Data Visualization with Python

Exploring Matplotlib, Seaborn, and Bokeh for Interactive Visualizations

Dr. Pooja

Data Visualization with Python

Exploring Matplotlib, Seaborn, and Bokeh for Interactive Visualizations

Dr. Pooja

bpb

Data Visualization with Python

Exploring Matplotlib, Seaborn, and Bokeh for Interactive Visualizations

Dr. Pooja

www.bpbonline.com

First published: 2023

Published by BPB Online
WeWork
119 Marylebone Road
London NW1 5PU

UK | UAE | INDIA | SINGAPORE

ISBN 978-93-55515-384

www.bpbonline.com

Dedicated to

This book is dedicated to my beloved parents, who have been my guiding lights. Your unconditional love, unwavering support, and boundless encouragement have shaped me into who I am today.

I would also like to dedicate this book to my wonderful daughters. You are the inspiration behind my pursuit of knowledge and the driving force behind my aspirations. Your curiosity, enthusiasm, and zest for learning constantly remind me of the joy that lies in discovery. May this book serve as a tribute to the love and joy you bring into my life.

I offer my deepest gratitude and love to my parents and my daughters. Your presence and influence have enriched my journey, and I dedicate this book to you with immense appreciation and affection.

About the Author

Dr. Pooja is an accomplished individual with almost two decades of experience in imparting education and making significant contributions in the field of Computer Science and Engineering. With a strong background in education, she has dedicated her career to sharing knowledge and inspiring others through her expertise. She has delivered numerous hands-on training sessions to students/learners/industry personnel on Artificial Intelligence and Machine learning using Python. Furthermore, she has worked with NITTTR, Chandigarh, towards the co-creation and delivery of courses for the national online education portal “Swayam portal” (NPTEL) on “Smart grid analytics” implementing a “Machine Learning module.”

Throughout her professional journey, she has published over 90 publications, which include national and international journal/conference papers and book chapters, including IEEE, Springer conferences, and Scopus Indexed Journals. The extensive body of work reflects their commitment to advancing knowledge and making valuable contributions to the field of Artificial Intelligence/Machine Learning and Deep Learning. Her expertise and insights have likely been sought after by peers, students, and fellow researchers alike.

About the Reviewers

- ❖ **Arun Kumar** is a Lead Data Scientist at SkyBridge Infotech with a Bachelor of Engineering degree. With a strong background in Fintech, Automotive, and Banking, he brings diverse industry experience to his role. He is currently dedicated to the fields of Artificial Intelligence (AI), Machine Learning (ML), and Deep Learning (DL).

  As an accomplished professional, Arun's expertise lies in leveraging data-driven insights to drive business growth and innovation. With a focus on AI, ML, and DL, he possesses in-depth knowledge of advanced analytics techniques and their practical applications. Arun's proficiency in these areas enables him to analyze complex datasets, develop predictive models, and extract valuable insights.

  Arun's experience working in different domains equips him with a comprehensive understanding of industry-specific challenges and opportunities. This knowledge allows him to evaluate technical content effectively, making him an ideal candidate for a technical reviewer role. His attention to detail, analytical mindset, and commitment to accuracy ensure the quality of technical materials.

  Beyond his professional endeavors, he likes to play volleyball and marathon running. You can reach Arun Kumar at [**https://www.linkedin.com/in/arunkumar040891**].

- ❖ **Sakil Ansari** is a versatile data scientist specializing in Artificial Intelligence (AI). He holds a bachelor's degree in Computer Science and Engineering from Jawaharlal Nehru Technological University Hyderabad, and a master's degree in Machine Learning from the same institution. Ansari has conducted research at esteemed institutions like the Indian Institute of Technology Madras (IITM) and the Indian Institute of Science (IISc) Bangalore. With twelve research papers published in international journals, he actively contributes to the scholarly community and has participated in numerous AI conferences. Ansari authored the book "Introduction to Natural Language Processing - A Practical Guide for Beginners" and has a passion for open-source initiatives, creating machine learning libraries for public use.

He is an experienced corporate trainer in data science and has a broad range of research interests, including natural language processing, speech processing, neural networks, and music information retrieval. With expertise in analytics problem-solving across various industries, such as Manufacturing, Retail, Finance, Entertainment, Sports, Automotive, and Healthcare, Ansari is adept at designing customized solutions based on advanced analytics and user requirements. He remains committed to leveraging data to create a better future for all.

Acknowledgement

There are several individuals and organizations I would like to express my gratitude for their unwavering support throughout the process of writing this book. Their encouragement and assistance have been invaluable, and I am truly thankful for their contributions.

First and foremost, I would like to extend my deepest appreciation to my family. Their continuous support and belief in my writing endeavors have been instrumental in completing this book. Without their unwavering encouragement, I would not have been able to accomplish this feat.

I am also grateful to the course and the companies that provided support during my learning journey of Data Visualization. I would also like to thank MTTF for providing me with the opportunity to deliver Data Visualization sessions, which compelled me to excel more in this area and initiate writing a book on it.

To all those who provided hidden support and assistance, I offer my heartfelt thanks. Your behind-the-scenes contributions have not gone unnoticed, and I am truly grateful for your help.

I would like to express my sincere acknowledgment to Arun Kumar and Sakil Ansari for their kind technical scrutiny of this book. Their expertise and valuable insights have greatly enhanced the quality of the content.

Furthermore, I would like to extend my gratitude to the team at BPB Publications. Their unwavering support and flexibility in allowing me ample time to complete the book and publish it is truly appreciated. Their understanding and cooperation have been instrumental in making this book a reality.

I extend my heartfelt thanks to all the individuals and organizations mentioned above for their continuous support, guidance, and belief in this project. It is through their contributions that this book has come to fruition.

Preface

In today's data-driven world, understanding and interpreting data is becoming increasingly crucial. Whether you are a researcher, a business professional, or simply someone interested in uncovering insights, the power of data visualization cannot be overstated. This book serves as a comprehensive guide to the art and science of data visualization, equipping you with the knowledge and tools to effectively communicate complex information through visually compelling representations.

The chapters in this book cover various aspects of data visualization, providing a structured approach to learning and applying these techniques. Here is a brief overview of what each chapter explores:

Chapter 1: Understanding Data- In this chapter, we lay the foundation by exploring the fundamentals of data. We discuss data types, sources, and formats and introduce key concepts such as variables, observations, and data structures. Understanding data is essential for effective visualization, as it allows us to identify relevant information and prepare it for visualization.

Chapter 2: Data Visualization - Importance- Building upon the understanding of data, we delve into the reasons why data visualization plays a vital role in understanding and communicating data, highlighting its ability to reveal patterns, trends, and correlations that might otherwise remain hidden.

Chapter 3: Data Visualization Use Cases- We examine a range of practical use cases for data visualization. From business analytics to scientific research, data visualization finds application in diverse fields, enabling us to make informed decisions, identify outliers, and communicate findings effectively.

Chapter 4: Data Visualization Tools and Techniques- To help you embark on your data visualization journey, we provide an overview of essential tools and techniques. We explore popular data visualization tools like various types of charts and plots, offering insights into their features and capabilities.

Chapter 5: Data Visualization with Matplotlib- In this chapter, we focus specifically on Matplotlib, a powerful library for creating static, animated, and interactive visualizations in Python. We cover its various plotting functions and customization options to create visually appealing visualizations.

Chapter 6: Data Visualization with Seaborn- In this chapter, we explore Seaborn, a high-level data visualization library built on top of Matplotlib. We discuss its specialized functions for statistical plotting and how it simplifies the creation of aesthetically pleasing visualizations.

Chapter 7: Data Visualization with Bokeh- In this chapter, we dive into Bokeh, a Python library for interactive data visualization. We explore its interactive plotting capabilities, including interactivity with web browsers and creating dynamic, interactive dashboards.

Chapter 8: Exploratory Data Analysis- We delve into the realm of exploratory data analysis, a crucial step in understanding and gaining insights from raw data. Through interactive visualizations and statistical techniques, you will learn how to uncover patterns, identify outliers, and much more that drive further analysis.

Throughout this book, we strive to strike a balance between theoretical concepts and practical examples. We provide clear explanations, step-by-step tutorials, and real-world case studies to help you grasp the principles and apply them to your own data visualization projects.

Whether you are a beginner or an experienced practitioner, this book aims to expand your understanding and proficiency in the art of data visualization. We invite you to embark on this journey with us as we explore the captivating world of visualizing data and unlocking its transformative power.

Happy visualizing!

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

https://discord.bpbonline.com

Table of Contents

CHAPTER 1
Understanding Data

"Data really powers everything that we do."

— Jeff Weiner

In this chapter, you will get familiar with "Data." The chapter will provide an understanding of what Data is and what are the ways to collect it. The chapter also presents different ways of categorizing data with suitable examples of each type. More importantly, we will discuss how data can be analyzed and how it can be used properly.

The chapter presents an introduction to data and various categories into which it can be segregated. For a better understanding of it, the attributes of data are also elaborated. For any application, data cannot be used directly. Data preprocessing is quite an essential stage. The chapter provides necessary techniques and ways to preprocess the data as well.

Structure

In this chapter, we will discuss the following topics:

- What is Data?
    - Categories of Data
    - Data attributes
    - Purpose of Data
- Data Collection
- Data Processing

Objectives

The foremost objective of this chapter is to make you familiar with data so that you have a basic understanding of what exactly you are working on in the upcoming chapter while visualizing the data. After studying this unit, you should be able to understand and analyze the data as various categories and data attributes, and further, you will be able to collect and prepare it well for further model building.

What is Data?

In our daily routines, we come across various important instances that can be termed as data. Let us assume you are on a stroll, and you meet someone. The conversation may start like this…*Hi! I am David. What is your name?* See here, "*David*" is important information and is a fact that the other person is referred to as David. Here, "David" is data, and if you are supposed to create a program/application which can fetch the names of the user, "David" will be considered as a string type data.

Let us take another example. You went to buy bread. The conversation might be:

A: "Do you have bread?

B: Yes, how much do you want?

A: Please pack 2. How much do I have to pay?

B: It will cost you 100.50 INR.

See here we have a piece of essential information viz the number of packs required and the cost to be paid. The quantity is "2," and the amount is "100.50". Again, if we want our machine/system to calculate the cost, this type of data will be considered as integer and float type, respectively.

Many times we fill up some kind of a form for, say, customer support by providing some information. Or you go for your medical check-up, and at the reception, you would be required to fill in the basic information about yourself. The form may consist of yes-no questions where you will 'Tick' mark the correct option. Actually, the data is being collected through this form. Here, the data may be in the form of a symbol.

The weather forecast on your mobile screen is another example of data; this data is processed data coming from the meteorological department after analyzing the historical data.

Our routines are full of data. The data captured by your smartwatch on your body parameters, the messages you type, the photographs you upload on social media, and so on. So, we can conclude that data is a collection of numbers, floating points, strings, and symbols that represents some value or situation. Data is information that can be used and translated into a form that is effective and effcient for processing. Data is facts and statistics collected together for reference or analysis. We rely on data mostly to make decisions or analyze a situation.

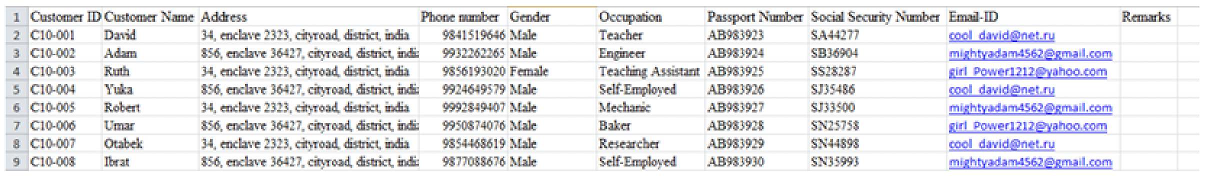
Note: Data is information that can be used and translated into a form that is effective and efficient for processing.

Categories of data

Two broad categories in which data can be classified on the basis of their format are:

Structured Data

Structured data is data that is ordered and may be recorded in a certain way. Structured data is presented in an organized manner. Structured data is often kept in a computer in a tabular (rows and columns) format, with each column representing distinct data for a specific parameter known as an attribute/ characteristic/variable and each row representing data of observation for multiple attributes. The data in the excel sheets, data pulled from finance teams, sales data, and CRM data are all structured data. Being in a pre-defined format, it is always easy to search for an element/data item from the whole dataset. Please refer to the following figure:



1	Customer ID	Customer Name	Address	Phone number	Gender	Occupation	Passport Number	Social Security Number	Email-ID	Remarks
2	C10-001	David	34, enclave 2323, cityroad, district, india	9841519646	Male	Teacher	AB983923	SA44277	cool_david@net.ru	
3	C10-002	Adam	856, enclave 36427, cityroad, district, indi	9932262265	Male	Engineer	AB983924	SB36904	mightyadam4562@gmail.com	
4	C10-003	Ruth	34, enclave 2323, cityroad, district, india	9856193020	Female	Teaching Assistant	AB983925	SS28287	girl_Power1212@yahoo.com	
5	C10-004	Yuka	856, enclave 36427, cityroad, district, indi	9924649579	Male	Self-Employed	AB983926	SJ35486	cool_david@net.ru	
6	C10-005	Robert	34, enclave 2323, cityroad, district, india	9992849407	Male	Mechanic	AB983927	SJ33500	mightyadam4562@gmail.com	
7	C10-006	Umar	856, enclave 36427, cityroad, district, indi	9950874076	Male	Baker	AB983928	SN25758	girl_Power1212@yahoo.com	
8	C10-007	Otabek	34, enclave 2323, cityroad, district, india	9854468619	Male	Researcher	AB983929	SN44898	cool_david@net.ru	
9	C10-008	Ibrat	856, enclave 36427, cityroad, district, indi	9877088676	Male	Self-Employed	AB983930	SN35993	mightyadam4562@gmail.com	

Figure 1.1: *Structured data (the data is generated synthetically)*

Unstructured data

Unstructured data is information that lacks a predefined data model or is not arranged in a predefined way. Unstructured data is often text-heavy, although it may also include data such as dates, figures, and facts. For instance, consider the data available on a webpage. It consists mostly of text; however, multimedia content is also available, viz. images, audio, video, and so on.

Consider the social media content; it includes text, emojis, special characters, GIFs, and so on. Social media content also falls under unstructured data.

Considering the data captured in the healthcare sector, the content written by the physician in the slip recommendation/notes is unstructured in nature. The data captured in the form of imaging is also unstructured in nature.

Thus, we can say that data, which are not in the traditional row and column structure, are unstructured in nature. It is always tedious to work on unstructured data due to the lack of any predefined format or schema. Further, unstructured data consumes more storage.

Note: Structured data is presented in an organized predefined manner like row-column format. Unstructured data lacks predefined format and is not organized.

Data can also be represented in the following categories:-

- Qualitative and Quantitative Data.
- Continuous and Discrete Data.
- Primary and Secondary Data.

Qualitative and Quantitative Data

Qualitative data are measurements of 'types,' and they can be represented by a name, symbol, or number code. Data concerning categorical variables constitute qualitative data (for example, what type). Qualitative data results from information, which has been classified.

Quantitative data are numerical variables' values (for example, how many; how much; or how often). Quantitative data occurs when the measurement of data is possible on a scale Quantitative data can also be discrete or continuous data varying on the elements being used and observed.

Refer to *Table 1.1* for better understanding. Here ‘age’ and ‘total marks’ are numeric variables containing quantitative data values (numeric values), while ‘Fail/Pass status’ and ‘Gender’ are categorical variables holding qualitative values.

Data Instance	**Quantitative data**		**Qualitative data**	
Student	Age	Total Marks obtained	Fail/Pass status	Gender
David	25	82	Pass	Male
Ruth	22	80	Pass	Female
Adam	23	40	Fail	Male
Luka	25	42	Pass	Male

Table 1.1 *: Qualitative - Quantitative data*

Some numeric variable examples:

- *"How many siblings do you have?"*
- *"How much do you earn?"*
- *"How many days do you work?"*
- *“How much is the area of your house?”*
- *“How often do you visit your aunt?”*
- *“How many employees are above 40?”*

In the *Table 1.2* below students have been categorized according to the age group bracket they fall in. Students falling or belonging to the same age group are grouped or huddled up together. These groupings are based on the age numbers of students, meaning the data is Numerical and thus referred to as Quantitative data.

Age	No. of Students
18 years and under	6
19-21 years	85
22-25 years	115
26-30 years	100
30 years and above	74

Table 1.2: *Number of students in an age group*

The *Table 1.3* shows the data of the different specific times that people tend and usually wake up. What is being observed or taken under consideration here is the time that these people usually wake up.

Wake up Time	No. of people
5AM-6AM	35
7AM-9AM	50
10AM-1130AM	60
12PM-130PM	20
2PM-3PM	15

***Table 1.3** : Number of persons as per wake-up time*

Some categorical variable examples-

- *“Are you a student?”*
- *"In which country were you born?"*
- *“What is the occupation of your father?”*
- *“Will they play today?” (Yes/No form)*
- *“Which category does this flower belong to?”*
- *“Is it a dog or a cat?”*

Flower features **sepal length, sepal width, petal length, petal width (cm)**	**Category**
5.1, 3.5, 1.4, 0.2	Iris Setosa
7.0, 3.2, 4.7, 1.4	Iris Versicolour
6.5, 3.2, 5.1, 2.0	Iris Virginica

Table 1.4: *Categories of flowers based on their characteristics (Iris dataset)*

Note: Quantitative data is the value of a numeric variable. Qualitative data is the value of categorical variable

Continuous and discrete data

Continuous data is data that can take any value. It appears as a sequence of values. Height, weight, temperature and length are all examples of continuous data. It represents the information that could be meaningfully divided into its finer levels. It can be measured on a scale or continuum and can have almost any numeric value. This type of data is referred to as Continuous data.

Discrete data is a type of data that includes whole, concrete numbers or categorical variables with specific and fixed data values determined by counting. Discrete data on the other hand may be shown in gaps in scale, with no real values to be found.

For example, the number of students in a class is an example of discrete data since we can count whole individuals but can’t count like 2.5, 3.75, kids. In simple words, discrete data can take only certain values and the data variables cannot be divided into smaller parts. It has a limited number of possible values for example days of the month.

Name	Gender	Age	Survive
Allen, Miss. Elisabeth Walton	female	29	1
Allison, Master. Hudson Trevor	male	0.9167	1
Allison, Miss. Helen Loraine	female	2	0
Allison, Mr. Hudson Joshua Creighton	male	30	0
Allison, Mrs. Hudson J C (Bessie Waldo Daniels)	female	25	0
Anderson, Mr. Harry	male	48	1
Andrews, Miss. Kornelia Theodosia	female	63	1

Table 1.5: *Titanic survival data instances*

In the *Table 1.5* above instances of ‘Titanic Survival’ dataset are pulled to elaborate on continuous and discrete data. Here the categorical data variable ‘Gender’ is discrete in nature because it has only two values (Countable) viz. male and female. Similarly the data values of ‘Survived’ are also discrete in nature (0 or 1) while the data in ‘age’ is continuous in nature and we can also subdivide the values into categories like adult, infant, senior citizens and so on.

Characteristics of Continuous data

- Continuous elements are not counted, but are measurable.
- Continuous data values can be categorized and further subdivided into smaller pieces with additional meaning.
- It is usually graphically displayed by histograms.

- Continuous data is first and foremost present and gives a better sense of variation.

Some continuous data examples:

- The weight of people.
- The height of footballers.
- The waking up time of people.
- Speed of cars.
- Weight of trucks.
- The height of children.
- House prices.
- Temperature

Characteristics of Discrete data

- Discrete data can be counted and is usually counted in whole numbers.
- Discrete data cannot be measured at all.
- Discrete data values and elements cannot be subdivided into smaller pieces.
- It is usually graphically displayed by a Bar Graph.
- Binary attributes are a special case of discrete attributes where the count of discrete values is always two (0/1, False/True).
- Discrete data may also be ordinal or nominal data.
- It may be ordinal data meaning when the values fit into one of many categories and there is an order or rank to the values.
- It may be nominal data meaning when the values fit into one or many categories, especially where there is not any order between the values.

Some discrete data examples:

- The number of students admitted to a College.
- The number of people attending a Seminar.

- The number of Football teams participating in a Tournament.
- The number of cars in a Car Dealership.
- The number of staff working in a company.
- The number of patients admitted to a hospital.
- The number of teachers working in a school.

Note: Continuous data is data that can take any value. It appears as a sequence of values. Discrete data is countable/fixed values. It can take only certain values

Primary and secondary data

Primary data is data that is collected by people or on behalf of the person who is going to make use of the data. We can say, it is the data collected for the first time. *For example*, if you contact children's parents and ask them about the educational qualifications of their children concerning their performance this also grants or gives them Primary data. Whereas Secondary data is data used by a person or by people other than the people whom it was intended for. We can say secondary data is the data that have already been collected by some other person

Characteristics of Primary Data

- Usually collected for the first time.
- Original and more reliable than most types of data.
- It is first-hand information gathered and collected usually by an Investigator or Surveyor.

Characteristics of Secondary Data

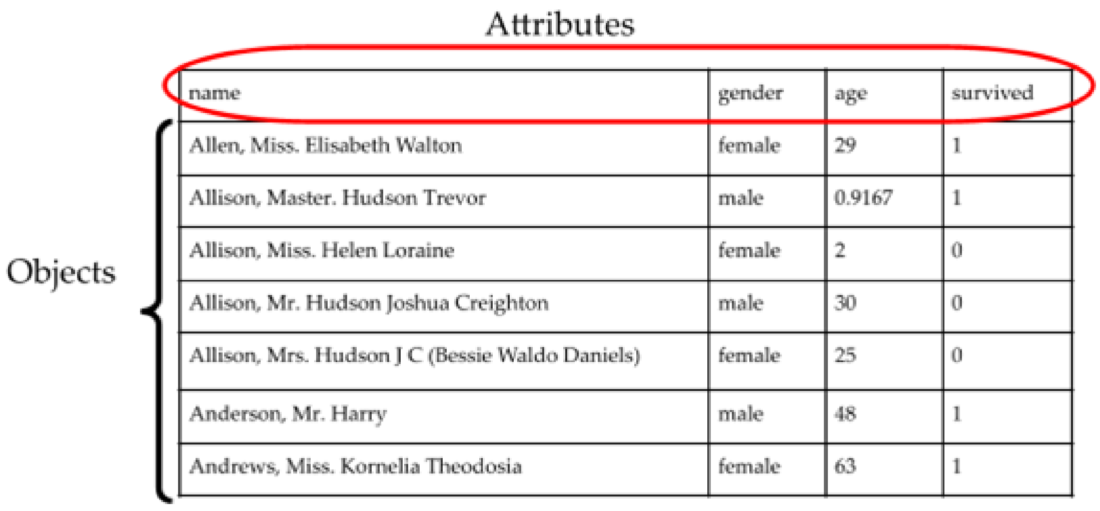
- It is second-hand information collected, gathered and reported.
- Usually obtained from already published or unpublished sources.
- Useful tips for using Secondary data.
- How should the data be collected and processed?

- Accuracy of the data.
- How far the data can and should be summarized.
- Comparing the data with other tabulations.
- How to interpret the data?

Note: Primary data is data that is collected for the first time. Secondary data is the data that have already been collected by some other person

Data attributes

Data is a collection of data objects and their attributes. In a particular dataset, we get the features and instances with feature values. These features are actually the attributes of the data while available instances are data objects. These instances possess values for all or some attributes.


Attributes
Objects

name	gender	age	survived
Allen, Miss. Elisabeth Walton	female	29	1
Allison, Master. Hudson Trevor	male	0.9167	1
Allison, Miss. Helen Loraine	female	2	0
Allison, Mr. Hudson Joshua Creighton	male	30	0
Allison, Mrs. Hudson J C (Bessie Waldo Daniels)	female	25	0
Anderson, Mr. Harry	male	48	1
Andrews, Miss. Kornelia Theodosia	female	63	1

Figure 1.2: *Data Attributes and Objects*

We can understand that an attribute is a property or characteristic of a data object. For instance, if we have to create a dataset of persons, eye color, hair color, height, weight, face shape can be considered as attributes. An attribute can also be referred to as variable, field, characteristic, dimension, or feature. and attribute values are numbers, symbols, values assigned to an attribute for a particular object. An object can be referred to as a point, instance, record, entry, and sample.

Note: Data Attribute is the property or characteristic of a data / data object.

Majorly the attributes can have the following type, [**NOIR**]:-

Qualitative (Categorical) data

- Nominal (N)
- Ordinal (O)

Quantitative (Numeric) data

- Interval (I)
- Ratio (R)

Nominal

A nominal attribute is used to name, label, or categorize certain measurements or features. It accepts qualitative values representing several categories, yet these categories are not intrinsically ordered. Although numbers can be used to code nominal variables, the order is arbitrary and arithmetic operations cannot be performed on the numbers.

A nominal variable is the simplest of all measurement variables and is one of two types of categorical variables. A person's phone number, national identification number, postal code, and other personal information are examples. A nominal value can be classified into two or more groups. Gender, for example, is a nominal variable that may accept the values as male/female or M/F. A nominal variable is qualitative, which implies that numbers are solely employed to categorize or identify items in this context. The number on the back of a player's shirt, for example, is used to indicate the position he or she is playing. They can also take numerical values. These quantitative values, however, lack numeric features. That is, they cannot be used for mathematical operations. They only possess the property of distinctness (equal or not equal).

Ordinal

The ordinal attribute value provides suffcient information to order the objects. They are built upon nominal scales by assigning numbers to objects to reflect a rank or ordering on an attribute. Also, there is no standard ordering in the ordinal variable scale. Ordinal data is information that is ranked or ordered. Examples include ranking one's favorite movies or arranging people in order of shortest to tallest or first, second, and third place in a competition. They possess the property of distinctness and order (<, >).

Interval

The interval attribute is used to define values measured along a scale, with each point placed at an equal distance from one another. Unlike ordinal variables that take values with no standardized scale, every point in the interval scale is equidistant. On Intervals, addition and subtraction operations can be performed.

Ratio

It measures variables on a continuous scale, with an equal distance between adjacent values. It is an extension of the interval variable and is also the peak of the measurement variable types. The only difference between the ratio variable and interval variable is that the ratio variable already has a zero value.

Due to the absolute point characteristics of a ratio variable, it does not have a negative number like an interval variable. On ratios, along with addition and subtraction, multiplication and division operations can also be performed. They possess the properties of distinctness, order, differences are meaningful and ratios are meaningful. (equal, not equal, <, >, +, -, *, /)

Although both interval and ratio data may be classified, sorted, and have equal spacing between consecutive values, only ratio scales contain a true zero. Temperature in Celsius or Fahrenheit, for example, is measured on an interval scale since 0 is not the lowest attainable temperature. However temperature, when measured in Kelvin, is an example of ratio variables.

The Purpose of data

- Data helps to improve the quality of life of people and for people. The sole purpose why organizations rely upon and use data is to improve the quality of their products, services and most importantly the quality of their customers' lives and experiences.

- Data equips one with Knowledge, knowledge is a powerful tool as it can be used to steer one to make informed decisions and to know what caution or risks to take. Data also provides you with concrete evidence for you to have as a backup when making informed decisions.
- Utilizing data well gives you the edge and allows you to monitor the health of important systems in your firm. It also gives you the upper hand and foresight to deal with challenges when they arise or to come up with counter strategies to withstand the hurdles ahead.
- Data allows or gives an organization the insight and information to review and verify the strategies they have come up with and evaluate and come up with solutions as well as view the statistics outcomes and results that follow. Collecting and storing such data will give organizations foresight on future outcomes and strategies.
- Data gives organizations the edge in determining the roots of problems and effectively coming up with strategies that will result in solutions in handling or fixing the problems. It allows firms to visualize and evaluate deeper relationships between departments in the firm and between staff members as well.
- Data acts as a key component or as a key advocate for you as it allows you to back up your arguments with facts. Data utilization helps in providing or providing a strong argument varying from any particular situation that arises be it advocating for increased funds or making changes to the organization.
- To keep it plain and simple, data allows explaining your points and illustrations much better to the organization’s stakeholders instead of just playing the guessing game. It allows you to be confident and put your best foot forward knowing the points and data you have been accurately researching and holding a strong case for you.
- Data helps to increase effciency that allows scarce resources to be effectively directed where they are needed. Data also supports and helps organizations to determine which particular areas or departments need the utmost priority compared to the others, meaning resources will be deployed where they are needed.

