BHOLA PRASAD

MATPLOTLIB UNLOCKED

Creating Beautiful Charts in Python

Matplotlib Unlocked

Creating Beautiful Charts in Python

Bhola Prasad

This book is available at https://leanpub.com/matplotlib-python

This version was published on 2025-07-05

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

© 2025 Bhola Prasad

To my Family and Friends

Contents

© 2024 Bhola Prasad AdaCode.io

© 2024 Bhola Prasad AdaCode.io

© 2024 Bhola Prasad AdaCode.io

© 2024 Bhola Prasad AdaCode.io

Chapter 1. Introduction

What is matplotlib?

Matplotlib is the most important library for creating data visualization in Python. It was originally developed by John D. Hunter in 2003. One of the key features of Matplotlib is its ability to produce a wide range of charts such as Line charts, bar charts, histograms, box plots and many other kind of plots. This versatility makes it a go-to tool for Data scientists, Engineers, and Analysts looking to visualize their data in Python.

Matplotlib integrates very well with many Data science libraries in Python such as Pandas, and Numpy making it an essential part of data analysis workflow in Python. Matplotlib provides lots of features and ways to create and modify charts in Python. Many data visualization libraries in Python are built on top of Matplotlib making it even more important to understand and get comfortable with it.

Why you should read this book?

Matplotlib Unlocked: Creating Beautiful Charts in Python is a comprehensive resource to learn and master the Matplotlib library in Python for Data Visualization and Insights. You will learn the tips and best practices of creating charts in matplotlib.

Whether you are taking your first step in data visualization or are already an advanced user looking for in-depth knowledge of matplotlib's capabilities, this book has something for everyone.

What sets this book apart is its approach to teaching. Rather than merely walking you through the functions and features of matplotlib, it empowers you by:

- Offering hands-on tutorial and exercises that teach you how to apply what you learned so you become more confident in using matplotlib for your data visualization tasks.
- Instead of using fake or dummy data, you will learn from practical real-world data and apply those skills to find hidden insights.
- You will learn to create charts incrementally. You will start from scratch and as you progress through more chapters and exercises, you will learn how to add more layers to your charts for complex visualization.
- You will learn all the skills required to create, modify, and present data that have the most impact on your audience.

If you ever felt intimated by the matplotlib library then join me in this journey and I will show you how to master it and get all the praise you deserve for your work.

Happy visualizing! May your data tell compelling stories.

Prerequisites

This book assumes that you have some Python programming experience and that you are familiar with Python's main data analysis and manipulation library Pandas.

If you don't know Python and Pandas yet, there are many great websites where you can learn them and are completely free. keep an eye on our new website AdaCode.io[1] where we will publish more tutorials to get you started with Python, Pandas and many

[1] https://adacode.io/

© 2024 Bhola Prasad AdaCode.io

other topics in Data Science and Programming in Python and other languages like javascript.

Installation

This book uses Python 3. please do not use Python 2, otherwise things not work as expected. There are various ways to install Python and Matplotlib but the easiest way is through Anaconda. We will use this for installtion.

Anaconda

Anaconda is a Python and R distribution for scientific computing which simplifies package installation and management. Anaconda comes with many libraries that Data scientists and Analysts need to do their work.

The process of installing Anaconda on Mac and Windows is almost same and since we will use a graphical user interface, installing it on either Mac or Windows is not that hard. If you encounter any problems then please search on Google (best way) or you can also email me at bholaprasad26@gmail.com

First, visit the Anaconda Website[2]

Then click on the Download button.

[2] https://www.anaconda.com/download

© 2024 Bhola Prasad AdaCode.io

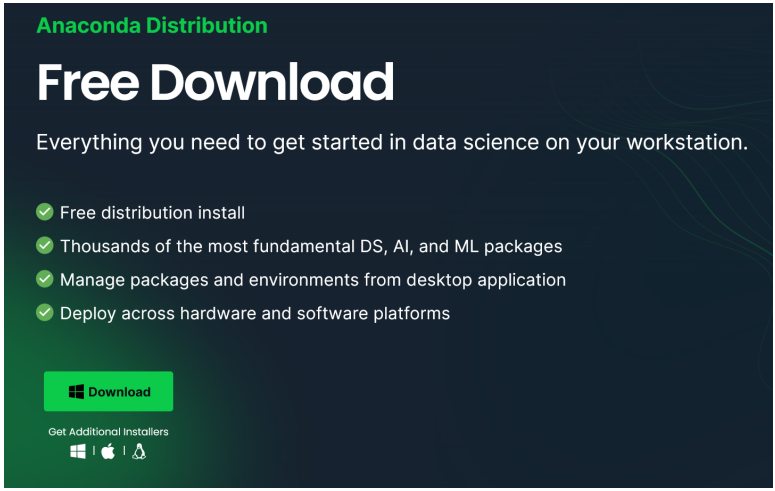

Figure 1. Download Python

This will download an executable file. Double click on it to start the installation process. Follow the instructions to install it as you do with any regular software on your machine.

Check installation

To verify that Python and Matplotlib get installed on your machine, run the following codes in Command prompt (windows) or Terminal (Mac or Linux). Don't include the `$` sign and the `>>>` sign.

```
$ python --version
'Python 3.11.5'

$ python
>>> import matplotlib
>>> print(matplotlib.__version__)
3.8.0
```

© 2024 Bhola Prasad AdaCode.io

Download code

The source code for this book's examples is available at the following Github repository: matplotlib-python-book[3] For those who are new to Git and Github, look for the download zip file from the `<code>` dropdown button at the top right-hand side. Those who are experienced with Git and Github can clone the repository from the command line. The instructions for cloning the project is included in the README.md file.

This repository also contains all the datasets used in this book. The data are in the `data` folder and the python notebooks are in the `notebooks` folder.

Jupyter notebook

Jupyter Notebook is an open-source web application that allows you to create and share documents that contains Python code, visualizations, Math equations, and texts. It is widely used in data science community for data cleaning and analysis.

Jupyter Notebook comes installed with Anaconda distribution so you don't need to install it separately.

To launch the jupyter notebook run the following command in Terminal.

```
$ jupyter notebook
```

This command will start the jupyter Notebook server and open it in your default web browser. It will look something like this.

[3] https://github.com/bprasad26/matplotlib-python-book

© 2024 Bhola Prasad AdaCode.io

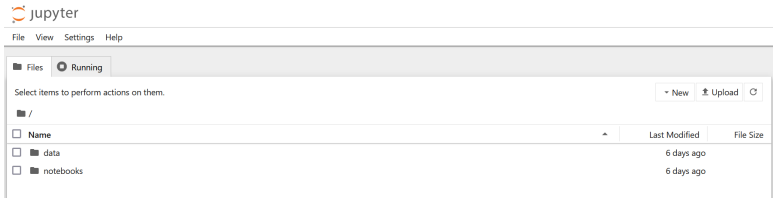

Figure 2. Jupyter Notebook

Click on the New dropdown menu at the top right corner and select notebook to create a new notebook.

To run a Python code type the following code in a notebook cell and hit the `run button` or press `Shift + Enter` on your keyboard.

```
1  print("Hello, World!)
```

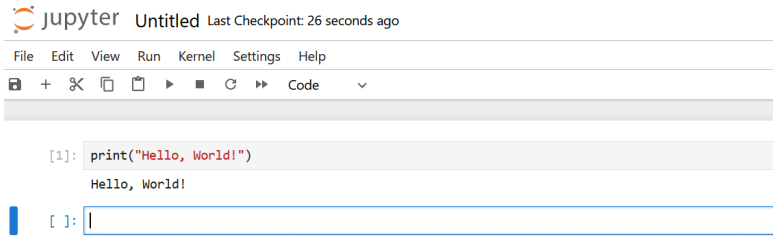

Figure 3. Running Python code in Notebook

To write text use the Markdown cell. Click on the Code dropdown menu at the toolbar and select `Markdown` or press `ESC` key and then `m`. This will change the code cell to a Markdown cell. To change a Markdown cell back to a code cell, use the dropdown menu or press `ESC` and then `y`.

© 2024 Bhola Prasad AdaCode.io

Chapter 2. Line Charts

A Line chart shows how a variable's value changes over time. The x-axis represents the time frame, such as months or years. The y-axis represents the value we are interested in measuring, like revenue or the number of visitors to a website. We can also create multiple line charts to compare related groups over time, such as the population of India, China, and the US in the past 20 years.

Line charts are handy for identifying trends in data over time. The continuous lines make it easy to see how value increases, decreases, or remain stable over time.

Read Data

We will use the Pandas library for data manipulation and analysis. Let's import it and read a dataset using the `read_csv()` function. We will also convert the `Date` column to `datetime` using the `parse_dates` parameter.

```
# silence warnings
import warnings
warnings.filterwarnings("ignore")
# import pandas library
import pandas as pd

# read the data into Pandas dataFrame
nvidia = pd.read_csv("../data/NVDA.csv", parse_dates=["Da\
te"])
nvidia.head()
```

	Date	Open	High	Low	Close	Adj Close	Volume
0	2019-02-19	39.227501	39.972500	39.035000	39.160000	38.855827	55189200
1	2019-02-20	39.455002	40.314999	39.342499	39.637501	39.329613	54098800
2	2019-02-21	39.764999	40.012501	38.794998	38.942501	38.640015	44854800
3	2019-02-22	39.465000	39.987499	39.327499	39.797501	39.488373	40174000
4	2019-02-25	40.639999	41.320000	39.584999	39.672501	39.364349	65602000

Figure 4. Nvidia Stock Prices Data

Create A Line Chart

To create a Line chart in Matplotlib, We use the plt.plot()[1] function and the plt.show()[2] function to show a figure.

Before creating any chart in Matplotlib, we must import the matplotlib library first using `import matplotlib.pyplot` as plt. Let's create a line chart with `Date` on the x-axis and `Adj Close` on the y-axis.

```
# import matplotlib library
import matplotlib.pyplot as plt

# create a line chart
plt.plot(nvidia["Date"], nvidia["Adj Close"])
plt.show()
```

[1] https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.plot.html
[2] https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.show.html

© 2024 Bhola Prasad AdaCode.io

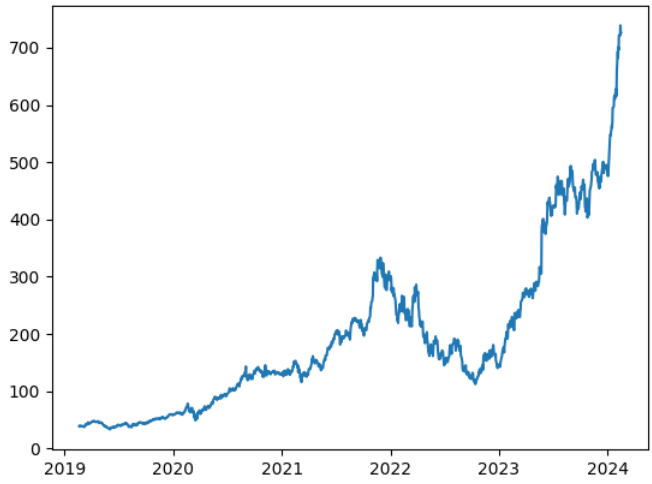

Figure 5. Line Chart in Matplotlib

Title and Axis Labels

Let's add a title to the plot using the plt.title()[3] and x and y-axis labels using the plt.xlabel()[4] and plt.ylabel()[5] respectively.

[3] https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.title.html
[4] https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.xlabel.html
[5] https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.ylabel.html

© 2024 Bhola Prasad AdaCode.io

```
# Line chart with title and axis labels
plt.plot(nvidia['Date'], nvidia['Adj Close'])
plt.title("Nvidia Stock Prices (2019-2024)")
plt.xlabel("Year")
plt.ylabel("Adj Close Price")
plt.show()
```

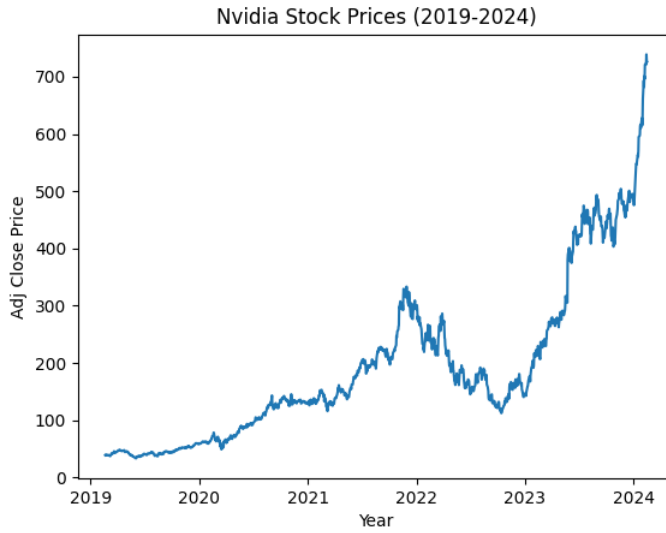

Figure 6. Line chart with title and axis labels

We can see that the share price of Nvidia was constantly going up from 2019-22, and then started to go down between 2022-23. It then rose again and went very high at the beginning of 2024.

The reason for such a hike in Nvidia stock prices is the explosion of Artificial Intelligence and their revenue from selling GPUs, which are needed to train large machine learning models. Nvidia controls more than 95% of the GPU market.

© 2024 Bhola Prasad AdaCode.io

Multiple Line Charts

We need to add another `plt.plot()` function to create multiple line charts on the same figure. This will instruct matplotlib to create another line with the same x and y-axis as the first figure.

Let's read some more data for this.

```
# read data
netflix = pd.read_csv("../data/NFLX.csv", parse_dates=["D\
ate"])
tesla = pd.read_csv("../data/TSLA.csv", parse_dates=["Dat\
e"])
```

We will create a line chart on the same plot for Nvidia, Netflix, and Tesla.

```
# Create Multiple Line charts
plt.plot(nvidia["Date"], nvidia["Adj Close"])

plt.plot(netflix["Date"], netflix["Adj Close"])

plt.plot(tesla["Date"], tesla["Adj Close"])

plt.title("Stock Prices (2019-2024)")
plt.xlabel("Year")
plt.ylabel("Adj Close Price")
plt.show()
```

© 2024 Bhola Prasad AdaCode.io

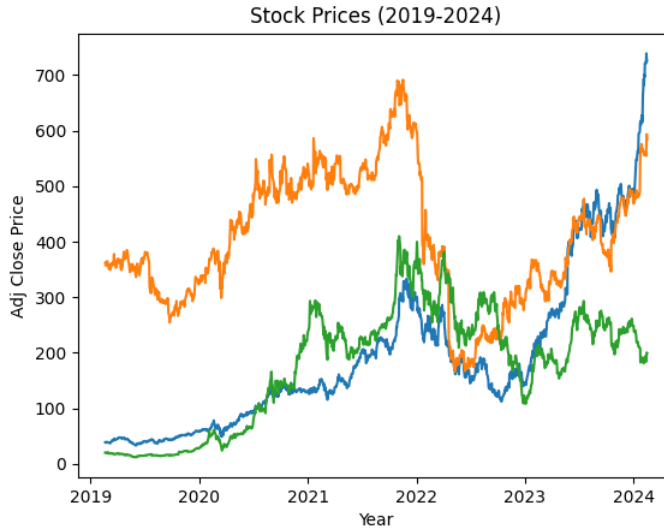

Figure 7. Multiple Line charts in Matplotlib

Legend

Although we plotted all the lines on the graph, which line corresponds to which company needs to be clarified. To solve this problem, we need to add a legend to the plot. First, we will add labels to each line and then use the plt.legend()[6] function to add a legend.

[6] https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.legend.html

© 2024 Bhola Prasad AdaCode.io

```
# Multiple Line charts with legend
plt.plot(nvidia["Date"], nvidia["Adj Close"], label="Nvid\
ia")

plt.plot(netflix["Date"], netflix["Adj Close"], label="Ne\
tflix")

plt.plot(tesla["Date"], tesla["Adj Close"], label="Tesla")

plt.title("Stock Prices (2019-2024)")
plt.xlabel("Year")
plt.ylabel("Adj Close Price")
plt.legend()
plt.show()
```

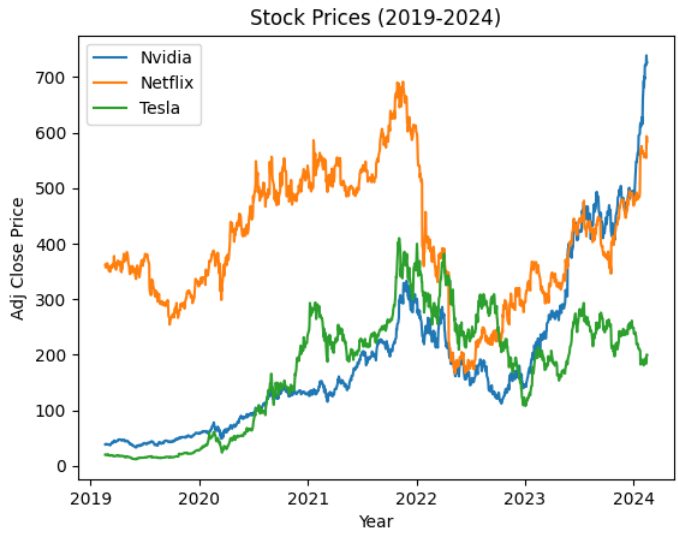

Figure 8. Multiple Line chart with Legend

© 2024 Bhola Prasad AdaCode.io

Line Style And Color

We can apply our own line styles and colors instead of default ones. Use the `linestyle` and `color` parameters to change the line style and color.

```
# Line chart with custom line style and color
plt.plot(tesla["Date"], tesla["Adj Close"], label="Tesla"\
, linestyle=":", color="blue")

plt.plot(
    nvidia["Date"],
    nvidia["Adj Close"],
    label="Nvidia",
    linestyle="--",
    color="seagreen",
)

plt.plot(
    netflix["Date"],
    netflix["Adj Close"],
    label="Netflix",
    linestyle="-.",
    color="crimson",
)

plt.title("Stock Prices (2019-2024)")
plt.xlabel("Year")
plt.ylabel("Adj Close Price")
plt.legend()
plt.show()
```

© 2024 Bhola Prasad AdaCode.io

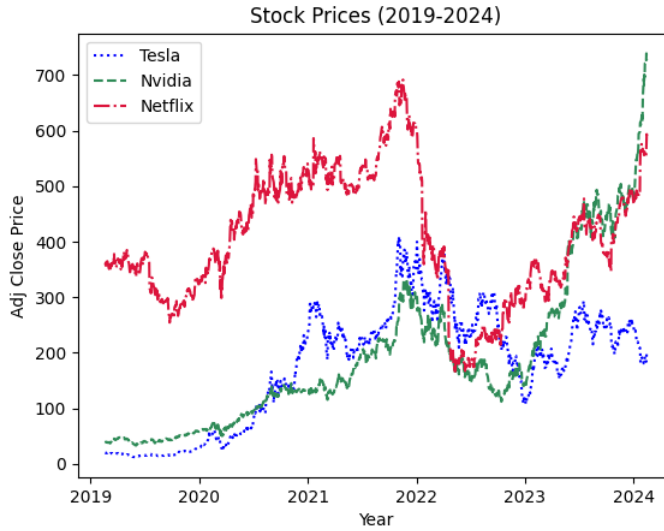

Figure 9. Line chart with custom line style and color

The plot with solid lines looked much better than this version. You can use many other parameters to change the appearance of a line chart; please refer to the document. line chart[7]

Style Sheets

Various style sheets are also available to change the appearance of your plots. You can list the styles available using the `plt.style.available` attribute. You can also see examples of them here - style sheets reference[8]

To add a style, use `plt.style.use('stylesheet_name')`.

