One way is to rotate the x-axis text. To do so, we use `plt.xticks` function to get access to positions that ticks (i.e. x-axis text) should be placed. Then, we can simply set rotation of x ticks to any value we like.

[3]:
```
# reproduce the above plot
# use plt.xticks to rotate the x-axis text by 90 degrees
# add labels/title
plt.plot(list(flights_1949["month"]), flights_1949["passengers"])
plt.title("Flight Passengers over a One-year Period")
plt.xlabel("Month")
plt.ylabel("Number of Passengers")
plt.xticks(rotation=90)
```

[3]:
```
([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11],
 <a list of 12 Text xticklabel objects>)
```



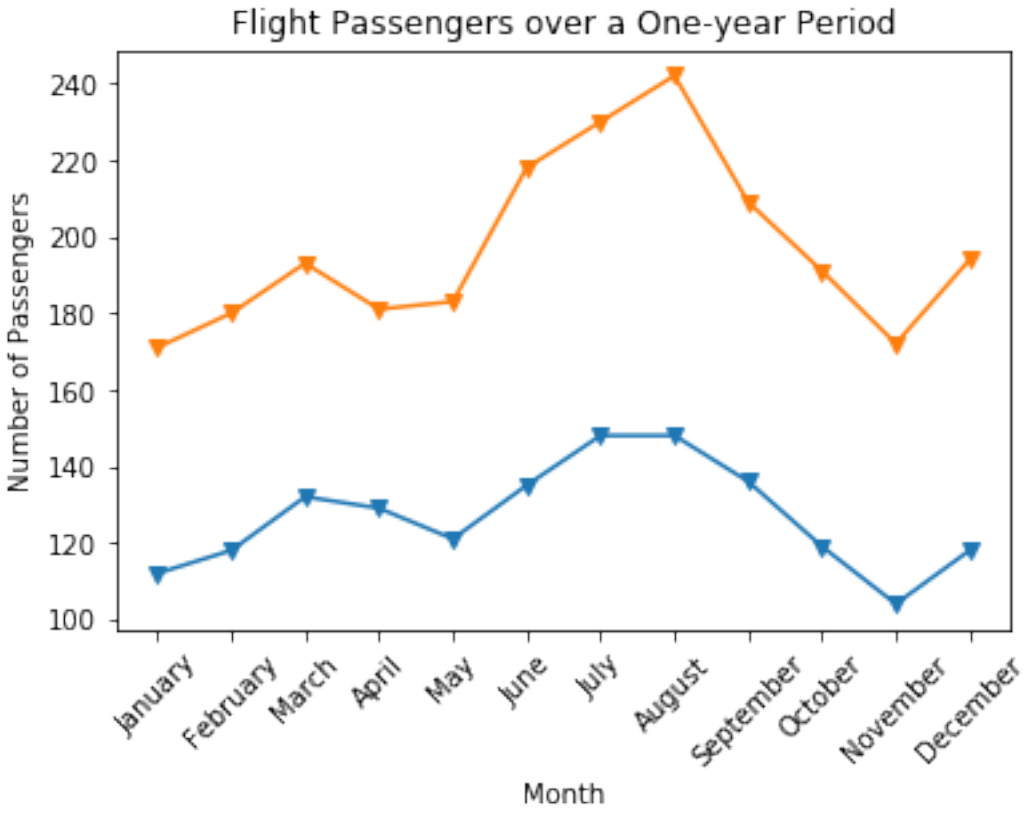
Nice plot!

Next, we want to compare the number of passengers in 1949 and 1952 on a monthly basis. For that we are going to show two line graphs on the same plot. We do this by calling the `plt.plot` function twice - once for the 1949 data and once for the 1952 data. By doing this, we are essentially overlaying these data plots over the base graph.

[4]:
```
# select all 12 samples of year 1952 and assign it to a new dataframe named␣
 ↪"flights_1952"
# to overlay the monthly number of passenger for years 1949 and 1960 in one␣
 ↪plot:
# 1st, plot the flights_1949 dataframe
# next, plot the flights_1952 dataframe
# don't forget to add triangular shape markers when plotting each of the two␣
 ↪datafeames
# rotate the x-axis text by 45 degrees
# add labels/title to the plot
flights_1952 = flights[flights['year'] == 1952]
plt.plot(list(flights_1949["month"]), flights_1949["passengers"], marker = 'v')
plt.plot(list(flights_1952["month"]), flights_1952["passengers"], marker = 'v')
plt.title("Flight Passengers over a One-year Period")
```

```
plt.xlabel("Month")
plt.ylabel("Number of Passengers")
plt.xticks(rotation=45)
```

[4]:
```
([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11],
 <a list of 12 Text xticklabel objects>)
```