- Data allows organizations to be able to recreate their areas of strength. Analyzing data will allow you to show you the areas of high performance in the company as well as high service areas and the high or hard-working performing staff or workers in the company.
- Analyzing data and proper storage of data allows the organization to effectively set up their goals and objectives in line with customers' needs to keep moving forward and accomplishing or meeting their goals in the end. It also helps organizations to set up their goals in line with the performance of the company and in the end celebrate the success. It allows organizations to also be more realistic with setting their benchmarks.
- For organizations to be able to pool funds or get Government grants or Shareholders' support in terms of funds they have to provide and present facts and proof that is driven by data. Data will be able to show the Shareholders or Investors how the company is performing and what amount to invest depending on the data given and shown.
- Most organizations pride themselves on having already been established and having the necessary resources and expertise to allow their staff and customers to begin and do their analysis. For example, HR offces already know and track information regarding their staff.

Data collection

In general, there are usually two sources of data and they are classified into; **Statistical** and **non-statistical**.

Statistical sources: this is data that is gathered and collected for offcial purposes, incorporating censuses and offcial administered surveys.

Non- statistical sources: this is a collection of gathered data for administrative purposes or the private sector.

The sources of data are further classified into two namely:

Internal sources

This is when the organization usually collects data and information from records and reports. For example, an annual report on the Profits and Loss of the business of that fiscal year.

External sources

This is when the organization collects and gathers information and data from outside sources. For example, if an airline like Air India wants to travel to a country like England it will first need to obtain information from the UK travel advisory board, this is what is called External sources.

Methods of Collecting Primary Data

- Direct personal investigation.
- Indirect oral investigation.
- Information through Correspondents.
- Telephonic Interview.
- Mailed Questionnaires.
- A questionnaire was filled out by enumerators.

***Figure 1.3**: Sources of data generation (**source:** https://www.seekpng.com/ipng/u2q8a9e6e6r5i1r5_our-competencies-multiple-data-sources/)*

Note: Enumerator is a person who helps or aids investigators with the collection and gathering of data.

Merits of direct personal investigation

- The data collected and gathered is reliable and accurate since its first hand in nature.
- Using this type of method means the questions can be adjusted varying to the level of the correspondent or other scenarios.
- Apart from the required information that is collected and gathered, additional information can also be added and used in investigations in the future.

Demerits of direct personal investigation

- It is not suitable for wide coverage or area.
- Using this type of method means the Investigator has to travel to different locations meeting different people to gather and collect information thus time- consuming.
- With various or large locations or different areas of coverage to be traveled this type of method is costly.
- At times the data collected and gathered may be based on personal biases depending on the investigator.

Note: Data collection: two sources of data and they are classified into; statistical and non-statistical.

Data processing

Data Processing cycle involves input and storage of data, its processing and generating output. Much of the raw data contained in databases or fetched from some data source is not preprocessed, incomplete, and noisy. For example, the data may contain fields that are obsolete or redundant, missing values, outliers. Data in a form not suitable for further use. In order to be useful for other purposes, the data needs to be processed. Further we are interested in data because it contains essential facts and information that may help us make decisions. However, drawing conclusions from enormous or massive amounts of data is impossible. Rather, data must be processed to produce outcomes, and those findings must be analyzed before we can draw conclusions or make judgments. Automated data processing may be found in circumstances such as online bill payment, complaint registration, ticket booking, and so on.

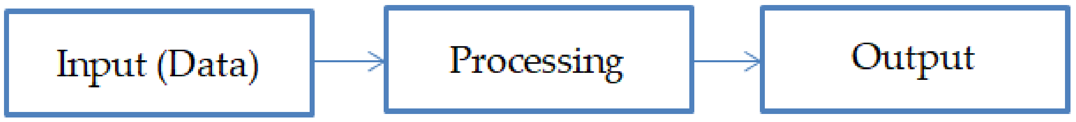


Figure 1.4*: Data processing cycle*

Data processing leads to output as required refer to figure ١.٤. However before the data is considered to be used as an input, it requires pre-processing so that the expected output is generated.

In the input phase data is collected, pre-processed and fed into the system for further processing. In preprocessing, we try to clean and transform the data. Data preprocessing is the most important of all steps, as the output is solely dependent on the data; if data is of good quality, the output will be accurate and precise.

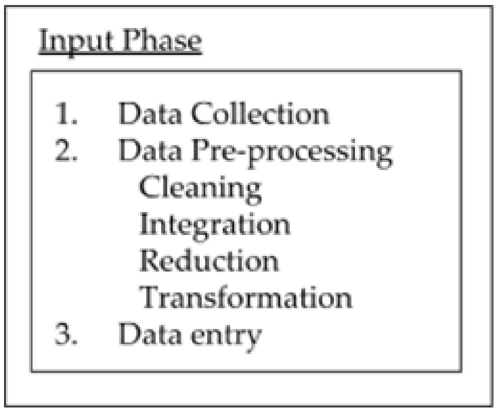


Figure 1.5: *Data input phase*

Depending on the source of data being processed (database, online databases, linked devices, and so on.) and the intended use of the output, this phase may differ slightly from process to process.

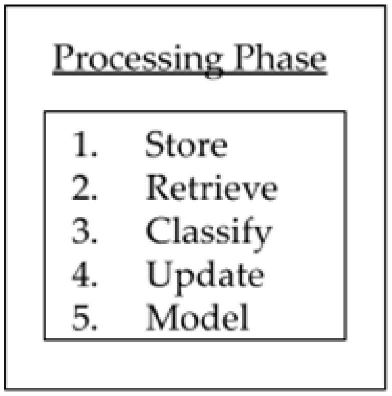


Figure 1.6: *Data processing phase*

Finally, the data is communicated and shown to the user in a readable format, such as reports, visualizations-graphs, documents, and so on. This output can be saved and processed further in the next data processing cycle.

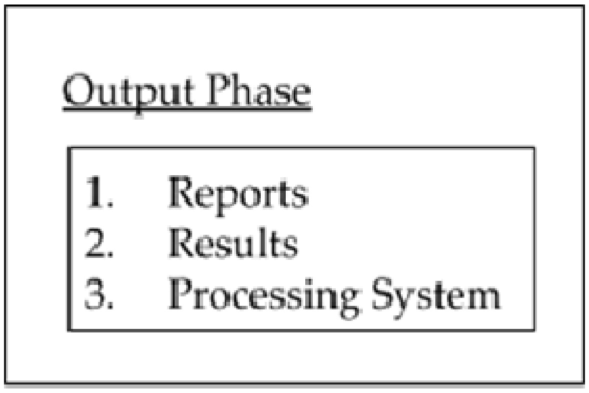


Figure 1.7*: Output phase*

Some real-life examples of data processing:

- Withdrawal of money from ATM
- Filling online examination form and getting the admit card as an output
- Self-driving cars taking data collected through sensors and taking decisions accordingly
- Recommendation of movies as per your historical choices on Netflix

Characteristics of data

- **Dimensionality (number of attributes)**: High dimensional data brings a number of challenges

- **Sparsity**: Only presence counts. Data collected may have some attributes with blanks or no values.
- **Resolution**: Patterns depend on the scale
- **Size**: Type of analysis may depend on the size of the data

Quality parameters of data

Data processing helps users examine the data quality. The following criteria can be used to assess the quality of data:

- **Accuracy**: to determine whether or not the data entered is correct.
- **Completeness**: to determine whether or not the data is available.
- **Consistency**: to determine if the same data is retained in all locations that match or do not match.
- **Timeliness**: data should be updated on a regular basis.
- **Believability**: data must be credible.
- **Data interpretability**: data's ability to be understood.

Conclusion

In this chapter, we tried to elaborate on basic concepts for understanding data. In the next chapter, you will learn about the importance of data visualization for better decision-making. It will highlight the importance of data visualization. The process of creating graphical representations of information using various methods such as pie charts, tables, scatter plots, graphs, geospatial maps, infographics, and so on is known as data visualization.

Note: Data Processing cycle involves input and storage of data, its processing and generating output.

Points to remember

- Data is information that can be used and translated into a form that is effective and effcient for processing.

- Data can be structured or unstructured.
- Structured data is presented in an organized predefined manner, like row-column format.
- Unstructured data lacks predefined format and is not organized.
- Data is usually categorized as Qualitative and Quantitative Data, Continuous and Discrete Data, Primary and Secondary Data.
- Quantitative data is the value of a numeric variable.
- Qualitative data is the value of the categorical variable
- Continuous data is data that can take any value. It appears as a sequence of values.
- Discrete data is countable/fixed values. It can take only certain values.
- Primary data is data that is collected for the first time.
- Secondary data is the data that has already been collected by some other person.
- Data Attribute is the property or characteristic of a data/data object.
- A nominal attribute is used to name, label, or categorize certain measurements or features.
- Ordinal data is information that is ranked or ordered.
- The interval attribute is used to define values measured along a scale, with each point placed at an equal distance from one another.
- Ratio measures variables on a continuous scale, with an equal distance between adjacent values.
- The Data Processing cycle involves the input and storage of data, its processing, and generating output.

Questions

1. What do you understand by data?
2. Differentiate between qualitative and quantitative data.
3. Differentiate between continuous and discrete data

4. Elaborate why data is important?
5. What are primary and secondary data?
6. Identify the type of data for the following scenarios:
    a. Age of students in a class
    b. Food items procured and sold by a shopkeeper
    c. Tweets on Twitter by your organization
    d. The temperature recorded for two weeks
    e. Attendance recorded for the students of a class
7. Explain the data processing steps
8. Write a note on NOIR (data attributes).
9. Differentiate range and interval.
10. What do you understand by data type?
11. Explain non- statistical sources of data collection.
12. What are the characteristics of data?
13. Debrief the quality parameters of data.
14. Write a note on data processing cycle.
15. Give some examples of data processing systems.

CHAPTER 2
Data Visualization – Importance

“This is my favorite part about analytics: Taking boring flat data and bringing it to life through visualization."

—John Tukey

Every set of data has important information to convey. Whether or not you can see it depends on how successfully you visualize it.

This chapter highlights the importance of data visualization. Creating graphical representations of information using various methods such as pie charts, tables, scatter plots, graphs, geospatial, maps, infographics, and so on is known as data visualization. The chapter commences with a brief description of data visualization. Combining data with the appropriate visual formats allows you to present information in a much more digestible manner.

Data visualization enables improved assimilation of critical business information, faster access to valuable business insights, a better understanding of business operations, and accurate customer sentiment analysis. It also enables data storytelling, pattern recognition, and more efficient decision-making in real time. And, with data volumes increasing at the speed of light, data visualization is one of the best ways to make the most of all your business data while staying ahead in a competitive market. The next section presents the importance of data visualization in various domains. It also includes the benefits of data visualization. The end of the chapter presents the latest trends in data visualization.

Structure

In this chapter, we will discuss the following topics:

- Introduction
- Data visualization
- Significance of visualization
- Benefits of data visualization
- Data visualization trends

Objectives

This chapter's foremost objective is to familiarize you with the concept of data visualization and its importance. After studying this unit, you should be able to understand and analyze the importance of data visualization and its use in various domains.

Introduction

One of the processes involved in data science is data visualization, which implies that after data has been gathered, processed, and analyzed, it must be displayed to conclude. A component of the larger field of data presentation architecture, which tries to locate, modify, prepare, and convey data in the most effective manner possible, is **data visualization**.

We need a mechanism to visualize that data to analyze it, because the business sector now collects so much information through data analysis. By providing it with a visual context via maps or graphs, data visualization helps us understand what the information means. As a result, it is simpler to spot trends, patterns, and outliers in enormous data sets since the data are easier for the human mind to understand.

Data visualization is, therefore, the graphical depiction of data and information in a pictorial or graphical format. The practice of translating information into a visual context is used to make data easier for the human brain to absorb and draw conclusions from. Data visualization's major objective is to make it simpler to spot patterns, tendencies, and oddities in big data sets. Developing decisions based on data requires the analysis of vast amounts of information and using data visualization tools and technology.

The idea of utilizing visuals to comprehend data has existed for a long time. Businesses required a way to acquire insight efficiently and simply into their data as they gathered enormous quantities of data in the formative days of the big data revolution. Tools for visualization fit in naturally.

Charts, tables, graphs, maps, and dashboards displays are examples of basic kinds of data visualization.

Data visualization

Data delivery is made more effective through data visualization. Data visualization, which is one of the crucial processes in the business intelligence process, takes the raw data, processes it, and then presents the data so that inferences may be drawn. Data experts are developing machine learning techniques in cognitive computing. This is being done to better combine crucial data into representations that are simpler to comprehend and interpret.

Data visualization, in particular, uses visual data to convey information in a way that is accessible, quick, and efficient. This method can assist businesses in determining what must be improved, what influences customer contentment and discontent, with what to do with particular items (where they should go and to whom they should be sold). Investors, entrepreneurs, and decision-makers may anticipate sales, profits, and growth prospects more accurately with visualized data.

With the seemingly infinite streams of data readily available to today's businesses across industries, the challenge lies in data interpretation, which is the most valuable insight into the individual organization as well as its aims, goals, and long-term objectives.

The issue lies in data quantitative reasoning, which is the most useful insight into the specific organization as well as its ambitions, goals, and long-term aspirations. With the unlimited data streams easily obtainable to today's organizations throughout all industries, this is a difficult task.

Making meaning of data starts with visualization. Data analysts employ a variety of techniques for data visualization, including charts, diagrams, maps, and so on., to translate and portray complicated data and relationships. The only way to make information and communication technology understandable is frequently by selecting the appropriate methodology and setting it up. Conversely, using poorly chosen strategies will prevent you from utilizing the full capability of your data or even render it ineffective.

Factors that influence data visualization choices

Let us now look at the following factors that influence the data visualization choices:

Content

The strategies you use will depend on the type of data you are working with. Line charts, for instance, are frequently used to illustrate the fluctuations of time-series data. Scatter plots are frequently used to depict the relationship between two variables. Bar charts, in turn, are useful for the comparison of results.

Audience

It is critical to customize data visualization for the intended audience. For instance, users of mobile fitness apps can easily use simple representations when browsing over their accomplishments. And from the other side, you can and frequently need something more than simple charts if data insights are meant for researchers and knowledgeable decision-makers who frequently deal with data.

Context

You can take input and use various data visualization techniques depending on the situation. Use the many shades of one color on the chart to draw attention to a certain number, such as a big increase in profits, and accentuate the greatest value with the boldest color. Opposing hues, on the other hand, can be used to distinguish aspects.

Purpose

The method of data visualization implemented depends on its purpose. Visualizations are assembled into dynamic, configurable dashboards with a variety of visual analytics tools for creating complicated analyses (formatting, filtering, comparison, and so on). However, dashboards are not required to display a single or seldom data insight.

Dynamics

There are many distinct sorts of data, and each type changes at varying rates. For instance, whereas trend analysis and tracking data are always evolving, financial results can be analyzed on a monthly or annual basis. In data mining, you might take into account continuous portrayal (steaming) or static data visualization techniques based on the fluctuation rate.

Significance of visualization

Data visualization, in particular, uses visual input to convey information in a way that is accessible, quick, and efficient. This method can assist businesses in determining what has to be improved, what influences customer contentment and dissatisfaction, and what to do with particular items (where they should go and whom they should be sold).

Investors, businesspersons, and decision-makers may anticipate revenues and profits, and future growth more accurately with the use of visualized data.

When working with data visualization, we will discover that it is an efficient and simple way to explain many complex and difficult subjects, typically in a more encompassing way. Additionally, we can test out various scenarios utilizing a dynamic visual that allows us to make small alterations as needed.

More reasons why data visualization is so important include:

Data visualization identifies data trends

Finding data patterns is data visualization's most crucial accomplishment. Since all the data is presented to you in a graphic form rather than a table, it is much simpler to spot data trends. For instance, Tableau data displays each customer's total revenue in descending order. This picture makes it extremely clear that even if some consumers may have a large turnover, they are still losing money. Otherwise, it would be quite challenging to observe this from a table.

Data visualization tends to give the data a perspective

Data visualization puts data into perspective by illustrating its significance with other factors. It shows where certain data references stand in relation to the bigger picture.

Data visualization provides the correct context for the data

Data visualization makes it exceedingly difficult to comprehend the relevance of the data. Analyzing statistics in a table without understanding the context of the data makes it very difficult to understand. The quantity of sales in each part of a country is shown using a Treemap as an illustration of data visualization using Tableau. Since the rectangle for one area is the largest, it is fairly clear from this data visualization that one area has the most amount of sales overall. However, without data visualization, this information is difficult to interpret when taken out of context.

Time is saved by data visualization

Instead, of just viewing a chart, data visualization makes it easier to glean some knowledge from big data. On Tableau, for instance, it is quite simple to pinpoint the parameter that has experienced a net loss as opposed to a profit. This is due to the heat map's red coloring of all the cells that have lost data, which makes it clear that the variables have also lost data. In a typical table, you would have to verify each field to see if it has a lower return to calculate a loss. In these circumstances, data visualization saves a ton of time.

A data story is told through data visualization

A data story can also be told to audiences through data visualization. The visualization can convey a tale and guide the audience to an obvious conclusion while presenting factual information in an understandable format. Like any other form of a story, this one about data should have a strong opening, a straightforward storyline, and a conclusion that it builds toward. For instance, if a data scientist is required to create a data visualization for company leaders that highlights the profits of different products, the data narrative can begin with the income and expenditure of different items before moving on to suggestions for how to address the losses.

Benefits of data visualization

Interactive visualizations of data can have a favorable impact on an organization's decision-making process. Because businesses can now comprehend data in graphical or pictorial forms, they can now spot trends more quickly.

Some more specific ways that data visualization can benefit an organization includes:

Fast comprehension of information

We can easily and quickly understand the information we see thanks to the images. We will be able to observe these vast amounts of data clearly and coherently when we use a graphical depiction of all that data about our business rather than going through charts and spreadsheets.

When one approaches it in this way, it is much simpler for a firm to address issues or provide timely responses to some of its main questions so that problems may be resolved without difficulty or without needing to worry about further harm.

Correlations in relationships

Finding correlations between independent variable relationships is difficult without data visualization. We can improve our business decisions if we can understand the meaning of those independent factors.

With the help of visuals, it is much simpler for a company to understand some of the various characteristics that are present and how they are closely related to one another.

It will be much simpler for our company to concentrate on the sectors that are most likely to have an impact on some of our top crucial objectives when we utilize these visualizations to help locate and understand these linkages.

Trends over time

Although it would seem like an obvious application for data visualization, this is one of its most beneficial uses. Without the required knowledge from the past and the present, it is impossible to make forecasts. Trends over time show us where we have been and where we might go.

Some of the anomalies that are there, the types which are more likely to have an impact on the caliber of your product, customer turnover, or other elements that may alter your business, are much easier to identify. Additionally, it will enable you to deal with problems before they develop into much greater ones that you must have to deal with.

Frequency

Frequency is closely tied to patterns over time. We can better understand how potential new consumers might behave and respond to various sales and customer acquisition efforts by looking at the rate, or how frequently, they make purchases and when they do.

Examining the market

Data visualization uses information from many markets and continues providing insights into the audiences you should pay attention to and those you should ignore. By presenting this information in various charts and graphs, we can see the potential inside those markets more clearly.

The visualizations help us to understand our consumers' locations, the kinds of things they would be most satisfied with, how we might offer them better services, and more. Many businesses choose to use data visualization to better understand their clients and make sure that their work will set them apart from the competition.

Risk and reward

Because we must analyze complex spreadsheets and figures without data visualization, analyzing value and risk measurements requires skill. When data is visualized, we may then identify regions that could or might not need action.

Responding to the market

The ability to obtain information quickly and easily with data displayed clearly on a functional dashboard allows businesses to act and respond to findings swiftly and helps to avoid making mistakes.

Simple data sharing

Organizations offer a different correspondence arrangement through the depiction of the information. Sharing visual data will pull in and transmit information that is easier to understand and digest than sharing dense information.

Some of the other ways that data visualization can help us out includes:

- Determine the areas that will require the most improvement and attention.
- Supports the decision-making over where to position each of the products.

- It can make clear which elements are most likely to affect a customer's behavior.
- It can make it simpler to determine and estimate our sales quantities, including whether they will be larger or less at a given time.
- Considering that data is now easier to acquire and interpret, do away with the necessity for data scientists.
- A better capacity to hold the audience's attention while providing information that they can understand.

Data visualization trends

The future of corporate strategy is data visualization, and this field is evolving right in front of our eyes.

The destiny of data visualization will be very different from its earlier iterations due to the combined effects of technological and artistic advancements. Modern data visualizations will go more toward fluid, real-time models capable of effectively portraying the totality of intricate data sets. The days of meticulously filling in graph paper squares manually are long gone.

Video visualization

It is impressive that the data retention numbers for video are much more compelling than those for static images.

We are all aware of how powerfully moving a good film can be. The fact is even more straightforward: even with less engaging material, like that in a corporate exhibition, our minds are designed to focus on the video.

With video, we retain information better. We naturally favor video when given the choice since we are aware that it is simpler for our minds to interpret. Instead of images or text, consumers prefer to watch a video to understand more about a new product launch.

These key variables confirm the viability of video infographics and, yes, video commercials. Their use in customer service and corporate strategy.

Data beyond visuals

Data visualization involves more than simply images. There are other additional components inside. Since charts and graphs are always the most visible components of any platform, many individuals get fixated on them. Data must be able to inform and empower teams to learn substantial knowledge in addition to these conventional images; this is how data visualization trends will change in the ensuing years.

The field of fresh data visualization is being aesthetically improved by the use of contemporary tools and technology like google charts and simple maps. Detailed maps and infographics are frequently employed to "educate and motivate" people to create innovative data visualization dashboards without coding expertise.

Data democratization

The utilized data frequently has a reputation for being difficult to comprehend; to reveal its secrets, professionals and other technical personnel are needed.

There is no longer a need for this. Your data can be processed and unlocked automatically by cutting-edge, no-code data analysis solutions. This makes it malleable and simple to present in any type of data visualization that your staff may conjure up regardless of their degree of technological sophistication.

This "democratization" of data, when combined with a wise decision for data visualization, can make big data outcomes accessible to teams at all organizational levels while leaving your technical staff alone.

Data democratization is the process through which anybody, at any time, can utilize data to create decisions without restrictions on access or comprehension. The most effective data visualization tools significantly contribute to the democratization of data and analytics and the accessibility of data-driven insights for people across an organization.

Users are given the ability to design dynamic dashboards and customize them with a few clicks, as well as access and visualize data via drag and drop. This also improves prospects for earning money and produces data-driven offerings for making decisions.

Data visualization is more social

Utilizing social platforms to encourage better engagements is another recent development in data visualization. Since they have seen this trend, data scientists are working to ensure Data Visualization is social media compliant.

Because social media users have a short attention span, your data needs to be presentable and visually appealing. The "less is more" method is used in data visualization for social sharing. Examples of social media visualizations include YouTube videos, GIFs, and 3D animation.

Real-time visualization

When a problem first appears, you need to be aware of it. Your bottom line can be significantly affected by early identification in business strategy, client retention, and brand presence.

This kind of timely identification enables Zoom to respond to a client issue fast. enlarge tweet This demonstrates why current data visualization is the best type of data visualization. Real-time data visualization has replaced static charts and is here to stay.

The Dashboard for *MonkeyLearn Studio* serves as an example. As you can see, it tracks user comments as they come in and compiles them into visually pleasing, comprehensible graphs. Here is a link to the dashboard's public demo.

You would be alerted to the problem as soon as the MonkeyLearn dashboard was operating. Users who were having what your company would refer to as a "screen sharing issue" would report their errors to your customer service, and these evaluations would appear in the center of the dashboard, already labeled and sentimentally analyzed.

Data storytelling

Numbers tell a significant tale. They count on you to speak for them in a resonant and persuasive manner. It is not surprising that people enjoy reading stories. Additionally, the notion of inventive data visualization for storytelling is becoming more popular. It is insufficient to simply visualize the data. Visualizers must develop into storytellers who can recognize and convey great experiences to the forefront of technology.

Nowadays, data visualization is more like storytelling, where the content is tailored, simple to grasp, and has a clear beginning and end. This data visualization trend can put you ahead of your rivals if you can successfully create simple tales out of complex data.

Data journalism is becoming more popular

We observe a growth in the number of unbelievable data visualizations provided by prominent media organizations around the world due to the availability of data and also fantastic tools for visualization and analysis. In the world of news reporting, data visualization is becoming more popular.

A great number of media professionals will likely be persuaded by data journalism's effectiveness and scope. By combining and summarizing information in a way that offers the reader more time, it takes up less screen space.

Mobile and social data visualization

The way that individuals surf the internet is also evolving. In February 2021, 56% of web access came from smartphones, according to Broadband Search.

What a large market share! As a result, it is critical to realize that data visualization may be utilized to increase customer loyalty, decrease churn, and even draw in new consumers by leveraging the persuasive power of images.

Mobile-optimized data visualization is crucial whether current or future clients are learning about your services on social media, on your website, or in an online discussion forum. Customers may leave your site for a rival out of annoyance if your content is totally at odds with mobile devices. Or it could simply indicate that they are not seeing a crucial graphic that you paid a lot of money to have placed in a strategic location.

The outcome is the same in both cases—the dreaded customer churn. It is simple but crucial to be at peak of your mobile data visualization game.

Artificial intelligence and machine learning

Every industry is embracing artificial intelligence, which is also crucial in determining future trends in business intelligence. Data visualization powered by **Artificial Intelligence** (**AI**) enables firms to identify the data among the vast amount of data now available that they should be examining.

There is a huge amount of information accessible now that has important insights that help produce improved outcomes. **Natural language processing** (**NLP**) and machine learning can work together to reveal key insights from data, reducing the strain associated with data visualization for people.

Artificial intelligence and machine learning are the foundation of any excellent data visualization and will become more and more important as technology advances. This might easily be number one.

Data visualizations without the support of cutting-edge AI are destined to be unreliable and ineffective over time. Since this turns out that people are equally inefficient at digesting actual data as they are at accurately sorting it.

The ideal way to handle customer feedback is using impartial software that can sort it in real-time and as per your requirements. This minimizes the need to assign personnel to complete many hours of arduous, biased hand-tabulation.

Mobile-friendly data

Mobile-friendly data is one of the upcoming trends in data visualization because mobile consumer experiences currently rule the world. Data Visualization Mobile is becoming more and more popular, and many companies are tackling the challenge of converting desktop applications to mobile-friendly formats, including the mobile-first strategy.

Therefore, it is vital to captivate the mobile audience by providing a top-notch Data Visualization experience on their devices. For instance, the global leader in consumer electronics, Apple, paid $30 million to acquire the mobile data visualization start-up MapSense last year.

Examples of current trends in data visualization and business intelligence

The data product chain has become more democratic with the development of self-service analytics. All of a sudden, advanced analytics are not just for analysts. Enterprises and process improvements have been hampered by the inflow of big data as well as the rate at which it is produced. Already, many businesses are adopting more data and a better strategy. The use of sensors and surveillance equipment is expanding, which aids in the data growth capacities of IoT devices. The transition is inevitable. By 2025, there will be 41.6 billion linked IoT devices, which together will produce 79.4 zettabytes of data.

The following are some examples of business intelligence and data visualization trends that will be prevalent in this century:

Augmented analytics

The first trend we'll discuss is augmented analytics, which combines human capabilities with artificial intelligence. In the coming years, this will propel the analytics industry. People are beginning to understand the power of technology thanks to ML, NLP, and AI, and according to IDC's semi-annual Worldwide Cognitive and Artificial Intelligence Systems Spending Guide, global spending on AI systems will double from its current level to reach $77.6 billion by 2022.

Plug and play analytics solutions

Organizations naturally adjust to an analytics solution that comes pre-loaded and meets the needs of one or more sections. The deployment of an analytics project requires enormous amounts of due diligence, from planning outages of current systems to rolling it out in stages. Companies spend a lot of time developing the proper analytics strategy since it is challenging to do it right the first time.

Data Quality Management

The quality of data analytics trends increased significantly during the past year. The development of business intelligence to analyze and derive value from the countless sources of data that we collect at a high volume also brought along several failures and low-quality reports: the disparity of data source types increased the complexity of the data integration process.

Data quality management (**DQM**) was identified as the most significant trend for 2020 in a poll performed by the Business Application Research Centre. The evolution of business intelligence will be primarily focused on the quality and context of data use and interpretation, in addition to the necessity of gathering as much information as possible.