[7] https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.plot.html
[8] https://matplotlib.org/stable/gallery/style_sheets/style_sheets_reference.html

© 2024 Bhola Prasad AdaCode.io

```
# Add a style sheet in matplotlib
plt.style.use("seaborn-v0_8-dark")

plt.plot(tesla["Date"], tesla["Adj Close"], label="Tesla"\
, color="blue")

plt.plot(nvidia["Date"], nvidia["Adj Close"], label="Nvid\
ia", color="seagreen")

plt.plot(netflix["Date"], netflix["Adj Close"], label="Ne\
tflix", color="crimson")

plt.title("Stock Prices (2019-2024)")
plt.xlabel("Year")
plt.ylabel("Adj Close Price")
plt.legend()
plt.show()
```

© 2024 Bhola Prasad AdaCode.io

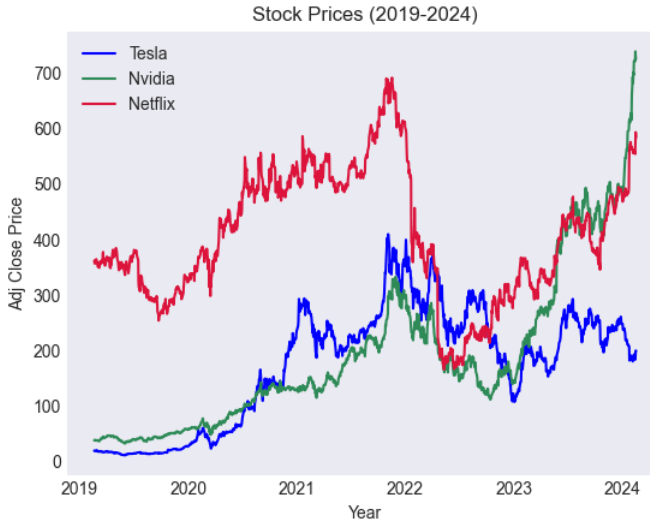

Figure 10. Style sheet in Matplotlib

Figure Size

To change the size of a figure, use the `figsize` parameter of the plt.figure()[9] function. The `figsize` takes a tuple of (`width`,`height`) in inches.

[9] https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.figure.html

© 2024 Bhola Prasad AdaCode.io

```
# Change figure size
plt.figure(figsize=(7, 5))

plt.plot(tesla["Date"], tesla["Adj Close"], label="Tesla"\
, color="blue")

plt.plot(nvidia["Date"], nvidia["Adj Close"], label="Nvid\
ia", color="seagreen")

plt.plot(netflix["Date"], netflix["Adj Close"], label="Ne\
tflix", color="crimson")

plt.title("Stock Prices (2019-2024)")
plt.xlabel("Year")
plt.ylabel("Adj Close Price")
plt.legend()
plt.show()
```

© 2024 Bhola Prasad AdaCode.io

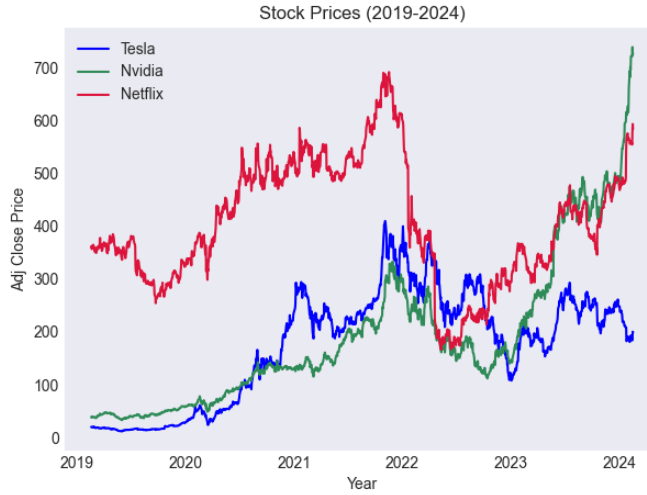

Figure 11. Custom Figure size in Matplotlib

Save A Figure

To save a figure in Matplotlib, use the plt.savefig("filepath/image.png")[10] function before calling the `plt.show()` function. You can also use other image extensions like `jpeg`, `pdf`, etc.

[10] https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.savefig.html

© 2024 Bhola Prasad AdaCode.io

```
plt.figure(figsize=(7, 5))

plt.plot(tesla["Date"], tesla["Adj Close"], label="Tesla"\
, color="blue")

plt.plot(nvidia["Date"], nvidia["Adj Close"], label="Nvid\
ia", color="seagreen")

plt.plot(netflix["Date"], netflix["Adj Close"], label="Ne\
tflix", color="crimson")

plt.title("Stock Prices (2019-2024)")
plt.xlabel("Year")
plt.ylabel("Adj Close Price")
plt.legend()
# save it in the image folder
plt.savefig("../images/stock_prices_2019_2024.png")
plt.show()
```

Exercise 2.1

1. Read the `marriage-age-2016.txt` in pandas with the sep='\t' parameter.
2. Use the same figure to create a line chart of men's and women's ages at marriage over time.
3. Create a new column, `age_diff` by subtracting Women's ages from Men's ages at marriage.
4. Create a Line chart of age differences over time.

© 2024 Bhola Prasad AdaCode.io

Summary

- To create a Line chart in matplotlib, use the `plt.plot()` function.
- Use `plt.title()` to add a title to the plot.
- For x and y-axis label, use `plt.xlabel()` and `plt.ylabel()`, respectively.
- Multiple line charts can be added to the same figure by adding another `plt.plot()` function.
- To add a legend, specify `label` for each line and then use `plt.legend()`
- Use `linestyle` and `color` to change line style and color.
- To use style sheets in matplotlib, use `plt.style.use('stylesheet_name')`.
- Use `figsize` to change the figure size.
- Use the `plt.savefig()` to save a figure.

Solution

Exercise 2.1

```
# ignore warnings
import warnings
warnings.filterwarnings("ignore")
# import required libraries
import pandas as pd
import matplotlib.pyplot as plt

plt.style.use("seaborn-v0_8-dark")

# Read the marriage-age-2016.txt in pandas
marriage = pd.read_csv("../data/marriage-age-2016.txt", s\
```

© 2024 Bhola Prasad AdaCode.io

```
ep="\t")

# Create a new column age_diff
marriage["age_diff"] = marriage["Men"] - marriage["Women"]

# Line chart of men's and women's ages at marriage
plt.plot(marriage["Year"], marriage["Men"], label="Men", \
color="crimson")

plt.plot(marriage["Year"], marriage["Women"], label="Wome\
n", color="seagreen")

plt.title("Age at Marriage By Gender (1890 - 2016)")
plt.xlabel("Year")
plt.ylabel("Age at Marriage")
plt.legend()
plt.show()
```

© 2024 Bhola Prasad AdaCode.io

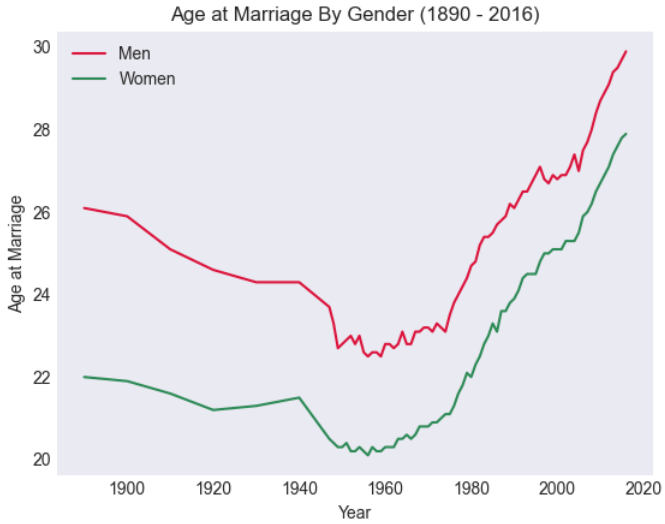

Figure 12. Age at Marriage by Gender

We can see that for both Men's and Women's age at Marriage dropped for a while then increased after 1980. There is an upward trend in People getting married late.

```
# Line chart of men's and women's age difference.
plt.plot(marriage["Year"], marriage["age_diff"], color="c\
rimson")

plt.title("Age Difference at Marriage (1890 - 2016)")
plt.xlabel("Year")
plt.ylabel("Age difference at Marriage")
plt.show()
```

© 2024 Bhola Prasad AdaCode.io

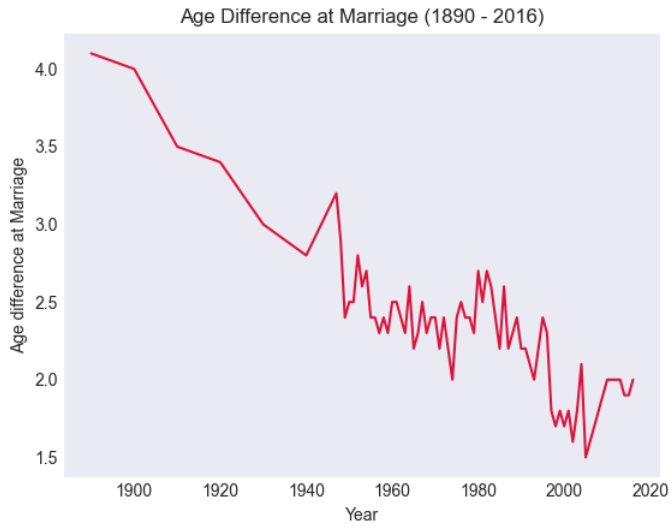

Figure 13. Age difference at Marriage

We can also see that the difference in age between men and women in marriage is dropping over time. More men and women are getting married to people of their age,and the age difference is reducing.

© 2024 Bhola Prasad AdaCode.io

Chapter 3. Scatter Plots

Scatter plots are used in statistics to visualize the relationship between two numerical variables. Each point in a scatter plot represents an individual record or observation from the dataset, plotted against each other, one on the x-axis and the other on the y-axis. Scatter plots help us understand if there is a correlation between two variables; it could be positive, negative, or no correlation.

Scatter plots can also reveal the presence of distinct groups or clusters within the data, which might indicate that the data points within each cluster share some common characteristics. They can also detect outliers or data points that significantly deviate from other data points.

Create A Scatter Plot

To create a Scatter plot in Matplotlib, use the plt.scatter()[1] function. Let's read the Penguin dataset and examine the relationship between `flipper_length_mm` and `body_mass_g`.

```
# silence warnings
import warnings
warnings.filterwarnings("ignore")
# import pandas and matplotlib
import pandas as pd
import matplotlib.pyplot as plt

# read data
```

[1] https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.scatter.html

```
df = pd.read_csv("../data/penguins.csv")
df.head()
```

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
0	Adelie	Torgersen	39.1	18.7	181.0	3750.0	MALE
1	Adelie	Torgersen	39.5	17.4	186.0	3800.0	FEMALE
2	Adelie	Torgersen	40.3	18.0	195.0	3250.0	FEMALE
3	Adelie	Torgersen	NaN	NaN	NaN	NaN	NaN
4	Adelie	Torgersen	36.7	19.3	193.0	3450.0	FEMALE

Figure 14. Penguins Data

We can see there are some missing values; let's drop them.

```
# drop missing values
df.dropna(inplace=True)
```

We will plot the `flipper_length_mm` on the x-axis and `body_mass_g` on the y-axis.

```
# create a scatter plot
plt.scatter(df["flipper_length_mm"], df["body_mass_g"])
plt.title("Flipper Length vs. Body Mass in Penguins")
plt.xlabel("Flipper Length")
plt.ylabel("Body Mass")
plt.show()
```

© 2024 Bhola Prasad AdaCode.io

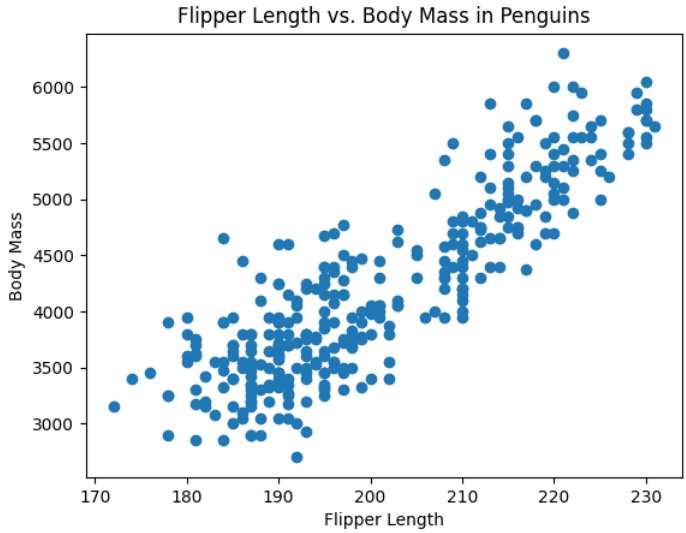

Figure 15. Scatter Plot in Matplotlib

Looking at the graph, I feel these variables have a positive correlation. On average, penguins with longer flippers weigh more than penguins with shorter flippers.

Marker Style

Use `marker` to change the marker style of a scatter plot. There are various marker styles available in matplotlib, which you can find here - marker styles[2]

[2] https://matplotlib.org/stable/api/markers_api.html#module-matplotlib.markers

© 2024 Bhola Prasad AdaCode.io

```
# scatter plots with different marker style
styles = ["^", "p", "*", "+"]
descriptions = ["triangle_up", "pentagon", "star", "plus"]

for style, desc in zip(styles, descriptions):
    plt.scatter(df["flipper_length_mm"], df["body_mass_g"]\
], marker=style)
    plt.title(f"Marker Style: {desc}")
    plt.xlabel("Flipper Length")
    plt.ylabel("Body Mass")
    plt.show()
```

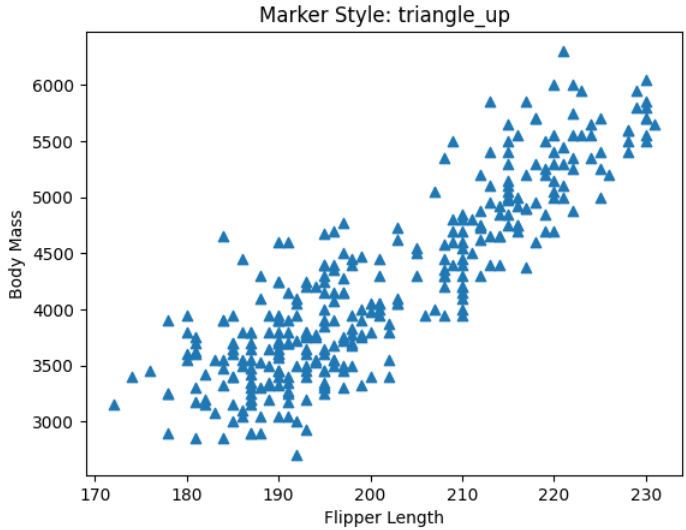

Figure 16. Marker style: triangle_up

© 2024 Bhola Prasad AdaCode.io

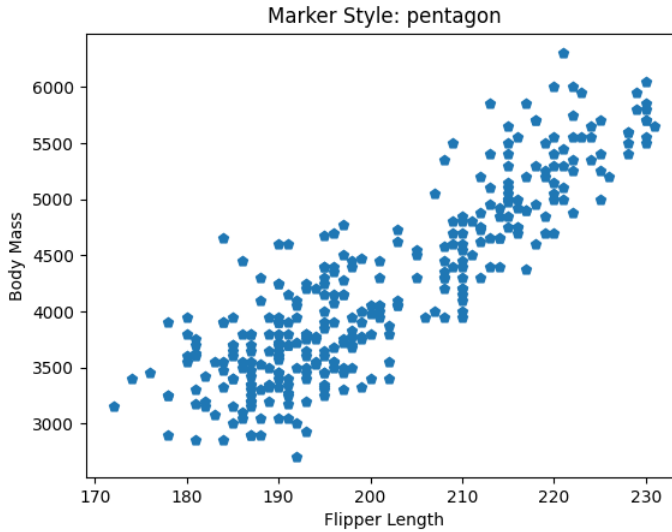

Figure 17. Marker style: pentagon

© 2024 Bhola Prasad AdaCode.io

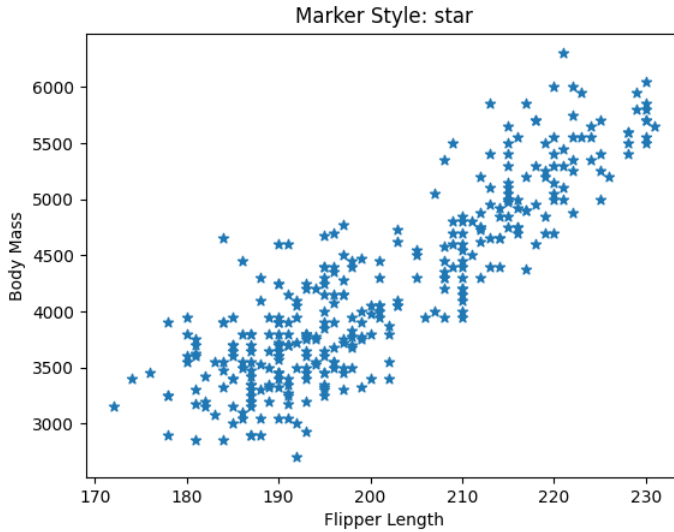

Figure 18. Marker style: star

© 2024 Bhola Prasad AdaCode.io

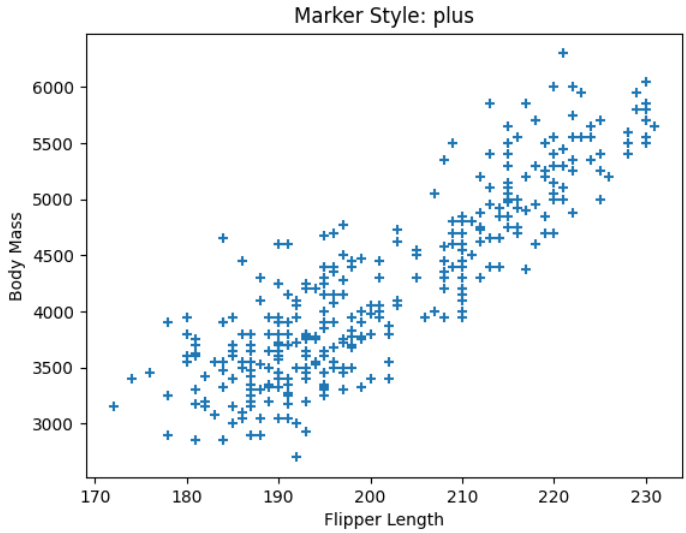

Figure 19. Marker style: plus

Marker Color

Use the `c` or `color` to change the marker color.

```
# scatter plot with custom color
plt.scatter(df["flipper_length_mm"], df["body_mass_g"], c\
="green", marker="+")
plt.title("Flipper Length vs. Body Mass in Penguins")
plt.xlabel("Flipper Length")
plt.ylabel("Body Mass")
plt.show()
```

© 2024 Bhola Prasad AdaCode.io

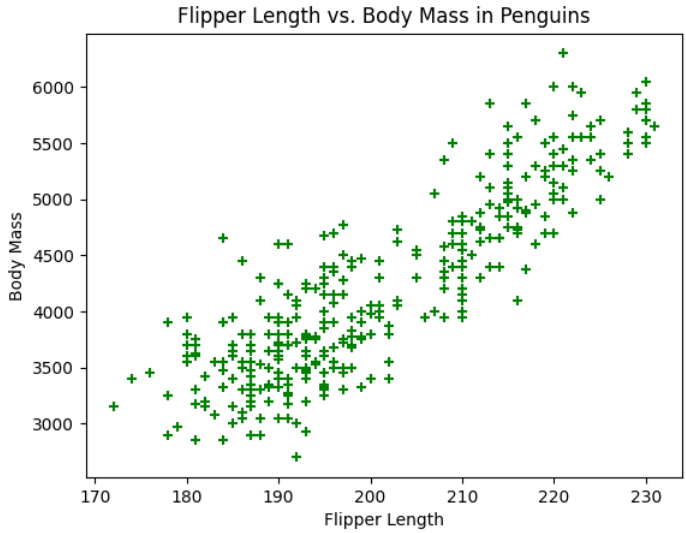

Figure 20. Scatter plot with custom color

Marker Size

The `s` controls the size of the markers on a scatter plot. You can specify `s` as a single number or an array of sizes to give each data point a different size. This is useful for adding an additional dimension to a two-dimensional scatter plot, as we do in bubble charts.