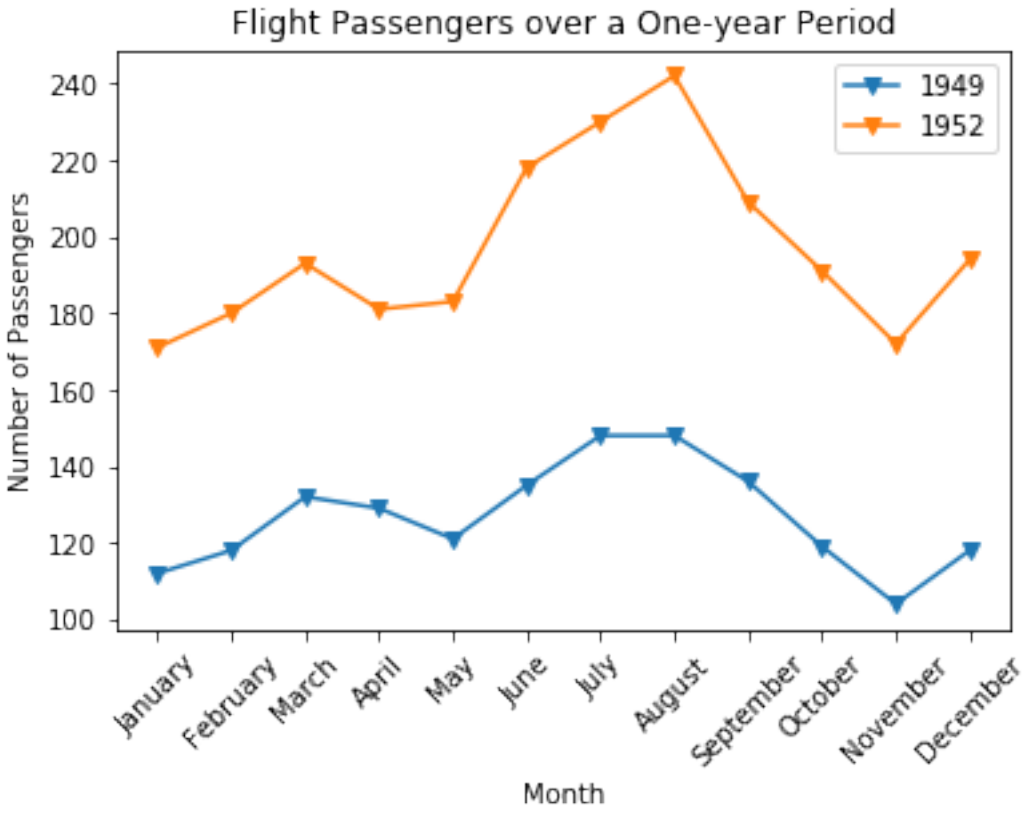
There is one final step left to have our perfect plot! Well, by only looking at this plot can you tell which line corresponds to year 1949 and which line corrsponds to year 1952?

To address this issue, we add legend to the plot by simply adding a `label` argument when plotting each of two line graphs, and then calling `plt.legend` function to show the defined labels.

[5]:
```
# reproduce the above plot
# add appropriate legends to the plot
flights_1952 = flights[flights['year'] == 1952]
plt.plot(list(flights_1949["month"]), flights_1949["passengers"], marker = 'v',␣
 ↪label = '1949')
plt.plot(list(flights_1952["month"]), flights_1952["passengers"], marker = 'v',␣
 ↪label = '1952')
plt.legend()
```

```
plt.title("Flight Passengers over a One-year Period")
plt.xlabel("Month")
plt.ylabel("Number of Passengers")
plt.xticks(rotation=45)
```

[5]:
```
([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11],
 <a list of 12 Text xticklabel objects>)
```



Does this plot give you any insight into the data? Do you see the same trend in the number of passengers during year 1949 and 1952? Which months have the maximum and minimum number of passengers? Can you guess why?

1.0.2 Scatter plot

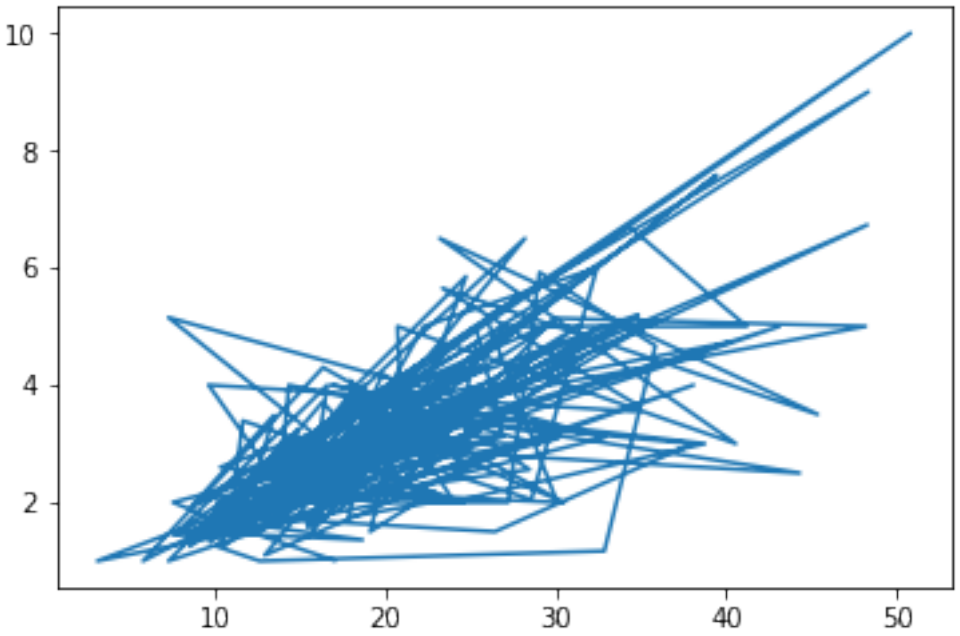
Before explaining the scatter plot, let's find out in what cases a line graph is not helpful!