Data governance

About Business Information technologies, managing data is becoming increasingly important due to the complexity and proliferation of data sources. The latter, as often as not, lays out the principles on which they base strategic decisions in today's data-driven business culture. The business decisions that follow from poor or erroneous data can be disastrous for the company. Only one individual will have access to and use data, thanks to data governance. This helps to gradually increase the data's reliability, leading to better and more precise business decisions.

Integrated capabilities

In business, quick decisions can frequently be as important as long-term plans. Companies struggle with fundamental efficiency in its absence. This leads to the inclusion of analytics capabilities and content in business applications.

All of these embedded analytics are anticipated to be a key BI trend in 2020. Integrating data analytics into the current corporate model in the form of ERPs, finance programming, CRMs, and marketing automation will assist firms in operating more intelligently.

NLP Driven Analytical Queries

Analytics chatbots that respond to user-conversational questions have a lot of potential in this area because usability and acceptance are major challenges for business intelligence and analytics projects. These bots have machine learning capabilities, so as you interact with them more, they learn more and become more insightful while requiring less work.

Any step skipped could result in a potentially disastrous business decision. A platform requires decision-makers to select filters, delve through dashboards, and then acquire actionable information. A safe and secure solution for such events is a chatbot.

Data-driven culture

We have talked about the value of data-driven decision-making in businesses, but one of the top priorities for business managers and business intelligence professionals in 2019 will be fostering a data-driven culture throughout the entire organization. This is one of the most talked-about trends in data analytics.

Making decisions without using data could result in potential harm that will be difficult to repair, but implementing a data culture across agencies can prove to be advantageous on all fronts: employees' attitudes will change, and data will be stored on the cloud where it is simple to access, precise market segmentation will become the norm, and costs will be considerably lower.

Clear and concise data visualization

The significance of effective data finding and visualization has already been emphasized by the Enterprise Applications Research Centre. Due to improved usability and a shorter time to insight, a good analytics dashboard increases adoption. It also fulfils the expectations of a genuinely self-service BI solution by removing the need for laborious filtering and helping to provide decision-makers with insights.

Collaborative business intelligence

As they deal with a workplace that is becoming more and more competitive, managers and employees must now engage differently. Collaborative business intelligence is a new type of business intelligence that is becoming more and more prevalent. To handle the new issues presented by the fast-track industry, a combination of web conferencing, encompassing social media as well as other technologies, has been developed. More analyses are conducted, and reports are amended due to this collaboration. The phrase "self-service Business intelligence" frequently appears when discussing collaborative BI in the sense that these self-service technologies do not necessitate an IT team to access, interpret, and comprehend all the data.

Mobile BI

The integration of mobile business intelligence into BI systems is increasing, and next year the trend will not go away. According to the research we cited at the beginning of the post, it is one of the most significant new trends in business intelligence that has been seen by approximately 3000 people working in the field.

Businesses may now access their data in real-time thanks to mobile business intelligence, enabling quicker responses to any business-related events and providing greater freedom to users who are currently out of the office but require access to their data.

Conclusion

This chapter highlights the significance of data visualization in making sense of large amounts of data. Since the human brain is not equipped to process raw data, visualization in the form of charts and graphs helps us comprehend patterns, trends and draw conclusions efficiently. As the era of Big Data progresses, visualization has become even more critical in reducing data noise and presenting essential information.

The next chapter will help you get familiarized with the methods for using data visualization to comprehend real-world situations and make decisions that will ultimately benefit companies.

Points to remember

- Data visualization is the graphical depiction of data and information in a pictorial or graphical format. It is the practice of translating information into a visual context.
- Data visualization makes data easier for the human brain to absorb and draw conclusions from.
- Data visualization's major objective is to make it simpler to spot patterns, tendencies, and oddities in big data sets.
- Interactive visualizations of data can have a favorable impact on an organization's decision-making process.
- By organizing the data into an understandable format and showing the trends and outliers, data visualization aids in telling stories.
- A strong visualization highlights important information while reducing data noise.
- The simplest graph could be too dull to draw attention or make a strong point, while the most striking visualization might completely

fail to communicate its point or could say a lot. Balancing is import for effective visualization.

Questions

1. What do you understand by data visualization?
2. Think and analyze, how can you visualize time series data?
3. Enlist some of the benefits of data visualization
4. Elaborate why is data visualization important?
5. Considering the field of sales and marketing, elaborate the applications and benefits of data visualization.

CHAPTER 3
Data Visualization Use Cases

"By visualizing information, we turn it into a landscape that you can explore with your eyes. A sort of information map. And when you're lost in information, an information map is kind of useful."

– David McCandless

Business users can utilize data visualization to comprehend their massive data sets. They gain by being able to spot fresh patterns and data mistakes. The users can focus on locations that show red flags or progress by making sense of these patterns. This procedure then propels the company forward. To help users understand, use, and benefit from data, data visualization generates visualizations of the interconnections between data and other aspects.

There are many ways to present data in a graphical or pictorial format. This chapter presents various use cases where decisions are taken using data visualization. Also, the chapter provides an encouraging explanation of domain-wise usage of visualization for better decision-making.

Structure

In this chapter, we will discuss the following topics:

- Introduction
- Various domains where data visualization implemented
- Possible reasons to use data visualization.

- Domain-wise use-case

Objectives

The foremost objective of this chapter is to make you familiar with the ways of utilizing data visualization for understanding the actual scenarios and making decisions for the overall benefit of organizations. After studying this unit, you will be able to elaborate on how data visualization is changing the way business units work and make decisions toward the growth of a business.

Introduction

Data visualization aids in the telling of notions by transforming data into a more understandable format, showing trends and anomalies. An exceptional visualization tells a story by reducing noise from data and emphasizing the most important facts. Visualization is necessary to help organizations recognize data trends quickly, which would otherwise be difficult. Analysts can visualize concepts and new patterns thanks to the pictorial depiction of data sets.

Data visualization involves taking into account raw data and then transforming it into graphs, charts, and infographics to offer a visual representation of the data. It is the information's visual or graphical representation, to put it in much simpler terms. Thus, Data visualization can assist in transforming large amounts of data into an understandable visual format. When the same information is presented in a visual format, it takes far less time to understand the complex information. The usage of the visualization for their data sets in the current global scenario is illustrated using a variety of use case scenarios.

Industries are becoming dominated by big data. Huge data strips are transformed into useful data points via data analytics. By swiftly presenting facts to a human brain, data visualization contributes to transferring information. Visualization has a lot of aesthetic value and can express and establish a concise message.

In general, data visualization aids in the following disciplines:

- Data visualization employs information visualizations such as tables and charts to present information to users clearly and efficiently.
- Data visualization makes it easier for consumers to analyze enormous amounts of data.
- Complex data is made more accessible, intelligible, and usable through data visualization.

Importance of use cases

Use cases create a goal-oriented chain of events that is simple for developers to follow by describing the functional needs of a system from the viewpoint of the end user. One primary or fundamental flow and several other flows make up a full use case.

Use cases are valuable because they aid in describing how the system ought to act and, in the process, aid in generating ideas for potential problems. They offer a list of objectives, and you can use this list to calculate the system's price and complexities.

Various domains where data visualization is implemented

Data visualization is a powerful tool that finds its application in a wide range of domains. From business and finance to healthcare and scientific research, data visualization helps in communicating complex information and insights in a more accessible and understandable way. Let us explore some of the various domains where data visualization is implemented.

National Geographic

National Geographic saved a lot of time in understanding its data with *Domo*. Because information was gathered from many sources, the information that National Geographic's marketing team provides has a restricted level of visibility.

Using the following techniques, Domo assisted National Geographic in getting rid of that problem:

- **Enhancing openness**: Using visuals to visually convey complex material in a way that is simple to understand.

  Domo's capability to graphically express large datasets in understandable graphics has helped us to release more data transparency from across the organization.

  These critical performance indicators can now be distributed across several platforms.

- **Significantly increasing production**: With Domo, it is quick and easy to produce monthly reports for traffic and engagement, increasing productivity.

  Overall, Domo helps users save two to three hours every day.

- **Mobile version of Domo**: The real-time data available on the *Domo* app can be used for communication between various teams. The software's capacity to allow groups to engage with the datasets in real time is one of its primary selling points.

When it comes to extracting insights from data, *Domo* is the magic solution. Observing a clear decisive moment demystifies complicated key performance indicators and helps us to unleash the power of many data sources. When it comes to extracting insights from data, *Domo* is the magic solution.

Improvado use case examples

PenPath, with the help of *Improvado* and *Tableau*, helped an e-commerce company improve its Sales.

PenPath is a marketing intelligence business that specializes in helping businesses and organizations see their data.

An e-commerce business from *St. Louis* that desired a deeper understanding of its data approached PenPath.

Essentially, the e-commerce corporation was seeking the following answers in its data:

- The current state of affairs with their clients.
- Platforms that clients utilized to place their orders.
- The reported purchase rates for all of their channels.

PenPath resorted to *Improvado* for this in the following ways:

- Through its dashboards, *Improvado* assisted in automating data and delivering insights from the data.
- PenPath was capable of giving the clients a glimpse of all the statistics they needed to track with the aid of *Improvado* and *Tableau's* visualization.
- The e-commerce company learned as a result that a large percentage of their clients used *Bing* as their primary search engine.
- The dashboards that *Improvado* and *Tableau* helped to construct allowed the organization to target the right audience with ease.

Without sampling, Google Analytics provides reports

Your marketing performance may suffer if you depend solely on sampling data. *Improvado* pulls data every day from *Google Analytics* to assist you in creating reports that are 100 percent correct. By doing so, you may avoid sample mistakes and present precise findings to your customers.

Put or streamline your analytics data into your preferred database, such as *Google BigQuery, Snowflakes,* or *Data Bricks.*

Build reports on the visualization platform of your choice, that is, *Google Data Studio, Qlik, Looker, Sisense, Metabase, Tableau*, and so on.

Revenue generated by email

One of the most effective methods of generating income is email. You can expedite your trials and get better results using finer analytics. Utilizing Improvado can simplify your email disclosures.

You can streamline data from applications like *MailChimp, Pardot, Oracle, Autopilot, Marketo, ActiveCampaign*, and so on. that you use for email automation.

Analytics for eCommerce

With Improvado, explore deep into eCommerce metrics 90 percent less human labor will be required, and your data will yield greater insights. Additionally, you may automate reporting on email effectiveness, SEO, and product performance.

Optimizing your website and promotional spending can boost exchange rates. This can be done by examining which website pages or product pages receive the most page views in connection to marketing campaigns or other referrals to determine which is more popular.

Constructing unique data pipelines with Improvado transforms;

- Platforms for e-commerce, that is, *Shopify, eBay*, and so on.
- Draw data from *Google Analytics, Adobe Analytics*, and so on.
- Adding revenue, refunds, and AOV from your CRM to reports will enhance them.
- You can integrate a database of deals if you have one.
- Create reports using your preferred graphical platform, such as *Domo, Tableau*, and so on.

Predictive modeling

Optimize your marketing initiatives to increase leads, reduce **Cost Per Action** (**CPA**) or **Cost Per Install** (**CPI**), and ensure sales increase. In terms of revenue, not all prospects are created equal. By linking each customer touchpoint to revenue, attribution models assist in overcoming this restriction.

There is also a prepared repository of pre-built attribution algorithms. Using *Improvado* Transforms, one can construct unique data pipelines or assist in the following:

- Compatible analytics programs include *Adobe Analytics, Google Analytics, Mixpanel, Adjust,* and others.
- The income from your customer relationship management, such as *Pipedrive, HubSpot, Salesforce,* and so on., should be combined with the customers' journey.
- You can link databases like *PostgreSQL, Databricks, Google Big Query, Snowflake,* and so on., if you store deals in them.
- Generating reports using your preferred visualization platform, such as *Metabase, Qlik, Looker,* and so on.
- Combining with the conversion from *AppsFlyer, Mixpanel, Google Analytics, Adobe Analytics,* and so on.
- Add deals, revenue, and from **Customer Relationship Management** (**CRM**), and there is turnover to your reports using the *HubSpot, Pipedrive*, or *Salesforce* platforms.
- Use a visualization tool of your choice to create reports, such as *Google Data Studio, Looker, Qlik, Sisense, Metabase,* and so on.

ROMI (return on marketing investment) analytics

An essential component of marketing analytics is **Return On Marketing Investment** (**ROMI**). Marketing campaigns that have been optimized primarily on ROMI typically produce up to 40% greater results. All of your market information, as well as purchases from analytics platforms or customer engagement, may be streamlined with Improvado.

You can use Improvado Integrate to modify your analytics solutions through;

- Obtaining information on advertising expenditures from *Facebook Ads, YouTube, Display* and *Video 360, Instagram, AdRoll, Taboola,* and so on.
- Combining conversions of data from sources like *Mixpanel, AppsFlyer, Adobe Analytics,* and *Google Analytics.*
- Adding deals, revenue, and churn data from your CRM to reports.

- You can connect it if you keep deals in a database.
- Building ROMI findings on your preferred visualization platform.

Evaluation of acquisition costs

Innovation and experimentation are the keys to growth marketing. If you want to create sustained growth, you must run as many tests as you can and act quickly. If your organization manually prepares CPA, CPI, or **Cost Per lead** (**CPL**) reports, none of it is even remotely feasible. Improvado assists businesses in streamlining their decision-making procedures and automating marketing reports.

- One can get statistics on advertising costs from *Facebook ads*, *YouTube*, and other sources through assessments.
- Add conversion data from *Mixpanel*, *AppsFlyer*, *Adobe Analytics*, and *Google Analytics*, among others.
- Adding sales, revenue, and churn data from your CRM will enhance reports.
- You can integrate it if you keep transactions in a database.
- Use the visualization sites or tools of your choice to create presentations.

Grafana

Grafana aided *DigitalOcean* in making its reporting of graphs simpler. The services provided by DigitalOcean, a supplier of cloud infrastructure, can be used by developers who deal with producing apps across a multitude of systems.

Teams at DigitalOcean were analyzing data using complex technologies. The majority of them employed different metrics, graphical representations, and internal graphic approaches.

Below are some ways that Grafana helped DigitalOcean:

- The supporting team was able to satisfy customers significantly because of the capability to now display graph snapshots.

- Every team used the same data visualization tool, which kept them all informed of the process and on the same page.
- Task allocation metric storage reduces costs.
- The platform team discovered that Prometheus integration made it easier to compile data from any source.

19E

19E Solved its Dashboard Reporting Requirements with **Klipfolio**. Analog engineering documentation is the main source of information for 19E. They were having dashboard-related problems.

19E used the Google spreadsheet before choosing the Klipfolio. But more often than not, it was a laborious effort with blunders.

A cloud-based web software called Klipfolio can help you to improve with data. By comprehending, visualizing, and monitoring the indicators and metrics which are most important to your company, you can expand it. You can collect, share, analyze, and gain knowledge from your data using Klipfolio in real time.

19E was searching for dashboard remedies with an emphasis on filing complaints. They are currently using Klipfolio to track three vital aspects:

- Production developments
- Primarily IT architectures
- Invoices

They used Klipfolio to track the assigned objectives for their companies*:*

- Processed documents
- The average number of documents handled per hour by a person.
- Charges for IT infrastructure per hour.

Wufoo (Infinity-Box)

Through the use of *Fusion Charts, Wufoo (Infinity-Box)* accelerated the pace of report generation. *Wufoo's* supplier is *Infinity Box Inc*. With the help of the web application **Wufoo**, customers can effectively obtain data by creating web questionnaires.

Before choosing Fusion Charts, Infinity-Box used the typically sluggish and non-adaptive *Wufoo* report creator. Users were able to peek into the grids and see the data they needed to see, thanks to the analysis. Their main worry was that because they processed data slowly, they were unable to process it promptly.

Infinity-Box found FusionCharts to be beneficial in different aspects:

- Creation of visually appealing infographics without challenging the status quo.
- *FusionCharts'* simple connection gave the development members access to *PHP*, *MySQL*, *JavaScript*, and *Flash*, which was a bonus.

Payback

Payback was able to derive insights into the data present with them because of *ThoughtSpot*.

Germany-based **Payback** is a tool for marketing and customer loyalty.

The company found it challenging to develop a good understanding of the huge amount of information that was available to them. It would take them a long time to analyze the data and come to the findings. It took a lot of time to do this.

It was assisted by ThoughtSpot in the following methods:

- Enabled the teams to build their pin boxes, which kept the information relevant to their specific search ready and saved time.
- They were more productive as a result of the openness of their lookup requests.
- Pinboards could be made and shared with ease.

- Facilitated the quicker analysis of consumer concerns, which was previously time-consuming.

Associated Press

Associated Press improved its election reporting with the help of *Microsoft Power BI*. When it comes to election coverage, the Associated Press has always been in the front. The media outlet is renowned for providing up-to-the-minute coverage of election outcomes.

To this day, after several years in the media world, Associated Press wanted to revolutionize the way ballots are reported.

Microsoft Power BI was made available, and AP was able to recognize particular inconsistencies:

- The outcome of the election might be presented to Associated Press members in simple-to-understand graphs and charts using Microsoft BI.
- AP was able to give a more in-depth assessment of the outcomes for the entire state.
- For improved results visualization, they even managed to create a real-time data refresh system.

IDS

Internet distribution system (**IDS**) solved its reporting problem related to returned data with **AnyChart**. The oil and gas sectors receive web-based information from different data suppliers. They collect and disseminate data from the web using the DataNet2 system, making it accessible to numerous users.

IDS was having issues with the following:

- Displaying the data that VisNet, a back-end search engine that drew information from the DataNet2 platform, had provided.
- Making interactive charts from the returned data and configuring them.

- High-quality visualizations should be customizable, as should the same visualization.
- AnyChart saved IDS by stepping in.

IDS benefited from AnyChart in the following methods:

- IDS received sophisticated and attractive animated infographics from it.
- Clients were able to view their information on the screen clearly and straightforwardly, thanks to the existence of a simple interface on top of the AnyChart framework.
- A greater number of alternatives for graphs, charts, and so on., will be available to users of the AnyChart library.
- VisNet benefited greatly from the visualization technologies offered by AnyChart.

Lifetime Brands

Lifetime Brands was able to have a complete view of its divisions with the help of *Qlik.* A pioneer in the advertising of cutlery and home furnishings is *Lifetime Brands*.

The difficulty they had was keeping track of the distribution network and the integrated feedback process they used to compile data from several business IT platforms. Qlik view was selected by lifetime brands. Employees were now examining a variety of accomplishments, including sales, forecasts, inventory reports, and so on.

Qlik View helped Lifetime Brands in the following ways:

- Eliminated wait time for reports.
- Consolidated different reports into one.
- Visibility into supply chain management and real-time sales analysis.

Informatica

Informatica was able to present reports on archived data for its customers, thanks to *JReport*. The industry expert in data and information management is Informatica. The firm's area of expertise is clients on data integration and governance.

Informatica was dealing with problems induced by:

- Presenting reports to consumers using old data was a difficulty for Informatica.
- Deliverance of a company's current reporting at every level.

JReport helped Informatica by providing the following:

- Clients who have self-service reporting capabilities within the data sets.
- Putting together pre-built report bundles that can be used to generate more income by being sold individually.
- Adoption of report creation improved by 80% as a result of JReport implementation.

QualiSystems

QualiSystems was able to know more about the preferences of their customers because of *Sisense*. Your firm can launch a demo on-demand on any cloud platform with the support of QualiSystems support in automating apps.

For end-users who wish to test the functionality of their apps, its platform offers self-service access to **sandboxes**, which comprise all the application-related material.

It was difficult for Quali to be transparent with its clients about how their products functioned.

Before choosing Sisense, Quali had to deal with other challenges:

- First, they had to gather all the data and provide it to their users in useful ways.

- Second, they had to keep tabs on who used the sandbox and why.
- Third, they had to give the non-technical personnel a relevant description of the results.

Sisense assisted Quali by assembling data from various sources onto one dashboard.

All user-related data was now captured and sent to Sisense whenever users deployed their cloud infrastructure.

This made it possible for clients to get data and quickly learn specifics like usage kind, resources section, and so on.

How different industries and sectors utilize data visualization

Every set of data has important information to convey. Whether or not you can see it depends on how successfully you have visualized it.

Whenever it comes to data, the adage *a picture is worth a thousand words* has never been more accurate. Businesses have the opportunity to collect terabytes of data on every element of their operations, but displaying that data is still difficult.

It might be challenging to understand the narrative that data is attempting to tell when presented in dry table form. But when data is removed and presented visually, it comes to life and provides businesses, employees, and customers with useful information.

Insurance: EmblemHealth

One of the top non-profit suppliers of health insurance in the US is *EmblemHealth*.

It was restricted by its current models but desired real-time internet connectivity to guide its strategic plan. However, managers found it challenging to comprehend the information provided in Excel spreadsheets and view data in real-time. Information that was out of date and erroneous was produced through scattered reports.

Oracle Cloud was used by EmblemHealth to acquire sufficient real-time information from a variety of data sources and networks.

They now have a single source of info, *Oracle Analytics Cloud,* that supports them in making important choices in real time. Management no longer has to rely on data that is four or five days old because they have fast accessibility to it.

Government: Singapore

Singapore's reaction to the **COVID-19** outbreak has gained accolades from all across the globe.

Its implementation of technology was one of its most important components. In just two months, they developed a thorough track-and-trace strategy with *SAP* and *PCI* that enabled users to observe the virus' progression through an app.

A pocket-sized, portable gadget that records complete, comprehensive, and confidential contact records was used as the answer. It assisted in delivering locals more knowledge concerning the virus's transmission and the specifics they required to protect themselves and their families.

Retail: The Home Depot

Around 2,500 *Home Depot* locations can be found throughout the country. Although no two shops are alike, it can occasionally be challenging to find certain things for both staff and consumers.

Home Depot used *Google Cloud* and *BigQuery* to enhance its stock management procedures to maintain a sufficient supply of even more than 50,000 items throughout all of its outlets. It delivers reminders when goods.

They can use mobile devices to access that data and make the most crucial client needs visible, as to where to find specific products in the market. When one of their colleagues needs to find anything, they just turn on their phone, and **Artificial Intelligence** (**AI**) directs them in the right direction has to be reordered and presents concrete information about inventory levels and supply.

They have changed the idea about what a database server can achieve with the aid of Google Cloud and developed among the most advanced inventory replenishment platforms ever made.

Energy: Geospatial

Intel, a producer of semiconductors, has a business called *Geospatial* that focuses on oil and gas corporations and electric utilities.

It created an open cloud-based architecture using **Amazon Web Services** (**AWS**) to view and analyze geospatial data gathered by satellites and drones.

It employs data to provide a 3D visualization tool that is immersive and could be used to highlight network faults. For instance, a geographic client with an oil field required a more reliable method to locate oil leaks. Every second counts because they could be costly and environmentally harmful. Detecting oil leaks is also challenging. They frequently have the appearance of shadows, which causes false-positive alarms. The system learned to recognize the essential elements of the oil leak using AI, allowing it to pinpoint issues.

Banking: Members First Credit Union

Understanding branch performance is essential for *Consumers First Credit Union* to provide their customers with the best value possible. Nevertheless, their human reporting processes made it challenging to comprehend and evaluate their data.

They used *Microsoft Azure* to automate their systems and put their data into statistics that could be read, understood, and taken action upon. They accomplished this using Power BI, business intelligence, and an interactive visualization tool for data analytics. It enabled staff to build reports and dashboards without relying on system admins or IT personnel.

To use Power BI to produce data visualizations for its branch briefings, the group needs around five months. They have removed anomalies in findings brought on by manual errors and made it quicker and simpler to acquire insights. Anyone in the organization has instant access to data, making it possible to read branch updates whenever needed.

Use cases for sales and marketing data visualization

Currently, there are strong new tools to gather and analyze data from your prospects and customers thanks to sales and marketing technology. Many agencies are based on using these technologies to benefit clients.

Now and then, technological innovation enables us to advance our understanding of the data and offer fresh perspectives that enhance sales and marketing effectiveness. An innovation involves quickly assembling data from many data sources, followed by the creation of visual displays that improve our capacity to decipher and respond to sales and marketing operations and trends.

Cloud-based data visualization from many data sources is offered by software businesses. For instance, *HubSpot* and *Databox* examines how their data visualization software performed in practical scenarios, and simplifying design firm reporting for customers is their topmost priority.

Seven significant use cases have been found for the product, which focuses on both new and future clients.

Scorecards for performance assessment

It is possible to provide clients with a breakdown of sales and marketing data through the usage of recurring scorecards. **Key Performance Indicators** (**KPIs**) can be found in the performance metrics, which are spreadsheets. The KPIs can be colored per gains or losses from the previous month or period, and monthly alerts can be supplied. A review of the scorecards is conducted during the monthly or periodic review meetings with the clients. Special attention is paid to the relative increases and decreases in KPIs, and recommendations are made for enhancing results. Please refer to the following figure:

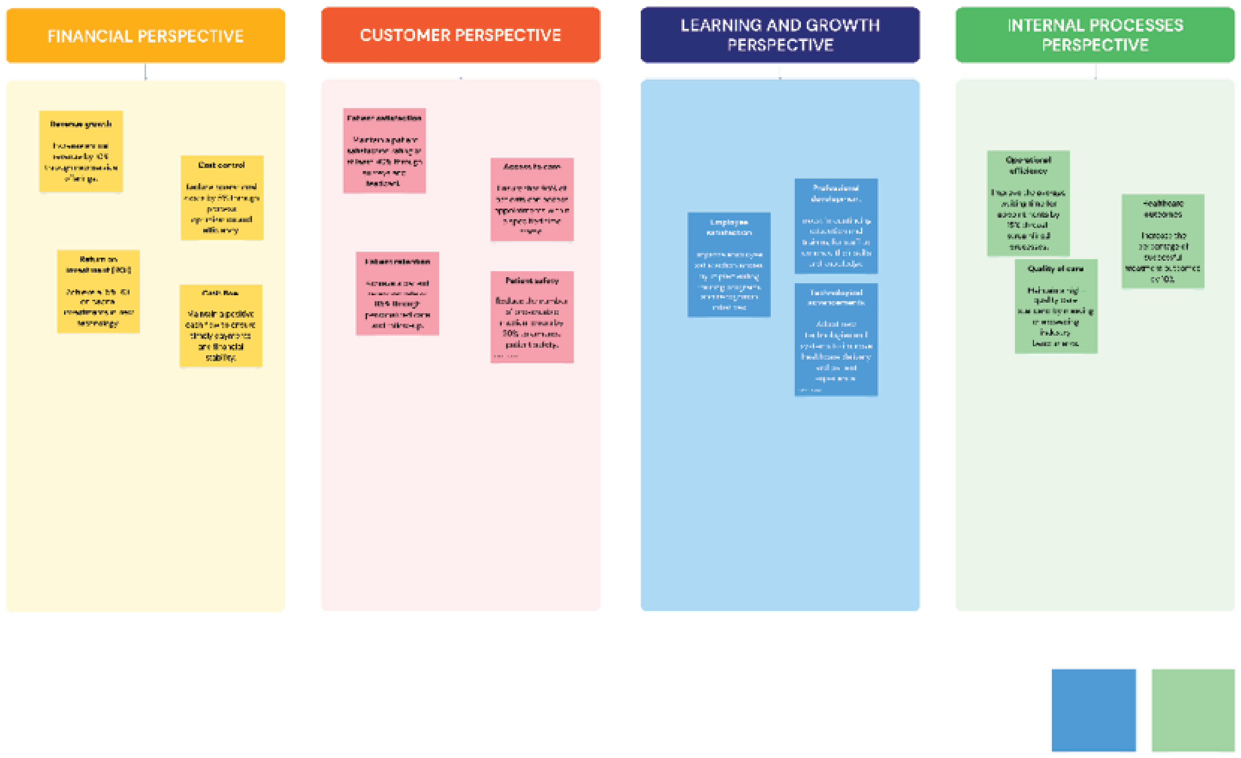


Figure 3.1: *Balanced scorecard in a given period*

The scorecards are an effective way to summarize monthly results and make comparisons month-over-month. More improvisations are in progress by adding more visual displays and by streamlining the preparation of reports.

We may include those upgrades using Databox through:

- Building visual templates that update automatically in close to real-time to reduce the amount of time, it takes us to prepare monthly reports.
- Offering clients access to online visual reports at any time (instead of once a month) and the ability to be notified when results are available.
- Facilitating the viewing and interpretation of patterns and irregularities for clients.
- Giving consumers accessibility to graphic reports via their cell phones and tablets.
- Offering professional novel insights into sales and marketing data by combining many data sources into a single visual report.