© 2024 Bhola Prasad AdaCode.io

```
# scatter plot with different marker size
plt.scatter(
    df["flipper_length_mm"], df["body_mass_g"], c="magent\
a", s=df["bill_length_mm"]
)
plt.title("Flipper Length vs. Body Mass in Penguins")
plt.xlabel("Flipper Length")
plt.ylabel("Body Mass")
plt.show()
```

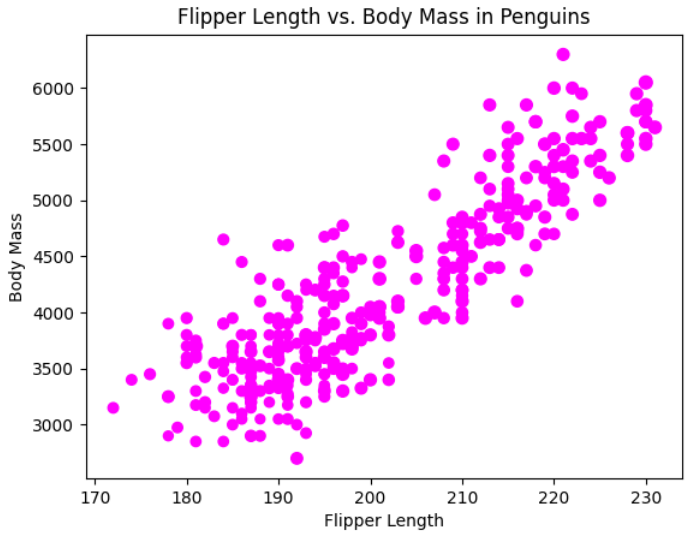

Figure 21. Scatter plot with different marker sizes

© 2024 Bhola Prasad AdaCode.io

Color Bar

We can also color the markers according to their size. For this, we must first set the `c` with the variable that specifies the size, then use the `cmap` to map the data to colors, and then use the plt.colorbar()[3] function to show the color bar on the plot. Use `label` to add a label to the color bar. You can find various options for `cmap` here - cmap options[4]

```
# scatter plot with a color bar
plt.scatter(
    df["flipper_length_mm"], df["body_mass_g"], c=df["bil\
l_length_mm"], cmap="plasma"
)
plt.title("Flipper Length vs. Body Mass in Penguins")
plt.xlabel("Flipper Length")
plt.ylabel("Body Mass")
plt.colorbar(label="Bill length")
plt.show()
```

[3] https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.colorbar.html
[4] https://matplotlib.org/stable/users/explain/colors/colormaps.html

© 2024 Bhola Prasad AdaCode.io

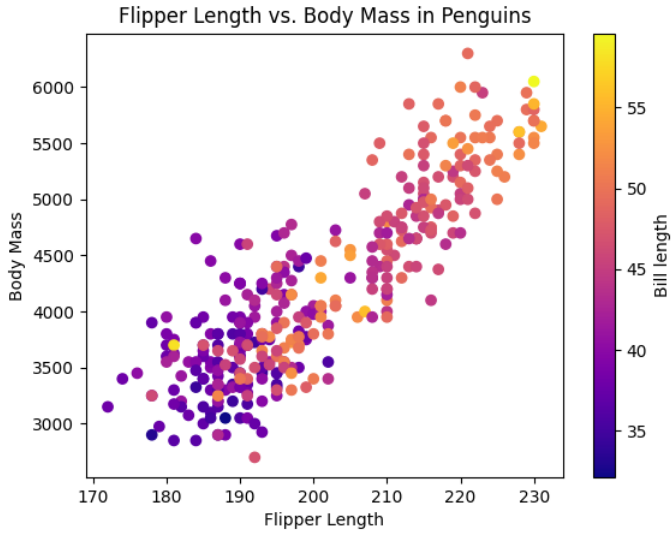

Figure 22. Scatter plot with color bar

Scatter Plot with Legend

To create a Scatter plot with a legend, add a `label` to the plot and then add a `plt.legend()` function. Adding a label and a legend will help us identify different classes or groups within a dataset. For example, we can see how the body mass and flipper length change based on the type of penguin species.

© 2024 Bhola Prasad AdaCode.io

```
# get unique species
unique_species = df["species"].unique()

# create a scatter plot for each species
for species in unique_species:
    df_species = df[df["species"] == species]
    plt.scatter(
        df_species["flipper_length_mm"], df_species["body\
_mass_g"], label=species
    )

plt.title("Flipper Length vs. Body Mass by Penguin Specie\
s")
plt.xlabel("Flipper Length")
plt.ylabel("Body Mass")
plt.legend()
plt.show()
```

© 2024 Bhola Prasad AdaCode.io

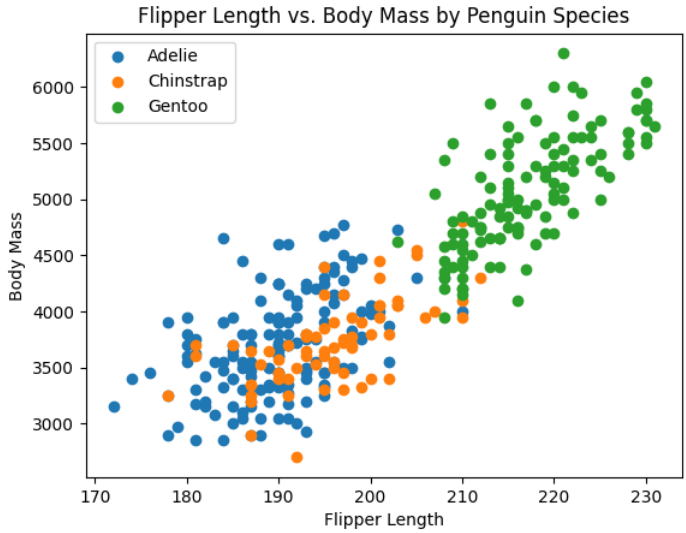

Figure 23. Scatter Plot with a Legend

This plot shows that `Gentoo` species tend to have considerably longer flipper lengths and higher body masses than `Adelie` and `Chinstrap` species.

Transparent Markers

The `alpha` controls the transparency level of the markers in the scatter plot. The `alpha` value ranges from `0` to `1`, where `0` is fully transparent (the markers are invisible) and `1` is fully opaque (the markers are completely solid). This is handy in scatter plots with many overlapping points.

© 2024 Bhola Prasad AdaCode.io

```
# scatter plot with alpha
plt.scatter(df["flipper_length_mm"], df["body_mass_g"], c\
="magenta", alpha=0.5)
plt.title("Flipper Length vs. Body Mass in Penguins")
plt.xlabel("Flipper Length")
plt.ylabel("Body Mass")
plt.show()
```

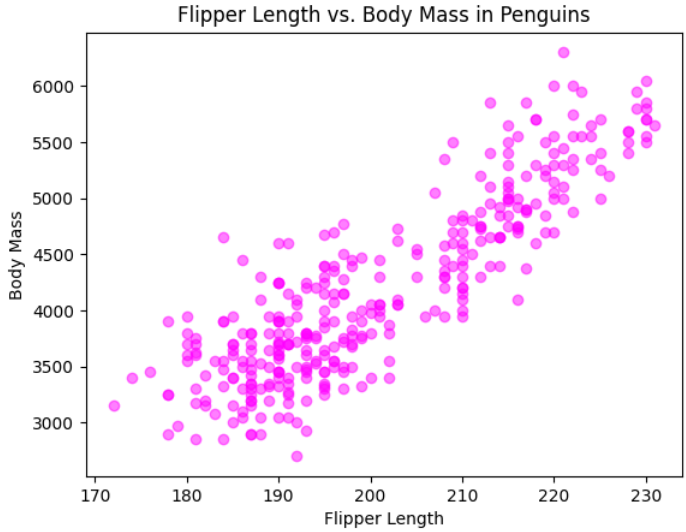

Figure 24. Scatter plot with alpha

Exercise 3.1

1. Read the `tips.csv` data in pandas.
2. Create a Scatter plot with `total_bill` on the x-axis and `tip` on the y-axis.

© 2024 Bhola Prasad AdaCode.io

3. Change the marker style to `tri_down` or your other choice.
4. Change the marker color to `magenta` or your favorite color.
5. Add a color bar using the `size` column, i.e. the number of people at the party.
6. Create a Scatter plot with a legend using the `sex` column.

Summary

- Scatter plots are used to visualize the relationship between two variables.
- Use the `plt.scatter()` function to create a Scatter plot in matplotlib.
- Marker styles can be changed with the `marker`.
- Marker color can be changed with `c` or `color`.
- The `s` change the marker size.
- To add a color bar, define `c` and `cmap`, then add `plt.colorbar()`.
- Use the `label` and `plt.legend()` to add a legend.

Solution

Exercise 3.1

© 2024 Bhola Prasad AdaCode.io

```
# Read tips data in a pandas
df = pd.read_csv("../data/tips.csv")

# Create a Scatter plot
plt.scatter(df["total_bill"], df["tip"])
plt.title("Total Bill Vs Tip")
plt.xlabel("Total Bill")
plt.ylabel("Tip")
plt.show()
```

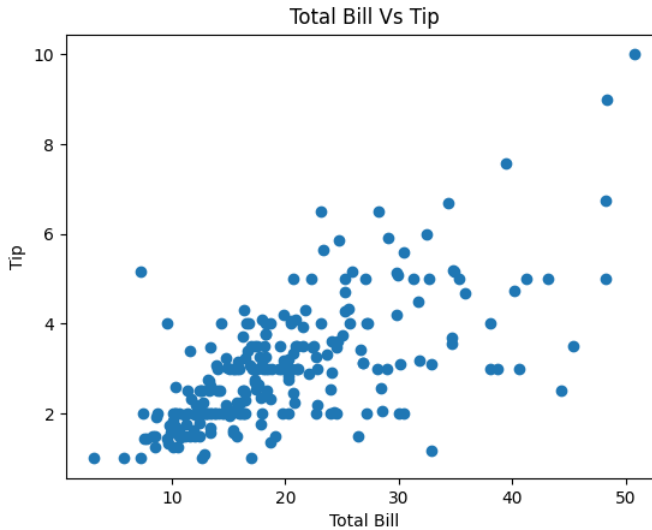

Figure 25. Scatter plot in Matplotlib

© 2024 Bhola Prasad AdaCode.io

```
# Scatter plot with style and color bar
plt.scatter(df["total_bill"], df["tip"], c=df["size"], ma\
rker="1", cmap="cool")
plt.title("Total Bill Vs Tip")
plt.xlabel("Total Bill")
plt.ylabel("Tip")
plt.colorbar(label="Number of People at the Party")
plt.show()
```

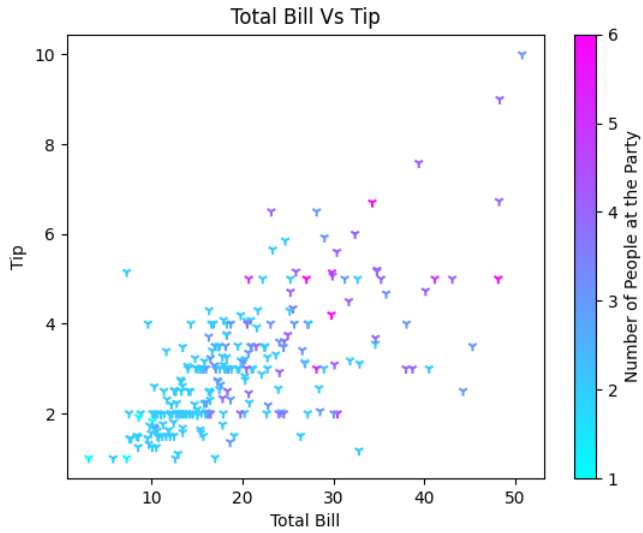

Figure 26. Scatter plot with style and color bar

© 2024 Bhola Prasad AdaCode.io

```
# Scatter plot with a legend
sex_types = df["sex"].unique()

for sex in sex_types:
    df_sex = df[df["sex"] == sex]
    plt.scatter(df_sex["total_bill"], df_sex["tip"], labe\
l=sex)

plt.title("Total Bill Vs Tip By Sex")
plt.xlabel("Total Bill")
plt.ylabel("Tip")
plt.legend()
plt.show()
```

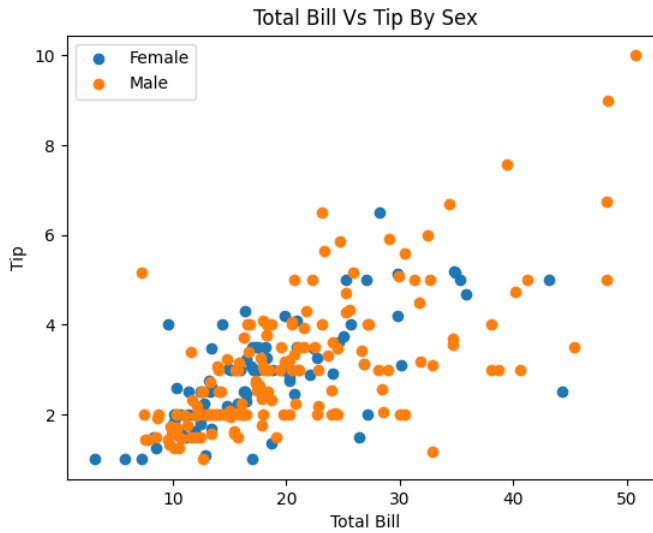

Figure 27. Scatter plot with a legend

© 2024 Bhola Prasad AdaCode.io

Chapter 4. Bar Charts

A Bar chart is a graph that uses bars to display information, allowing easy comparison between different categories. The bars can be vertical or horizontal depending on how you display information.

Bar charts can be used in many situations, such as comparing monthly sales of different products or categories, determining how many books you read each month, how many cats and dogs your neighborhood has, and what kind of ice cream flavors sell more in a store. Bar charts make it easier to understand numbers by showing them as bars; the longer the bar, the bigger the number.

Create A Bar Chart

Use the plt.bar(x, height)[1] function to create a Bar chart in Matplotlib. The `x` represents the x coordinates of the bars, and the height represents the height of each bar.

Let's read a dataset. We will work with the insurance charges data.

```
# filter warnings
import warnings
warnings.filterwarnings("ignore")
# import pandas and matplotlib
import pandas as pd
import matplotlib.pyplot as plt

# read data
df = pd.read_csv("../data/insurance.csv")
df.head()
```

[1] https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.bar.html

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520

Figure 28. Insurance charges Data

Vertical Bar Chart

The most common type of bar chart is the vertical bar chart. In this type of chart, categories are displayed along the horizontal axis, and the lengths of the bars are displayed along the vertical axis.

Let's try to understand the average insurance charges for smokers compared to non-smokers. We must first calculate each group's average insurance charges to visualize this.

```
# average insurance charges for smokers vs. non-smokers
average_charges = (df.groupby('smoker')['charges']
                   .mean()
                   .round(2)
                   .reset_index())
```

Now, let's use matplotlib to create a Vertical bar chart.

© 2024 Bhola Prasad AdaCode.io

```
# create a Vertical bar chart
plt.bar(average_charges["smoker"], average_charges["charg\
es"])
plt.title("Average Insurance Charges: Smokers vs. Non-Smo\
kers")
plt.xlabel("Smoker")
plt.ylabel("Average Insurance Charges")
plt.show()
```

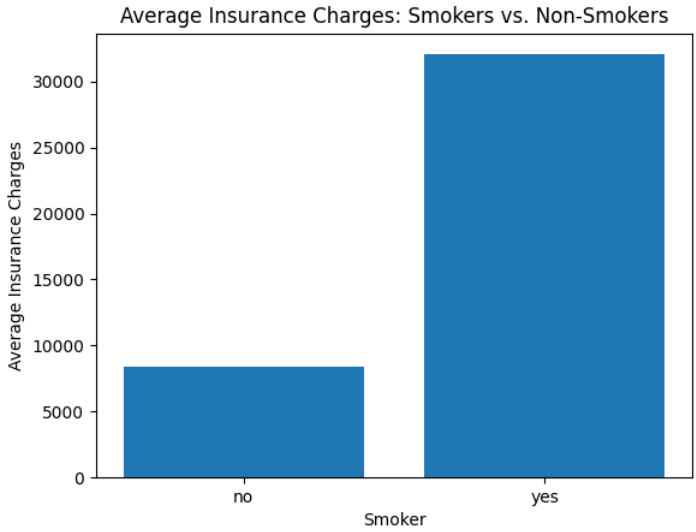

Figure 29. Bar chart in Matplotlib

As we can see, smokers tend to have significantly higher average insurance charges than non-smokers, highlighting the impact of smoking on insurance costs.

We can further improve this chart by changing the colors of each bar. We will color the bar red for smokers and green for non-smokers.

© 2024 Bhola Prasad AdaCode.io

```
# Create a vertical bar chart
plt.bar(
    average_charges["smoker"], average_charges["charges"]\
, color=["seagreen", "crimson"]
)
plt.title("Average Insurance Charges: Smokers vs. Non-Smo\
kers")
plt.xlabel("Smoker")
plt.ylabel("Average Insurance Charges")
plt.show()
```

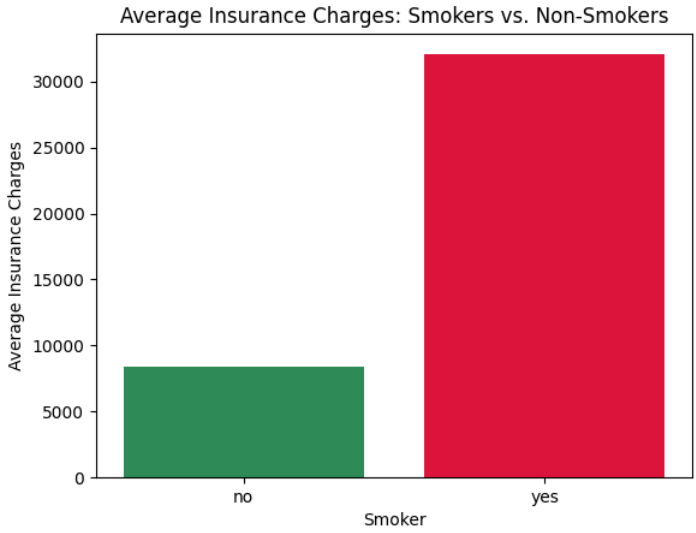

Figure 30. Bar chart with custom colors

Here, I am using different shades of red and green. Since the first bar represents non-smokers and the second bar represents smokers, I passed the list of colors in that order.

