Data Visualization

Hasan Can Aslan

Submission Date: 10th of May 11:59 PM

1 Introduction

Each day, individuals across the globe collect terabytes of digital data for different purposes. Although the data vary in nature, we can think of data in their raw form as resources that lead us to new understandings. However, raw data could be noisy.

Data visualization is the process of mapping, graphing, plotting, and charting data to discover relationships and patterns. We will be creating data visualizations using `matplotlib` and `pandas` throughout this project. Your assignment consists of two parts: Ahmet's Lemons and Ceren's Garden Center. References and warm up exercises about related packages are provided before each part.

1.1 Submission

We will use *OK* platform for assignment submissions. We already registered you, you should be able to see our class when you sign in using your Koç University mail address. Please contact us immediately if you have any issues. You should submit two files named cerens_garden_center.py and ahmets_lemons.py.

1.2 Academic Honesty

Koç University's *Statement on Academic Honesty* holds for all the homework given in this course. Failing to comply with the statement will be penalized accordingly. If you are unsure whether your action violates the code of conduct, please consult with your instructor.

1.3 Aim of the Project

In your homework assignments, the **functionality** and **style** of your programs are **both important.** A program that "works" is not necessarily a good program. A good program is **comprehensible, readable and well structured.** Therefore you're expected to decompose the main problem into **simpler subtasks** and **implement helper functions** for these subtasks. You're also expected to write comments if needed, be careful about **indentation**, and use **descriptive names** for helper functions. Even if your program functions well according to the project requirements, you can still improve your code style and how you decompose the task. The goal of this project is to teach you how to see the meaning of a dataset and convey that information to others using Python.

1.4 Further Questions

For further questions **about the project** you may contact **Hasan Can Aslan** at [haslan16@ku.edu.tr]. Note that it may take up to 24 hours before you receive a response so please ask your questions **before** it is too late. No questions will be answered when there is **less than two days** left for the submission.

2 Tasks

2.1 Pandas Reference

This part consists of several functions and usages of `pandas`. Before proceeding to your assignment please read this part carefully. You **don't need to submit** any file for this part but we strongly recommend trying it on your own. Your next task will be built on these functions.

2.1.1 Installing Pandas

To use `pandas`, you need to install it from the PyPI. You can install it by writing your terminal following line:

```
python -m pip install -U pandas
```
For Mac users:
```
python3 -m pip install -U pandas
```

2.1.2 Pandas

`pandas` is an open-source library that is used to analyze data in Python. It takes in data, like a CSV or SQL database, and creates an object with rows and columns called a data frame. `pandas` are typically imported with the alias `pd`.

```
import pandas as pd
```

2.1.3 Pandas DataFrame Creation

The fundamental `pandas` object is called a DataFrame. It is a 2-dimensional size-mutable, potentially heterogeneous, tabular data structure.

A DataFrame can be created in multiple ways. It can be created by passing in a dictionary or a list of lists to the `pd.DataFrame()` method, or by reading data from a CSV file.

```
# Ways of creating a Pandas DataFrame
# Passing in a dictionary:
data = {'name':['Anthony', 'Maria'], 'age':[30, 28]}
df = pd.DataFrame(data)

# Passing in a list of lists:
data = [['Tom', 20], ['Jack', 30], ['Meera', 25]]
df = pd.DataFrame(data, columns = ['Name', 'Age'])

# Reading data from a csv file:
df = pd.read_csv('students.csv')
```

2.1.4 Selecting Pandas DataFrame Rows Using Logical Operators

In `pandas`, specific rows can be selected if they satisfy certain conditions using Python's logical operators. The result is a DataFrame that is a subset of the original DataFrame.

Multiple logical conditions can be combined with OR (using `|`) and AND (using `&`), and each condition must be enclosed in parentheses.

```
# Selecting rows where age is over 20
df[df.age > 20]

# Selecting rows where name is not John
df[df.name != "John"]

# Selecting rows where age is less than 10
# OR greater than 70
df[(df.age < 10) | (df.age > 70)]
```

2.1.5 Pandas apply() Function

The `pandas apply()` function can be used to apply a function on every value in a column or row of a DataFrame, and transform that column or row to the resulting values.