Possible reasons to use data visualization

Justifications for using data visualization in decision-making and the significance of data visualization are explained in this section of the chapter.

Using visual data, data visualization offers a rapid and efficient approach to conveying information to all audiences. Additionally, the process can help firms detect problem areas or those that require more attention, uncover elements that influence consumer habits, make data more unforgettable for clients, determine the best times and locations to sell particular products, and forecast gross margins.

Data visualization also offers the following advantages:

- A greater capacity to keep the audience interested with the information they can understand; a better understanding of the next measures that must be taken to strengthen the organization.
- Easy information dissemination improves the chance for everybody concerned to offer ideas; do away hence the need for data scientists because the data is easier to obtain and comprehend.
- Good Information Big Data can be unlocked through visualization. Any data inefficiencies can be resolved, and it can quickly and easily take in enormous volumes of data that are presented in visual representations.
- Visualization may quickly boost the speed of decision-making by allowing consumers to comprehend data fast. Any organization needs to be able to act quickly and avoid becoming weighed down by redundancies. By taking prompt action, one can avoid losses and profit from any market circumstance.
- The survival of any firm depends on a big disclosure of any deviations in the trends and patterns. Knowing what is driving higher losses or what is necessary to optimize gains is crucial.
- Quickly spotting data flaws and inaccuracies is made possible via visualization.

- Businesses may get facts and significantly help managerial tasks by using visualization. To boost operational performance and boost production, the choice might leverage visualization and on-demand data.
- It encourages narration in the most powerful way possible. The most effective technique to get the intended message through to the audience is through visuals.
- Exploring market intelligence helps move business goals on the right path by using data visualization. Correlating the information from graphical or visual representations is helpful. It enables quick examination and rapidly assimilates crucial metrics.
- It lets businesses remain competitive by learning about the newest trends through data visualization tools.
- Without data visualization, businesses would have to spend tons of their time customizing reports and modifying dashboards, replying to ad hoc requests, and so on.
- Benefits of data visualization technologies include data optimization and rapid data retrieval through customized reports, which considerably reduce employee time.

While big data visualization has its merits, there are also some *drawbacks* for businesses. These would include:

- A visualization specialist needs to be recruited if big data visualization techniques are to be used to their full potential. To ensure that businesses are providing the best use of their data, this professional must be capable of recognizing the best data sets and visualization techniques.
- Since big data visualization demands strong computer technology, effective storage systems, and even a shift to the cloud, it frequently necessitates managerial and IT engagement.
- Big data visualization can only yield insights that are as accurate as the data being displayed. As a result, it is crucial to have systems in

place for managing and regulating the quality of corporate data, metadata, and data sources.

Domain-wise use-case

Data visualization is an essential tool that enables individuals and organizations to make sense of complex data and identify meaningful patterns and insights. With the growth of big data, the need for effective data visualization has become increasingly important. From healthcare to finance, businesses, and education, data visualization finds its application in numerous domains. In this section, we will explore the domain-wise use-cases of data visualization, highlighting how it is being used to solve real-world problems and drive better decision-making across industries.

- **Business:** Data visualization is widely used in business to analyze sales figures, customer behavior, and market trends. By creating interactive dashboards, graphs, and charts, businesses can quickly identify patterns, spot outliers, and make informed decisions.
- **Healthcare:** Data visualization plays a crucial role in healthcare, allowing doctors and researchers to analyze patient data and medical research. Visualizations of medical data can provide insights into disease patterns, treatment outcomes, and population health trends.
- **Finance:** In finance, data visualization helps traders and analysts to monitor stock prices, track financial trends, and identify risks. It can also be used to create dashboards and reports that visualize financial data for clients.
- **Education:** Data visualization is used in education to track student performance, identify trends, and inform teaching strategies. By visualizing student data, teachers can better understand individual student needs and identify areas where additional support is needed.
- **Science:** Data visualization is used extensively in scientific research to analyze experimental results, visualize complex scientific concepts, and communicate findings to other researchers and the public.

- **Government:** Governments use data visualization to inform policy decisions, monitor public services, and communicate information to citizens. It can be used to create dashboards that visualize data related to public health, transportation, crime, and other government services.
- **Social media:** Social media platforms use data visualization to track user behavior, identify trends, and inform marketing strategies. It can also be used to create interactive visualizations that allow users to explore social media data in real-time. Please refer to the following figure:

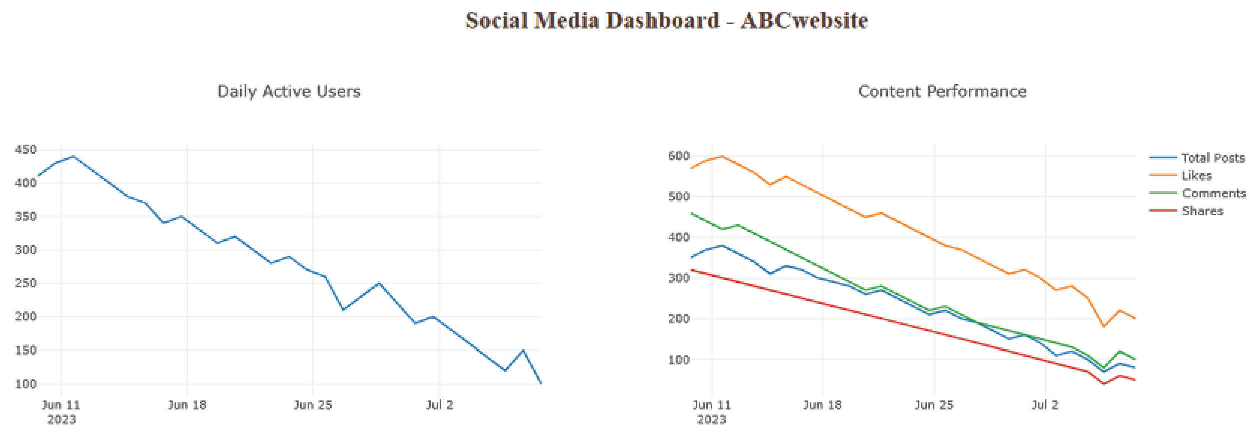


***Figure 3.2**: Social media Websites' performance dashboard*

Other common data visualization use cases

Other common use cases for data visualization include the following:

- **Politics:** The party for which each region or constituency is elected is shown on a geographic map, which is a popular application of data visualization in politics.
- **Healthcare:** Choropleth maps are commonly used by health providers to display crucial health data. In connection to a set of variables, a choropleth map shows split geographic locations or districts that have been given a specific hue. Specialists can use choropleth maps to visualize variations in a parameter, such as the mortality incidence of heart disease.

- **Scientists:** Scientific representation, also known as SciVis, enables research scientists to learn more from actual experimental results than it has ever.
- **Logistics:** The most efficient transshipment lines can be found using visualization tools by shipping corporations.
- **Data scientists and researchers:** Data scientists frequently create visualizations for their use or to communicate the data to a small group of people. The modules and techniques of choice are used to create the graphical representations. Scientists and data scientists typically employ specialized software or open-source programming languages, like Python, for complex data analysis. These data research teams use data visualization to better analyze large data sets and spot trends and patterns that could otherwise go missed.

Conclusion

In this chapter, we have understood that data visualization in today's business world can't be ignored. With the addition of tons of data, it has proved to be a sigh of relief for modern analysts. A crucial set of tools and methods for obtaining a qualitative understanding is provided by data visualization. To help users understand, use, and benefit from data, data visualization generates visualizations of the interconnections between data and other aspects.

In the next chapter, you will explore more data visualization tools and techniques which will help you find the trends and patterns in the data.

Points to remember

- Data visualization is a critical tool that helps organizations make sense of complex data and identify meaningful patterns and insights.
- Data visualization finds its application in numerous domains, including healthcare, finance, education, science, government, and social media.

- In business, data visualization helps analyze sales figures, customer behavior, and market trends, leading to better-informed decisions.
- In healthcare, data visualization is used to analyze patient data and medical research, providing insights into disease patterns and treatment outcomes.
- In education, data visualization helps teachers track student performance, identify trends, and inform teaching strategies.
- In scientific research, data visualization is used to analyze experimental results, visualize complex scientific concepts, and communicate findings to other researchers and the public.
- In government, data visualization is used to inform policy decisions, monitor public services, and communicate information to citizens.
- In social media, data visualization is used to track user behavior, identify trends, and inform marketing strategies.
- When selecting a visualization method, it is important to consider the type of data, the intended audience, and the user's level of expertise with the tool.

Questions

1. What is data visualization, and why is it important?
2. In what domains is data visualization widely used?
3. How does data visualization help businesses analyze data and make informed decisions?
4. What role does data visualization play in healthcare, and how is it used by doctors and researchers?
5. How can data visualization be used to track student performance and inform teaching strategies in education?
6. What are some examples of how data visualization is used in scientific research?

7. How can governments use data visualization to inform policy decisions and communicate information to citizens?
8. How do social media platforms use data visualization to track user behavior and inform marketing strategies?
9. Analyze factors that should be considered when selecting a visualization method for a given dataset.
10. How can data visualization be used to identify patterns and outliers in large datasets?

CHAPTER 4
Data Visualization Tools and Techniques

"Data visualization doesn't live in an ethereal dimension, separated from the data. When there's a large number of pie charts in a report or a presentation, there is something wrong in the organization, and it's not the pie. A pie chart is a potential symptom of lack of data analysis skills that have to be resolved."

– Jorge Camoes

Data visualization tools and techniques aid in the discovery of patterns and trends that lie beneath the surface of mountains of complex data. They enable businesses/organizations to present data-driven insights and findings in formats that are easily understood by all stakeholders. This chapter presents a brief introduction to various ways in which data can be visualized.

Structure

In this chapter, we will discuss the following topics:

- Introduction
- Different types of visualizations
- Most Popular data- visualization tools and techniques
- Other available tools and Techniques
- Use-case of most commonly used tools

Objectives

The foremost objective of this chapter is to familiarize you with various ways of visualizing data, viz., bar chart, tree map, pie chart, and donut chart, to name a few. You will also explore the real use case of data visualization techniques in well-known organizations like *Facebook*, *Google*, and *Twitter.*

Introduction

Data visualization is the process of putting information into a visual framework, such as a map or graph, to make it simpler for the human brain to comprehend and draw conclusions from the data. During the presentation of data, these graphics produce a clear image and improve audience comprehension. Its major goal is to spot outliers, trends, and patterns in huge data sets. Additionally, data visualization is the study of giving meaning to data by setting it in a visual framework so that patterns, trends, and correlations that may not otherwise be seen can be shown.

Python offers us several excellent libraries chock full of unique characters. Whether you want to build interactive graphs or ones that are heavily customized, Python has an excellent library. As follows:

- **Matplotlib**: low level, provides lots of freedom
- **Pandas Visualization**: easy-to-use interface built on Matplotlib
- **Seaborn**: high-level, great default styles
- **Plotline**: based on R's ggplot2, uses Grammar of graphics
- **Plotly**: can create interactive plots

Nonetheless, big data visualization refers to the implementation of more contemporary visualization techniques to illustrate the relationships within data. It includes strategies that can show real-time changes and more graphic images other than pie, bars, and other charts. As technology has improved in its advantages and accessibility, there is still a gap in the capabilities of individuals in data management. Linked data refers to a well-established protocol for publishing and managing structured information on the internet, assembling and bringing together knowledge from different scientific and commercial domains.

Given the inclusion of a significant amount of Linked Data in recent years, it is critical to offer helpful tools for researchers, scientists, and subject matter experts to display and interact with significant amounts of data sets.

Different types of visualizations

Let us discuss the different types of visualizations:

Bar chart

One of the most popular data visualizations on this list is the bar chart or bar graph. They are also sometimes known as column charts. To compare data along two axes, bar charts are employed. The numerical axis is on one of the axes, and the visual axis shows the categories or subjects being measured.

A bar chart can have either horizontal or vertical bars. Numerical values appear on the y-axis (vertical axis) of vertical bar graphs and the x-axis of horizontal bar graphs (horizontal axis.)

Examine your data before deciding which type of bar graph to use. A horizontal bar chart is the best option if the names of your qualitative data are lengthy and detailed. Utilizing a visual axis system similar to the one in the template above is another inventive choice. Use a color-coding scheme and a legend instead of posting the category name next to the corresponding bar. Refer to the following figure:

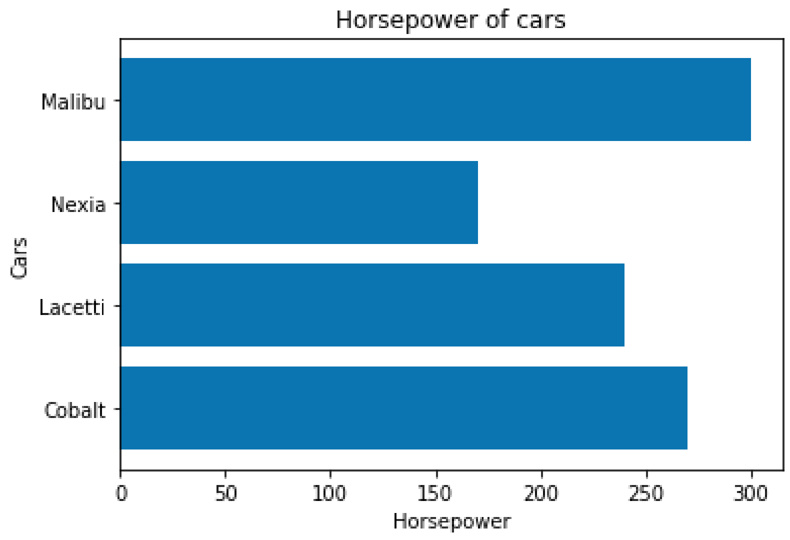


***Figure 4.1**: Bar Chart*

Use 3D bars, dynamic effects, and picture backgrounds for a more imaginative approach. You can also try making a stacked bar chart.

Pie chart

The second most common data visualization on this list is the pie chart. The data in a pie chart represent parts of a whole. The entirety of the circle is whole, and each wedge is a relevant section.

The best type of data for a pie chart has no more than five or six parts. Any more than this makes the wedges too thin at the center. If more than three values are similar to each other, it will be difficult to discern the difference. The best pie charts use contrasting colors that fit well together, making each wedge visually different from the one next to it. Please refer to the following figure:

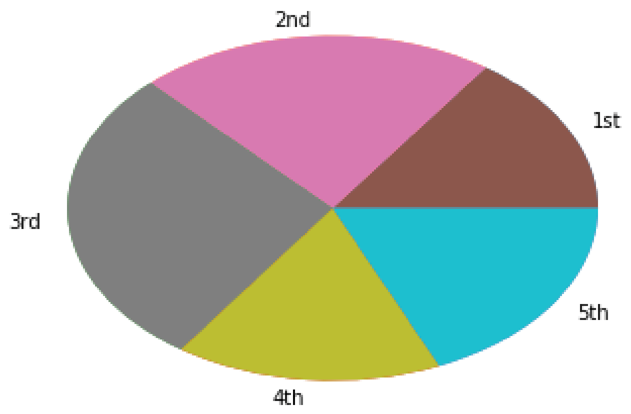


Figure 4.2*: Pie Chart*

If you have more than six sections to visualize, consider using a donut chart instead.

Donut chart

A donut chart is much like a pie chart but with the center area taken out. The difference between them is essentially visual. You can have more sections than a pie chart in a donut chart, and it will still be readable.

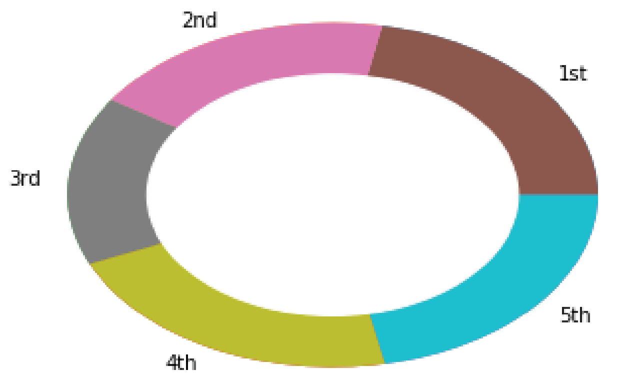


Figure 4.3: *Donut Chart*

The same rule about colors applies to donut charts; choose contrasting colors to separate the sections visually. To make them more attractive, add a 3D feature to the donut, which has more visual depth. If you're working on a project to share online, consider adding an animation to the chart.

Half donut chart and gauge chart

The half-donut chart is exactly what its name implies, half of a donut chart. It’s a good choice of data visualization type when you need to showcase small data sets. Preferably, don’t use more than three wedges in a half-donut chart. Please refer to the following figure:

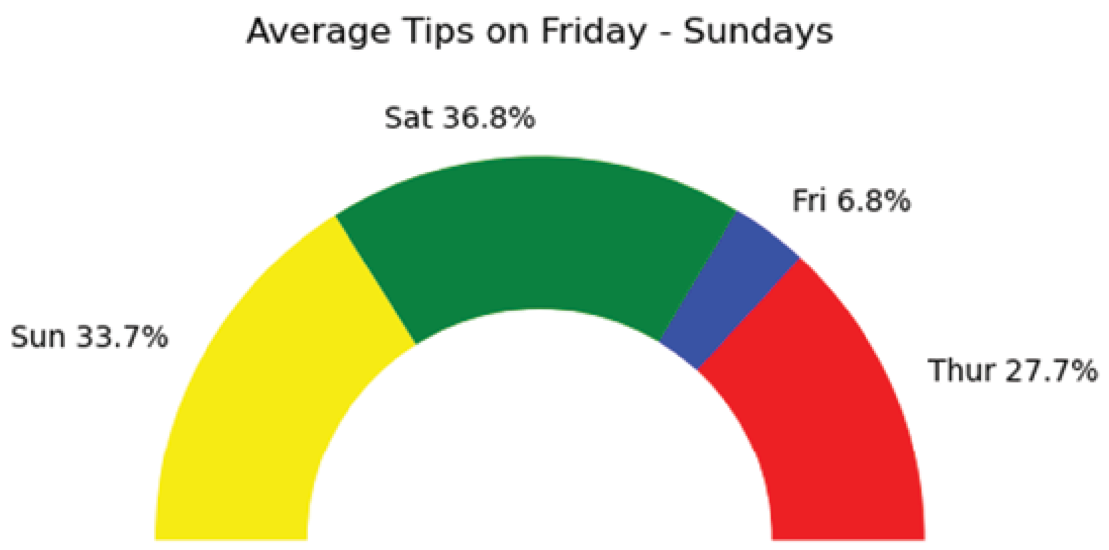


Figure 4.4: *Half Donut Chart*

Remember to use contrasting colors and use percentage values to make your half-doughnut chart easier to read at a glance. A gauge is another visualization type for percentages such as population growth during 4 quarters of a year.. The shape resembles a half donut with a pointer pointing toward the values on the chart. This is a great choice if you're dealing with a small amount of data.

Nested pie chart

Use pie charts and donut charts in unison to create a nested pie chart. These visualizations work well for representing complex data.

You can see a multilayer pie chart below depicting Male - Female student percentages in three sections of class 12th. The outside donut chart is the top-level category, The sections of the 12th class viz. A-Medical, B- Non Medical, C-Arts. On the second layer are the descriptive sections on Male-Female percentage that fit inside each main category. Please refer to the following figure:

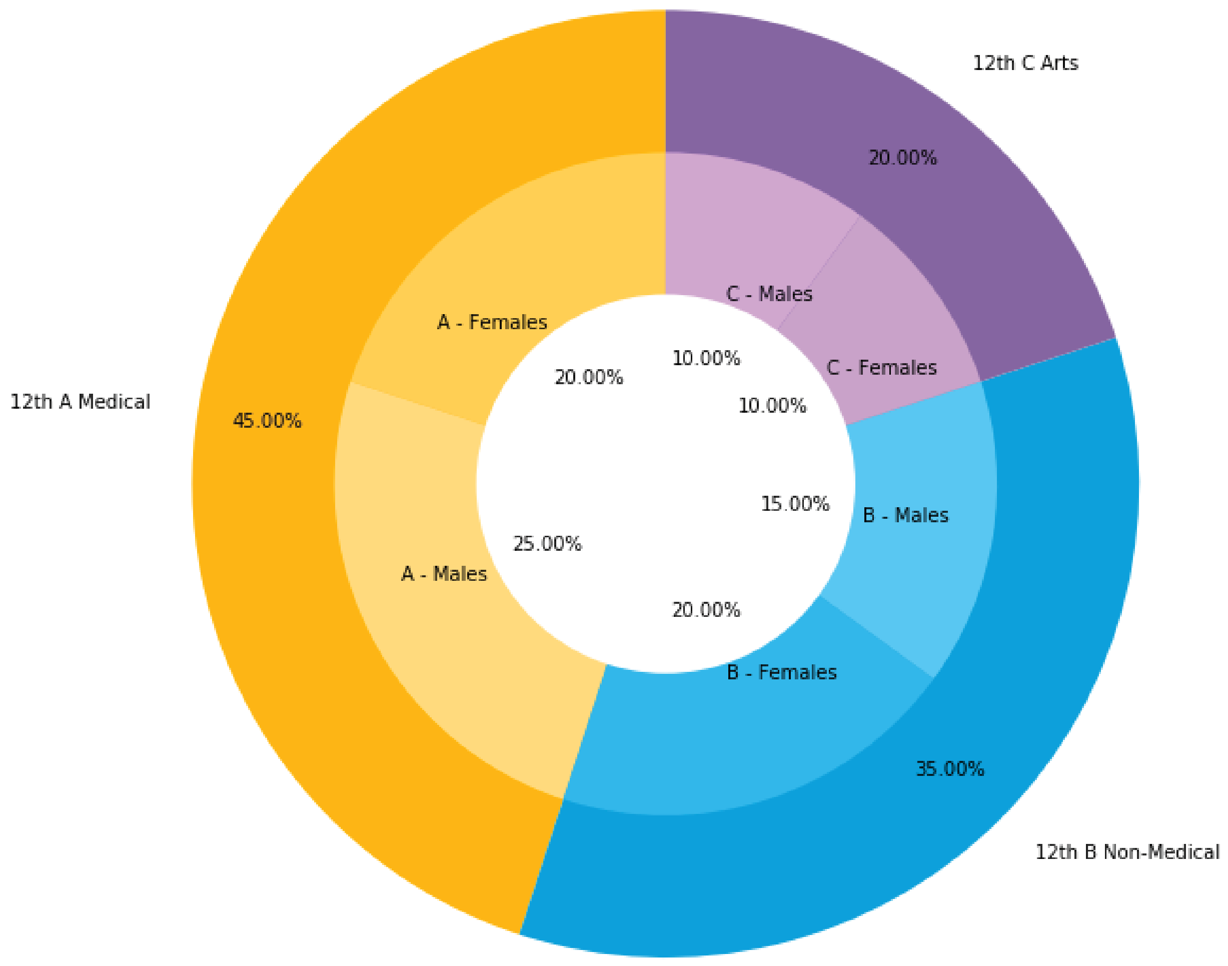


Figure 4.5: *Nested Pie Chart*

This data visualization type isn’t as easy to create as others; it does take some strategizing for all the categories to fit together and be easy to understand. In technical terms, this visualization is three pie charts layered over each other.

Line chart

A line chart or line graph is a data visualization type that showcases changing data over time. Like a bar graph, the line chart has an x and y-axis. The difference is that both axes contain numerical values representative of the data.

To create a line chart, input the relevant time frame along the x-axis and the quantitative measurement on the y-axis. Plot the data in the graph by connecting the time value and the numeric value. After plotting all the dots, connect them with a line.

A line graph can have one line or several. In the case of a chart with several lines, each one represents a category. Every category has a color, and the description is detailed in the legend:

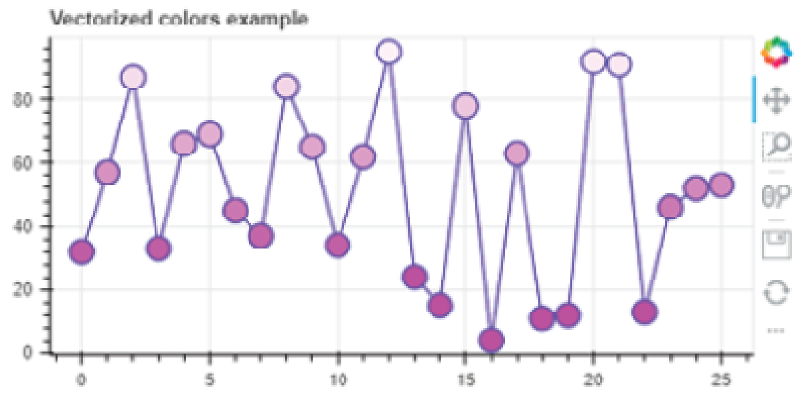


Figure 4.6: *Line Chart*

For an effective line graph, use no more than four or five lines and make sure the colors are different enough to be differentiated visually.

Scatter plot

A scatter plot is a data visualization version adopted to analyze the correlation between variables. The data is marked on the graph as dots over the crossing based on its two values. When there are dots outside of the expected range, these are called outliers and should be considered when analyzing the data. Make use of scatter plots wherein your variables are related to each other regarding a group of test subjects.

Some of these could be the relationship between weight and height in children less than 18 years old, temperature-dependent sales in an ice cream shop, diabetes, and obesity rates. Please refer to the following figure:

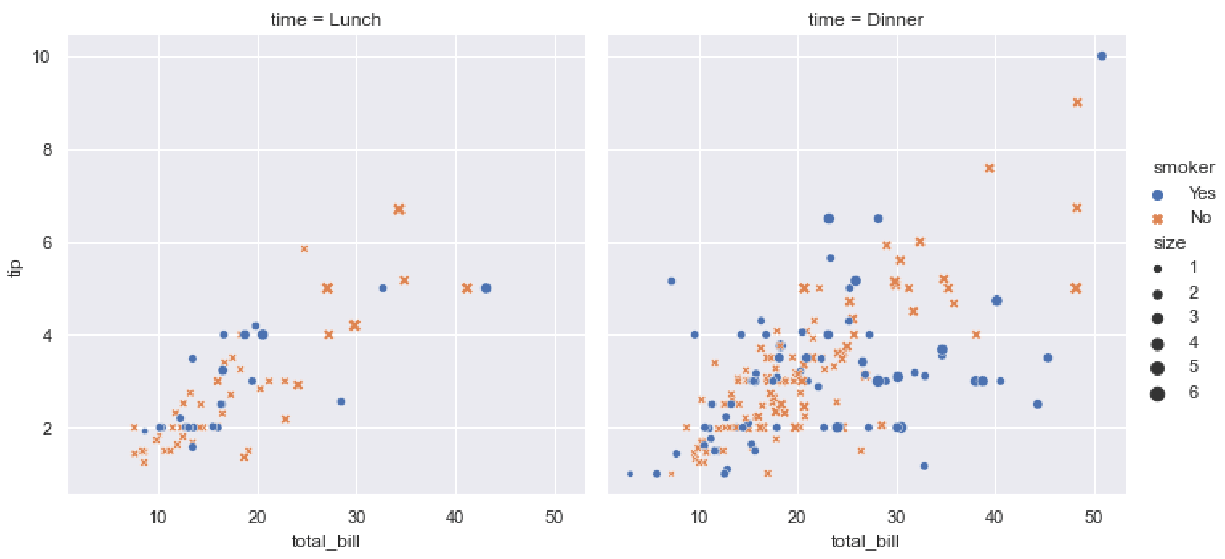


Figure 4.7: *Scatter Plot*

Avoid plotting too many data points upon a scatter plot or it will become impossible to review. Use no more than two different color dots and always use a legend if that's the case.

Cone chart

The cone chart is another data visualization type that shows parts of a whole, similar to pie charts. The difference is that a cone chart also visualizes hierarchy. The data with the highest value sits highest on the cone with the widest area. Other values flow in descending order toward the bottom tip of the cone.

Use contrasting colors to visualize the different values or select a monochromatic palette to add depth to the visual hierarchy. Don’t use more than seven or eight values, as using too many will make the cone chart difficult to understand. Include a color-coded legend for more straightforward analysis.