Let's also add a grid to this plot.

© 2024 Bhola Prasad AdaCode.io

Grid Lines

To add grid lines in Matplotlib, use the plt.grid()[2] function. In the following code, I have added a grid with `alpha` to add transparency to the grid lines becuase I don't want the grids to dominate the chart. The `axis` lets you specify whether you want the grid on the x-axis, y-axis, or both.

```
# vertical bar chart with grid
plt.bar(
    average_charges["smoker"], average_charges["charges"]\
, color=["seagreen", "crimson"]
)
plt.title("Average Insurance Charges: Smokers vs. Non-Smo\
kers")
plt.xlabel("Smoker")
plt.ylabel("Average Insurance Charges")
plt.grid(alpha=0.3)
plt.show()
```

[2] https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.grid.html

© 2024 Bhola Prasad AdaCode.io

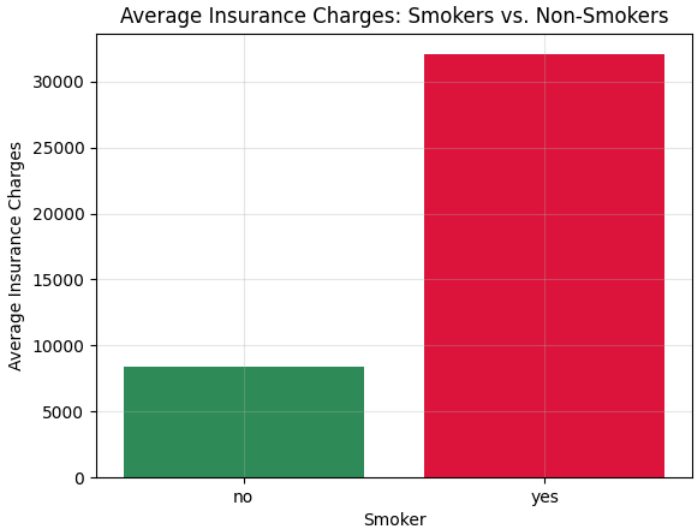

Figure 31. Bar chart with Grid Lines

Width Of The Bars

To change the width of the bars, use the `width`. By default, it is 0.8.

```
# bar chart with custom bar width
plt.bar(
    average_charges["smoker"],
    average_charges["charges"],
    color=["seagreen", "crimson"],
    width=0.4,
)
plt.title("Average Insurance Charges: Smokers vs. Non-Smo\
kers")
plt.xlabel("Smoker")
plt.ylabel("Average Insurance Charges")
```

© 2024 Bhola Prasad AdaCode.io

```
12  plt.grid(alpha=0.3)
13  plt.show()
```

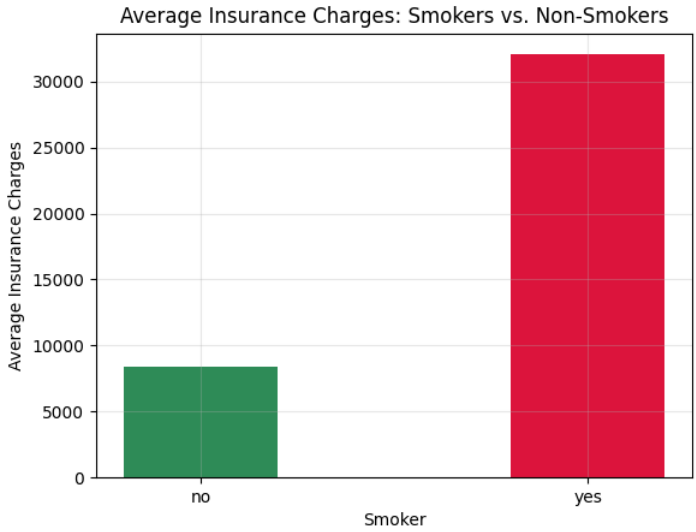

Figure 32. Bar chart with custom bar width

Horizontal Bar Chart

A Horizontal bar chart compares the size of categories using horizontal bars. In this chart, categories are displayed on the vertical axis and their values on the horizontal axis. This type of chart is handy for displaying data with long category labels or when there are a large number of categories.

To create a Horizontal bar chart in Matplotlib, use the plt.barh()[3]

[3]https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.barh.html

© 2024 Bhola Prasad AdaCode.io

function. For example, let's look at the average insurance charges by region.

```
# Calculate the average insurance charges by region
avg_region = (
    df.groupby("region")["charges"]
    .mean()
    .round(2)
    .reset_index()
    .sort_values(by="charges")
)
avg_region
```

	region	charges
3	southwest	12346.94
1	northwest	12417.58
0	northeast	13406.38
2	southeast	14735.41

Figure 33. Insurance charges by region

Looking carefully, you can see that I have also sorted the data by charges in ascending order, creating a natural progression of the bars when plotted. The bar with the highest charges will appear at the top, and the bar with the lowest will be at the bottom. This makes comparing bars much easier. You can also reverse the order of the bars by changing the sorting order.

© 2024 Bhola Prasad AdaCode.io

```
# horizontal bar chart for insurance charges by region
plt.barh(
    avg_region["region"],
    avg_region["charges"],
    color=["tab:cyan", "tab:blue", "tab:orange", "tab:red\
"],
)
plt.title("Average Insurance Charges by Region")
plt.xlabel("Average Insurance Charges")
plt.ylabel("Region")
plt.grid(axis="x", linestyle="--", alpha=0.7)
plt.show()
```

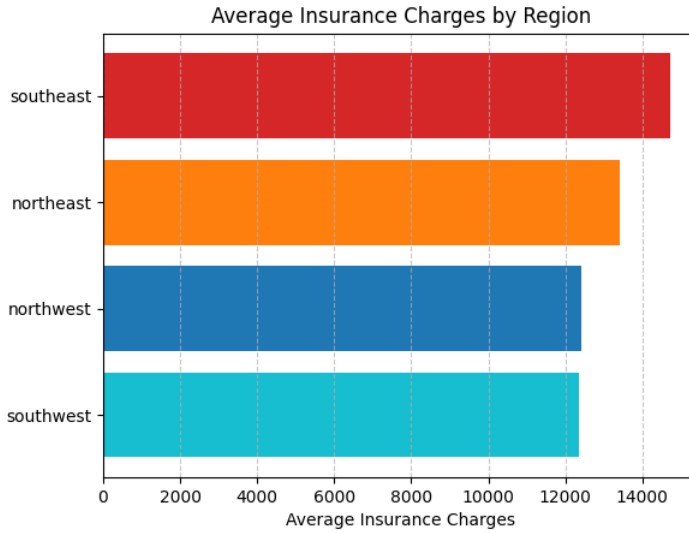

Figure 34. Insurance charges by region

Tip: In `plt.barh(y, width)`, the second argument is the `width` of the bar, which corresponds to the value

© 2024 Bhola Prasad AdaCode.io

they represent in a horizontal bar chart, not their height, as might be intuitively thought. This contrasts with `plt.bar(x, height)` function for vertical bar charts, where the second argument represents the heights of the bars. For horizontal bars, the `height` controls the thickness of the bars, which can be set manually to adjust how thick or thin the bars appear. This distinction is crucial when plotting data in horizontal versus vertical bar charts.

```
# horizontal bar chart with custom bar height
plt.barh(
    avg_region["region"],
    avg_region["charges"],
    color=["tab:cyan", "tab:blue", "tab:orange", "tab:red\
"],
    height=0.5,
)  # set the height
plt.title("Average Insurance Charges by Region")
plt.xlabel("Average Insurance Charges")
plt.ylabel("Region")
plt.grid(axis="x", linestyle="--", alpha=0.7)
plt.show()
```

© 2024 Bhola Prasad AdaCode.io

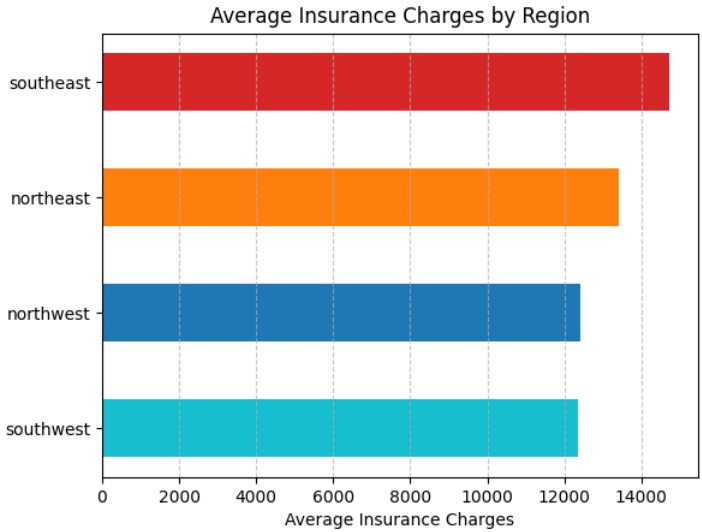

Figure 35. Horizontal bar chart with bar height

The average insurance charges are highest in the `southeast` region and lowest in the `southwest` region.

Exercise 4.1

1. Calculate the average insurance charges based on `sex`.
2. Create a vertical bar chart for average insurance charges by `sex`.
3. Change the colors of the bars other than the default. You can find more info here - matplotlib colors[4] or HTML color names[5]
4. Add Grid lines to the plot and experiment with different grid options available in matplotlib. Grid document[6]

[4] https://matplotlib.org/stable/gallery/color/named_colors.html
[5] https://htmlcolorcodes.com/color-names/
[6] https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.grid.html

© 2024 Bhola Prasad AdaCode.io

5. Change the width of the bars.

Exercise 4.2

1. Read the `penguins.csv` data in pandas.
2. calculate the average body mass of penguins by species.
3. Create a Horizontal bar chart by species.
4. Change the height and colors of the bars.

Stacked Bar Chart

In data visualization, a Stacked Bar chart segments the categories into sub-categories, where the total length of each bar represents the combined total of its sub-categories.

Let's segment each region by smokers and non-smokers. First, we will group the data by `region` and `smoker` and then calculate the average insurance charges for each sub-category.

```
region_smoker = df.groupby(["region", "smoker"])["charges\
"].mean().round(2).unstack()
```

Next, we will set up the plot. We will define the positions for each region on the x-axis and separate the data for non-smokers and smokers. Separating the data is optional but will make understanding what we are doing easy.

© 2024 Bhola Prasad AdaCode.io

```
# Separate the data for non-smokers and smokers
non_smokers = region_smoker["no"]
smokers = region_smoker["yes"]

# positions for the regions on the x-axis
positions = range(len(region_smoker))
```

Next, we will plot the bars for non-smokers and then for smokers, stacked on top of non-smokers, using the `bottom` parameter. We will also use the plt.xticks()[7] to set the tick locations and labels.

```
# Create a stacked bar chart
plt.figure(figsize=(8, 6))

# Plot bars for non-smokers
plt.bar(positions, non_smokers, width=0.5, label="Non-Smo\
kers", color="seagreen")

# Plot bars for smokers, stacked on top of non-smokers by\
 using the bottom parameter
plt.bar(
    positions, smokers, width=0.5, bottom=non_smokers, la\
bel="Smokers", color="crimson"
)

# Add some details
plt.title("Average Insurance Charges by Region and Smoker\
 Status")
plt.xlabel("Region")
plt.ylabel("Average Insurance Charges")
plt.xticks(positions, region_smoker.index)
plt.grid(alpha=0.2)
plt.legend()
plt.show()
```

[7] https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.xticks.html

© 2024 Bhola Prasad AdaCode.io

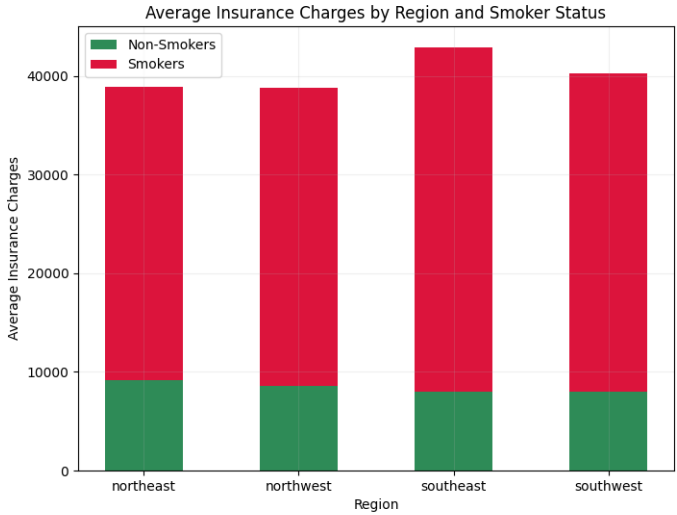

Figure 36. Stacked Bar Chart

We can see average insurance charges are very high for somkers in all regions compared to non-smokers.

Exercise 4.3

1. Group the penguin's data by `species` and `sex` and calculate the `average` body mass.
2. Separate the data for females and males.
3. Define the positions for each species on the x-axis.
4. Create a Stacked bar chart for each species segemented by sex.

© 2024 Bhola Prasad AdaCode.io

Grouped Bar Chart

A Grouped Bar chart, also known as a clustered bar chart, is another type of bar chart used to compare different categories. Instead of stacking the bars as we did with the stacked bar chart, the grouped bar chart plots each sub-category side by side.

Let's turn our previously created stacked bar chart into a grouped bar chart. First, we need to determine the positions of the bars on the x-axis. To plot bars side by side, we will adjust these positions by shifting slightly left or right from the central positions for each group.

```
import numpy as np

# Define the width of the bars and the positions
bar_width = 0.35
n_regions = len(region_smoker.index)
index = np.arange(n_regions)

# Calculating positions for smokers and non-smokers
positions_non_smokers = index - bar_width / 2
positions_smokers = index + bar_width / 2
```

For the non-smokers, we subtract half the bar width from each index to place these bars slightly to the left, and for smokers, we add half the bar width to each index to place these bars to the right.

Now, we have to plot two bar charts, one for the non-smokers and one for the smokers.

© 2024 Bhola Prasad AdaCode.io

```
# Plotting the bars for non-smokers and smokers
plt.figure(figsize=(8, 7))

plt.bar(
    positions_non_smokers,
    non_smokers,
    width=bar_width,
    label="Non-Smokers",
    color="seagreen",
)

plt.bar(positions_smokers, smokers, width=bar_width, label="Smokers", color="crimson")

# Adding chart details
plt.xlabel("Region")
plt.ylabel("Average Insurance Charges")
plt.title("Average Insurance Charges by Region and Smoker Status")
plt.xticks(index, region_smoker.index)
plt.grid(alpha=0.2)
plt.legend()
plt.show()
```

© 2024 Bhola Prasad AdaCode.io

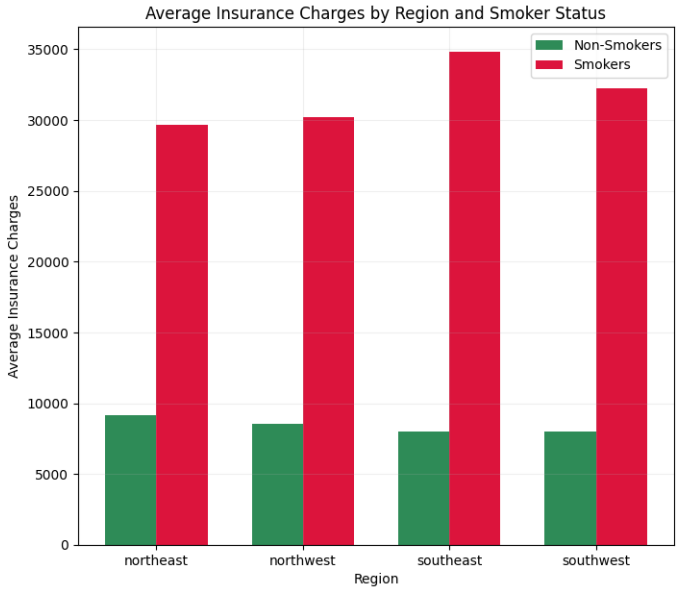

Figure 37. Grouped Bar Chart

Exercise 4.4

1. Convert the Stacked bar chart from Exercise 3.3 to a Grouped Bar chart.

Summary

- To create a Vertical Bar chart in matplotlib, use the `plt.bar()` function.

© 2024 Bhola Prasad AdaCode.io

- The width of the bars in the Vertical Bar chart is adjusted with the `width`.
- To create a Horizontal Bar chart, use the `plt.barh()` function.
- The `height` parameter adjusts the thickness of the bars in the Horizontal Bar chart.
- In a Stacked Bar chart, sub-categories are stacked on each other for every category.
- In a Grouped Bar chart, sub-categories are placed side by side for each category.