By default, it will apply a function to all values of a column. To perform it on a row instead, you can specify the argument `axis=1` in the `apply()` function call.

```
# This function doubles the input value
def double(x):
  return 2*x

# Apply this function to double every value in a specified column
df.column1 = df.column1.apply(double)

# Lambda functions can also be supplied to `apply()`
df.column2 = df.column2.apply(lambda x : 3*x)

# Applying to a row requires it to be called on the entire
    DataFrame
df['newColumn'] = df.apply(lambda row:
  row['column1'] * 1.5 + row['column2'],
  axis=1)
```

2.1.6 Pandas DataFrames Adding Columns

`pandas` DataFrames allow for the addition of columns after the DataFrame has already been created, by using the format `df['newColumn']` and setting it equal to the new column's value.

```
# Specifying each value in the new column:
df['newColumn'] = [1, 2, 3, 4]

# Setting each row in the new column to the same value:
df['newColumn'] = 1

# Creating a new column by doing a
# calculation on an existing column:
df['newColumn'] = df['oldColumn'] * 5
```

2.2 Ceren's Garden Center

Now you're the data analyst for a chain of gardening stores called Ceren's Garden Center. As Ceren's business growing, getting harder to keep track of inventory. Help them analyze their inventory! Starter code-named `cerens_garden_center.py` for this part is provided. **You need to submit your answers for this part.**

2.2.1 Answer Customer Emails

- Data for all of the locations of Ceren's Garden Center is in the file `inventory.csv`. Load the data into a DataFrame called `inventory`.
- Inspect the first 10 rows of `inventory`.

  Hint: You can slice dataframe like strings. For example, `df[:5]` or `df.head(5)` will return first 5 rows.
- The first 10 rows represent data from your Istanbul location. Select these rows and save them to `istanbul`.
- A customer just emailed you asking what products are sold at your Istanbul location. Select the column `product_description` from `istanbul` and save it to the variable `product_request`.

  Hint: You can select a column by writing its name in square brackets like `df['product_description']`
- Another customer emails to ask what types of seeds are sold at the Bursa location.

  Select all rows where `location` is equal to Bursa and `product_type` is equal to `seeds` and save them to the variable `seed_request`.

  Hint: You can look at section 2.3.4 for examples about filtering.

2.2.2 Inventory

- Add a column to `inventory` called `in_stock` which is `True` if `quantity` is greater than 0 and `False` if `quantity` equals 0.

  Hint: You can use the code in part 2.3.5 as an example with following changes: paste lambda below in `inventory.apply()`, and set its axis to `axis=1` to apply for each row.

```
lambda row: row['quantity'] > 0
```

- Ceren's Garden Center wants to know how valuable their current inventory is.

  Create a column called `total_value` that is equal to `price` multiplied by `quantity`.

  Hint: You can use the code in part 2.3.5 and task above as an example apply following lambda and set its axis to `axis=1` to apply for each row.

```
lambda row: row['price'] * row['quantity']
```

- The Marketing department wants a complete description of each product for their catalog.

  The following lambda function combines `product_type` and `product_description` into a single string:

```
combine_lambda = lambda row: '{} - {}'.format(row.product_type
    , row.product_description)
```

  Paste this function into `cerens_garden_center.py`.

- Using `combine_lambda`, create a new column in inventory called `full_description` that has the complete description of each product.

  Hint: You just need to write `combine_lambda` directly in the `inventory.apply()` and set its axis to `axis=1` to apply for each row.

2.3 Matplotlib Reference

This part consists of several functions of `matplotlib`. You can find a matplotlib cheat sheet at the **Resources** part at the end of the handout. You **don't need to submit** any file for this part, but we strongly recommend trying it on your own. Your next task will be built on these functions.