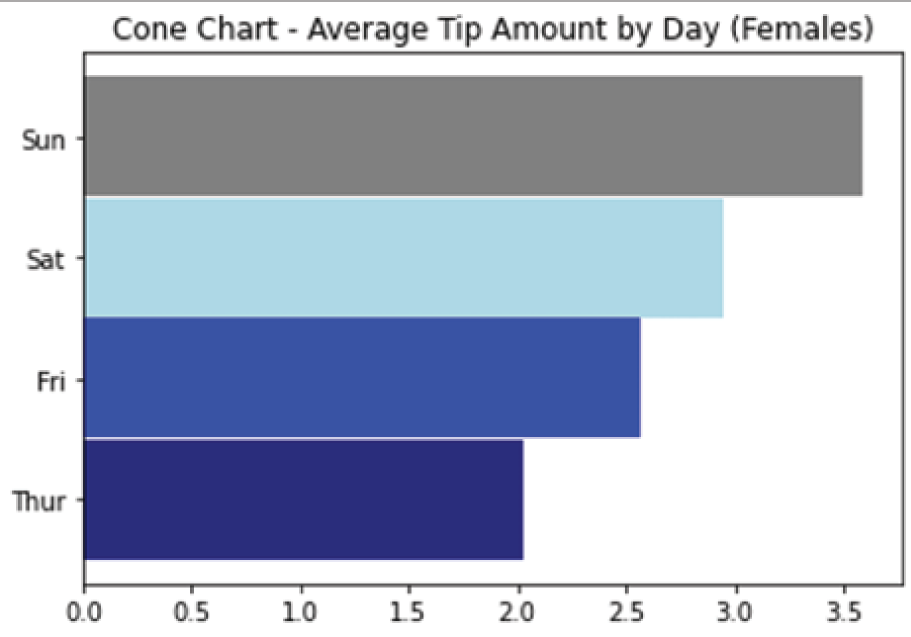


Figure 4.8: *Cone Chart*

Pyramid chart

A pyramid is much like a cone chart but placed the other way around. The smallest data set is at the top, while the largest is at the bottom. Deciding whether you want to use a cone chart or a pyramid chart depends on how you want to present data; in ascending order or descending order.

Pyramid charts can also be created without numerical data. The sections are separated into equal parts to show a hierarchy of steps or components of a whole that are only visually hierarchical. Such is the case in the example below with the pyramid in violet tones.

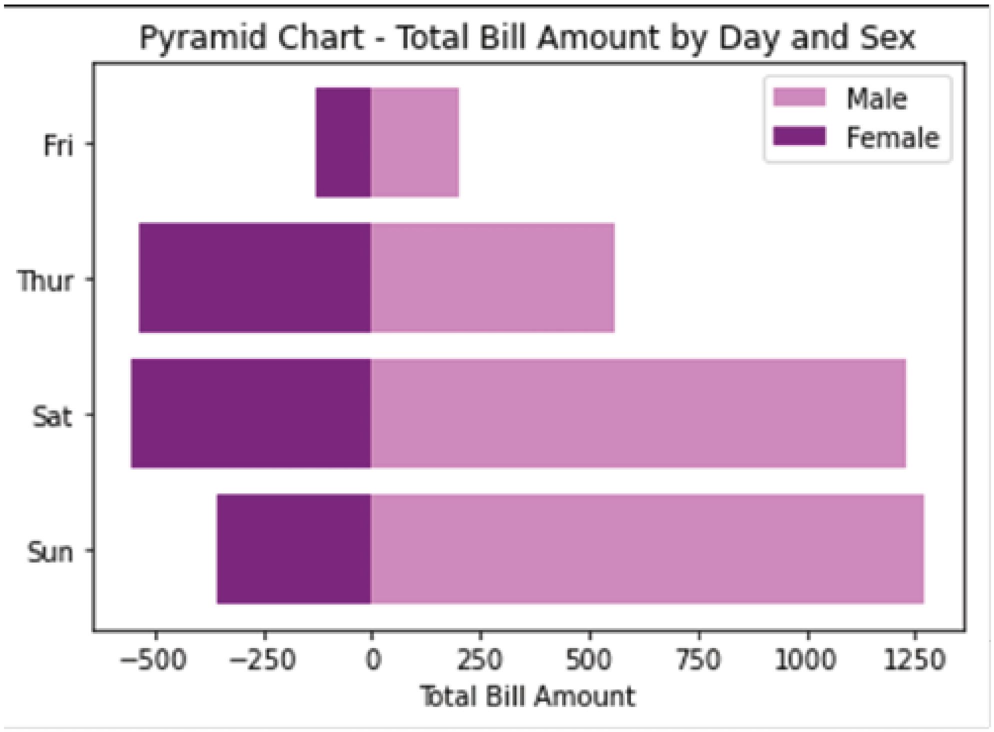


Figure 4.9: *Pyramid Chart*

Funnel chart

A funnel chart is similar to a cone chart in shape but has a slightly different purpose. The main idea of a funnel chart is to visualize a sequential process from top to bottom. Generally, the data set at the top of the process is larger than the bottom as the process diminishes the quantity as it flows down. Please refer to the following figure:

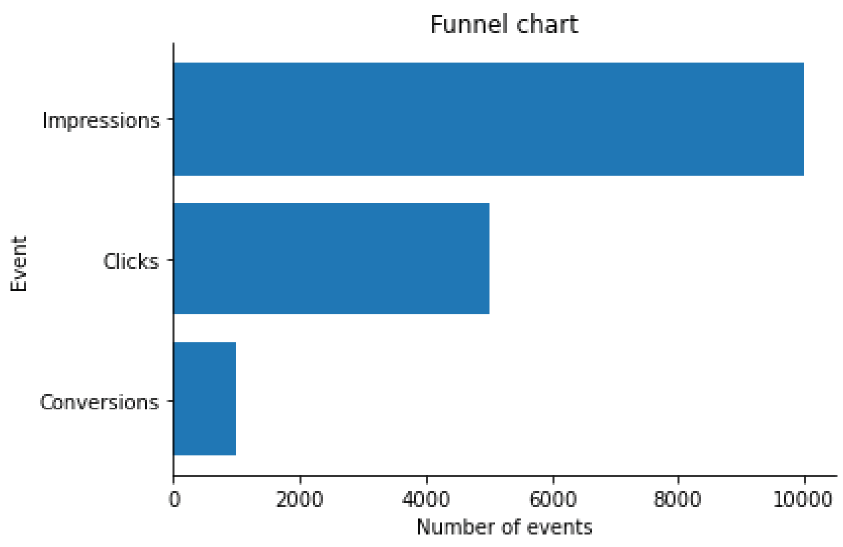


Figure 4.10: *Funnel Chart*

Funnel charts are useful in various domains and applications where there is a clear sequence of stages or steps in a process, and where there is a desire to visualize the conversion rates or progression from one stage to the next. One of the most common use cases of funnel charts is in sales and marketing to track the progression of potential customers through the sales funnel, from initial lead generation to final conversion. Funnel charts help identify areas where the conversion rate is low, and where improvements can be made in the sales process. Additionally, they can be used in website analytics to track the flow of visitors through a website, from initial landing pages to final conversions or sign-ups. Funnel charts can help identify areas where visitors are dropping off or abandoning the site, and where improvements can be made in the user experience. Moreover, funnel charts can be used in recruitment to track the progress of job applicants through the hiring process, from initial screening to final job offer. They can help identify areas where the application process is slow or inefficient and where improvements can be made to streamline the hiring process. It's essential to create a visual difference between sections.

Radar triangle

Radar charts are a data visualization type that helps analyze items or categories according to a specific number of characteristics. The radar chart layout is a circle with concentric circles where the data are plotted as dots.

The dots are then connected to create a shape. Each item or category is a shape.

A radar triangle is a radar graph that compares items or categories based on three characteristics. Each dot is one corner of the triangle. The triangle can be composed only of lines or with a transparent color fill.

It’s important to remember that you can’t add too many layers to a radar graph, or it will be impossible to analyze. Please refer to the following figure:

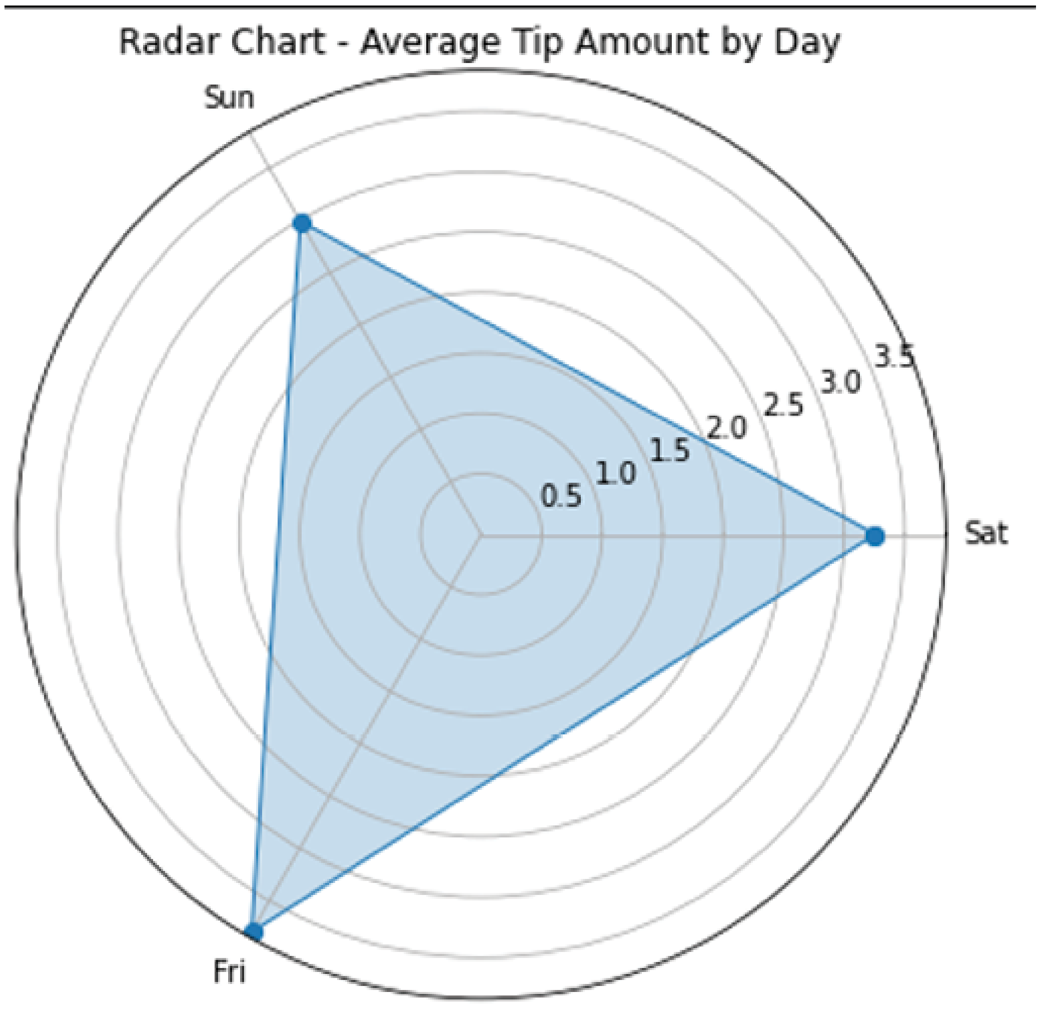


Figure 4.11: *Radar Triangle Chart*

Radar Polygon

A Radar polygon is the same as a radar triangle, but the resulting shape is different. A radar triangle has three points for characteristic data, while a radar polygon has four or more. The maximum number of points is 9 or 10, and the max layer of items is 4 or 5.

When choosing colors for each item, select ones that will layer well and not become a dirty mess where they all overlap — your best choice is to use a series of monochromatic tones with one base color, For example, shades of blue and purple or shades of red and orange.

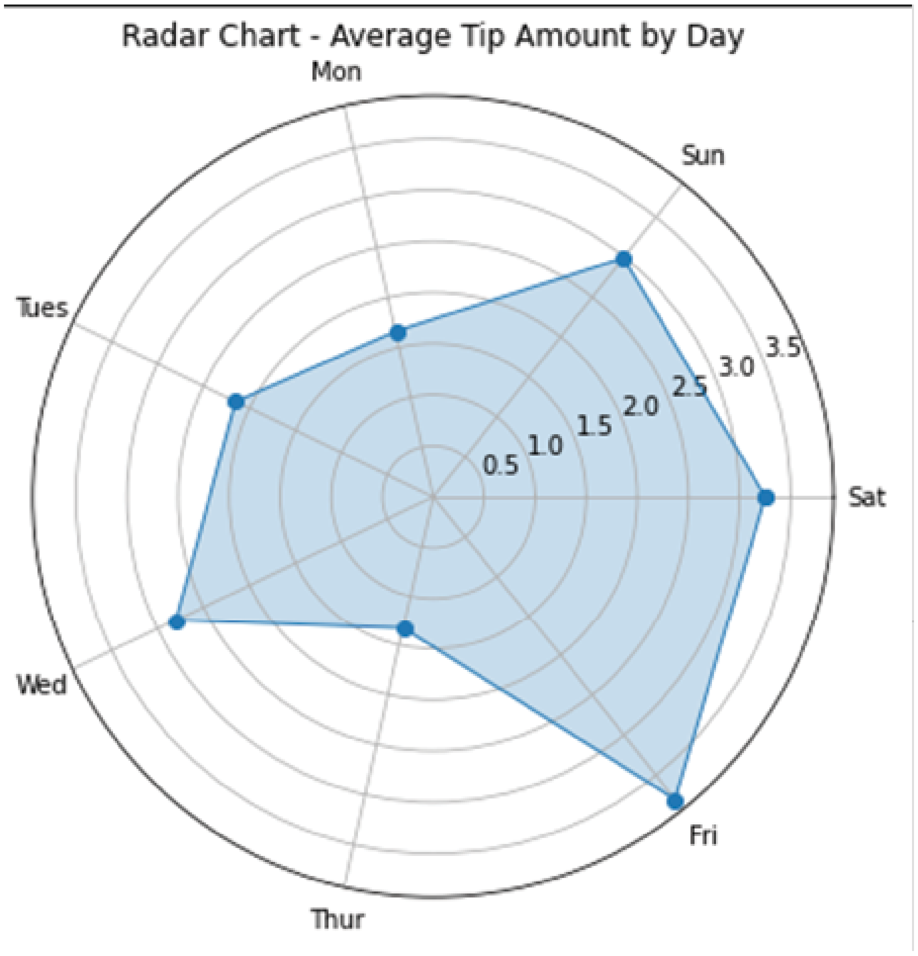


Figure 4.12: *Radar Polygon Chart*

Polar chart

A polar chart has the same circular base as a radar chart, but the data plots differently. Instead of connecting points to each other, wedges expand outwards from the center:

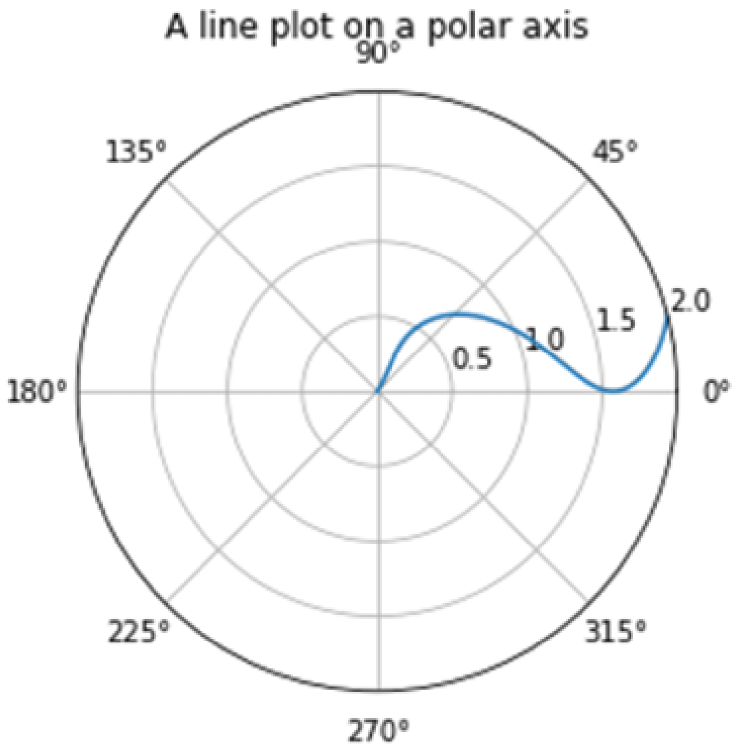


Figure 4.13: *Polar Chart*

The difference is primarily visual. Choose a polar graph if the data values are very different from each other. Otherwise, it can be challenging to read at a glance. A polar chart is simply a cartesian chart where the X-axis is wrapped around the perimeter. It can render common cartesian series types like line, column, area, or area range.

Area chart

The area chart is a variation of the line chart. The difference is that the area between the baseline and the values plotted on the line is colored in. The color fill is semi-transparent so that the overlapping regions are easy to read.

Even though you can switch any line chart into an area chart, it’s not always the best practice. An area chart can’t have more than four or five datasets simultaneously; the possibility of occlusion is too high. Area charts are sometimes stacked, separating the data into sections as part of whole relationships or as cumulative data.

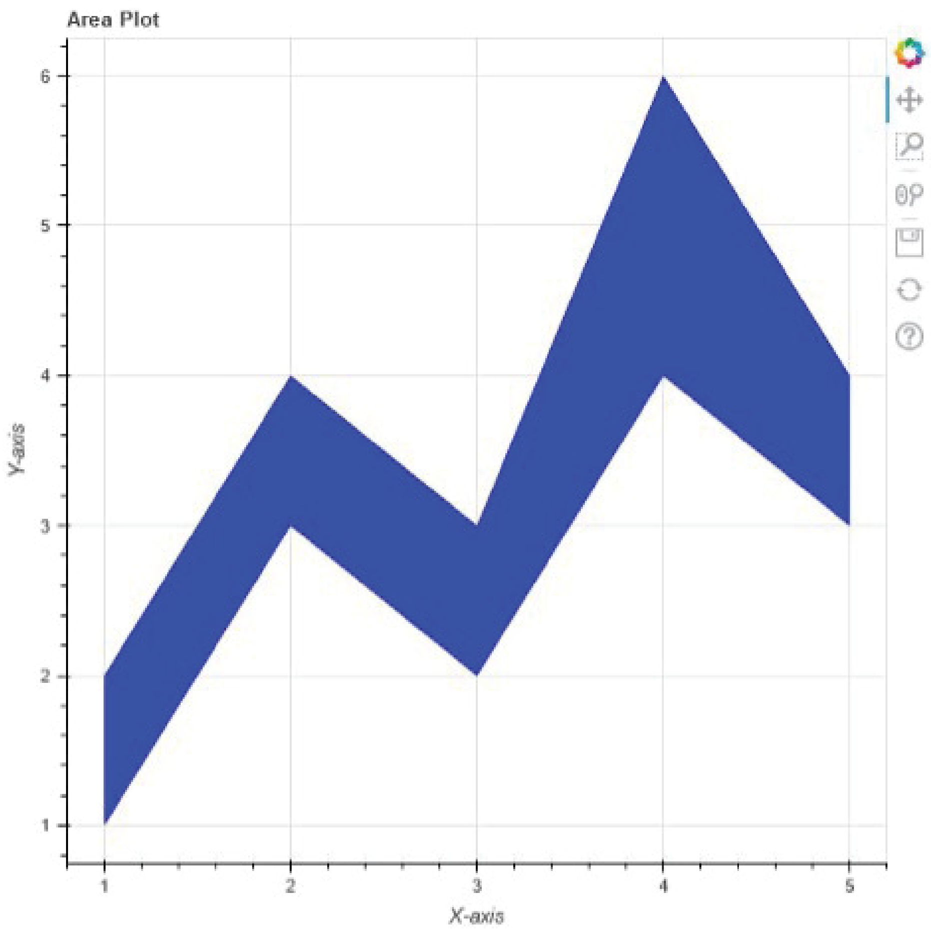
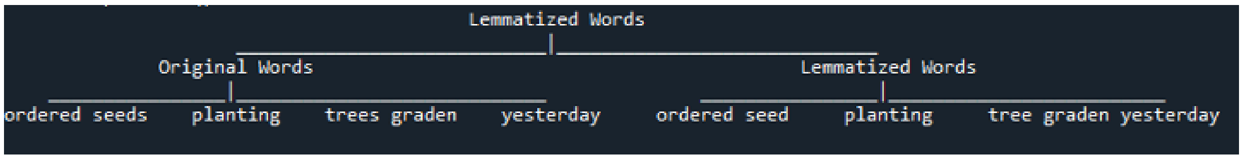


Figure 4.14*: Area Chart*

Tree chart

A tree chart or tree diagram is more of a visual data visualization than one for detailed numerical data. The main idea of a tree chart is to visualize data as parts of a whole inside a category. For a more complex tree chart, lay out different categories next to each other.

Choose a tree chart when your visualization doesn’t depend on granular numerical data. Better yet, if the data is hierarchical, a tree chart does a good job. Please refer to the following figure:



```
                                          Lemmatized Words
                    ____________________________|____________________________
             Original Words                                                  Lemmatized Words
    ________________|____________________________                  ________________|_________________________
ordered seeds     planting     trees graden     yesterday     ordered seed     planting     tree graden yesterday
```

Figure 4.15: *Tree Chart*

Flowchart

A flowchart is a highly versatile type of data visualization. Use a flowchart to visually describe a process, hierarchical data of items or persons, and even a mind map for brainstorming strategy.

The best part about flowcharts is that they are easy to customize for any project — for example, a training manual or strategy proposal. Inside a pitch deck or welcome kit, a flowchart can visualize the hierarchy of the company's teams.

Visually, flowcharts start with one header shape that branches out to a series of shapes and lines that connect. Please refer to the following figure:

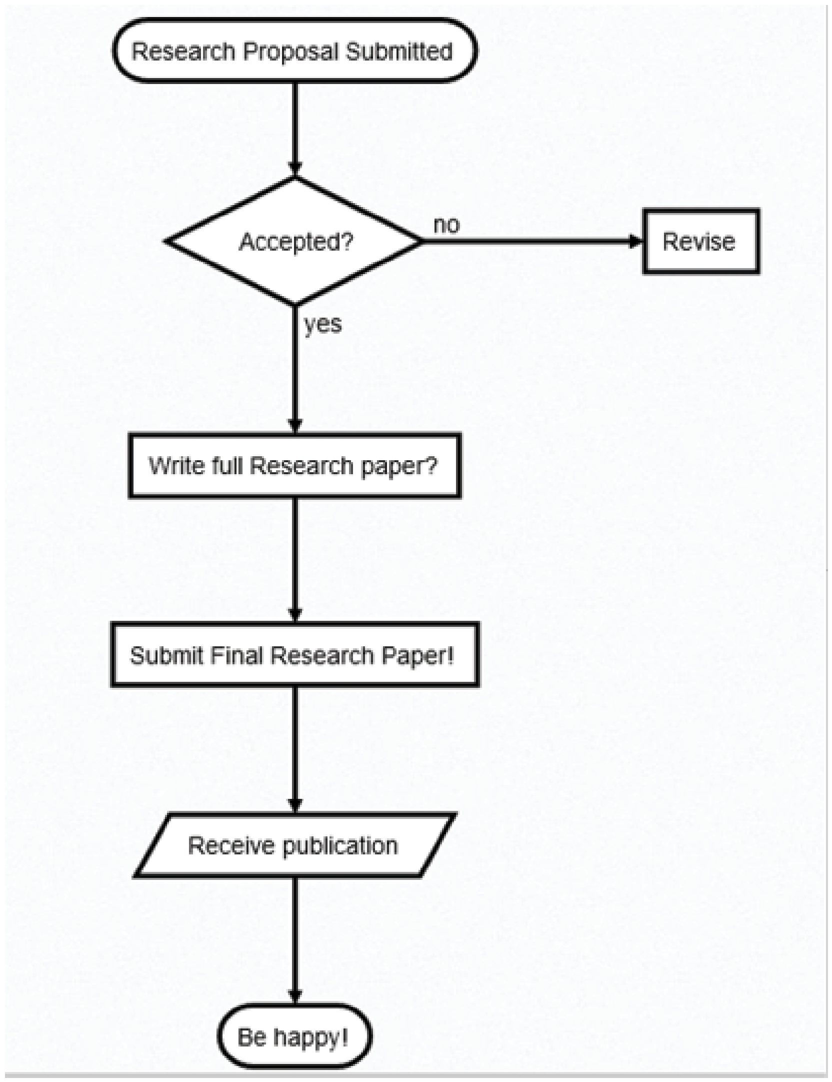
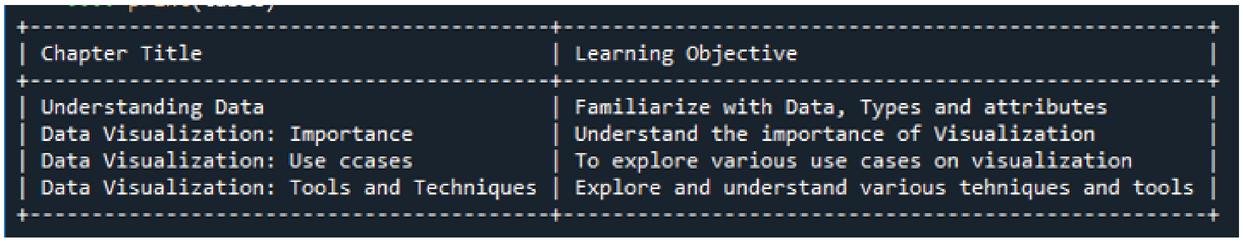

Research Proposal Submitted
Accepted?
no
Revise
yes
Write full Research paper?
Submit Final Research Paper!
Receive publication
Be happy!

Figure 4.16: *Flowchart*

Table

Tables are like mini spreadsheets and show data in rows and columns. Use a table to display pricing for a service, comparative features of a product, school reports, and more.

This data visualization type fits well inside visual documents like reports, proposals, and training manuals. For a unique take on a table visualization, use dots or icons to represent yes or no data about a specific category. Please refer to the following figure:



```
+------------------------------------------+---------------------------------------------------+
| Chapter Title                            | Learning Objective                                |
+------------------------------------------+---------------------------------------------------+
| Understanding Data                       | Familiarize with Data, Types and attributes       |
| Data Visualization: Importance           | Understand the importance of Visualization        |
| Data Visualization: Use ccases           | To explore various use cases on visualization     |
| Data Visualization: Tools and Techniques | Explore and understand various tehniques and tools |
+------------------------------------------+---------------------------------------------------+
```

***Figure 4.17**: Table*

Geospatial map and choropleth map

Maps are the ideal visualization for any data that has to do with geolocation. A data map has many uses, from country-by-country information to detailed regional analysis. A choropleth map is based on a geographic map but has a specific purpose. A choropleth map is a geographical representation of statistical values according to region. For example, population density in a country is visualized by state.

Values are divided into equal portions and given color each. The map's associated areas are then color-coded to correspond to their values. These visualizations are ideal for non-profit organizations, businesses in the healthcare industry, or anybody who wants to visualize statistical data of a specific geographic area. Please refer to the following figure:

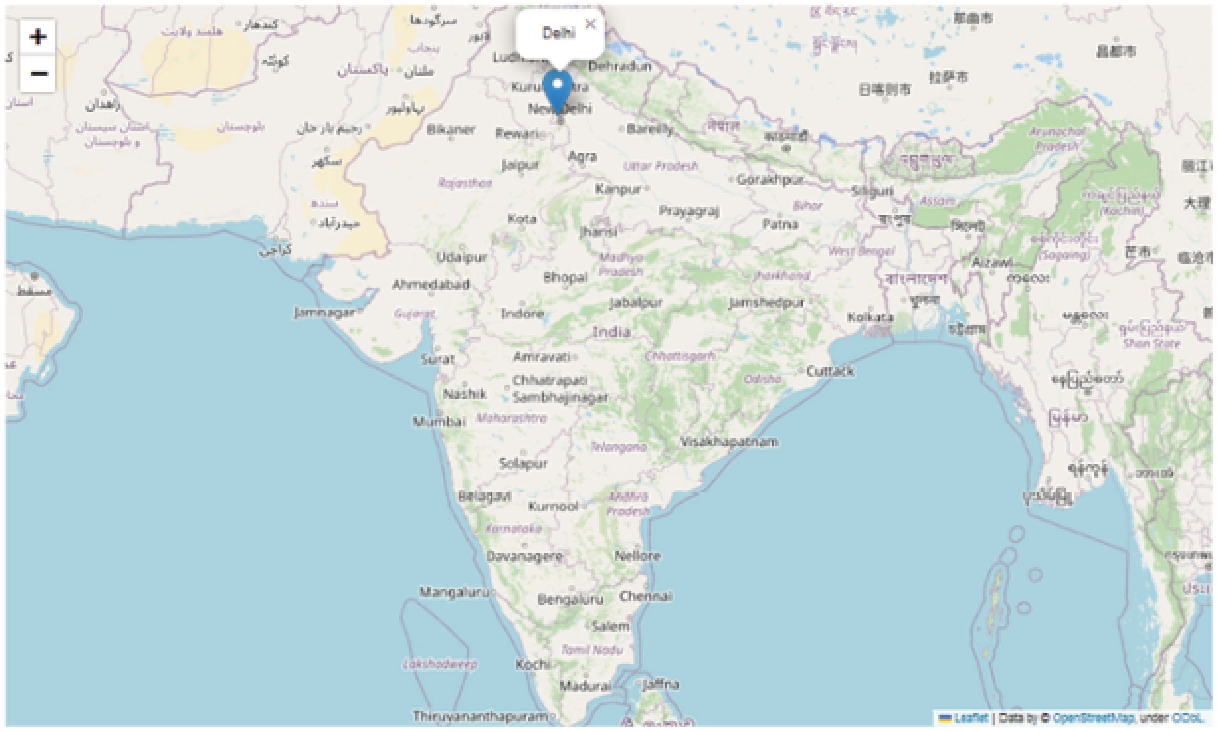


Figure 4.18: *Choropleth Map*

When working with huge data sets, a choropleth map is ideal for interactive data visualization. Popup data labels providing details about the data being used can be assigned to each colored section.