Solution

Exercise 4.1

```
# ignore warnings
import warnings
warnings.filterwarnings("ignore")
# import libraries
import pandas as pd
import matplotlib.pyplot as plt

# read insurance data
df = pd.read_csv("../data/insurance.csv")

# calculate average insurance charges by sex
avg_charges_sex = df.groupby("sex")["charges"].mean().res\
et_index()

# create a vertical bar chart
plt.bar(
    avg_charges_sex["sex"],
    avg_charges_sex["charges"],
    color=["crimson", "teal"],
```

© 2024 Bhola Prasad AdaCode.io

```
    width=0.3,
)

plt.title("Average Insurance Charges by Sex")
plt.xlabel("Sex")
plt.ylabel("Average Insurance Charges")
plt.grid(axis="y", linestyle="--", alpha=0.4)
plt.show()
```

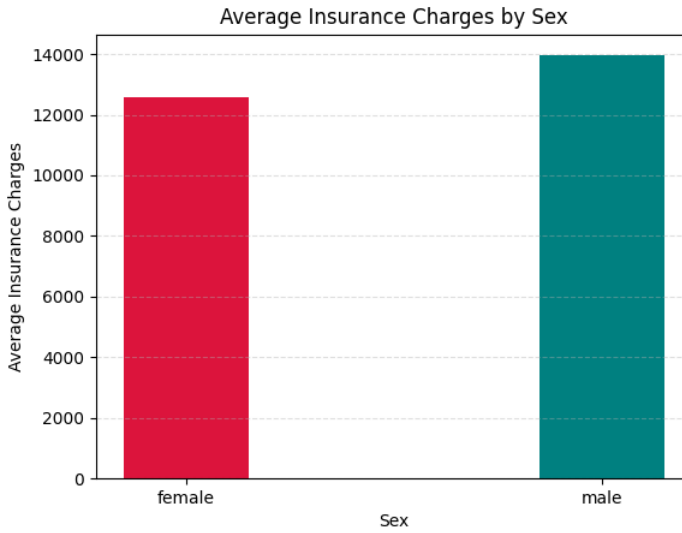

Figure 38. Vertical Bar chart

Exercise 4.2

© 2024 Bhola Prasad AdaCode.io

```
# read penguins data
penguins = pd.read_csv("../data/penguins.csv")

# calculate the average body mass of the penguin species
avg_species = (
    penguins.groupby("species")["body_mass_g"]
    .mean()
    .reset_index()
    .sort_values(by="body_mass_g")
)

# create a horizontal bar chart
plt.barh(
    avg_species["species"],
    avg_species["body_mass_g"],
    color=["tab:cyan", "tab:blue", "tab:orange"],
    height=0.5,
)

plt.title("Average Body Mass By Penguin Species")
plt.xlabel("Average Body Mass")
plt.ylabel("Penguin Species")
plt.show()
```

© 2024 Bhola Prasad AdaCode.io

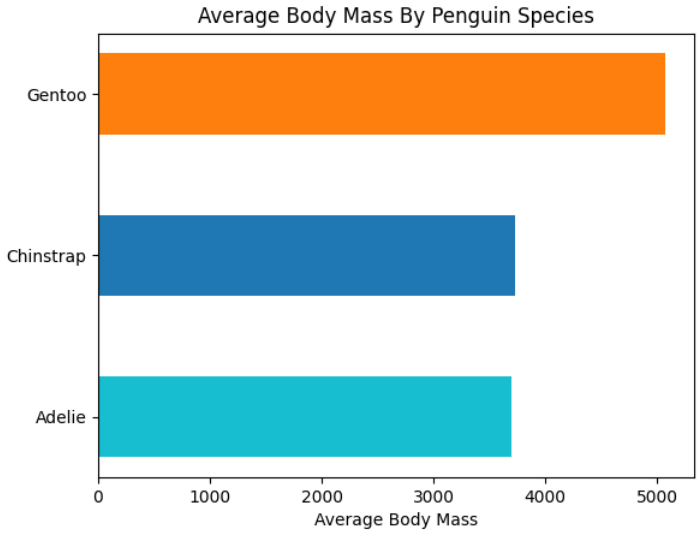

Figure 39. Horizontal Bar chart

Exercise 4.3

```
# Group by species and sex and calculate average body mas\
s.
species_sex = penguins.groupby(["species", "sex"])["body_\
mass_g"].mean().unstack()

# Seperate the data for male and female
penguins_male = species_sex["MALE"]
penguins_female = species_sex["FEMALE"]

# define the positions of the groups on the x-axis
positions = range(len(penguins_female))

# create a Stacked Bar chart for each species.
```

© 2024 Bhola Prasad AdaCode.io

```
plt.bar(positions, penguins_male, width=0.4, label="Male"\
, color="teal")

plt.bar(
    positions,
    penguins_female,
    width=0.4,
    bottom=penguins_male,
    label="Female",
    color="deeppink",
)

plt.title("Average Body Mass by Penguins Species and Gend\
er")
plt.xlabel("Species")
plt.ylabel("Average Body Mass")
plt.xticks(positions, species_sex.index)
plt.legend()
plt.show()
```

© 2024 Bhola Prasad AdaCode.io

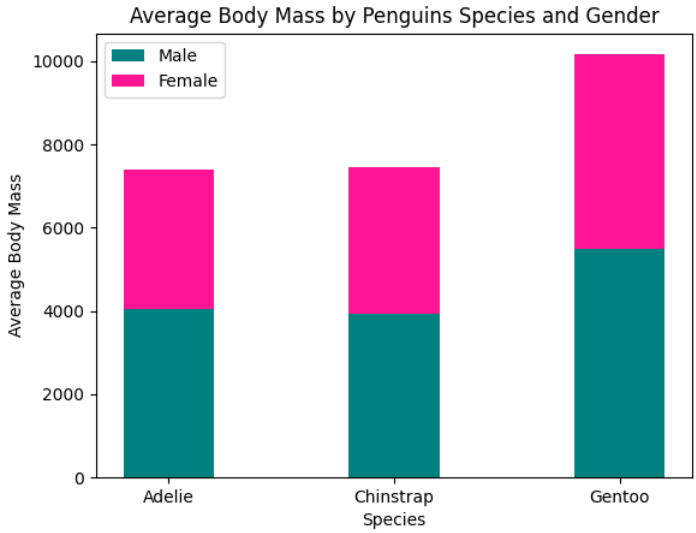

Figure 40. Stacked Bar chart

Exercise 4.4

```
import numpy as np

# bar width and index positions
bar_width = 0.35
n_species = len(penguins_female.index)
index = np.arange(n_species)

# define the positions for males and females
positions_male = index - bar_width / 2
positions_female = index + bar_width / 2

# create a grouped bar chart
plt.bar(positions_male, penguins_male, width=bar_width, l\
```

© 2024 Bhola Prasad AdaCode.io

```
abel="Male", color="teal")
plt.bar(
    positions_female, penguins_female, width=bar_width, l\
abel="Female", color="deeppink"
)
plt.title("Average Body Mass by Penguin Species and Gende\
r")
plt.xlabel("Species")
plt.ylabel("Average Body Mass")
plt.xticks(index, penguins_female.index)
plt.legend()
plt.show()
```

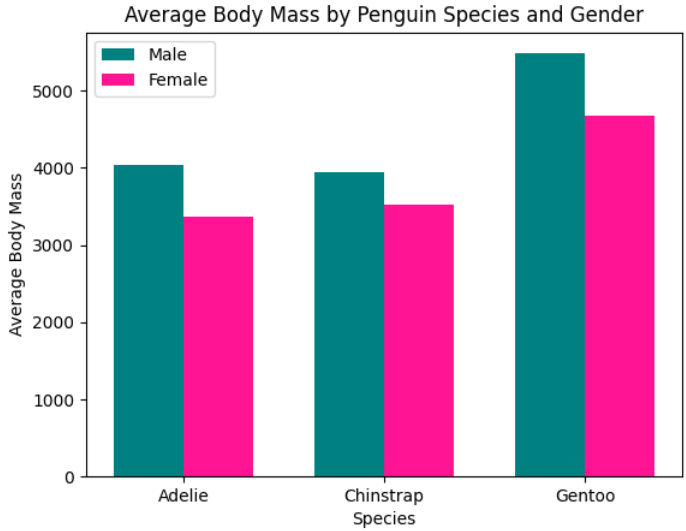

Figure 41. Grouped Bar chart

© 2024 Bhola Prasad AdaCode.io

Chapter 5. Histograms

Histograms are used in data visualization to show the distribution of numerical data. Each bar in a histogram represents the frequency(the number of occurences) of data points within a specific range of values called a bin.

Histograms are particularly useful for understanding the shape of the data, such as whether the distribution is symmetric or skewed or if there are any outliers or unusual patters.

Create a histogram

To create a Histogram in matplotlib, use the plt.hist()[1] function. This function requires at least one parameter `x`, which specifies the data values you wish to plot in the histogram. The `x` parameter can be a single array of values or a sequence of arrays if you wish to plot multiple datasets on the same histogram for comparison.

```
import warnings

warnings.filterwarnings("ignore")

import pandas as pd
import matplotlib.pyplot as plt

# read Nvidia Share market data
nvidia = pd.read_csv("../data/NVDA.csv", parse_dates=["Da\
te"])
nvidia.head()
```

[1] https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.hist.html

	Date	Open	High	Low	Close	Adj Close	Volume
0	2019-02-19	39.227501	39.972500	39.035000	39.160000	38.855827	55189200
1	2019-02-20	39.455002	40.314999	39.342499	39.637501	39.329613	54098800
2	2019-02-21	39.764999	40.012501	38.794998	38.942501	38.640015	44854800
3	2019-02-22	39.465000	39.987499	39.327499	39.797501	39.488373	40174000
4	2019-02-25	40.639999	41.320000	39.584999	39.672501	39.364349	65602000

Figure 42. Nvidia stock prices

Let's create a histogram of `Adj close` price column.

```
# create a histogram
plt.hist(nvidia["Adj Close"])
plt.title("Histogram of Nvidia Adj Close Prices")
plt.xlabel("Adjusted Close Price")
plt.ylabel("Frequency")
plt.show()
```

© 2024 Bhola Prasad AdaCode.io

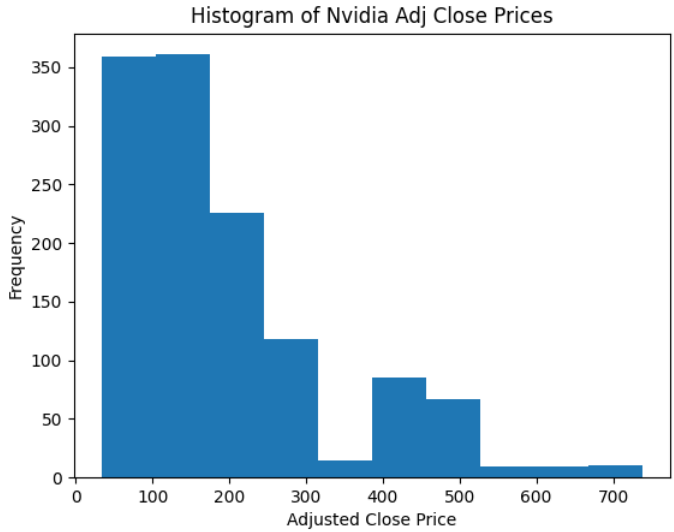

Figure 43. A Histogram in matplotlib

The data is right-skewed, and most of the values are between $40 and $300. There is a long tail on the right. To get a better sense of the data, we can supplement the histogram with summary statistics, which can be calculated easily using the pandas describe method.

```
# summary statistics
nvidia['Adj Close'].describe()
```

© 2024 Bhola Prasad AdaCode.io

	Adj Close
count	1259.000000
mean	192.245246
std	141.305020
min	33.257328
25%	84.799519
50%	153.062927
75%	244.593521
max	739.000000

Figure 44. Summary statistics

The minimum close price is $33, and the maximum is $739. The mean close price is around $192, and the median is $153.

Bin size

The `bins` parameter can be used to adjust the histogram's bin size. By default, the bin size is 10. Let's try bin sizes of 20 and 30.

© 2024 Bhola Prasad AdaCode.io

```
# histogram with custom bin size
for bin_size in [20, 30]:
    plt.hist(nvidia["Adj Close"], bins=bin_size, label=f"\
Bin Size= {bin_size}")
    plt.title("Histogram of Nvidia Adj Close Prices")
    plt.xlabel("Adjusted Close Price")
    plt.ylabel("Frequency")
    plt.legend()
    plt.show()
```

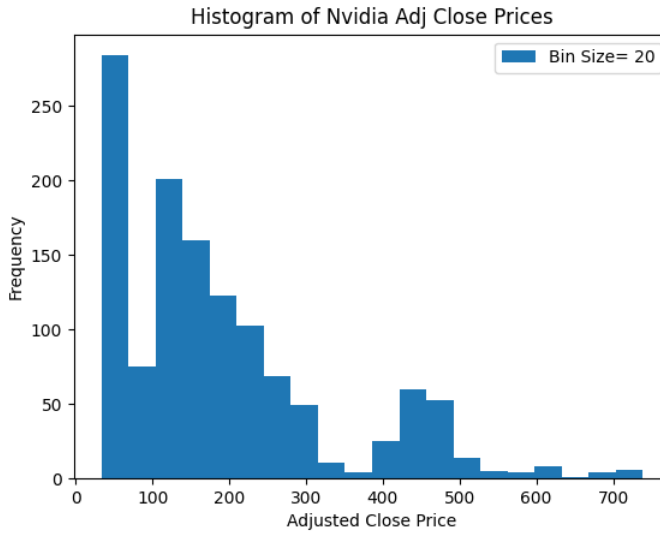

Figure 45. Histogram with bin size=20

© 2024 Bhola Prasad AdaCode.io

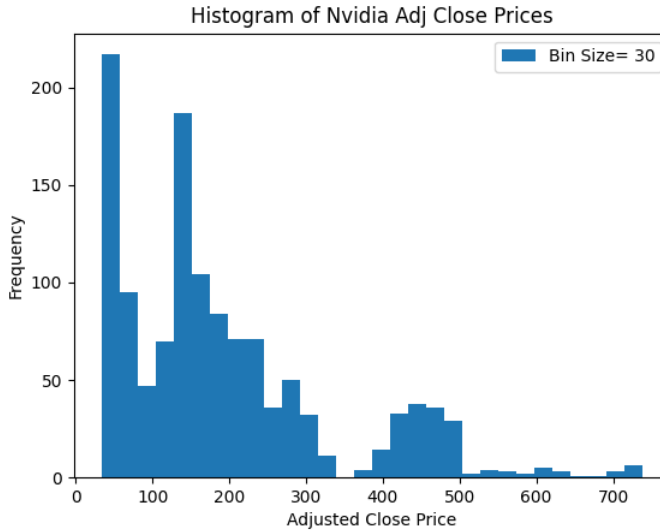

Figure 46. Histogram with bin size=30

Color

We will use the `color` to change the color as usual. Since Nvidia's brand color is green, let's use that.

```
# histogram with custom color
plt.hist(nvidia['Adj Close'], color='seagreen', bins=30)
plt.title('Histogram of Nvidia Adj Close Prices')
plt.xlabel('Adjusted Close Price')
plt.ylabel('Frequency')
plt.show()
```

© 2024 Bhola Prasad AdaCode.io

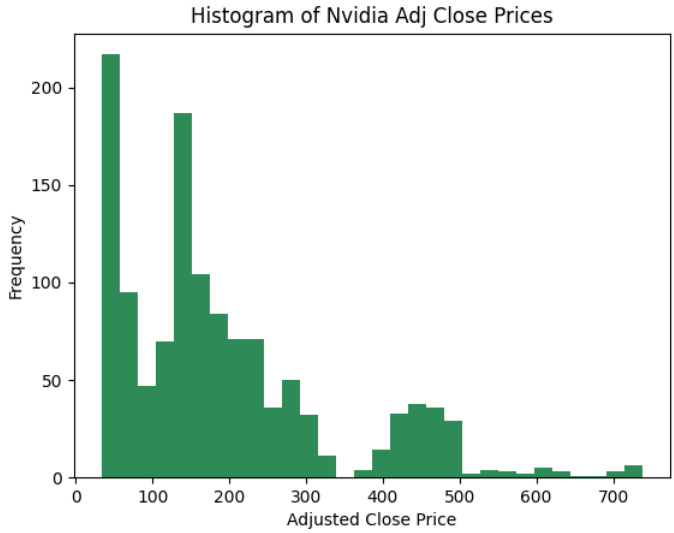

Figure 47. Histogram with custom color

Multiple histograms

You can also plot multiple histograms on the same figure. To illustrate this, let's read the data from the Framingham Heart study.

```
framingham = pd.read_csv('../data/framingham.csv')
```

Now, Let's plot a histogram of Systolic blood pressure for males and females.

© 2024 Bhola Prasad AdaCode.io

```
male_df = framingham[framingham["sex"] == "Male"]
female_df = framingham[framingham["sex"] == "Female"]

# Plot histograms for Systolic blood pressure for Male an\
d Female
plt.hist(male_df["sysBP"], bins=30, label="Male", color="\
tab:blue")
plt.hist(female_df["sysBP"], bins=30, label="Female", col\
or="tab:red")
plt.title("Histogram of Systolic Blood Pressure by Gender\
")
plt.xlabel("Systolic Blood Pressure (sysBP)")
plt.ylabel("Frequency")
plt.legend()
plt.show()
```

© 2024 Bhola Prasad AdaCode.io

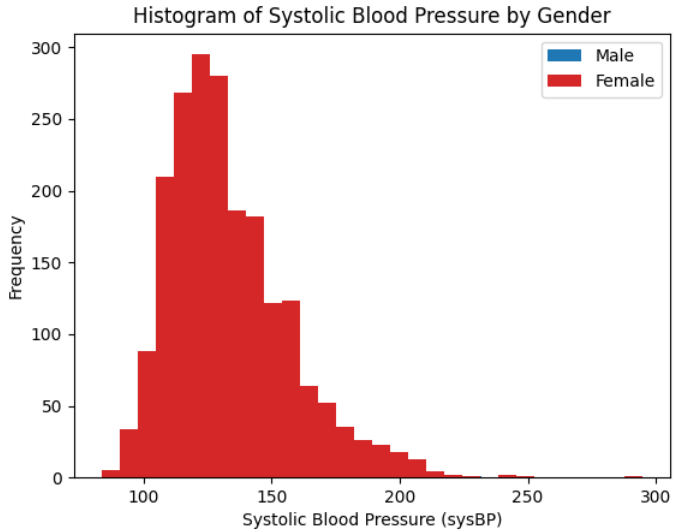

Figure 48. Multiple Histograms

Although we plotted the histogram for males and females, the male data was hidden behind the females. We can solve this problem using several methods. A straightforward solution is to use the `alpha` to add transparency to the plot.

```
# histograms for Male and Female
plt.hist(male_df["sysBP"], bins=30, label="Male", color="\
tab:blue")

plt.hist(female_df["sysBP"], bins=30, alpha=0.5, label="F\
emale", color="tab:red")

plt.title("Histogram of Systolic Blood Pressure by Gender\
")
plt.xlabel("Systolic Blood Pressure (sysBP)")
plt.ylabel("Frequency")
plt.legend()
```

© 2024 Bhola Prasad AdaCode.io

```
13  plt.show()
```

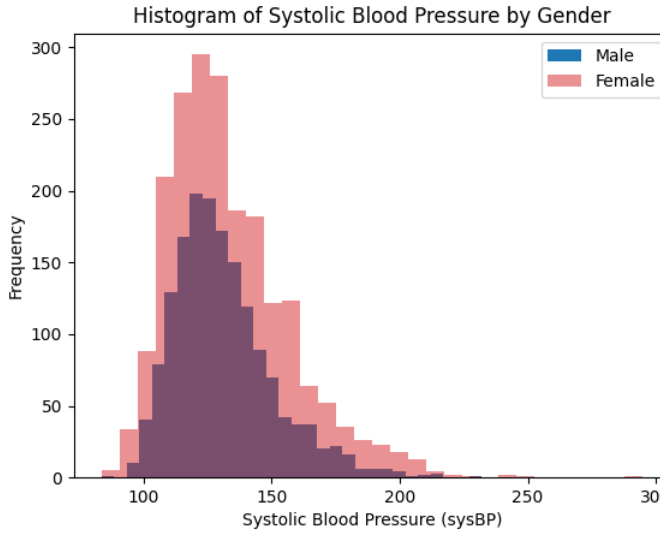

Figure 49. Multiple Histogram with alpha

Histogram types

Another method for solving this problem is using the `histtype` in `plt.hist()`. Setting it to `step` will generate a line plot that is, by default, unfilled. The other histtype are bar(default), which is a traditional bar-type histogram, `barstacked`, where multiple data are stacked on top of each other; and `stepfilled`, which is another variation of the step, but by default, it is filled.

© 2024 Bhola Prasad AdaCode.io

```
# histograms with step hist type
plt.hist(male_df["sysBP"], bins=30, histtype="step", labe\
l="Male", color="blue")

plt.hist(female_df["sysBP"], bins=30, histtype="step", la\
bel="Female", color="red")

plt.title("Histogram of Systolic Blood Pressure by Gender\
")
plt.xlabel("Systolic Blood Pressure (sysBP)")
plt.ylabel("Frequency")
plt.legend()
plt.show()
```

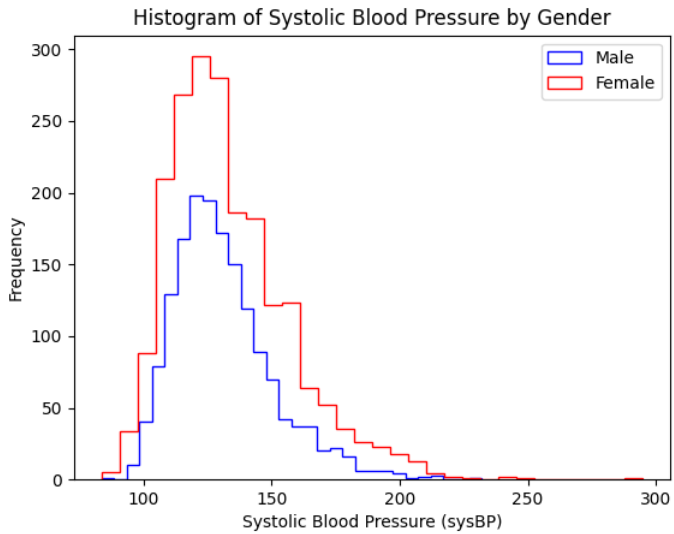

Figure 50. Histogram with histtype='step'

© 2024 Bhola Prasad AdaCode.io

Density histogram

The `plt.hist()` function also has a parameter called `density`, which plots the probability density instead of frequency. When `density=True`, the area under the curve becomes 1, which means that the height of each bar will show the probability of observations falling within each bin relative to the total dataset.