2.3.1 Installing Packages

To use `matplotlib`, you need to install it from PyPI (Python Package Index) using `pip` (Package installer for Python). `pip` comes pre-installed with modern distributions of Python. You can install `matplotlib` by writing your terminal following line:
`python -m pip install -U matplotlib`
For Mac users:
`python3 -m pip install -U matplotlib`

2.3.2 Subplots in Matplotlib

In Python, the `matplotlib`'s `pyplot.subplot()` the function can be used to create a figure with a grid of subplots. This function accepts the number of rows, number of columns, and the current index as arguments.

```
import matplotlib.pyplot as plt
# Datasets
x = [1, 2, 3, 4]
y = [1, 2, 3, 4]

# First Subplot
plt.subplot(1, 2, 1)
plt.plot(x, y, color='green')
plt.title('First Subplot')

# Second Subplot
plt.subplot(1, 2, 2)
plt.plot(x, y, color='blue')
plt.title('Second Subplot')

# Display both subplots
plt.show()
```

2.3.3 Figures in Matplotlib

In `matplotlib` package, a figure is a container that holds plots. It can hold a single plot or multiple plots. When a figure holds multiple separate plots, those are called subplots.

2.3.4 Setting Linestyle and Color in Matplotlib

In `matplotlib` package, the `pyplot.plot()` function can accept parameters to set the color(`color`), linestyle(`linestyle`) and marker(`marker`) for line graph. Color values can be HTML color names or HEX codes. Line styles can be dashed(`'--'`) or dotted(`'..'`). Markers can be circles(`'o'`), squares(`'s'`), stars(`'*'`), or other shapes.

```
pyplot.plot(days, money_spent, color='green', linestyle='--')
pyplot.plot(days, money_spent_2, color='#AAAAAA',  marker='o')
```

2.3.5 Pyplot Functions

The `matplotlib` package contains the pyplot module, which provides users with an interface for graphing data. Pyplot contains over 100 functions, from acorr to yticks. You must import the module, and we recommend using `plt` to refer this module for your convenience.

```
from matplotlib import pyplot as plt
```

Here are some of the most common pyplot functions:

Function	Description
`plot`	plots y versus x as lines and/or markers
`show`	displays a figure
`axis`	sets some axis properties
`xlabel`	sets the label for the x-axis
`ylabel`	sets the label for the y-axis
`title`	sets a title for the axes
`subplot`	adds a subplot to the current figure
`subplots_adjust`	tunes the subplot layout
`legend`	places a legend on the axes
`figure`	creates a new figure
`savefig`	saves the current figure

2.3.6 X-ticks and Y-ticks in Matplotlib

In Python's `matplotlib`, the x-tick and y-tick marks of the plot can be changed using functions `ax.set_xticks()` and `ax.set_yticks()`. These functions accepts an array of values representing tick mark positions.

```
import matplotlib.pyplot as plt

ax = plt.subplot()
plt.plot([0, 1, 2, 3, 4], [0, 1, 4, 9, 16])
plt.plot([0, 1, 2, 3, 4], [0, 1, 8, 27, 64])
ax.set_xticks([1, 2, 4])
```

2.4 Ahmet's Lemons

In this part of the assignment, you will be acting as a data analyst for Ahmet who wants to start online retailing for his lemon business called Ahmet Lemons. People all over the country can use this product to get the freshest, best-quality lemons delivered to their door. However, Ahmet doubts this job will work and he would like to gain insight into the customers and their habits. Using `matplotlib`, you'll create some different line graphs to show this information effectively. Starter code-named `ahmets_lemons.py` for this part is provided. **You need to submit your answers for this part.**

2.4.1 Importing Data

You will use `pandas` package to import data into your code like you've done at Ceren's Garden Center part.

- Data for sales and visitors of the Ahmet's Lemons is in the file `lemon_data.csv`. Load the data into a DataFrame called `lemon_data` using `pandas`.
- Save every column's data from DataFrame called `lemon_data` to related variables as `months`, `visits_per_month`, `citron_lemons_per_month`, `sweet_lemons_per_month` and `lisbon_lemons_per_month`.