Percentage bar

A progress or percentage bar is a simple data visualization type used to display a percentage value. A percentage bar chart is a specific type of bar chart that represents the relative proportions or percentages of different categories or groups. In a percentage bar chart, the height of each bar represents the percentage value of a category relative to the total. These come in handy when creating an informational infographic or progress report. Since percentage bars are so small, they work well as a group. The main difference between a percentage bar chart and a regular bar chart is that the y-axis of a percentage bar chart represents percentages (ranging from 0% to 100%), while a regular bar chart typically represents the actual values of the data being plotted.

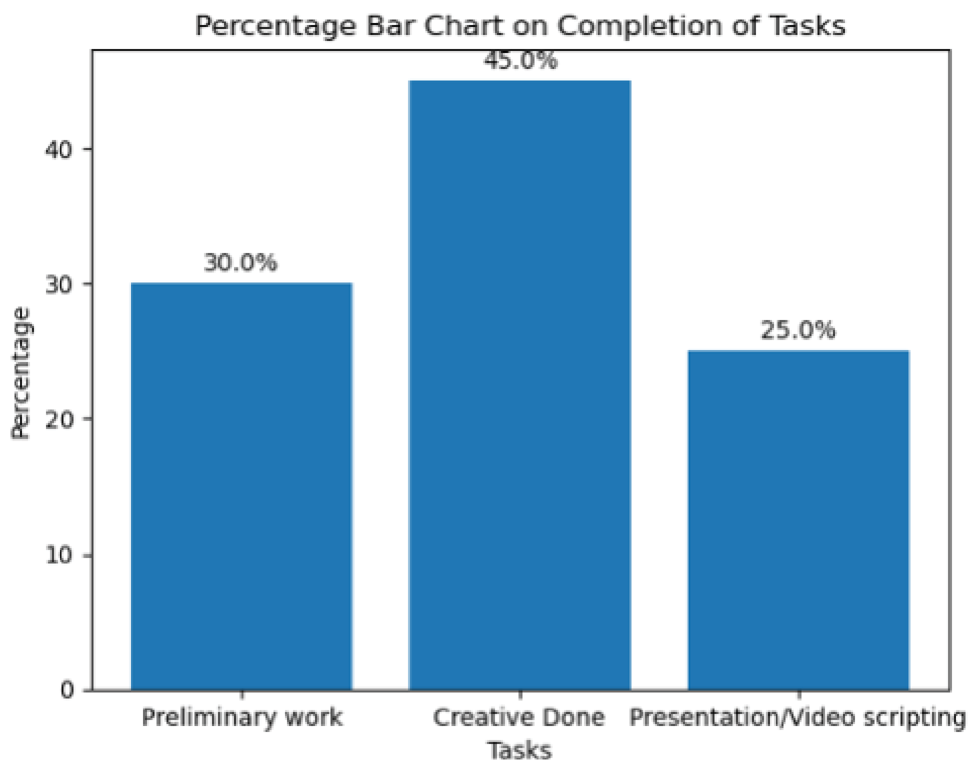


Figure 4.19: *Percentage Bar*

In a percentage bar chart, the bars may not have the same height or total length as in a regular bar chart, as the height of each bar is proportional to the percentage it represents within the whole dataset.

Radial wheel

Another data visualization type for percentage values is the radial wheel. A radial wheel chart (also known as a radial bar chart or a circular bar plot) displays data as bars that extend radially from the center of a circle. Each bar represents a category or data point, and its length or position from the center represents the value or magnitude of the data. Radial wheel charts are often used to show cyclic or periodic data, such as data grouped by days of the week, months, or any other cyclical pattern. Use a radial wheel for infographics, social media visuals, blogs, statistical reports, and more. Though it looks like a donut chart sometimes, donut charts emphasize the relative proportions of different categories within a whole, while radial wheel charts focus on visualizing values or magnitudes in a cyclical or radial manner.

Customize the radial wheel with the colors in your project and personalize the way the values are presented. Like percentage bars, radial wheels are great for group layouts.

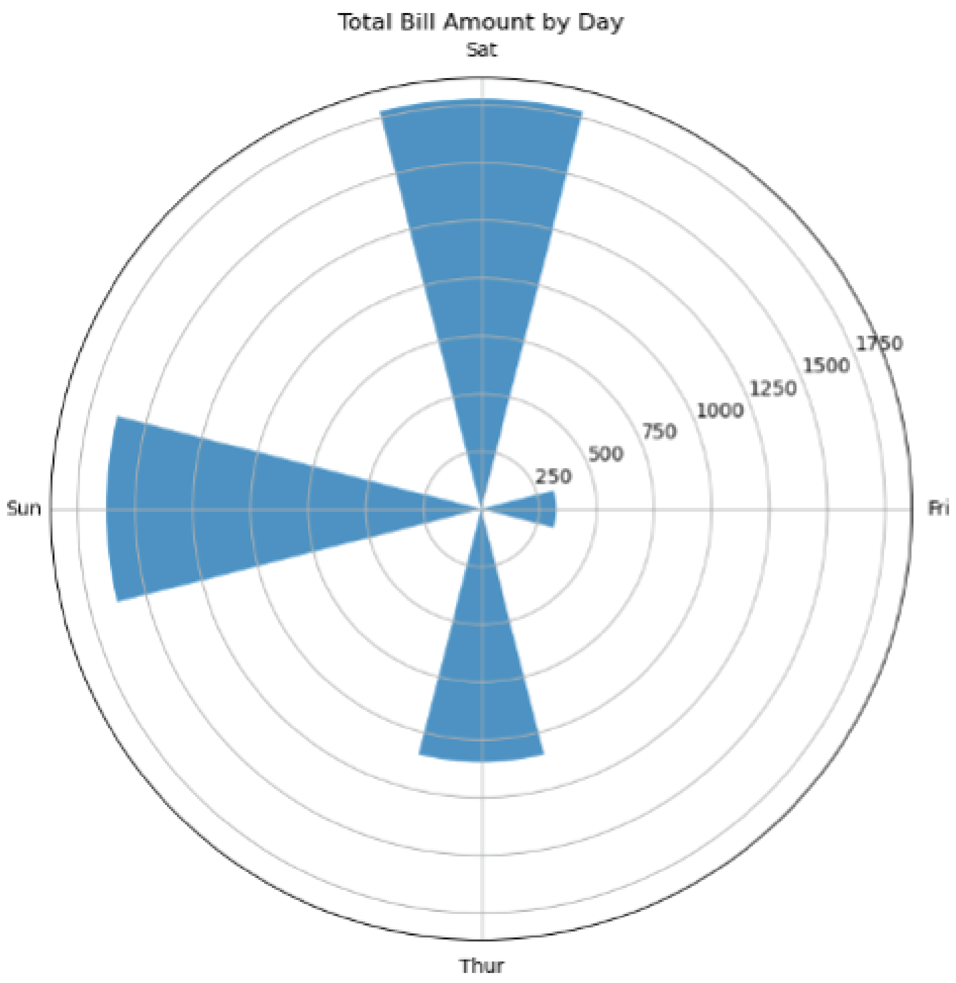


Figure 4.20: *Radial Wheel/Circular bar plot*

Concentric Circles

A concentric circle data visualization is like a line chart on a circular axis. Each category or data item is a circle in the chart, and each circle has its color and is plotted along the circular axis according to the data. Also, the circles are arranged concentrically. This visualization can be particularly valuable for representing hierarchical structures or order, such as organizing individuals within a neighborhood, city, state, or nation. By using concentric circles, the relative positions and relationships between entities can be easily displayed. For instance, larger circles can represent higher-level entities like a nation, while smaller circles nested within them can signify lower-level entities like states or cities. This hierarchical arrangement helps to visualize the organization and order of different levels within a system, facilitating better understanding and analysis of complex structures.

For an easy-to-read chart, there should be no more than six concentric circles:

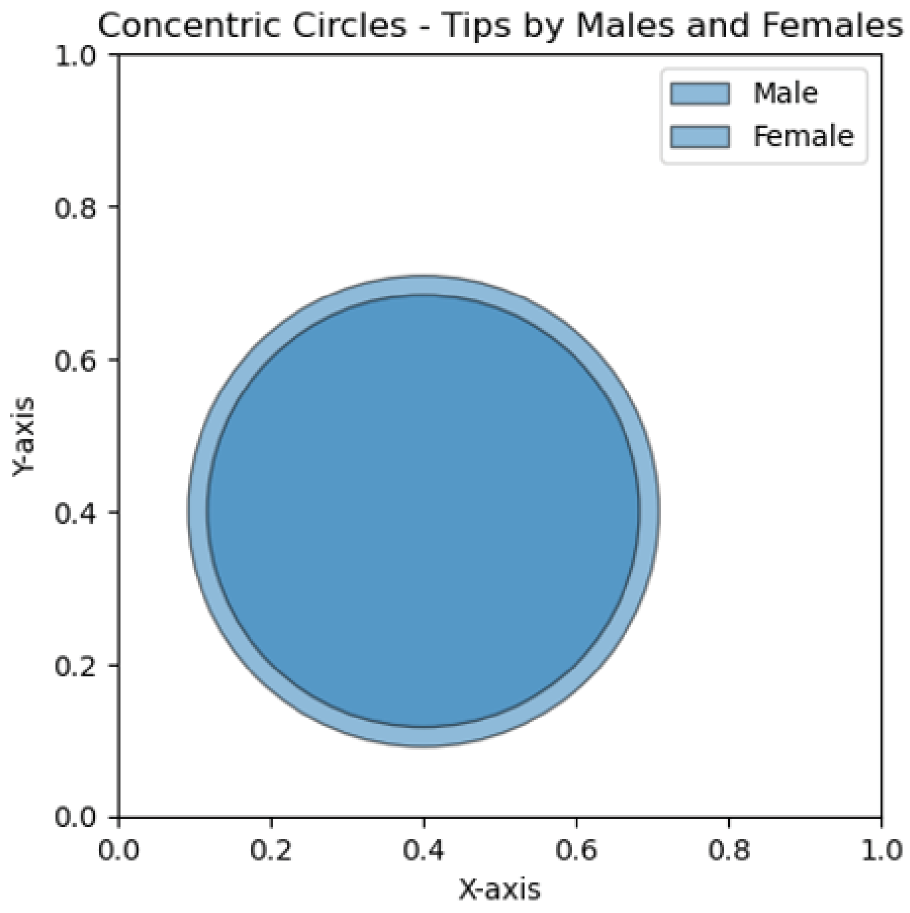


Figure 4.21: *Concentric Circles*

Gantt chart

Gantt charts are based on horizontal bar graphs but are different in a big way. In a Gantt chart, it’s not about how the data changes over time but rather how long it takes to complete over a specific range of time.

Each item on the chart is represented by a rectangle that stretches from left to right. Each one has a different size, depending on how long each task takes to complete.

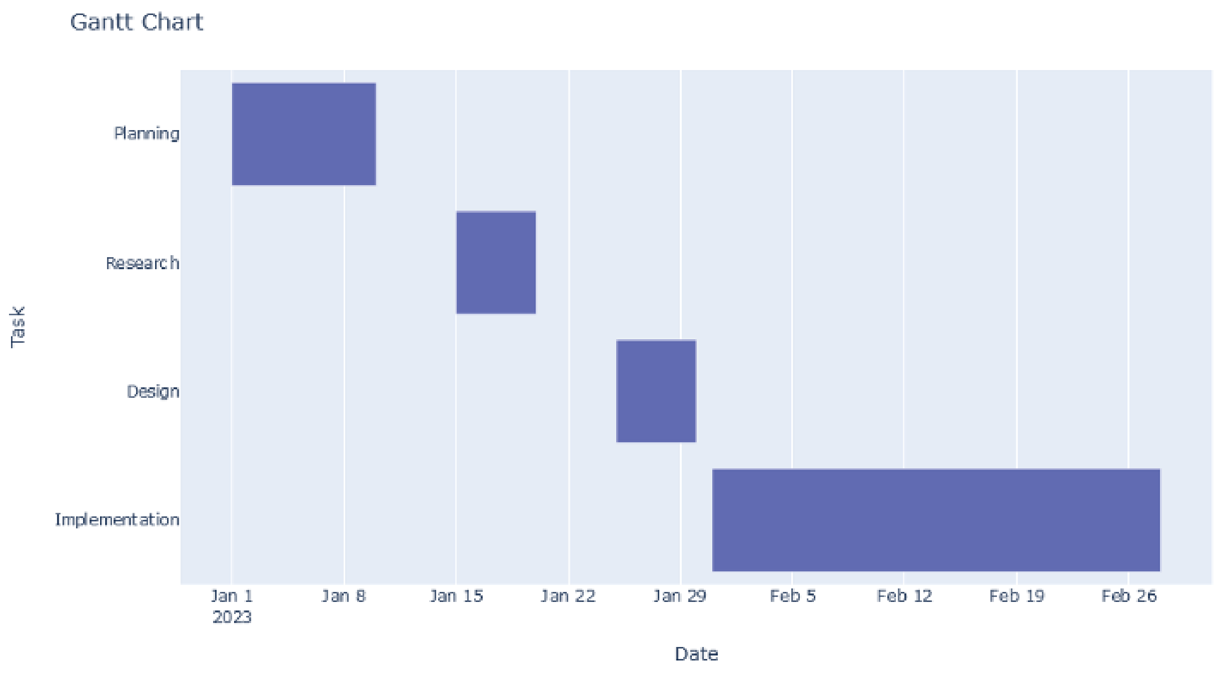


Figure 4.22: *Gantt Chart*

Network diagram

A circuit diagram is a type of flowchart that visualizes concepts like technical circuits, network setups, and other technical connections. These are generally simply designed diagrams without much fanfare. It shows the interconnection between a set of entities. Each entity is represented by vertices/nodes. Connection between nodes is represented through edges:

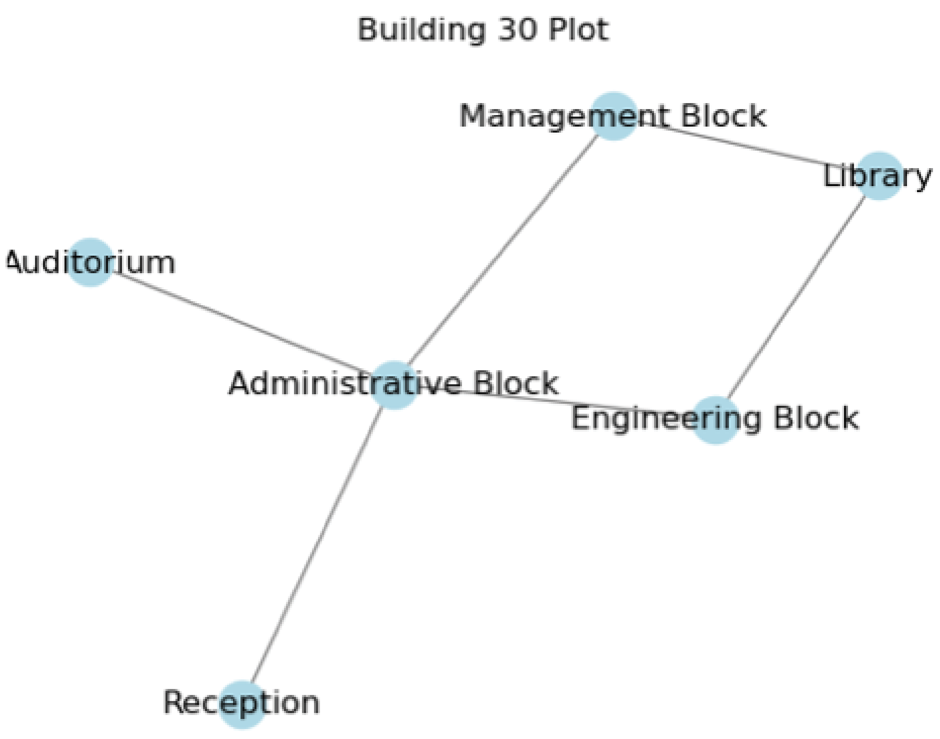


Figure 4.23: *Network Diagram*

Network diagrams are versatile tools used in a wide range of fields to visually represent relationships and connections between entities. In social networks, they help depict friendships, collaborations, and interactions among individuals, enabling the identification of key influencers and communities. In communication networks, network diagrams visualize information flow within organizations or computer networks, aiding in the optimization of communication pathways and identification of critical points. In supply chain management, these diagrams model suppliers, manufacturers, distributors, and customers, providing insights into dependencies, logistics optimization, and the impact of disruptions. For project management, network diagrams such as PERT or Gantt charts are essential for scheduling and tracking tasks, highlighting dependencies and critical paths. Biological networks utilize these diagrams to represent gene regulatory networks, protein-protein interactions, or metabolic pathways, aiding in the understanding of complex biological systems. In web analysis, network diagrams reveal website structures and hyperlinks, facilitating web analytics and optimizing website design. Lastly, transportation networks benefit from network diagrams for traffic flow analysis, route optimization, and infrastructure planning.

Timeline

Timelines are visualizations that show events that have happened or will happen over a specific period. Timelines work great in vertical layouts and horizontally on one presentation slide or several consecutive ones. Use this data visualization type for informational reports about topics with a backstory or for visualizing a company's growth story. Alternatively, use a timeline to explain a plan or objective for a project:

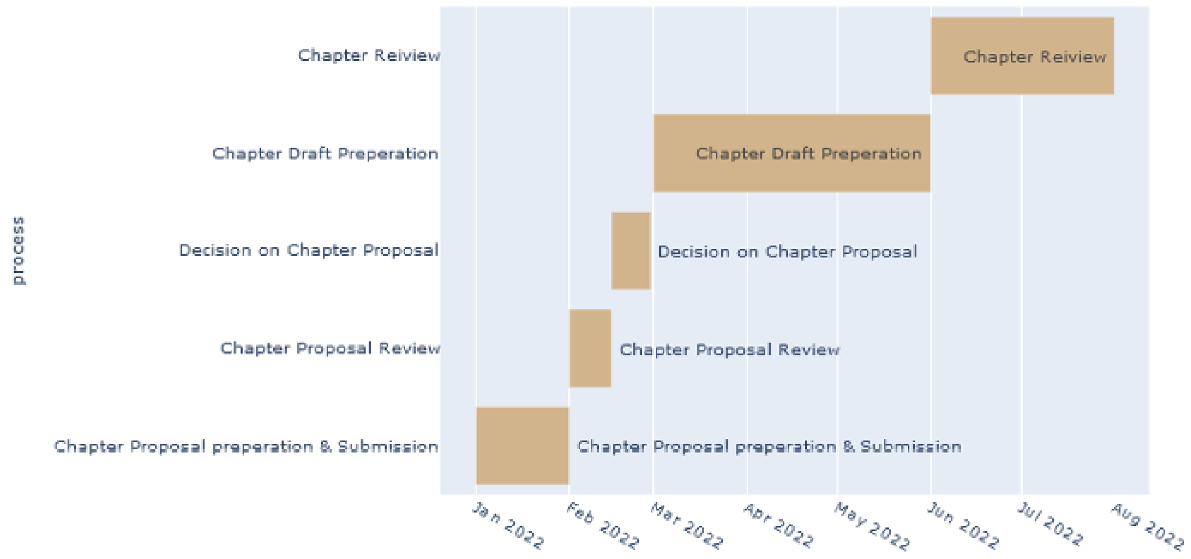


Figure 4.24: *Timeline*

Venn Diagram

A Venn diagram is a data visualization type that aims to compare two or more things by highlighting what they have in common. The most common style for a Venn diagram is two circles that overlap. Each circle represents a concept, and the area that connects them is what the two have in common. Venn diagrams can have up to four or five concept circles where the combined areas show what's in common between them:

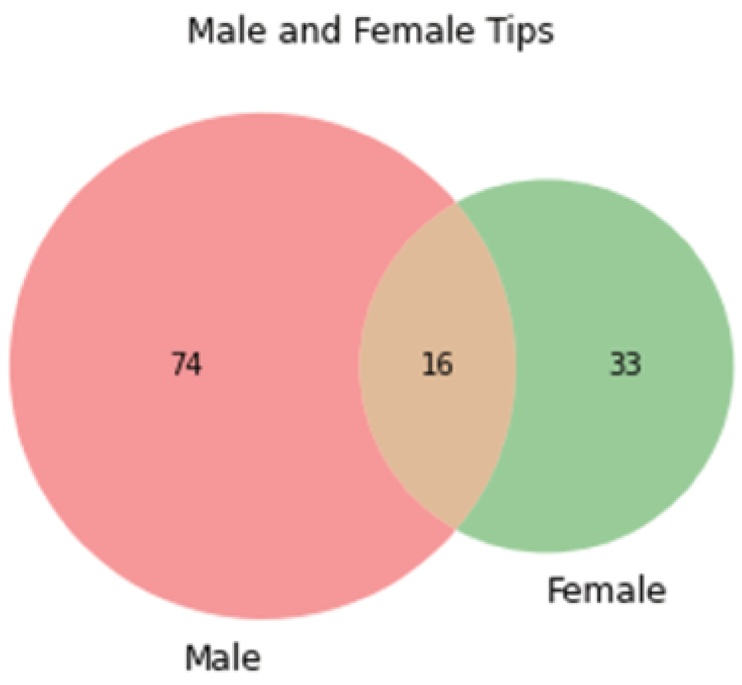


Figure 4.25: *Venn Diagram*

Using more than three or four circles or shapes in a Venn diagram gets very complicated.

Histogram

A histogram is similar to a bar graph but has a different plotting system. Histograms are the best data visualization type to analyze ranges of data according to a specific frequency. They're like a simple bar graph but specifically to visualize frequency data over a specific period.

Histograms can only be vertical, different from how bar charts can be both vertical and horizontal.

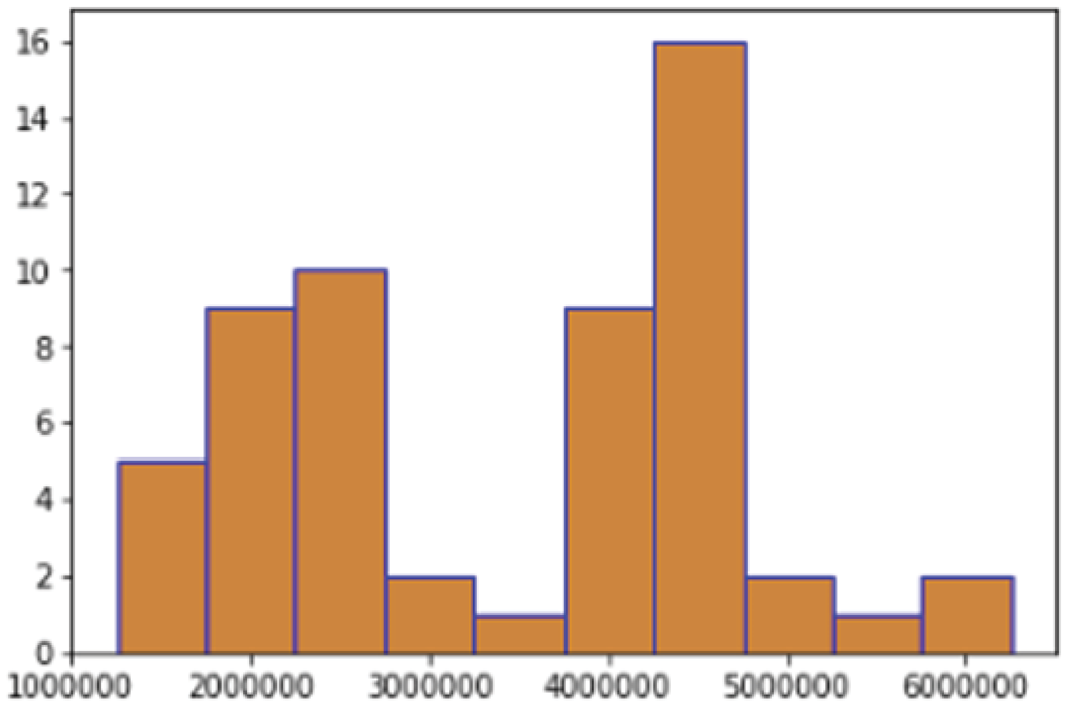


Figure 4.26*: Histogram*

Mind map

A mind map is another data visualization type that helps brainstorm and organize ideas. Visually, a mind map is a web of shapes organized by concept and connected in order of hierarchy. A mind map can be small with only a few connected shapes or extremely large, with many shapes branching out from one or two main ideas. They are graphical representations that allow individuals to visually capture, explore, and present ideas, concepts, and relationships. Mind maps are often used for brainstorming, problem-solving, planning, note-taking, and knowledge organization.

The main components of a mind map include a central topic or idea, which is placed at the center of the map, and branches or spokes radiating outwards from the central topic. These branches represent subtopics or related ideas, and they can further extend into additional subtopics or details:

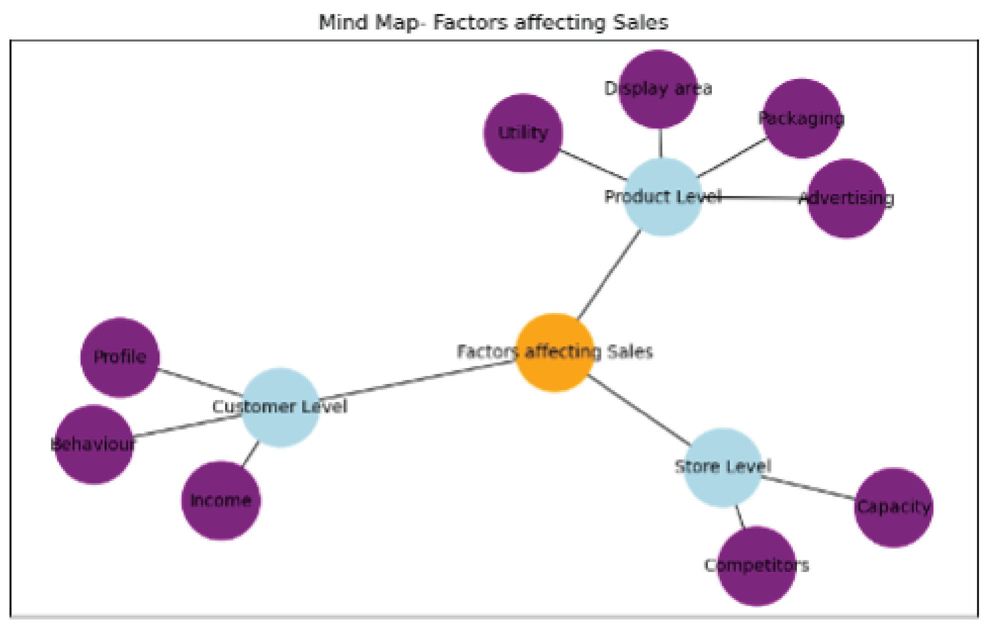


***Figure 4.27**: Mind Map*

Dichotomous key

A dichotomous key is another type of flowchart visualization whose purpose is to help with decision-making. They are typically presented as a series of questions with two possible answers (but there can be three or four depending on the key's length and complexity). Each question narrows down the choices by eliminating one of the possibilities. Usually represented in a linear format, where each question leads to the next until a final identification is made. As you answer question after question, you move along the flowchart toward the appropriate answer.