This is useful when comparing the shape of distributions rather than their absolute counts, especially when dealing with datasets of different sizes.

```
# histograms with density=True
plt.hist(
    male_df["sysBP"], bins=30, histtype="step", label="Ma\
le", color="blue", density=True
)

plt.hist(
    female_df["sysBP"],
    bins=30,
    histtype="step",
    label="Female",
    color="red",
    density=True,
)

plt.title("Normalized Histogram of Systolic Blood Pressur\
e by Gender")
plt.xlabel("Systolic Blood Pressure (sysBP)")
plt.ylabel("Density")
plt.legend()
plt.show()
```

© 2024 Bhola Prasad AdaCode.io

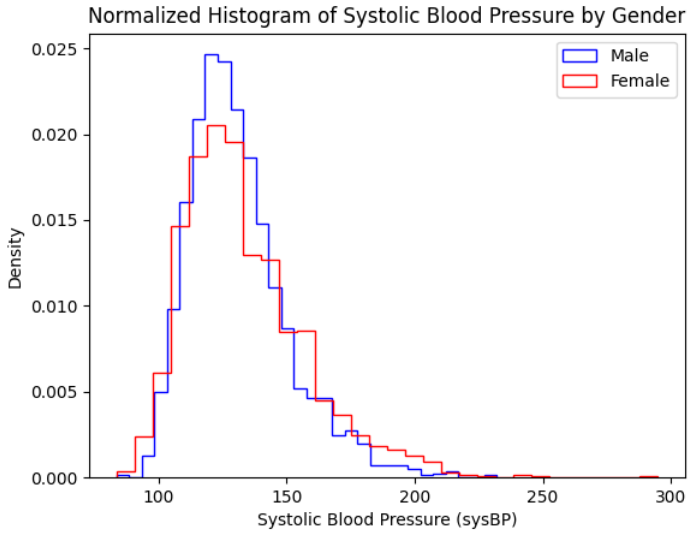

Figure 51. Density Histogram

Orientation

The `orientation` in `plt.hist()` specifies the orientation of the histogram. By default, histogram are plotted vertically, but you can change it by setting `orientation='horizontal'`, which creates a Horizontal histogram.

© 2024 Bhola Prasad AdaCode.io

```
# Horizontal histogram
plt.hist(
    male_df["sysBP"],
    bins=30,
    histtype="step",
    label="Male",
    color="blue",
    orientation="horizontal",
)

plt.hist(
    female_df["sysBP"],
    bins=30,
    histtype="step",
    label="Female",
    color="red",
    orientation="horizontal",
)

plt.title("Horizontal Histogram of Blood Pressure by Gend\
er")
plt.xlabel("Frequency")
plt.ylabel("Systolic Blood Pressure (sysBP)")
plt.legend()
plt.show()
```

© 2024 Bhola Prasad AdaCode.io

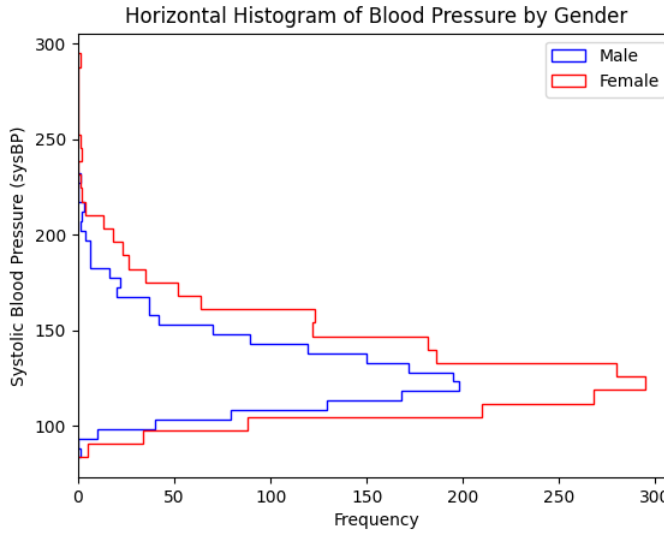

Figure 52. Horizontal Histogram

2D histogram

A 2D Histogram represents the joint distribution of two variables by dividing the plane into bins and counting the number of observations in each bin. It helps visualize the relationship between two numerical variables, similar to a scatter plot, but with a focus on the density of points.

To create a 2D Histogram in matplotlib, use the plt.hist2d()[2] function. Let's make a 2D Histogram to visualize the relationship between systolic and diastolic blood pressure.

[2]https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.hist2d.html#matplotlib.pyplot.hist2d

© 2024 Bhola Prasad AdaCode.io

```
# 2D histogram of blood pressure
plt.hist2d(framingham["sysBP"], framingham["diaBP"], bins\
=30)
plt.colorbar()
plt.title("2D Histogram of Blood Pressure")
plt.xlabel("Systolic Blood Pressure")
plt.ylabel("Diastolic Blood Pressure")
plt.show()
```

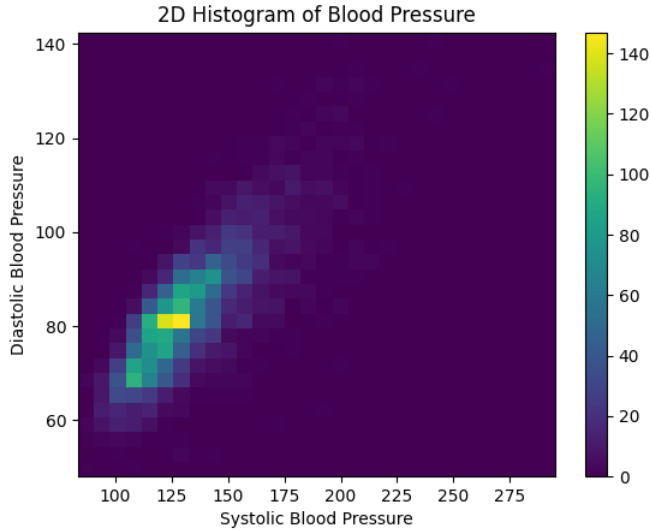

Figure 53. 2D Histogram

Exercise 5.1

1. Create a Histogram of `BMI`(Body Mass Index).
2. Create a Multiple Histogram of BMI based on Gender.

© 2024 Bhola Prasad AdaCode.io

3. Apply various strategies to rectify the plot if there is too much overlap.
4. Create a Density histogram of `BMI`.
5. Create a 2D Histogram of `BMI` and `totChol` (Total cholesterol).

Summary

- To create a Histogram in matplotlib, use the `plt.hist()` function.
- To change the bin size, use `bins`.
- To change color of the histogram, use `color`.
- Add another `plt.hist()` function to create another histogram on the same plot.
- Use `alpha` to add transparency to the histogram.
- Use `histtype` to create different types of histograms.
- Use `density` to create a Density histogram.
- Set `orientation='horizontal'` to create a Horizontal histogram.
- To create a 2D Histogram, use the `plt.hist2d()` function.

Solution

Exercise 5.1

© 2024 Bhola Prasad AdaCode.io

```
# Create a Histogram of BMI
plt.hist(framingham['BMI'], color='crimson')
plt.title('Histogram of BMI')
plt.xlabel('Body Mass Index')
plt.ylabel('Frequency')
plt.show()
```

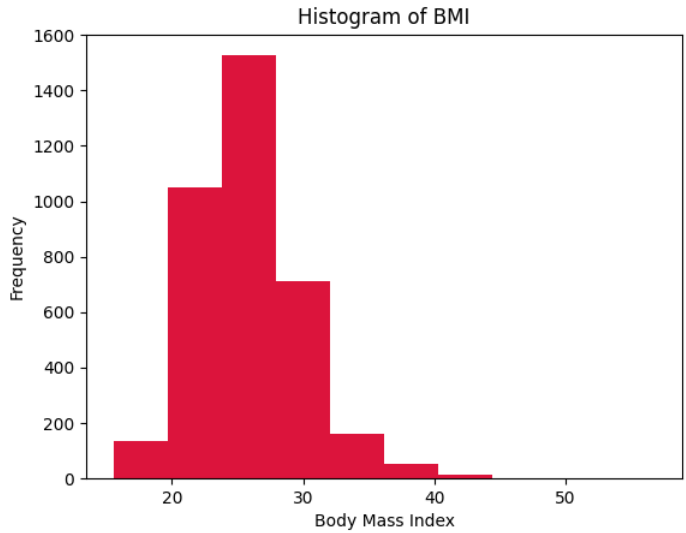

Figure 54. Histogram of Body Mass Index

© 2024 Bhola Prasad AdaCode.io

```
# Create a Multiple Histogram of BMI based on Gender.
male_df = framingham[framingham["sex"] == "Male"]
female_df = framingham[framingham["sex"] == "Female"]

plt.hist(male_df["BMI"], color="tab:blue", label="Male")
plt.hist(female_df["BMI"], color="seagreen", label="Femal\
e")
plt.title("Histogram of BMI by Gender")
plt.xlabel("BMI")
plt.ylabel("Frequency")
plt.legend()
plt.show()
```

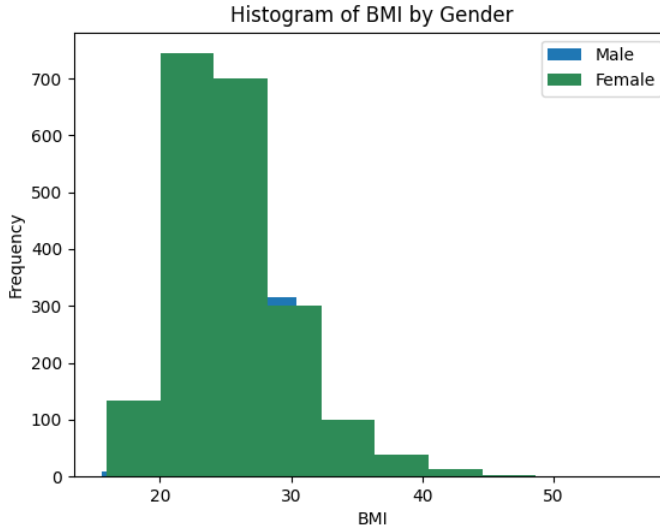

Figure 55. Histogram of BMI by Gender

© 2024 Bhola Prasad AdaCode.io

```
# Rectify the problem of overlapping by using alpha
plt.hist(male_df['BMI'], color='tab:blue', label='Male')
plt.hist(female_df['BMI'], color='seagreen', alpha=0.5, l\
abel='Female')
plt.title('Histogram of BMI by Gender')
plt.xlabel('BMI')
plt.ylabel('Frequency')
plt.legend()
plt.show()
```

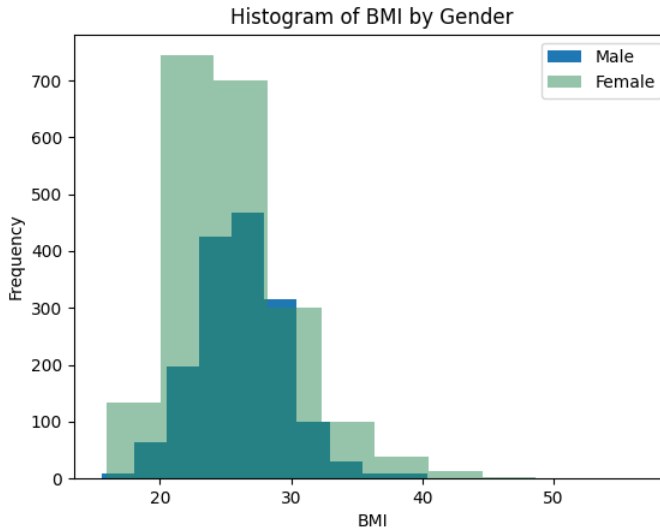

Figure 56. Histogram with alpha

© 2024 Bhola Prasad AdaCode.io

```
# Rectify overlapping by changing the histogram type
plt.hist(male_df["BMI"], color="tab:blue", histtype="step\
", label="Male")

plt.hist(female_df["BMI"], color="seagreen", histtype="st\
ep", label="Female")

plt.title("Histogram of BMI by Gender")
plt.xlabel("BMI")
plt.ylabel("Frequency")
plt.legend()
plt.show()
```

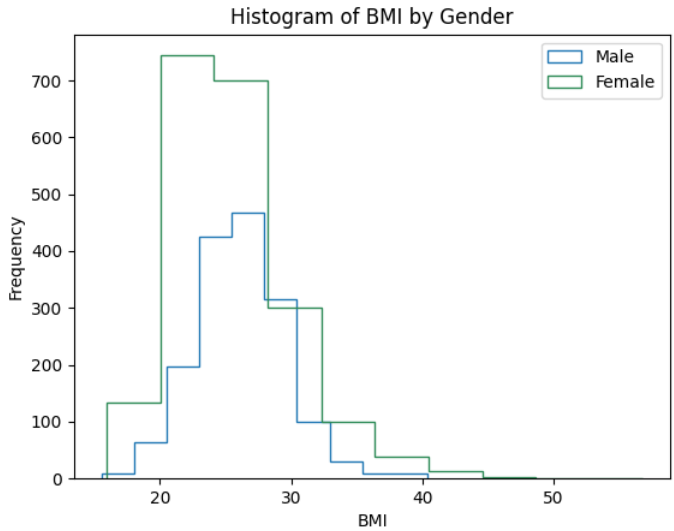

Figure 57. Histogram with histtype='step'

© 2024 Bhola Prasad AdaCode.io

```
# Create a density histogram.
plt.hist(male_df["BMI"], color="crimson", density=True, h\
isttype="step", label="Male")

plt.hist(female_df["BMI"], color="green", density=True, h\
isttype="step", label="Female")

plt.title("Histogram of BMI by Gender")
plt.xlabel("BMI")
plt.ylabel("Density")
plt.legend()
plt.show()
```

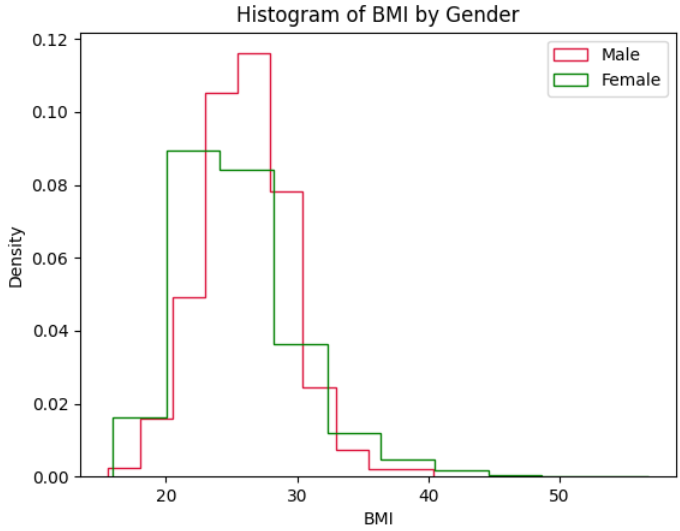

Figure 58. Density Histogram

© 2024 Bhola Prasad AdaCode.io

```
# 2D histogram of BMI and totChol
plt.hist2d(framingham['BMI'], framingham['totChol'])
plt.colorbar()
plt.title('2D Histogram of BMI and Total Cholesterol')
plt.xlabel('BMI')
plt.ylabel('Total Cholesterol')
plt.show()
```

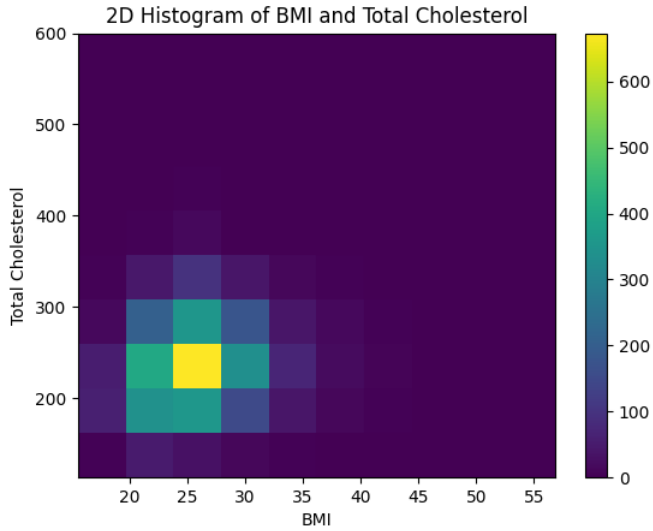

Figure 59. 2D Hist

© 2024 Bhola Prasad AdaCode.io

Chapter 6. Box plot

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/matplotlib-python.

Create a box plot

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/matplotlib-python.

Multiple box plots

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/matplotlib-python.

Colors

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/matplotlib-python.

Notched box plot

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/matplotlib-python.

Outliers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/matplotlib-python.

Remove outliers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/matplotlib-python.

Horizontal box plot

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/matplotlib-python.

Exercise 6.1

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/matplotlib-python.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/matplotlib-python.

© 2024 Bhola Prasad AdaCode.io

Solution

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/matplotlib-python.

Exercise 6.1

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/matplotlib-python.

© 2024 Bhola Prasad AdaCode.io

Chapter 7. Axes and Subplots

Axes and Subplots are the core building blocks for creating data visualizations in matplotlib. Instead of being limited to a single plot, subplots gives you the ability to arrange multiple plots within one figure. This lets you easily compare different datasets side-by-side. Whether you're exploring relationships within your data or building complex visual dashboards,axes and subplots are the key to unlocking Matplotlib's full potential.

Figure, axes, axis, and subplots

Before we dive into creating more complex charts in matplotlib, It's essential to grasp some basic terms: figure, axes, axis, and subplots. If we're unclear on these from the start, things can get confusing. Understanding these terms helps make everything else much easier to handle as we go along.

Figure

In Matplotlib, a **Figure** acts as a container that holds all the plot elements. Think of it as the canvas on which your plots will be drawn. Creating a Figure in matplotlib is usually the first step in plotting data. You do this by calling `plt.figure()`, which initializes a new Figure object. This object provides the foundation upon which you'll build your visualization, and add axes, labels, and other elements.