  Hint: You can select a column by writing its name in square brackets like `lemon_data['months']`

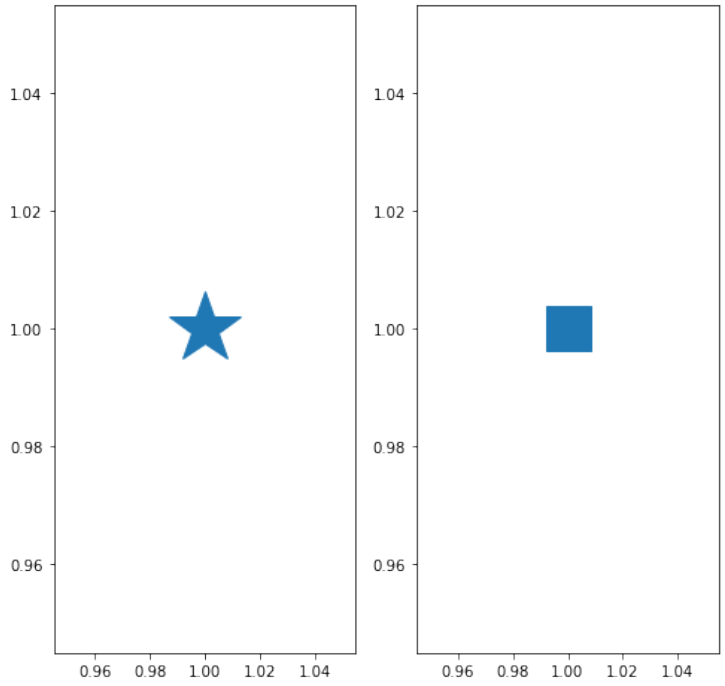
2.4.2 Understand the Data and Set Up Subplots

```
1 from matplotlib import pyplot as plt
```

This line of code provided allows you to use `matplotlib` library in your project. You don't need to create any object instance, you can use functions directly by accessing the `pyplot` module. For example, `plt.show()`.

- We have provided some data in different lists in **ahmet_lemons.py**. Look through these lists and try to understand what each one represents.
- Create a figure of width 12 and height 8.
  Hint: You can use `plt.figure` and write as an argument the `figsize=(width, height)` to be width and height.
- We are going to make two charts in one figure, laid out side-by-side. In other words, the figure will have one row and two columns, like figure below.
- Write the command to create the left subplot (the one that would correspond to the plot with a star in our example figure). Save this subplot in a variable called `ax1`.
  Hint: First, you will have to create the left subplot. Calling `plt.subplot(1,2,1)` before calling `plt.plot()` would put a plot in the first column of a 2 column layout with one column.

  How would you select the first column of a 2 column layout with one row, instead? What will happen if you change 3rd argument of subplot? i.e. plt.subplot(1,2,2).



- Write the command to create the right subplot (the one that would correspond to the plot with a square in our example figure).

  Save this subplot in a variable called `ax2`.

2.4.3 Page Visits Over Time

In the left subplot, we are going to plot the total page visits over the past year as a line.

- First, let's create the list of x-values, which is `range(len(months))`. Store this in a variable called `x_values`
- Make sure this happens after the line where you created `ax1`, but before the line where you've created `ax2`, so that the plot goes in the subplot on the left.
- Plot the total page visits against these `x_values` as a line.
- Give the line markers that will help show each month as a distinct value. **Hint:** Remember that you can change the marker style inside the call to `plt.plot` by setting `marker=<style>` (with `<style>` representing any of the marker types).
- Label the x-axis and y-axis with descriptive titles of what they measure. **Hint:** `plt.xlabel` and `plt.ylabel` will be useful for this.
- Set the x-axis ticks to be the `x_values`.

- Label the x-axis tick labels to be the names stored in the `months` list.
  Hint: The `set_xticklabels` function of `ax1` will help you do this task.