Dichotomous keys are used widely in scientific education; they help classify organisms by answering questions about their characteristics. You can represent a dichotomous key using a flowchart as well. Each question or decision in the dichotomous key can be represented as a diamond-shaped decision box in the flowchart, with arrows indicating the flow based on the user's response (refer to the *figure 4.16* flowcharts). Dichotomous keys are primarily used in the field of taxonomy and biology to identify and classify organisms based on their characteristics. However, the concept of dichotomous decision-making can be applied in data science as well, particularly in the context of data classification and feature selection. In data science, dichotomous decision-making can be used to build decision trees, which are a popular machine learning algorithms. Decision trees are constructed by recursively partitioning the data based on binary decisions at each node, similar to the process of a dichotomous key.

PERT chart

Another style of data visualization based on the tried-and-true flowchart is available. Circuit diagrams and process maps are combined in PERT charts. Following each item as a process is a premise behind a PERT chart. The previously connected shape must be completed before the next one can be completed unless otherwise specified in the chart.

An effective PERT chart uses different shapes or colors to represent each step's specific characteristics:

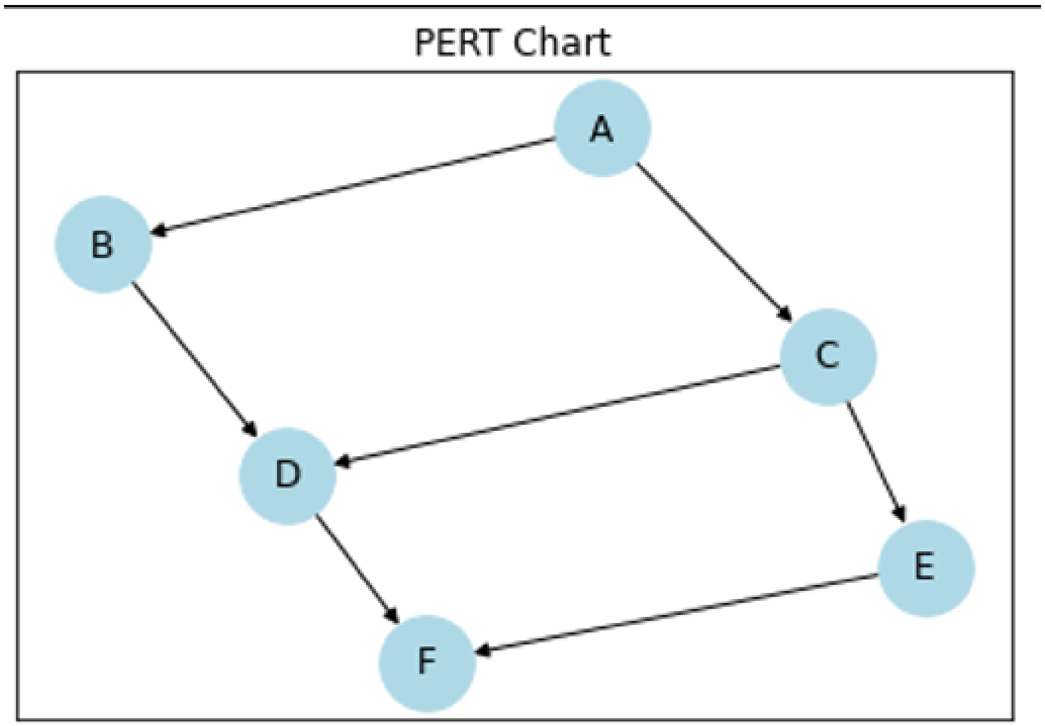


Figure 4.28: *Pert Chart*

Box plot

Box plots are used to show distributions of numeric data values, especially when you want to compare them between multiple groups. They are built to provide high-level information immediately, offering general information about a group of data’s symmetry, skew, variance, and outliers.

Median (Q2/50th percentile): The middle value of the data set

First Quartile (Q1/25th percentile): The middle number between the smallest number (not the “minimum”) and the median of the data set.

Third Quartile (Q3/75th percentile): The middle value between the median and the highest value (not the “maximum”) of the dataset.

Interquartile Range (IQR): 25th to the 75th percentile, the distance between Q3 and Q1 is known as the interquartile range (IQR).

Whiskers: IQR plays a major part in how long the whiskers extending from the box are. Each whisker extends to the furthest data point in each wing, which is within 1.5 times the IQR.

If the data do not extend to the end of the whiskers, then the whiskers extend to the minimum and maximum data values

Outliers (shown as a bubble beyond the whisker): An outlier is defined as a data point that is located outside the whiskers of the box plot.

“maximum”: Q3 + 1.5*IQR

“minimum”: Q1 -1.5*IQR

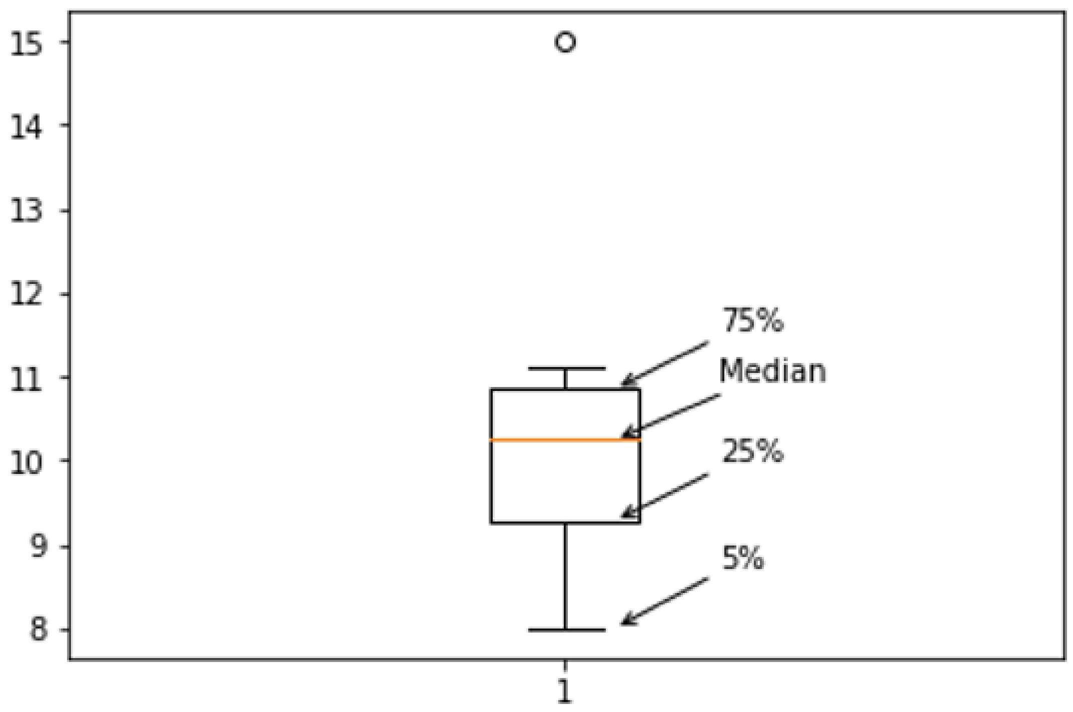


Figure 4.29: *Box Plot*

Heatmap

A heatmap is a type of data visualization that represents data points in a tabular format using colors to represent the data values. It is often used to show the correlation between two or more variables, where each cell in the table represents a combination of values for the variables. The color of the cell is determined by the magnitude of the data value, with darker or brighter colors indicating higher or lower values, respectively.

Heatmaps are useful for quickly identifying patterns and trends in large data sets. They allow users to quickly visualize the magnitude and distribution of the data and to identify clusters or outliers. They can also be used to identify correlations between variables and detect patterns of missing data.

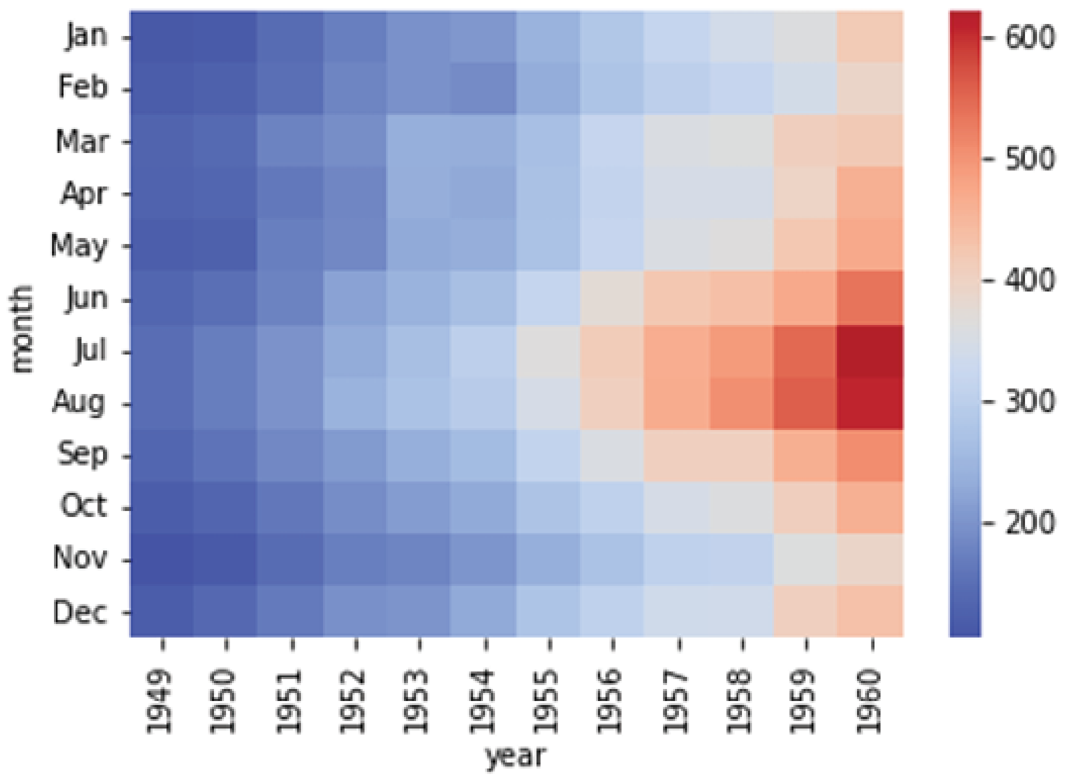


Figure 4.30: *Heatmap*

Heatmaps are widely used in fields such as biology, finance, marketing, and social science. In biology, heatmaps are often used to represent gene expression data, where each row represents a gene, and each column represents a sample, and the color of each cell indicates the level of expression of the gene in that sample. In finance, heatmaps are used to represent the performance of stocks or other financial instruments, where each row represents a stock, and each column represents a time, and the color of each cell indicates the percentage change in the value of the stock during that time.

In addition to their use in tabular data, heatmaps can also be used to represent geographical data, where the color of each region or point on a map is determined by the data value for that location. This type of heatmap is commonly used in weather forecasting, traffic analysis, and disease mapping.

Most popular data- visualization tools and techniques

Some of the most popular data-visualization tools and techniques have been discussed in the following:

Tableau

Various data sources with it and quickly produce visuals.

The first version was the Tableau desktop. It doesn't produce dynamic maps; instead, it is designed to produce static visualizations that can be published on one or more web pages.

With some restrictions, Tableau Public is the free distribution of desktop applications.

Several tools enable you to learn how to use Tableau, but it takes time and practice to become proficient. Tableau is the most valuable tool for you to understand and use in your daily tasks as a data scientist.

QlikView

Qlik View is not just another data visualization tool. It is a data discovery platform that empowers users to make faster, more informed decisions by accelerating analytics, revealing new business insights, and increasing the accuracy of results.

It has been an intuitive software development kit that has been used in organizations around the world for many years. It can combine various kinds of data sources with visualizations in color-coded tables, bar charts, line graphs, pie charts, and sliders.

It has been developed on a “drag and drop” visualization interface**,** allowing users to easily add data from many different sources, such as databases or spreadsheets, without having to write any code. These characteristics also make it a relatively easier tool to learn and grasp.

Microsoft Power BI

Microsoft Power BI is a data visualization tool that is used for business intelligence type of data. It is and can be used for reporting, self-service analytics, and predictive analytics.

Furthermore, it provides an end-user platform to create reports and share insights with others in their organization. It acts as a centralized repository for all your business data which can be accessed by all your business users.

On top of all this, Power BI also provides integration with other SaaS products like Google Analytics, Mail Chimp, Office 365, and so on.

Through such integrations, the reports created can be shared within the organization, thus making it a very important tool for organizations looking for a centralized data reporting system.

Data wrapper

A data wrapper is an online data visualization tool that can be used in various contexts. It is very easy to use, and it has a clean and intuitive user interface.

The data wrapper allows users to create charts and maps directly in the browser by uploading their data files. The charts and maps created in the Data wrapper are responsive and designed for all kinds of devices, so readers will be able to view them on any device that they are using.

Data wrapper is free for everyone; however, there are certain limitations in the free version.

For example, it only lets you upload 500 rows of data and one sheet (or 5MB) of data at a time. The available chart types include Line, Bar, Area, Column, Pie, and Scatter. The data files that can be uploaded are.csv, tsv, or .txt files.

Plotly

Plotly is a data visualization tool that is used to create interactive graphs, charts, and maps. You can also use Plotly to create a visualization of a dataset and then share the link of that visualization with your readers on social media or your blog.

Plotly graphs can be shared easily because they are interactive and have a special URL. By hovering over data points and examining information about them, readers can investigate how you created them.

It is ideal for sharing both interactive plots and datasets with your audience because readers can explore all the data interactively rather than attempting to comprehend your code.

With Plotly's user-friendly interface, you can generate stunning graphs faster than ever before. You can select from a wide range of plots and maps thanks to Plotly's extensive collection of open-source visualization types.

Sisense

Sisense is a data visualization tool that allows you to easily create interactive visualizations from your data. With Sisense, you can quickly and easily create extensive, informative dashboards that will help you understand your data better.

It has a very powerful yet simple and intuitive interface that allows you to drag and drop your data onto the canvas and create visualizations with a few clicks of a mouse.

It is also fully integrated with several BI tools, such as Microsoft Excel, BIRT, Pentaho, QlikView, and Tableau.

Sisense utilizes multi-dimensional in-memory technology that is designed for Big Data. It also has an embedded artificial intelligence engine with predictive analytics, allowing you to easily visualize data trends and discover hidden patterns in your data.

Excel

Microsoft Excel is a tool for data visualization that has a simple interface, so using it doesn't have to be challenging.

In Excel, data may be shown in a variety of ways. One of these is using **scatter plots**, which show the connection between two datasets you want to compare. Additionally, you can examine the relationships between various variables to decide whether or not they are related.

For objectives like market research or financial planning, many data analysts utilize scatter plots to analyze statistical, scientific, medical, and economic data.

Zoho Analytics

You can simply generate unique reports and dashboards using the data visualization and reporting tool Zoho Analytics.

Zoho Analytics enables you to:

- Quickly create custom reports and dashboards with drag-and-drop ease.
- Get insights into your data with interactive charts and graphs.
- Share reports and dashboards with colleagues or customers in just a few clicks.

Apart from this, it is part of the Zoho Office Suite, which also includes Zoho Writer, Zoho Sheet, and Zoho Show. You can use Zoho Analytics to report on data from any of these applications, as well as from external sources such as MySQL, SQL Server, Oracle, and Google Sheets.

Other available tools and techniques

The following are some other available tools and techniques:

Infogram

Charts, reports, and maps can be produced using Infogram, another well-liked option.

The ability to produce infographics—hence the name—sets Infogram apart from the other tools on this list, making it particularly well-liked among creative workers. The application also features a drag-and-drop editor, which is beneficial for beginners.

For use online, visualizations can be saved as HTML or picture files and GIFs to be integrated with reports and other documents. Infogram features tiered pricing, ranging from a free to enterprise-level edition, like the majority of the other tools on this list.

Fusion charts

With over 1000 different types of maps and 150 different types of charts and graphs, Fusion Charts is a popular JavaScript-based data visualization application for displaying data in a visual style. Instead of limiting data visualization to plain images, this application strives to provide the dashboards that business analysts prefer. This drives up the price of the item and puts a strain on your finances, particularly if you run a small business.

Fusion Charts, however, is also acknowledged as one of the most potent readily available data visualization tools on the market. It is an excellent technique to create a variety of lines, bars, column charts, pie charts, and much more from large amounts of material and data. Each chart does not need to be created from scratch. Using the greatest visualizations on websites is made simple by the pre-existing templates for presenting the data and embedding the images. Check out the iOS and Android mobile versions as well.

Multiple programming languages and frameworks can be combined with Fusion Charts. What if you want services that work across browsers? Simple. The API can be used to support many browsers. Additionally, it provides much more functionality than comparable apps and data visualization tools.

D3.js

To work with data, D3.js makes use of a JavaScript library. One of the open-source data visualization tools may be completely tailored to meet your company's needs. To use this software and produce interactive visualizations, however, you must be familiar with programming languages.

Of course, you can continue utilizing the software by using apps created to assist non-programmers in using data visualization tools created using programming languages. NVD3, which enables D3 JavaScript users to work on reusable charts, is one example of this.

To show the data for business intelligence, Plotly (another data visualization service) includes a Chart Studio software that enables you to make WebGL and other charts.

When big data sets are tied to SVG objects and different software functions, pie charts, bar charts, graphs, and dynamic maps are produced.

The emphasis is on web standards and features that help construct and design responsive and interactive maps. It produces excellent visions and is effective and potent. It is cross-browser compatible and free to use. The source code is available for download, and you can modify it to suit your company's needs.

High charts

High Charts is yet another JavaScript-based charting toolkit and data visualization tool aimed at programmers and developers. Including interactive infographics on websites is convenient (similar to Fusion-Charts). The tool supports responsive websites and offers a selection of charts.

To begin using High Charts, not many modifications or changes are necessary. Working with this data visualization tool is quick, flexible, and requires the bare minimum of technical knowledge. The prepared charts are visually appealing and are filled with insightful data. Although it might be viewed as a benefit, High Charts is a little out of date when compared to the most recent tools and doesn't provide as many alternatives for expanding the charts. Small and new enterprises can benefit from it. To use High Charts paid-for edition, you must purchase a license. To embed charts on the website, schools, individuals, and NGOs can access the free version.

But it's also one of the most popular online tools for data visualization. Try High Charts if you want to create thorough visualization reports.

Fine Report

Fine Report was created for use at the corporate level. It mixes data entry and data visualization and is entirely built-in in Java. Interactive visualization is provided via the BI reporting and dashboard.

Fine Report provides excellent relief if you are concerned that you will need to study programming to master visualization approaches. This free data visualization tool doesn't require coding, in contrast to others that are based on languages. Error risk is reduced to a minimum.

Yes! To utilize Fine Report, you don't need to know how to code. You only need to drag and drop objects into the desired positions to build dashboards or create data visualizations. Your projects will have the exact appearance that you desire. Utilizing Fine Report has the added benefit of being able to connect to practically any data source. This platform is ideal for you if you work with several sources.

The tool may be easily customized. There's more, too. Even beginners can use Fine Report easily because of its user interface's resemblance to Excel. The company provides the personal version without charge. Small businesses can launch with more affordable plans.

Use-case of commonly used tools

Let us now discuss the use cases of some commonly used tools:

Used tools on Facebook

Techniques for exploring a dataset using data visualization are quite helpful. The data science ecosystem uses a wide range of visualization types. The qualities of the data and variables determine what is most appropriate for a certain task.

This January, Facebook released Hi-Plot, a Python package to allow the visualization of a high-dimensional data table. It is renowned for its clever interactive parallel plot in particular. Hi-Plot specializes in visualizing high-dimensional data. It can handle datasets with numerous variables, making it easier to understand complex relationships and patterns in the data. By visualizing multiple dimensions simultaneously, Hi-Plot helps users uncover hidden connections and identify important features. It offers different plot types, color schemes, labels, and markers, enabling users to create aesthetically pleasing and informative visualizations.

Used tools in Google

Google offers a free tool called Google Data Studio if you're new to data visualization. You can quickly customize the charts, graphs, and reports in Data Studio, and it offers a wide range of options for building shared reports that are all readily available.

Benefits of using Data Studio include:

- Built-in connections to over 200+ data sources eliminating the need to schedule periodic data to refresh for reports.
- No licenses, and it's free for anyone.
- No downloads are necessary since everything is online.

Used tools on Twitter

To see how two terms are associated in a group of tweets, a technique known as the Twitter Venn diagram can be employed. Tweepsmaps is an additional tool that analyzes and displays the network of Twitter accounts. It displays the percentage-based distribution of followers across the map. Additionally, the app lists inactive users who haven't tweeted in the last three to six months. You may gain insightful knowledge and understand the geographical dynamics of Twitter connections using TweepMaps.

TweepMaps allows you to explore the geographic distribution of Twitter data by visualizing tweets and user activity on a map. Discover geographical patterns, activity hotspots, and regional trends in connection with subjects or events.

TweepMaps' contextualization capabilities are what make it so appealing. You may examine tweets in relation to their actual locations to gain a better understanding of how content and geography are related by mapping Twitter data.

Additionally, follow me is a free service that enables users to examine key data about Twitter users. It offers a user-friendly UI. It investigates hashtags, mentions, and themes. Autonomy is a solid platform that offers a unique approach to analysis and visualization, to sum up. Its functions include gathering insightful information, tracking follower growth, backup/exporting, and analyzing tweets, among others.

Used tools on Netflix

Netflix uses DataViz to track the viewing experience of its customers in a specific geographical area. DataViz provides real-time data on the viewing experience. DataViz is a dynamic layout that shows a map view of the performance of a particular show, connection-related issues, if there are any, shows the frequency of viewer login, and so on. The DataViz interface can be customized by each user to suit his or her preference.

DataViz helps Netflix to generate real-time information for decision-making. In light of creating television shows to suit the preference of views, Netflix used DataViz to analyze the audience of British shows and came to the conclusion that viewers enjoyed movies directed by *David Fincher*. This helps Netflix recreate new series directed by David Fincher. Hence, Netflix owes a majority of its growth to the ways data is being analyzed in real-time and how products are couched to meet the viewing experience of people in a geographical area.

Used tools by Intel

Intel uses data visualization tools produced by *Kanjoya Inc*. (which was later acquired by Ultimate Software, an American multinational technology company that provide AI based HCM and workforce management solutions to its customers) to internally analyze the sentiments of its employees. Sentiment analysis is finding trends, negative or positive, in data. Kanjoya Inc. uses a machine learning algorithm to decipher the emotions of its employees.

Intel analyzes the social media post of all its employees to find a pattern in their posts. The pattern helps Intel to determine if their employees are dissatisfied with activities at work or whether they are satisfied with the working environment. Emotions can be deceptive. Therefore, sentiment analysis helps to know exactly how people feel and address it.

Used tools by retails companies

Retail Companies use market basket analysis to determine the buying behavior of their customers. Instead of relying on new customers, Retail companies seek to know the preference of existing customers. The buying history of customers is checked over a period of time; historical buying database. Buying behavior, the of the customer is discovered, helps Retail companies to place the preferred item at the top of the shelf, that way making it visible to all its customers. Market basket analysis has a mathematical backing. This is why eatable and non-eatable products are not lumped into one portion of a Retail shop.

Market basket analysis falls under association clustering in machine learning, where products are recommended to customers based on the product being bought. An example is Amazon. In most cases, after a purchase is made, similar items are recommended with the phrase "*Customers buy these together*." The chances of getting a customer to instantly make a purchase for the recommended product is high, and this help to boost sales. Hadoop is used to analyze the data, and data is collected in the form of receipts.

Conclusion

In this chapter, we learned about various data visualization tools and techniques, ranging from simple charts to more advanced techniques such as heatmaps. Different visualization tools and techniques are used depending on the data being analyzed and the objectives of the analysis. The most popular visualization tools include Python libraries like Matplotlib, Seaborn, bokeh, and Plotly, as well as other software such as Tableau and Excel. The right choice of tool is important to create effective visualizations that can convey complex information in a clear and concise manner, leading to better insights and decision-making.

In the upcoming chapter, we will dig deeper into the Matplotlib library for data visualization.

Points to remember

- Some of the most common types of visualizations include bar charts, line charts, scatter plots, histograms, heatmaps, and pie charts.
- The choice of visualization depends on the type of data being represented, the purpose of the visualization, and the audience for the visualization.
- Bar charts are used to compare different categories of data by plotting bars of different lengths or heights. They are useful for showing changes over time, ranking data, and comparing data across different categories.
- Line charts are used to show trends over time by plotting data points connected by lines. They are useful for showing changes over time, identifying patterns, and highlighting differences between groups.
- Scatter plots are used to show the relationship between two variables by plotting data points on a two-dimensional grid. They are useful for identifying correlations, outliers, and trends in data.
- Histograms are used to show the distribution of a dataset by dividing the data into intervals and plotting the frequency of data points within each interval. They are useful for identifying patterns, trends, and outliers in data.
- Heatmaps are used to show the distribution of data over a two-dimensional grid by using colors to represent the intensity of the data. They are useful for identifying patterns and trends in large datasets.
- Pie charts are used to show the relative proportions of different categories within a dataset by dividing a circle into segments proportional to the data. They are useful for displaying small datasets with clear categories.
- Choropleth maps are used to display data for geographic areas, such as countries, states, or counties. They use color-coding to represent the

intensity of the data. They are useful for showing regional patterns and comparing data across different areas.

- Funnel charts are used to show the different stages of a process or the progression of data from one stage to another. They are shaped like a funnel, with each section representing a different stage of the process. They are useful for identifying areas of improvement in a process and tracking progress toward a goal.
- Gantt charts are used to show the timeline of a project, with each task represented as a bar spanning the duration of the task. They are useful for showing the dependencies between tasks, identifying critical paths, and tracking progress towards a deadline.
- Donut charts are similar to pie charts but with a hole in the center. They are used to show the relative proportions of different categories within a dataset, with each segment representing a different category. They are useful for displaying small datasets with clear categories, and the hole in the center can be used to display additional information.

Questions

1. What are the main categories of data visualizations?
2. How do scatter plots differ from line charts?
3. What are the benefits of using bar charts for comparing data?
4. When is it appropriate to use a pie chart?
5. What are some common charting libraries and tools used for data visualization?
6. What is a choropleth map, and how is it useful for visualizing data?
7. What is a box plot, and what information does it convey about a data set?
8. What is a funnel chart, and how is it used to visualize data?
9. What is a heat map, and how can it be used to visualize the correlation between variables?

10. What are some best practices for creating effective data visualizations?

CHAPTER 5
Data Visualization with Matplotlib

“This is my favorite part about analytics: Taking boring, flat data and bringing it to life through visualization.”

– John W. Tukey

The main objective of this book is to provide knowledge on using python libraries for data visualization.

Data visualization is the process of creating visual representations of data in order to gain insights and better understand the data. It is a powerful tool for exploring and analyzing data, as it allows us to quickly identify patterns and trends. Data visualization can be used to create charts, graphs, maps, and other visualizations that can help us better understand the data. Data visualization can also be used to communicate data-driven stories and insights to a wider audience.

This chapter presents various functions available in the matplotlib library of python which are used for presenting data in graphical representations. Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. With this library you can create and design from basic plots to three dimension (3D) plots. However, before plotting you need to understand the dataset, we will work on exploratory data analysis in the last chapter of this book, wherein we will learn to find the insights of data for better understanding.

In this chapter, you will learn ways to make use of matplotlib library to its best for plotting various plots/charts as mentioned in the previous chapters of this book. For better illustrations, we will try to create synthetic datasets for plotting or we will utilize datasets available at various sources for research purposes. Whichever way, the links/datasets will be made available to you at the end of this chapter. Also steps to install python are available in the annexure at the end.

Structure

In this chapter, we will discuss the following topics:

- Introduction
- Importing Matplotlib and its documentation
- Understanding Figure and subplots
- Basic plots
- Advanced plots
- Saving the plot

Objectives

In this chapter, Matplotlib library is discussed in detail, which is used for plotting the data. Our objective is to introduce the generally used 'plot styles' and 'features' of the Matplotlib library, which are required for plotting the outcomes acquired by the simulations or visualizing the data during various applications based on machine learning, data analytics, business analytics and so on.