To understand it visually, create a Python file `figure.py`, which we will use to demonstrate, as running the following code in Jupyter Notebook doesn't show the figure.

```
import matplotlib.pyplot as plt

fig = plt.figure(figsize=(5, 4), facecolor="teal")
plt.show()
```

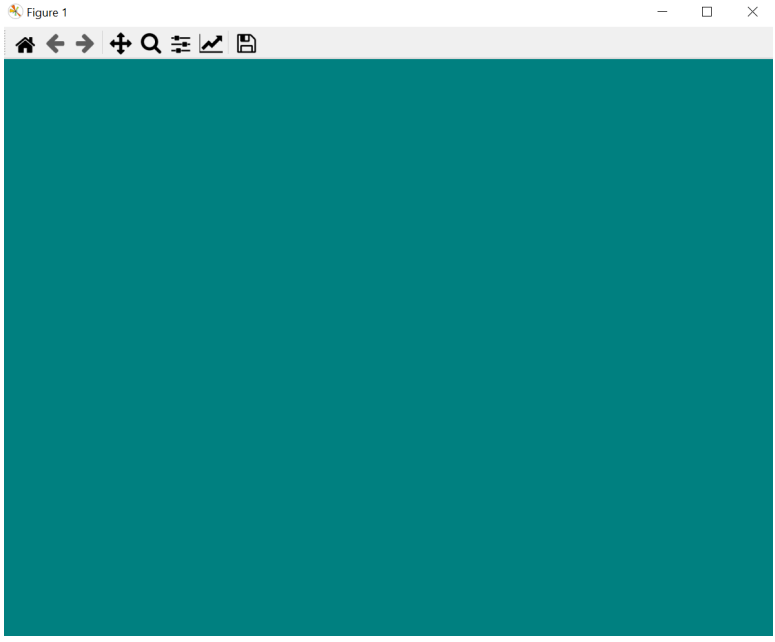

Figure 60. Figure with teal background

This teal window is the figure in matplotlib, everything will be plotted upon this canvas. You can add anything such as image, text, plots, labels, etc. Let's add the text "Hello, World!". Close the previous figure window before running this code.

© 2024 Bhola Prasad AdaCode.io

```
fig = plt.figure(figsize=(5, 4), facecolor="teal")

fig.text(0.5, 0.5, "Hello, world!", ha="center", va="cent\
er", size=20, color="white")
plt.show()
```

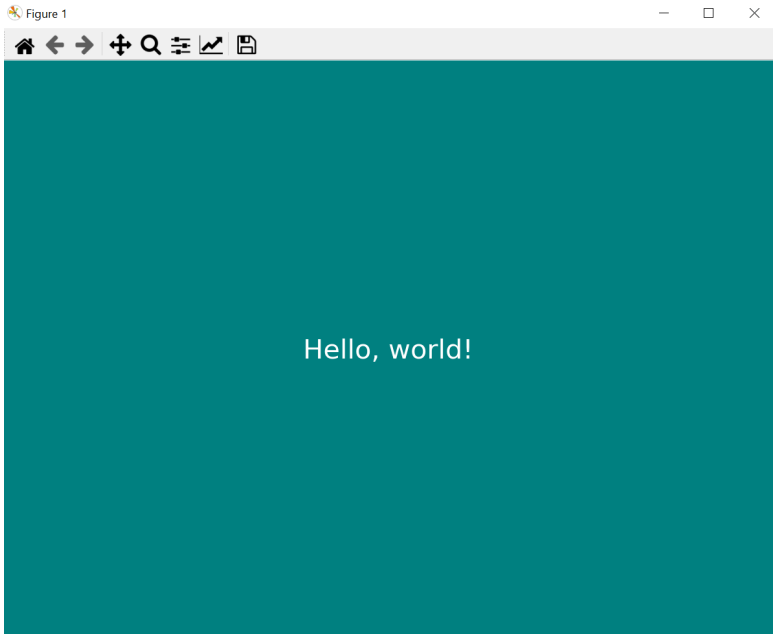

Figure 61. A Text on a Figure

You can learn more about how to control the widget here - Interactive nav[1]

[1] https://matplotlib.org/stable/users/explain/figure/interactive.html#interactive-navigation

© 2024 Bhola Prasad AdaCode.io

Axes

In Matplotlib, **Axes** refers to the part of your plot where the data is actually drawn, and it's what you might commonly think of as the plot itself. An axes object includes elements like lines or bars that represent your data, the labels for x-axis and y-axis, the ticks on those axes, and other elements.

Each axes objects sits in a figure, and a single figure can contains one or more axes objects. This means you can have multiple plots in one figure. when you hear `axes` in the context of matplotlib, think of it as the specific area on the grid in a figure where your data is visualized.

Let's add an axes to our figure. Now, You can run the following code in Jupyter Notebook as usual.

```
# Create a figure
fig = plt.figure(figsize=(6, 4), facecolor="teal")

# Add a single axes to the figure
# The (1, 1, 1) mean (rows, columns, subplot index)
ax = fig.add_subplot(1, 1, 1)

# Optionally, set the axes background color
ax.set_facecolor("deeppink")
plt.show()
```

© 2024 Bhola Prasad AdaCode.io

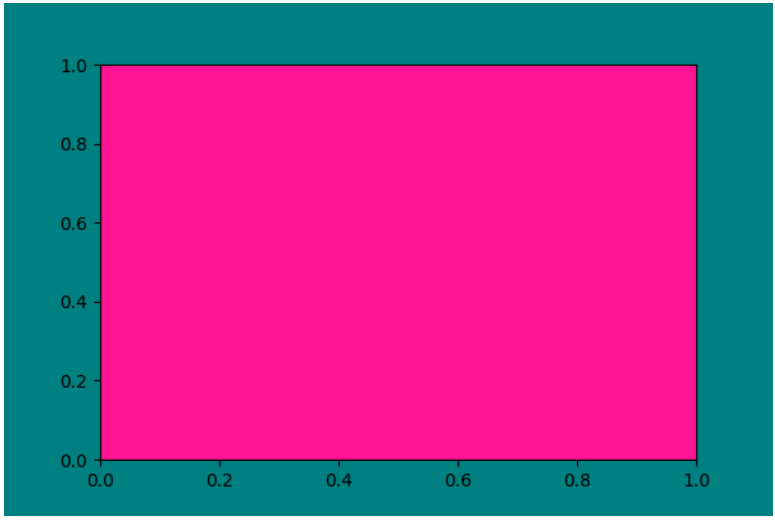

Figure 62. Figure with an Axes

The `Figure` is the entire space with the `teal` background, while the `axes` is the area with a deep pink background where data will be plotted. By default, matplotlib has added some ticks and tick labels which will change when we add some data.

Let's add a plot to this axes. We will add a simple line chart to it.

```
import warnings

warnings.filterwarnings("ignore")

import matplotlib.pyplot as plt
import pandas as pd

# read data
nvidia = pd.read_csv("../data/NVDA.csv", parse_dates=["Da\
te"])

# Create a figure
```

© 2024 Bhola Prasad AdaCode.io

```
fig = plt.figure(figsize=(6, 4), facecolor="teal")

# add a line chart
ax = fig.add_subplot(1, 1, 1)
ax.plot(nvidia["Date"], nvidia["Adj Close"])
ax.set_title("Nvidia Stock Prices (2019 - 2024)")

# set the color for the axes
ax.set_facecolor("deeppink")
plt.show()
```

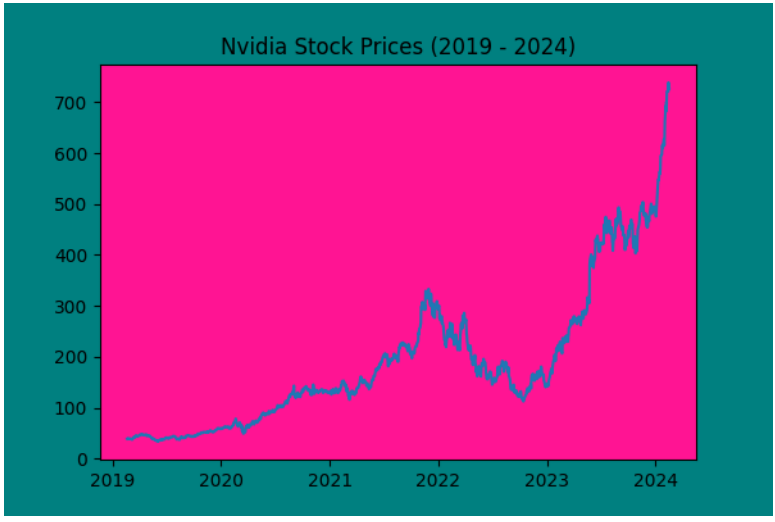

Figure 63. Axes with a plot on a Figure

Axis

In Matplotlib, when we talk about an `axis`, we are referring to the lines you see on a graph that mark the edges of the plotting area.

© 2024 Bhola Prasad AdaCode.io

These lines have numbers on them, which help you understand the size or value of what you are looking at. Every graph has at least two axis: the x-axis, which goes from left to right, and the y-axis, which goes up from bottom.

The `Axis` also has little marks (we call these `ticks`), which help us read specific values more easily. In Matplotlib, you can change how these axis look and what numbers they show. Simply put, the axis in Matplotlib is like the measuring tape that helps you and everyone else understand the size and scale of your data.

Let's customize the x-axis and y-axis and see it in action.

```
# import libraries
import matplotlib.pyplot as plt
import matplotlib.dates as mdates

# Create a figure
fig = plt.figure(figsize=(6, 4), facecolor="teal")

# add a line chart
ax = fig.add_subplot(1, 1, 1)
ax.plot(nvidia["Date"], nvidia["Adj Close"])
ax.set_title("Nvidia Stock Prices (2019 - 2024)")


# Customize the x-axis to show every 2 years
ax.xaxis.set_major_locator(mdates.YearLocator(2))
ax.xaxis.set_major_formatter(mdates.DateFormatter("%Y"))
ax.set_xlabel("Year")

# customize the y-axis to show $200 interval
ax.set_yticks(range(0, 701, 200))
ax.set_ylabel("Adj Close Price")


# set the axes background color
```

© 2024 Bhola Prasad AdaCode.io

```
ax.set_facecolor("deeppink")

# show plot
plt.show()
```

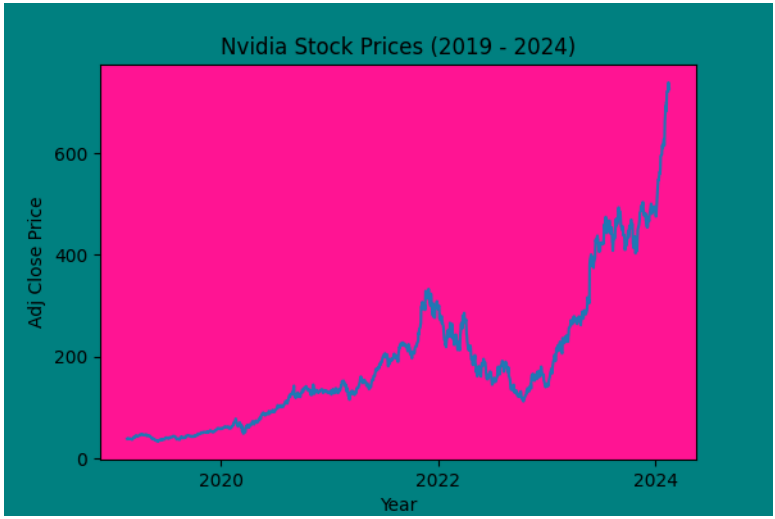

Figure 64. Line chart with custom x-axis & y-axis

Subplots

Subplots in Matplotlib are a way to arrange multiple axes (plots) within a single figure. Each subplots represents an individual plot, with its own axes and chart type. Subplots are very useful for comparing data, showing trends, or highlighting relationship between different datasets.

When you create subplots, you are essentially divide the figure into a grid, where each cell of the grid contains its own axes object with

© 2024 Bhola Prasad AdaCode.io

a distinct plot. The grid is defined by rows and columns, allowing for a flexible layout of your visualization.

In essence, an Axes is about the `what` of your data visualization (What you are plotting), while subplots are about the `how`(how you are organizing multiple plots in relation to each other).

```
import matplotlib.pyplot as plt

# Create a figure
fig = plt.figure(figsize=(7, 4), facecolor="teal")

# Add the first subplot to the figure. This creates an ax\
es
# in the left half of the figure.
ax1 = fig.add_subplot(1, 2, 1)  # (1 row, 2 columns, 1st \
subplot)
ax1.set_facecolor("deeppink")
ax1.set_title("First Axes")

# Add the second subplot to the figure, creating another \
axes
# in the right half of the figure.
ax2 = fig.add_subplot(1, 2, 2)  # (1 row, 2 columns, 2nd \
subplot)
ax2.set_facecolor("dodgerblue")
ax2.set_title("Second Axes")

plt.show()
```

© 2024 Bhola Prasad AdaCode.io

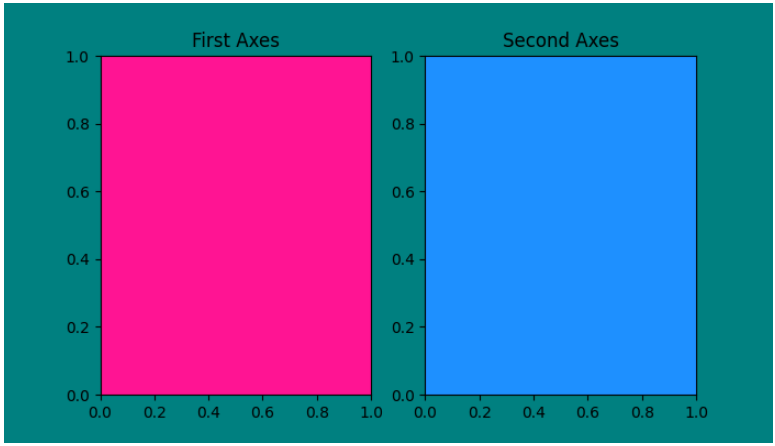

Figure 65. Subplots in Matplotlib

Figure: The entire window or canvas, which in this case, has a teal background. it's the base where all plots are drawn.

First Axes (left subplot): This is the first plot area within the figure, with a pink background. It's labeled as `First Axes`, illustrating that it's an independent plot area where you could visualize data.

Second Axes(Right subplot): This is the second plot area, distinct from the first, with a dodger blue background. It's labeled as `Second Axes`, showing another space for a different data visualization or the same data presented differently.

In Matplotlib, adding a subplot to a figure automatically creates an axes object, but there are different things, although used interchangeably by people. In simple terms, the act of adding a subplot is like saying "I want a plot here in this part of the figure', and the axes is the realization of that plot, providing the space and tools to actually draw and display the data.

I hope this clarifies things about Figure, Axes, Axis, and Subplots in Matplotlib.

© 2024 Bhola Prasad AdaCode.io

Now, let's tackle one more confusing aspect of Matplotlib: how we create charts or plots in Matplotlib?

Matplotlib application interface

If you notice carefully, the way we create figures in this chapter differs from how we created figures in chapters 2-6. The Matplotlib Application Interface (API) provides two ways of creating figures: the `pyplot` interface (also called the implicit interface) and the `object-oriented` interface (also called the explicit interface).

You can use either interface or mix both interfaces if you want to; this is where all the confusion arises.

Pyplot Interface

The `pyplot` interface resembles MATLAB's plotting interface. In this interface, figures and axes are created implicitly, which means pyplot creates and manages the figures and axes for us and adds other objects to it. This interface is helpful if you want to create simple plots quickly. From chapters 2-6, we used this interface to create plots. Let's read a dataset and see an example of this interface again.

© 2024 Bhola Prasad AdaCode.io

```
# ignore warnings
import warnings
warnings.filterwarnings('ignore')

# import libraries
import pandas as pd
import matplotlib.pyplot as plt

# read data
df = pd.read_csv("../data/penguins.csv")
df.dropna(inplace=True)
df.head()
```

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
0	Adelie	Torgersen	39.1	18.7	181.0	3750.0	MALE
1	Adelie	Torgersen	39.5	17.4	186.0	3800.0	FEMALE
2	Adelie	Torgersen	40.3	18.0	195.0	3250.0	FEMALE
4	Adelie	Torgersen	36.7	19.3	193.0	3450.0	FEMALE
5	Adelie	Torgersen	39.3	20.6	190.0	3650.0	MALE

Figure 66. Penguins Data

We will create a simple scatter plot. This time, we will plot `bill_depth_mm` on the x-axis and `body_mass_g` on the y-axis.

```
# create a figure
plt.figure(figsize=(6, 5))

# create a scatter plot
plt.scatter(df['bill_depth_mm'], df['body_mass_g'],
            color='magenta')
plt.title("Penguins Bill depth vs Body mass")
plt.xlabel("Bill Depth")
plt.ylabel("Body Mass")
plt.show()
```

© 2024 Bhola Prasad AdaCode.io

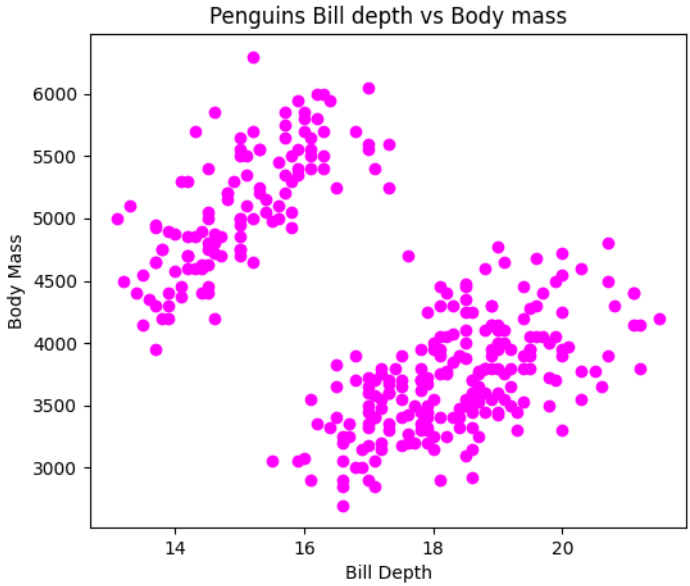

Figure 67. Scatter plot with pyplot interface

In this example, the `plt.figure()`, creates a figure and `plt.scatter()` plots the data into the current figure. You can skip the `plt.figure()` call if you do require figure size or other properties, the `pyplot` interface will create a figure for us with default figure size and other default figure properties.

This behavior highlights the convenience and ease of use of the `pyplot` interface. It makes it straightforward to create visualizations without needing to explicitly manage figures and axes objects. It is convenient for quick data exploration and creation of simple plots.

© 2024 Bhola Prasad AdaCode.io

Object-oriented interface

The `Object-oriented` interface is more powerful and provides interface is more powerful and provides finer control over the figure and axes we create. In this interface, figures and axes are created explicitly, and we use methods on them to create other objects step by step. This method is useful when we want to create more complex plots that require more control and customizations. We introduced this interface in this chapter. Let' recreate the above plot using the `object-oriented` interface.

```
# import matplotlib
import matplotlib.pyplot as plt

# Explicitly creating a figure and an axes object
fig, ax = plt.subplots(figsize=(6, 5))

# Creating a scatter plot using the axes object
ax.scatter(df["bill_depth_mm"], df["body_mass_g"], color=\
"magenta")

# Adding title and labels
ax.set_title("Penguins Bill depth vs Body mass")
ax.set_xlabel("Bill depth")
ax.set_ylabel("Body mass")
plt.show()
```

© 2024 Bhola Prasad AdaCode.io

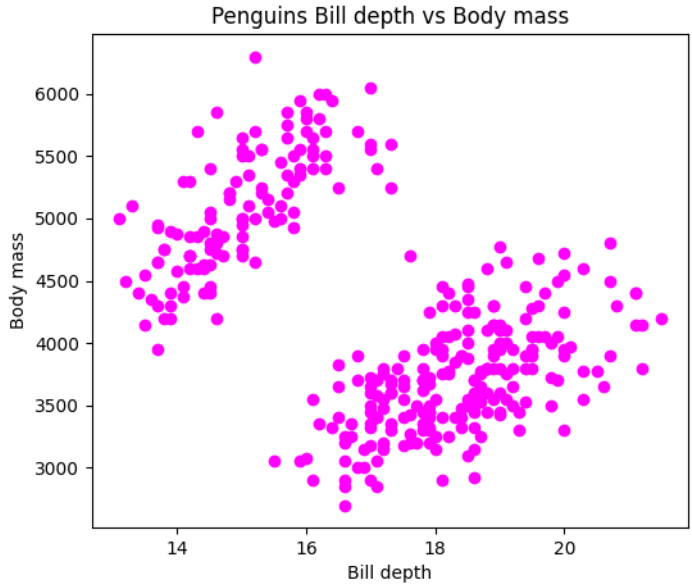

Figure 68. Scatter plot with Object-oriented interface

Here, we've recreated the scatter plot using the `object-oriented` interface of matplotlib. This approach involved explicitly creating a figure and axes object; then, we used the axes object `ax` to generate the scatter plot, add a title, and label the x and y-axis.