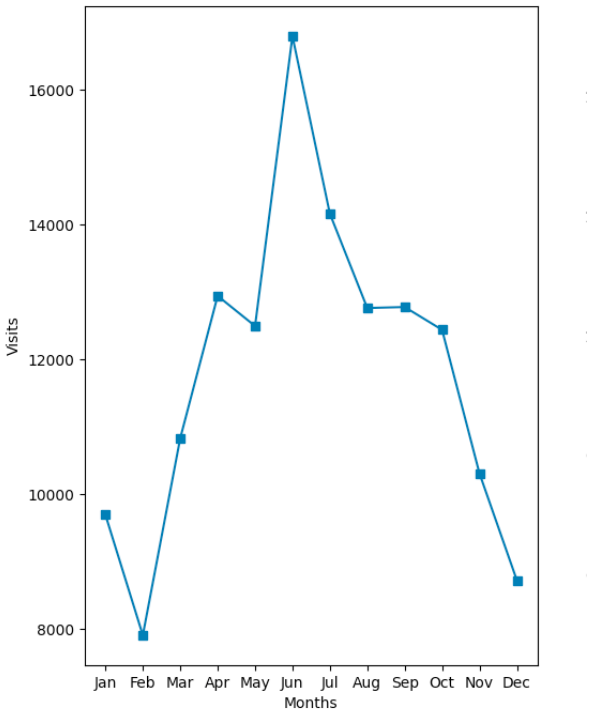


Figure 1: Your left subplot should look like this figure.

2.4.4 Plotting Multiple Lemon Species

In the subplot on the right, we are going to plot three lines on the same set of axes. The x-values for all three lines will correspond to the months, so we can use the list of `x_values` we used for the last plot.

- On one plot, create the three lines:
  - number of citron lemons sold vs `x_values`
  - number of Lisbon lemons sold vs `x_values`
  - number of sweet lemons sold vs `x_values`

  Make sure this happens after the line where you created `ax2`, so that it goes in the subplot on the right.
- Give each line a specific color of your choosing.
- Add a legend to differentiate the lines, labeling each lime species.
- Set the x-axis ticks to be the `x_values`, and the tick labels to be the `months` list.

2.4.5 Labeling and Saving

- Add a title to each of the two plots you've created, and adjust the margins to make the text you've added look better.
- Now, save your figure as a *.png* with a descriptive file name.

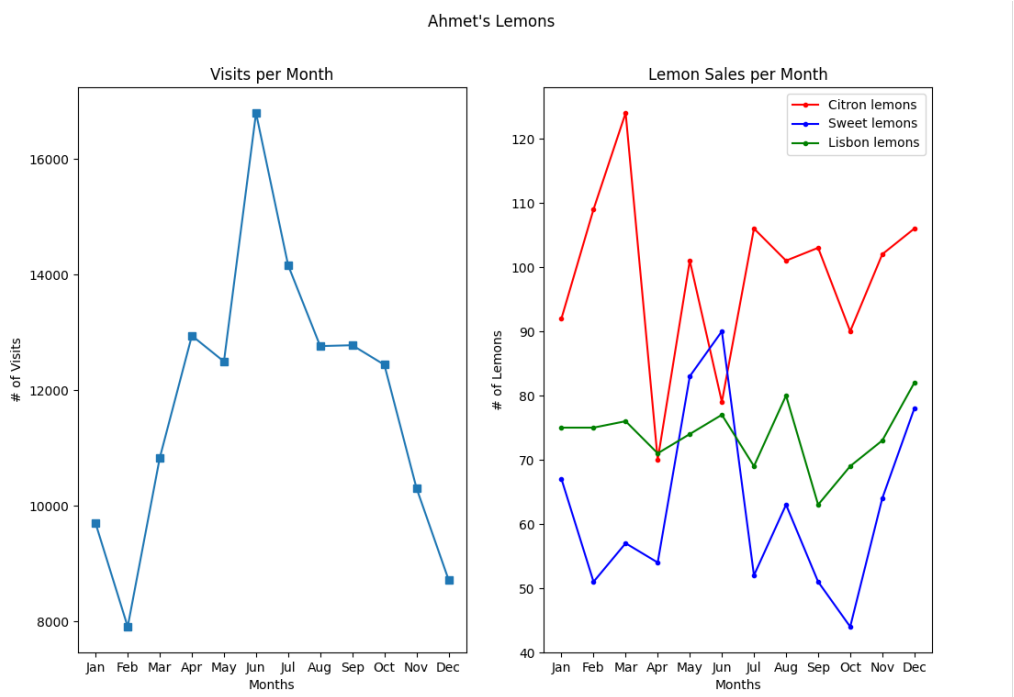


Figure 2: Your final plots should look like these figures.