[0]:
```
# load "tips" dataset from seaborn package and assign it to a variable called␣
 ↪tips
tips = sns.load_dataset("tips")
```

This dataset includes the amount that servers receive in tips in restaurants ,`tip`, based on a few factors including the total bill, `total_bill`.

[7]:
```
# use plt.plot to plot the amount of tip on the y-axis versus the totall bill␣
↪on the x-axis
plt.plot(tips["total_bill"], tips["tip"])
```

[7]:
```
[<matplotlib.lines.Line2D at 0x7f2eec789550>]
```

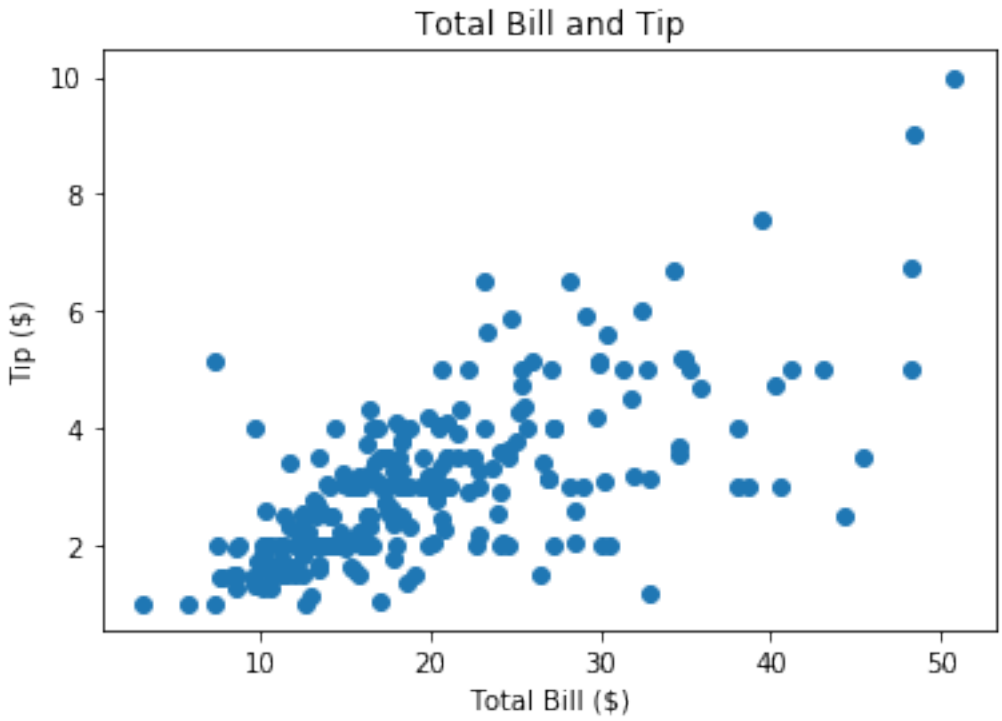


Do you see the problem?

Yes, you're correct! `plt.plot` function, by default, thinks this is a time series data and connects consecutive samples by lines. To avoid this problem, we use scatter plots, which simply plots points without connecting lines to visualize observations and illustrate the relationship between two variables. To create a scatter plot, use `plt.scatter` function. The same as `plt.plot` function,`plt.scatter`'s first variable correponds to the measurement we would like to show along the x-axis, and the second argument correspond to the measurement along the y-axis.

[8]:
```
# show the amount of tip on the y-axis versus the total bill on the x-axis␣
↪using a scatter plot
# add labels/title
plt.scatter(tips["total_bill"], tips["tip"])
plt.title("Total Bill and Tip")
plt.xlabel("Total Bill ($)")
plt.ylabel("Tip ($)")
```

[8]: `Text(0, 0.5, 'Tip ($)')`



Of course you can change the size, marker style, color of points, as well as make them tranparent. Google how you can play with these arguments!

In this lesson you learned to: * improve a line graph plot by overlaying multiple lines * add legends to the plot * when to use a scatter plot rather than a line graph * generate a scatter plot