This example demonstrates how the `object-oriented` interface offers detailed control over the plot through methods applied directly to figure and axes objects. This method enables more complex customizations and management of plot elements, and it is particularly useful for creating advanced plots.

Bonus Points if you can add a third variable, `species` to this plot using the object-oriented interface without looking at the code below. I assume you have given it a try. Let's see how to do it.

© 2024 Bhola Prasad AdaCode.io

```
# unique species
unique_species = df["species"].unique()

# create a figure and an axes
fig, ax = plt.subplots(figsize=(6, 5))

# create scatter plots
for species in unique_species:
    species_df = df[df["species"] == species]
    ax.scatter(species_df["bill_depth_mm"], species_df["b\
ody_mass_g"], label=species)

# Add a legend to the plot
ax.legend()

# Add titles and labels
ax.set_title("Bill depth vs Body Mass by Species")
ax.set_xlabel("Bill depth")
ax.set_ylabel("Body mass")

# add grid lines
ax.grid(True)
plt.show()
```

© 2024 Bhola Prasad AdaCode.io

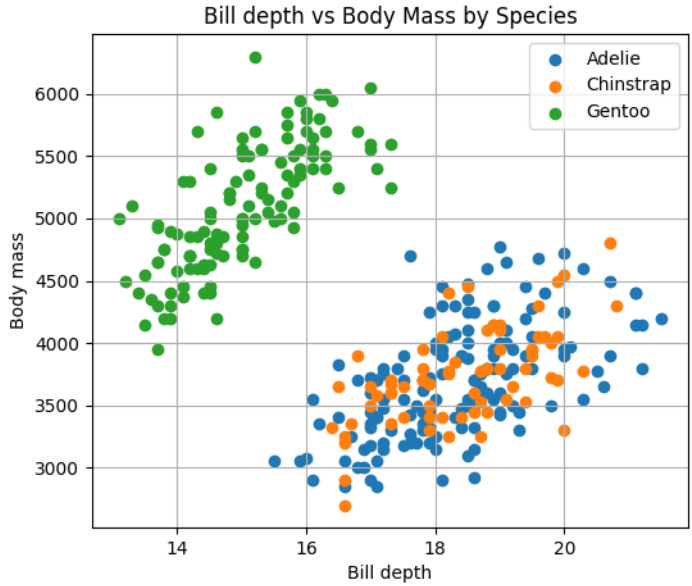

Figure 69. Scatter plot with Object-oriented interface

plt.subplot() vs plt.subplots()

Let's unpack one more thing: What is the difference between `plt.subplot()` and `plt.subplots()`, and when should we use which one?

In matplotlib, both `plt.subplot()` and `plt.subplots()` are used to create multiple plots within a single figure, but their approach is different. Understanding the difference can help you shoose the proper function for your plots.

© 2024 Bhola Prasad AdaCode.io

plt.subplot()

The plt.subplot()[2] function adds a single subplot to an existing figure at a specified grid position. It allows us to create multiple subplots in the same figure, one at a time. Each subplot will have its own Axes.

The `plt.subplot(nrows, ncols, index)` function defines the layout of the grid and the position of the subplot in that grid. The grid is determined by the number of rows (`nrows`) and columns (`ncols`), and the `index` specifies the position of the subplot in the grid. The index starts from 1, left to right, and top to bottom.

```
# calculate average body mass by species
species_bm = df.groupby("species")["body_mass_g"].mean().\
reset_index()

# create a figure
plt.figure(figsize=(8, 6))

# 1 row, 2 columns, 1st subplot for scatter plot
plt.subplot(1, 2, 1)
plt.scatter(df["bill_length_mm"], df["body_mass_g"], colo\
r="seagreen")
plt.xlabel("Bill Length")
plt.ylabel("Average Body Mass")

# 1 row, 2 columns, 2nd subplot for bar chart
plt.subplot(1, 2, 2)
plt.bar(
    species_bm["species"],
    species_bm["body_mass_g"],
    color=["tab:blue", "tab:green", "tab:red"],
)
plt.xlabel("Penguin Species")
```

[2] https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.subplot.html

© 2024 Bhola Prasad AdaCode.io

```
plt.ylabel("Average Body Mass")


# Add a title for the whole figure
plt.suptitle("Penguin Bill Length and Body Mass Analysis")
plt.tight_layout()

plt.show()
```

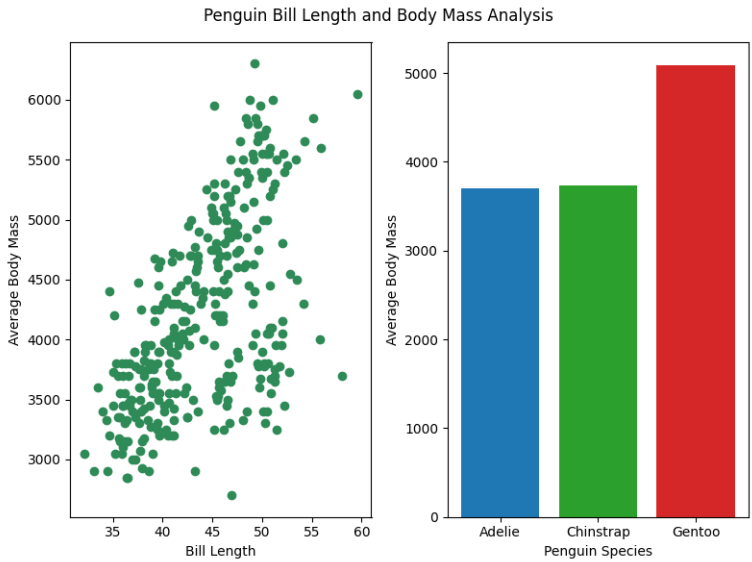

Figure 70. plt.subplot() in Matplotlib

The `plt.suptitle()` is used to add a centered title for the whole figure.

© 2024 Bhola Prasad AdaCode.io

plt.subplots()

The plt.subplots()[3] function offers a more convenient way to create a figure and a set of subplots simultaneously. It returns a figure and an array of Axes objects. Unlike `plt.subplot()`, which adds subplots one at a time, `plt.subplots()` initializes all the subplots simultaneously.

The `plt.subplots(nrows, ncols)` has two main parameters, the number of rows (`nrows`) and columns (`ncols`) to specify the grid layout and it returns a figure and an array of axes obejcts that can be manipulated individually.

Let's recreate the above plot using `plt.subplots()`.

```
# Create a figure and a grid
# with 1 row and 2 columns
fig, axs = plt.subplots(1, 2, figsize=(8, 6))

# Scatter plot on the first subplot
axs[0].scatter(df["bill_length_mm"], df["body_mass_g"], c\
olor="seagreen")
axs[0].set_xlabel("Bill Length")
axs[0].set_ylabel("Average Body Mass")

# Bar chart on the second subplot
axs[1].bar(
    species_bm["species"],
    species_bm["body_mass_g"],
    color=["tab:blue", "tab:green", "tab:red"],
)
axs[1].set_xlabel("Penguins Species")
axs[1].set_ylabel("Average Body Mass")

# Add a title for the whole figure
```

[3] https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.subplots.html

© 2024 Bhola Prasad AdaCode.io

```
21 plt.suptitle("Penguin Bill Length and Body Mass Analysis")
22 plt.tight_layout()
23 plt.show()
```

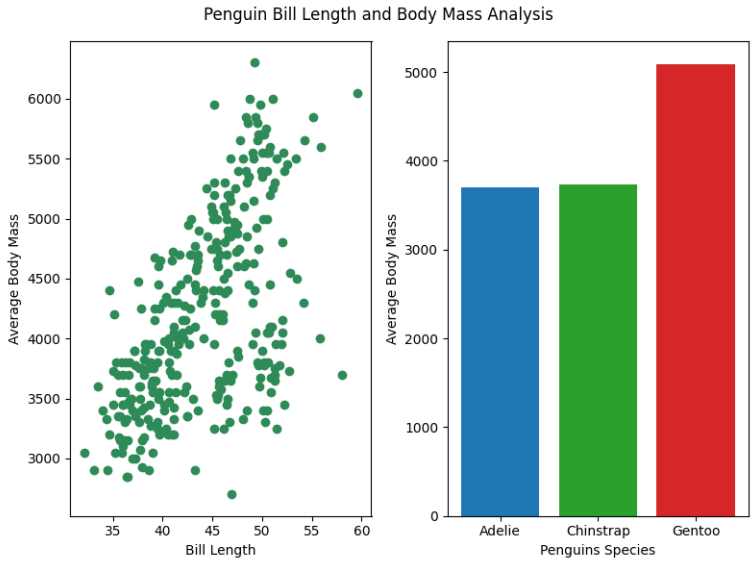

Figure 71. plt.subplots() in matplotlib

Let's take a break. In the next chapter, we will learn an even more complex way of arranging multiple axes on a figure. But before we go, let's do a quick exercise to reinforce everything we learned in this chapter.

Exercise 7.1

1. Load the datasets for `TSLA`, `NVDA`, `NFLX`, and `BRK-B` into separate pandas dataframe.

© 2024 Bhola Prasad AdaCode.io

2. Using the `plt.subplot()` function from the pyplot interface, create a single figure with four subplots arranged in 1 row and 4 columns.
3. In each subplot, generate a line chart for one of the companies. Use the `Date` column for x-axis and `Adj Close` price for the y-axis.
4. Add x and y-axis labels and a legend that identifies the company.
5. Add a title for the whole figure.
6. Repeat the visualization process, this time using the plt.subplots() function. Arrange the subplots in a grid with 2 rows and 2 columns.
7. For the new figure, organise the subplots in the following positions: `NVDA: [0,0]`, `TSLA: [0,1]`, `NFLX: [1,0]`, and `BRK-B: [1,1]`.
8. If you encounter difficulties, refer to the solution provided at the end of this exercise. A more detailed explanation and walkthrough will be available in the next chapter.

Summary

- A **Figure** in Matplotlib acts as a container for all plot elements, allowing the addition of plots, texts, images, or any other graphical objects.
- An **Axes** in Matplotlib represents a single plot or graph. It is the primary building block for creating various data visualizations, such as line and bar charts.
- An **Axis** refers to the number lines we usually see on graphs. It controls the graph limits, tick marks, and their labels. An axes object contains at least two axis for 2D and three for 3D plots.
- Matplotlib provides two interfaces for creating plots: the pyplot interface and the object-oriented interface.

© 2024 Bhola Prasad AdaCode.io

- The `pyplot` interface is creates plots implicitly and is useful if you want to create simple plots quickly.
- The `object-oriented` interface involves explicitly creating figures and axes, offering more control and flexibility. It is preferred for creating complex plots that require detailed customization.
- The `plt.subplot()` function adds a single subplot to an existing figure at a specified grid position, creating multiple subplots within the same figure, one at a time.
- The `plt.subplots()` function simultaneously creates a figure and a set of subplots. It returns a figure and an array of Axes objects, which can be manipulated individually.

Solution

Exercise 7.1

```
import warnings
warnings.filterwarnings('ignore')

import pandas as pd
import matplotlib.pyplot as plt

# read data
nvidia = pd.read_csv("../data/NVDA.csv", parse_dates=['Da\
te'])
tesla = pd.read_csv("../data/TSLA.csv", parse_dates=['Dat\
e'])
netflix = pd.read_csv("../data/NFLX.csv", parse_dates=['D\
ate'])
berkshire = pd.read_csv("../data/BRK-B.csv", parse_dates=\
['Date'])

```

© 2024 Bhola Prasad AdaCode.io

```
# create a figure
plt.figure(figsize=(12, 5))

# 1 row, 4 columns, 1st subplot
plt.subplot(1, 4, 1)
plt.plot(nvidia["Date"], nvidia["Adj Close"],
         label='Nvidia', color='seagreen')
plt.xlabel("Year")
plt.ylabel("Adj Close price")
plt.legend()

# 1 row, 4 columns, 2st subplot
plt.subplot(1, 4, 2)
plt.plot(tesla["Date"], tesla["Adj Close"],
         label='Tesla', color='black')
plt.xlabel("Year")
plt.legend()


# 1 row, 4 columns, 3rd subplot
plt.subplot(1, 4, 3)
plt.plot(netflix["Date"], netflix["Adj Close"],
         label='Netflix', color='red')
plt.xlabel("Year")
plt.legend()

# 1 row, 4 columns, 4th subplot
plt.subplot(1, 4, 4)
plt.plot(berkshire["Date"], berkshire["Adj Close"],
         label='Berkshire', color='dodgerblue')
plt.xlabel("Year")
plt.legend()

plt.suptitle("Stock Prices (2019-2024)")
plt.tight_layout()
```

© 2024 Bhola Prasad AdaCode.io

```
52  plt.show()
```

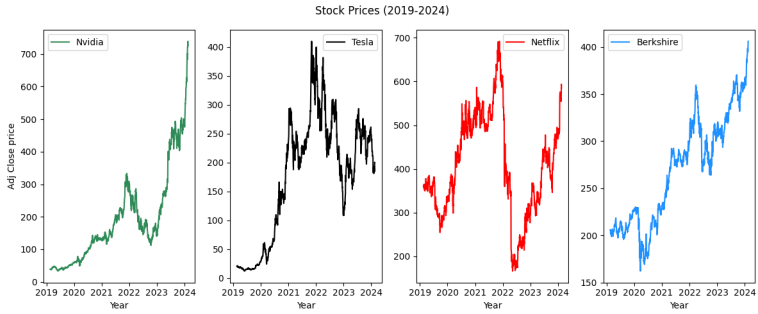

Figure 72. Stock prices (2019-2024)

Let's recreate this figure using the `Object-oriented` interface.

```
1   import matplotlib.pyplot as plt
2
3   # create a figure and 4 axes
4   # using 2 rows and 2 columns
5   fig, axs = plt.subplots(2, 2, figsize=(8, 5))
6
7   # 1st row, 1st column
8   axs[0, 0].plot(nvidia["Date"], nvidia["Adj Close"],
9            label='Nvidia', color='seagreen')
10  axs[0, 0].set_xlabel("Year")
11  axs[0, 0].set_ylabel("Adj Close Price")
12  axs[0, 0].legend()
13
14  # 1st row, 2nd column
15  axs[0, 1].plot(tesla["Date"], tesla["Adj Close"],
16           label='Tesla', color='black')
17  axs[0, 1].set_xlabel("Year")
18  axs[0, 1].legend()
```

© 2024 Bhola Prasad AdaCode.io

```
19
20  # 2nd row, 1st column
21  axs[1, 0].plot(netflix["Date"], netflix["Adj Close"],
22           label='Netflix', color='crimson')
23  axs[1, 0].set_xlabel("Year")
24  axs[1, 0].set_ylabel("Adj Close Price")
25  axs[1, 0].legend()
26
27  # 2nd row, 2nd column
28  axs[1, 1].plot(berkshire["Date"], berkshire["Adj Close"],
29           label='Berkshire', color='dodgerblue')
30  axs[1, 1].set_xlabel("Year")
31  axs[1, 1].legend()
32
33  fig.suptitle("Stock Prices (2019-2024)")
34  fig.tight_layout()
35  plt.show()
```

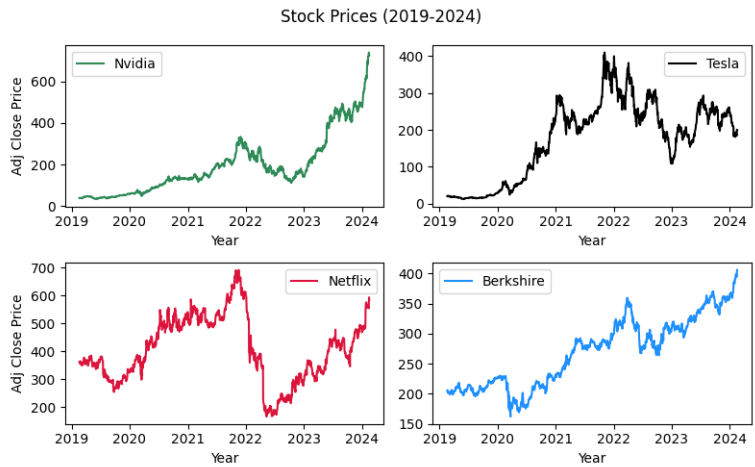

Figure 73. Stock prices (2019-2024)

© 2024 Bhola Prasad AdaCode.io

Chapter 8. Arranging Multiple Axes in a Figure

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/matplotlib-python.

Creating grids

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/matplotlib-python.

1x2 grid

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/matplotlib-python.

subplots

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/matplotlib-python.

subplot_mosaic

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/matplotlib-python.

2x1 grid

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/matplotlib-python.

2x2 grid

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/matplotlib-python.

Axes spanning rows or columns in a grid

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/matplotlib-python.

Variable widths and heights in a grid

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/matplotlib-python.

Nested axes layouts

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/matplotlib-python.

© 2024 Bhola Prasad AdaCode.io

subfigures

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/matplotlib-python.

Exercise 8.1

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/matplotlib-python.

Exercise 8.2

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/matplotlib-python.

Exercise 8.3

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/matplotlib-python.

Solution

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/matplotlib-python.

Exercise 8.1

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/matplotlib-python.

© 2024 Bhola Prasad AdaCode.io

Exercise 8.2

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/matplotlib-python.

Exercise 8.3

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/matplotlib-python.

© 2024 Bhola Prasad AdaCode.io

Chapter 9. Axis Scales and Ticks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/matplotlib-python.

Linear Scale

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/matplotlib-python.

Logarithmic Scale

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/matplotlib-python.

Symmetrical Logarithmic Scale

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/matplotlib-python.

Custom Scale

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/matplotlib-python.

Other Scales

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/matplotlib-python.

Tick Locations and Labels

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/matplotlib-python.

Locators and Formatters

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/matplotlib-python.

AutoLocator

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/matplotlib-python.

MultipleLocator

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/matplotlib-python.

MaxNLocator

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/matplotlib-python.

© 2024 Bhola Prasad AdaCode.io

FixedLocator

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/matplotlib-python.

LogLocator

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/matplotlib-python.

FormatStrFormatter

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/matplotlib-python.

FuncFormatter

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/matplotlib-python.

ScalarFormatter

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/matplotlib-python.

LogFormatter

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/matplotlib-python.

© 2024 Bhola Prasad AdaCode.io

PercentFormatter

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/matplotlib-python.

Styling Ticks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/matplotlib-python.

Basic Tick Stylying

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/matplotlib-python.

Advanced Tick Styling

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/matplotlib-python.

© 2024 Bhola Prasad AdaCode.io

Chapter 10. Legends

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/matplotlib-python.

Create a Legend in Matplotlib

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/matplotlib-python.

Location of the Legend

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/matplotlib-python.

Multiple Legend on the Same Axes

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/matplotlib-python.

Exercise 10.1

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/matplotlib-python.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/matplotlib-python.

Solution

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/matplotlib-python.

Exercise 10.1

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/matplotlib-python.

© 2024 Bhola Prasad AdaCode.io