2.5 End of Project

Your project ends here. You may continue to tinker with the code to implement any desired features and discuss them with your section leader. However, **do not** include any additional features that you implement after this point into your submission. You are just required to submit part 2.2 and 2.4 which are `cerens_garden_center.py` and `ahmets_lemons.py`.

3 Resources

You can find all course and section related material in our website. You can download related course slides from here. We've also created a *matplotlib* cheatsheet below which can be helpful while implementing the assignment. You can also refer to the official tutorials of *matplotlib* and *pandas*.

3.1 Matplotlib Cheatsheet

- `plt.plot(x_values, y_values)`

  Creates a line. `x_values` and `y_values` are lists of numbers (i.e., `[1, 2, 3, 4]`). Also accepts the following keyword arguments:

  - `marker`: a symbol that will appear at each (x, y) point. Options include '*' for a start, 'o' for a circle, or 's' for a square.
  - `linestyle`: whether the line is solid ('-') or dashed ('–' or ':') or no line at all ('').
  - `linewidth`: a number representing the thickness of the line; default is 1.
  - `color`: the color of the line (can be a HEX code or any html color name).

- `plt.show()`

  Displays any previous plot commands.

- `plt.close('all')`

  Closes all previous figures.

- `plt.figure(figsize=(width, length))`

  Creates a new figure with a specified length and width. `width` and `length` are both numbers in inches.

- `plt.title('My Chart Title')`

  A title for a chart.

- `plt.xlabel('My X-Label')`

  A label for the x-axis.

- `plt.ylabel('My Y-Label')`

  A label for the y-axis

- `plt.subplot(num_rows, num_cols, subplot_index)`

  Creates a subplot for a grid with `num_rows` rows and `num_cols` columns. The new subplot is at a position defined by `subplot_index`. For instance, `plt.subplot(2, 3, 4)` would create a grid with 2 rows and 3 columns and would create a plot in the second row and the first column (4 "steps" if moving left to right and top to bottom).

- `ax = plt.subplot()`

  Creates and axes object (`ax`) that can be used for adjusting the position and labeling of x- and y-axis tick marks.

- `plt.legend(['label1', 'label2'])`

  Creates a legend using the labels given.

- `plt.legend()`

  Creates a legend using any `label` keywords that were given in `plt.plot` commands.

- `ax.set_xticks([0, 1, 2, 3, 4])`

  Creates tick marks at positions `[0, 1, 2, 3, 4]` on the x-axis.
  Requires that you created an axes object by using `ax = plt.subplot()`.

- `ax.set_yticks([0, 1, 2, 3, 4])`

  Creates tick marks at positions `[0, 1, 2, 3, 4]` on the y-axis.
  Requires that you created an axes object by using `ax = plt.subplot()`.

- `ax.set_xticklabels(['label1', 'label2', 'label3'])`

  Modifies the first three labels of the x-axis ticks marks to be `'label1'`, `'label2'`, `'label3'`
  Requires that you created an axes object by using `ax = plt.subplot()`.
  You'll probably want to start by specify the positions of your x-ticks using `ax.set_xticks`.

- `ax.set_yticklabels(['label1', 'label2', 'label3'])`

  Modifies the first three labels of the y-axis ticks marks to be `'label1'`, `'label2'`, `'label3'`
  Requires that you created an axes object by using `ax = plt.subplot()`.
  You'll probably want to start by specify the positions of your y-ticks using `ax.set_xticks`.

- `plt.bar(x_values, heights)`

  Creates a bar chart with bars at each value of `x_values` using the heights given in `heights`. If you're only creating one bar chart, `x_values` can be equal to `range(len(heights))`.

- `plt.bar(x_values, heights, bottom=other_heights)`

  Creates a second bar chart stacked on top of another bar chart whose heights are `other_heights`.