Introduction

Data visualization means graphical or photographic depiction of the data utilizing graphs, charts, and so on. The function of outlining information is to imagine variant or program connections between variables. Visualization likewise assists to effectively communicate information to intended individuals. Web traffic signs, ultrasound reports, Atlas book of maps, speedometer of a car, receivers of tools are few instances of visualization that we come across in our day-to-day lives. Visualization of data is successfully made use of in areas like wellness, financing, science, mathematics, design, and so on. In this phase, we will certainly find out how to imagine information making use of the Matplotlib library of Python by outlining charts such as line, bar, scatter with respect to the different kinds of data. For making 2D-plots or figures that are static, animated, or interactive, utilise the Matplotlib package. A package is a python component which can contain other modules or recursively, various other plans. It is the kind of python script that you import in your Python code. Packages help in ensuring the reusability of code. The files in packages that contain python code are called modules. Modules are used to break down large code into smaller and more understandable parts.

Importing Matplotlib and its documentation

Matplotlib is a plotting library for Python used to create 2D graphs and plots by using python scripts. It provides a wide variety of plots and graphs such as line plots, histograms, bar charts, pie charts, scatter plots, stack plots, 3D graphs, and contour plots. It can be used to visualize data in a variety of ways such as line graphs, bar charts, histograms, scatter plots, and heatmaps. Matplotlib can be used to create interactive visualizations, as well as static ones. It is also used to create animations. It can be mounted utilizing the **pip** command from the command prompt:

```
C:\>pip install matploltlib
```

Pip is among one of the most popular as well as widely used bundle administration systems to set up and also take care of software written in Python and located in **Python Package Index** (**PyPI**). Conversely, pip stands for "preferred installer program".

Python 2.7.9 as well as later on (on the python2 collection), and also *Python 3.4* and also later consist of **pip** (**pip3** for Python 3) by default.

You can check the version of pip with the following command:

```
C:\>pip –version
```

Let us check out some convenient commands to use pip.

To install the latest version of a package:

```
C:\>pip install 'PackageName’
```

like we did earlier for installing the matplotlib package.

To install a certain version, type the bundle name followed by the required variation:

```
C:\>pip install 'PackageName== 1.6'
```

To upgrade a currently set up package:

```
C:\>pip install --upgrade PackageName
```

Uninstalling/removing a plan is very easy with pip:

```
C:\>pip uninstall PackageName
```

To access any module or file from a Python package, use an import statement in the Python source file where you want to access it. Import of modules from Python Packages is done using the dot operator (.). Importing modules and packages helps us make use of existing functions and code that can speed up our work:

```
import module1
```

```
import module1, module2
```

Where import is a keyword used to import the module, module1 is the module to be imported. You can also import more than one module at a time. The import statement is used to import all the functionality of one module to another. Must take a note that you can use the functionality of any python source file by importing that file as the module into another python source file. You can import multiple modules with a single import statement, but a module is loaded once regardless of the number of times it has been imported into our file.

For visualization, making use of Matplotlib, we require to import the module first in the script:

```
import matplotlib as mp
```

More specifically we need its '**Pyplot**' component, we can import it with the help of from statement. You can import that specific function/class/variable, by using from <**module_name**> import <**class/function/variable_name**> statement:

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

Or

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

Here, **plt** is taken as an alias or a different name for **matplotlib.pyplot**. As a convention it is taken as **plt**, however you can name it as per your choice. The **pyplot** component of matplotlib contains a collection of features that can be used to work on plots. In the next section of this chapter, we will try to understand the anatomy of the plot in matplotlib.

Understanding figure and subplots

Matplotlib graphs your data on Figures (for example, windows, Jupyter widgets, and so on), each of which can consist of one or more Axes, an area where points could be specified in terms of x-y coordinates (or theta-r in a polar plot, x-y-z in a 3D plot, and so on). The easiest way of creating a Figure with an Axes is using **pyplot.subplots**. We can then use **Axes.plot** to draw some data on the Axes:

A figure is the total window where the outcomes of **pyplot** functions are outlined. A figure includes a plotting area, legend, axis tags, ticks, title, and so on. Refer to Figure *5.1* on anatomy of a plot, this is picked from the matplotlib's official documentation (website):

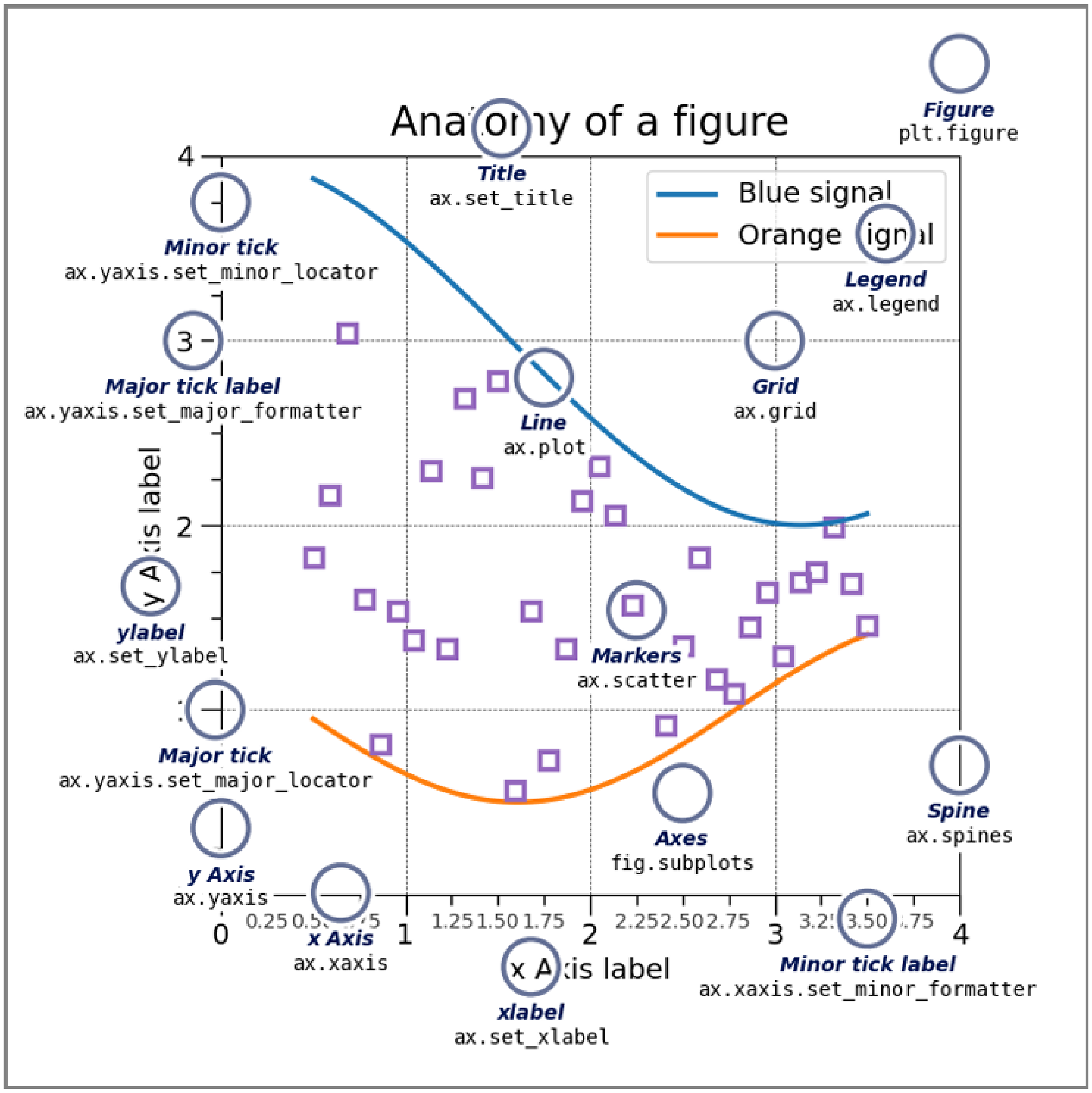


Figure 5.1: Anatomy of a plot in matplotlib (*Source:* [*https://matplotlib.org*](https://matplotlib.org))

One by one you will learn all the functions/methods to create such an informative graph for your data. But before that, let's understand the plot anatomy in detail.

The anatomy of a matplotlib plot refers to the different components and elements that make up a typical plot. The basic anatomy of a Matplotlib plot includes:

- **Figure:** The overall container for the plot. A figure can contain multiple subplots or axes.

- **Axes:** The area where the data is plotted. An axe can contain one or more plots, such as lines, bars, or scatter points.
- **Title:** The title of the plot, usually located at the top of the plot.
- **X-axis and Y-axis:** The horizontal and vertical axes, respectively, that define the coordinate system for the data.
- **X-axis Label and Y-axis Label:** The labels for the X-axis and Y-axis, respectively, that describe the data being plotted.
- **Ticks:** The marks on the X-axis and Y-axis that represent the values of the data.
- **Tick Labels:** The labels for the ticks that display the values associated with the data.
- **Legends:** A key that explains the meaning of different markers or colors in the plot.
- **Gridlines:** The lines that divide the plot into smaller units and make it easier to read the data.

In addition to these basic elements, a Matplotlib plot can also contain other elements such as annotations, color bars, and more. The appearance of a plot can be customized by changing the font size, line width, color palette, and other properties.

As you are aware of the major components of the figure, you are prepared to go ahead with plotting the data. Once data is imported in the script, you can plot a figure using the subplots functions:

```
import matplotlib.pyplot as plt

import numpy as np

x=[1,2,3,4]

y=[2,4,6,8]

fig, ax = plt.subplots()

ax.plot(x,y, color ='blue', linestyle ='dashed', marker='o', linewidth=2.0)

ax.set(xlim=(0,5), xticks=np.arange(1,5),

    ylim=(0,10),yticks=np.arange(1,10))

plt.show()
```

The **plot()** function is used to create a plot in Matplotlib. It takes in the data points as arguments and creates a plot based on the data. It can also be used to customize the plot with various parameters such as linewidth, color, line type, and marker type. To outline x versus y, we can compose plot (x, y). Further you can set the axis details using the **ax.set()** function by placing the required values for various parameters viz. **xlimit (xlim)**, **ylimit(ylim, xticks, yticks** and so on. The result generated is a line graph.

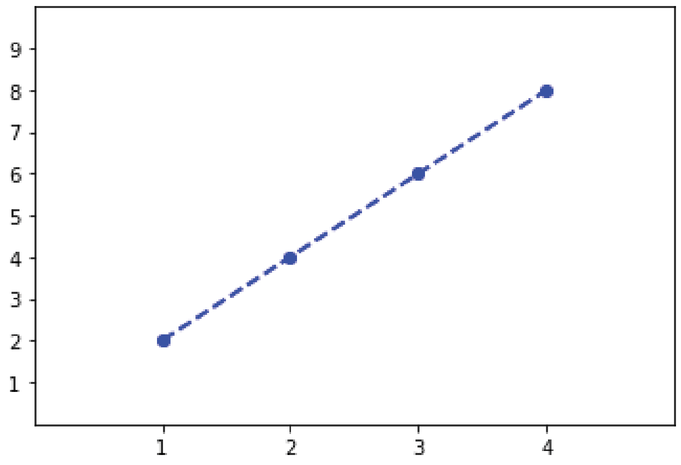


Figure 5.2: *Output of subplot() - a line graph*

This will generate a plot with blue dashed lines and circles as markers.

You can also include the title, **xlabel**, **ylabel** in **ax.set()** function like below:

```
import matplotlib.pyplot as plt

import numpy as np

x=[1,2,3,4]

y=[2,4,6,8]

fig, ax = plt.subplots()

ax.plot(x,y, color ='blue', linestyle ='dashed', marker='o', linewidth=2.0)

ax.set(xlim=(0,5), xticks=np.arange(1,5),

    ylim=(0,10),yticks=np.arange(1,10),

    title="Plot of x versus y",

    xlabel="Values of x", ylabel="Values of y")

plt.show()
```

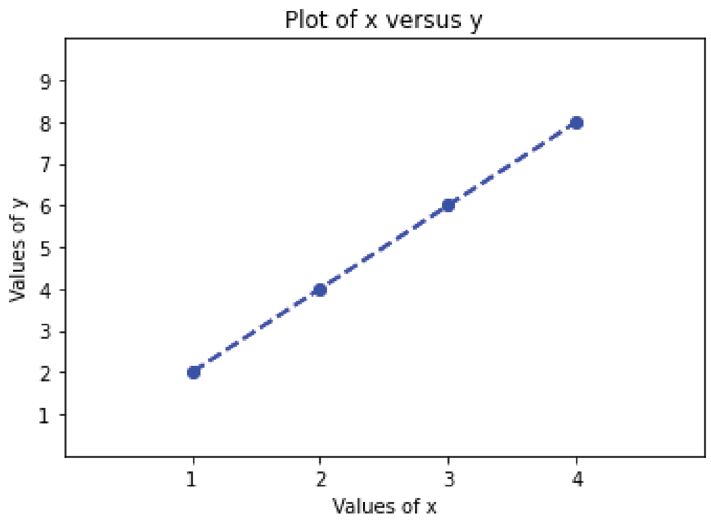


Figure 5.3: *Output of subplot()* **-** *a line graph with title and axis labels included*

Well, there is a lot more to it, let's add facecolor and annotate the plot with the value of intersection point (x=3, y=6). '**facecolor**' is added in the **ax.set()** function, while for including annotations, make use of **ax.annotate()** function:

```
import matplotlib.pyplot as plt

import numpy as np

x=[1,2,3,4]

y=[2,4,6,8]

fig, ax = plt.subplots()

ax.plot(x,y, color ='blue', linestyle ='dashed', marker='o', linewidth=2.0)

ax.set(xlim=(0,5), xticks=np.arange(1,5),

     ylim=(0,10),yticks=np.arange(1,10),

     title="Plot of x versus y",

     xlabel="Values of x", ylabel="Values of y", facecolor="yellow")

     ax.annotate('3,6 point', xy=(3.1, 5.9), xytext=(3.5,5.5),

     arrowprops=dict(facecolor='black', shrink=0.05))

plt.show()
```

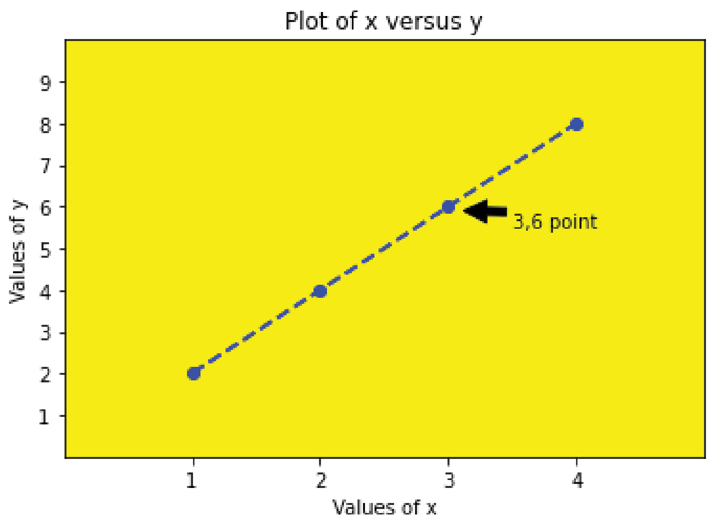


Figure 5.4: *A line graph with face color and annotation included*

With the help of **axx.grid()** function, you can always include a grid in the figure. Also, you can use **tick_parameter** function to change the appearance of ticks:

```
import matplotlib.pyplot as plt
import numpy as np
x=[1,2,3,4]
y=[2,4,6,8]
fig, ax = plt.subplots()
ax.plot(x,y, color ='blue', linestyle ='dashed', marker='o', linewidth=2.0)
ax.set(xlim=(0,5), xticks=np.arange(1,5),
    ylim=(0,10),yticks=np.arange(1,10),
    title="Plot of x versus y",
    xlabel="Values of x", ylabel="Values of y")
    ax.annotate('3,6 point', xy=(3.1, 5.9), xytext=(3.5,5.5),
    arrowprops=dict(facecolor='black', shrink=0.05))
ax.grid(True, linestyle='--')
ax.tick_params(labelcolor='r', labelsize='medium',width=3)
plt.show()
```

Please make note here, that the facecolor has been removed so default white facecolor is displayed in the plot. Additionally, the color of the arrow is changed in this code to red, just to match it with the tick color. Please refer to the following figure:

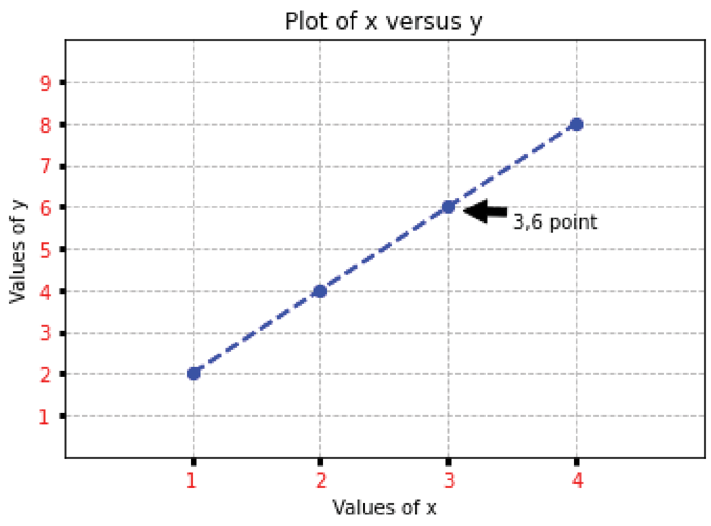


Figure 5.5: *A line graph with grid and tick parameters included*

Matplotlib provides you with plenty of options for creating beautiful linestyles, and markers along with a variety of color options to be used to beautify the plots and graphs. TMentioned below are the marker options available in matpotlib for using with plot and scatter functions:

Marker	Symbol	Description
"."	●	**point**
","	·	**pixel**
"o"	⬤	**circle**
"v"	▼	**triangle_down**
"^"	▲	**triangle_up**
"<"	◀	**triangle_left**

Marker	Symbol	Description
">"	▶	**triangle_right**
"1"	⋎	**tri_down**
"2"	⅄	**tri_up**
"3"	≺	**tri_left**
"4"	≻	**tri_right**
"8"	●	**octagon**
"s"	■	**square**
"p"	⬟	**pentagon**
"P"	✚	**plus (filled)**
"*"	★	**star**
"h"	⬢	**hexagon1**
"H"	⬣	**hexagon2**
"+"	+	**plus**
"x"	×	**x**

Marker	Symbol	Description
"X"	✖	**x (filled)**
"D"	◆	**diamond**
"d"	♦	**thin_diamond**
"|"	|	**vline**
"_"	—	**hline**
0 (TICKLEFT)	–	**tickleft**
1 (TICKRIGHT)	–	**tickright**
2 (TICKUP)	╵	**tickup**
3 (TICKDOWN)	╷	**tickdown**
4 (CARETLEFT)	◀	**caretleft**
5 (CARETRIGHT)	▶	**caretright**
6 (CARETUP)	▲	**caretup**
7 (CARETDOWN)	▼	**caretdown**
8 (CARETLEFTBASE)	◀	**caretleft (centered at base)**

Marker	Symbol	Description
9 (CARETRIGHTBASE)	▶	**caretright (centered at base)**
10 (CARETUPBASE)	▲	**caretup (centered at base)**
11 (CARETDOWNBASE)	▼	**caretdown (centered at base)**

Table 5.1: *Marker Options*

For colors, you can use the notation of a single character like:

‘r’ for red color,

‘g’ for green,

‘b’ for blue,

‘k’ for black,

‘w’ for white,

‘c’ for cyan,

‘m’ for magenta,

‘y’ for yellow.

However, you can also make use of hex representation of colors as used in html viz. '**#0f0f0f**'. Matplolib also supports string type full name representation for color. The names should be mentioned without any space in between like ‘*aquagreen*’.

Additionally, you can also use ‘0’ for black, ‘1’ for white and in between any floating number as grey shade like 0.7 for light grey.

Basic plots

Data visualization is an important aspect of data analysis as it helps in understanding and communicating the insights derived from the data.

Matplotlib is a widely used data visualization library in Python that provides a variety of plotting and charting functions for creating static, animated, and interactive visualizations.

Some of the common types of visualizations that can be created using Matplotlib include:

- **Line plots:** Line plots are used to visualize trends in data over time. They are useful for plotting time-series data, such as stock prices or weather data.
- **Bar charts:** Bar charts are used to compare different categories of data. They can be used to visualize categorical data, such as the number of purchases made by different customers.
- **Histograms:** Histograms are used to visualize the distribution of continuous data. They are useful for understanding the distribution of data, such as the distribution of ages or heights of a group of people.
- **Scatter plots:** Scatter plots are used to visualize the relationship between two continuous variables. They are useful for understanding the relationship between two variables, such as the relationship between height and weight.
- **Pie charts:** Pie charts are used to visualize proportions of different categories of data. They are useful for visualizing the composition of data, such as the composition of expenses or revenue.

In addition to these basic visualizations, Matplotlib also provides advanced visualization techniques, such as 3D plotting and animation. The library also supports customization of visualizations, allowing users to fine-tune the appearance of their plots and charts.

Overall, Matplotlib is a powerful tool for data visualization and can help in gaining insights from data and communicating results effectively.

Line plot

A line plot (also known as a line chart or line graph) is a type of graphical representation used to display the relationship between two variables. It plots individual data points as points on a two-dimensional coordinate system and then connects the points with a line.

Line plots are useful for exploring trends and patterns in a dataset over time or across categories. They can be used to show the following:

- **Trends**: A line plot can show whether the values of a variable are increasing, decreasing, or remaining constant over time.
- **Patterns**: A line plot can reveal patterns in the data, such as seasonal trends, fluctuations, or spikes.
- **Comparison**: Line plots can be used to compare different variables, showing how they change over time or across categories.
- **Outliers**: Outliers, or data points that fall outside the expected range of values, can be easily identified on a line plot.

Line plots are a useful tool for exploring the relationship between two variables and are often used in exploratory data analysis. They can be created using the plot function in the Matplotlib library in Python.

Here is an example of how to create a simple line plot using Matplotlib in Python:

```
import matplotlib.pyplot as plt
import numpy as np

#Sample data
x=np.linspace(10,20,100)
y=np.cos(x)

#Plot the data
plt.plot(x,y)

#Add labels and title
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.title('Cosine wave')

#Show the plot
plt.show()
```

In this example, we generate a sample data set x and y using the **linspace()** and **sin()** functions from the **NumPy** library. We then use the **plot()** function to plot the data, and add labels and a title to the plot using the **xlabel()**, **ylabel()**, and **title()** functions. Finally, we use the **show()** function to display the plot:

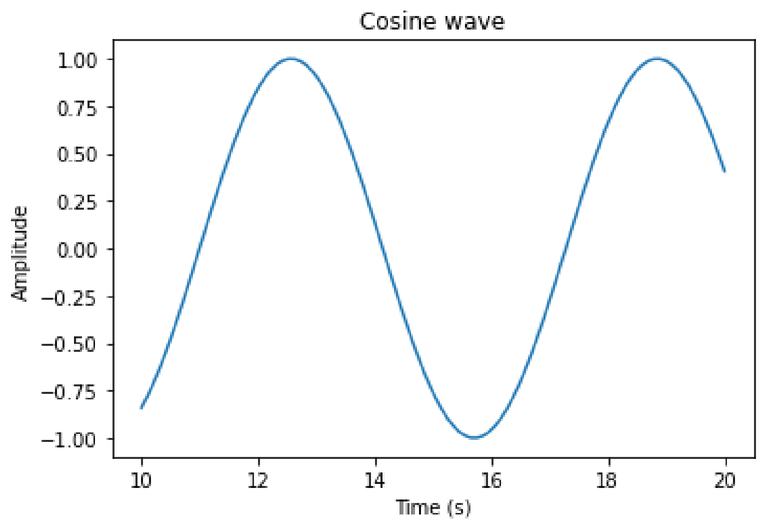


Figure 5.6: *A line plot*

This will create a line plot of the sine wave with the x-axis labeled as "Time (s)" and the y-axis labeled as "Amplitude". The title of the plot will be "Cos wave".

Bar chart

A bar chart (also known as a bar graph) is a type of graphical representation that is used to compare different values or categories. It consists of rectangular bars, with the length of each bar proportional to the value it represents.

The following elements make up a bar chart:

- **X-axis**: The x-axis represents the categories or labels that are being compared.
- **Y-axis**: The y-axis represents the values or quantities that are being compared.
- **Bars**: The bars represent the values for each category, with the length of each bar proportional to the value it represents.

- **Labels**: Labels are used to label each bar and associate it with the corresponding category.

Bar charts can be used to visualize simple comparisons between categories, such as the number of items sold in different regions or the popularity of different products. They can also be used to compare more complex data, such as the distribution of a continuous variable by different categories.

Bar charts can be created using the bar function in the Matplotlib library in Python. They can be created vertically (with the bars extending upward) or horizontally (with the bars extending to the right), and can be customized with different colors, labels, and other features to make the data easier to understand.

Here is an example of how to create a simple bar chart using Matplotlib in Python:

```
import matplotlib.pyplot as plt

#Sample data
categories = ['A', 'B', 'C', 'D']
values = [1, 4, 2, 5]

#Plot the data
plt.bar(categories, values)

#Add labels and title
plt.xlabel('Categories')
plt.ylabel('Values')
plt.title('Bar Chart Example')

#Show the plot
plt.show()
```

In this example, we create a sample data set of categories and values. We then use the **bar()** function to plot the data, and add labels and a title to the plot using the **xlabel()**, **ylabel(),** and **title()** functions. Finally, we use the **show()** function to display the plot:

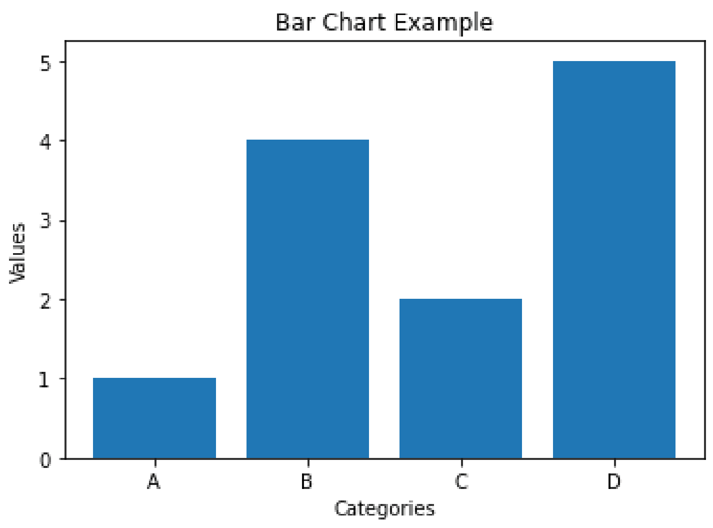


Figure 5.7: *A Bar Plot*

This will create a bar chart with the categories on the x-axis and the values on the y-axis. The x-axis will be labeled as "Categories" and the y-axis as "Values". The title of the plot will be "Bar chart example".

Histogram

A histogram is a type of graphical representation used to show the distribution of a dataset. It is an estimate of the probability distribution of a continuous variable, showing the number of observations that fall within specified ranges (or "bins") of values.

A histogram is created by dividing the range of values of the dataset into a series of intervals, called bins. The height of each bar in the histogram represents the number of observations that fall within the corresponding bin.

The following elements make up a histogram:

- **X-axis:** The x-axis represents the range of values in the dataset, divided into bins.
- **Y-axis:** The y-axis represents the frequency of observations in each bin.
- **Bars:** The bars represent the number of observations in each bin. The height of each bar is proportional to the frequency of observations in that bin.

- **Bin width:** The width of each bin represents the range of values that fall within that bin.

Histograms are useful for visualizing the distribution of a dataset and for identifying patterns or features such as skewness, outliers, and multimodality. They are often used in exploratory data analysis and can be created using the hist function in the Matplotlib library in Python.

Here is an example of how to create a histogram of the "Age" attribute of the Titanic survival data using Matplotlib in Python:

```
import matplotlib.pyplot as plt

import pandas as pd

#Load the Titanic dataset

df = pd.read_csv('titanic.csv')

#Plot the histogram

plt.hist(df.age, bins=20, edgecolor='black', alpha =0.7)

#Add labels and title

plt.xlabel('Age')

plt.ylabel('Frequency')

plt.title('Histogram of Titanic Passengers \'Age\'')

#Show the plot

plt.show()
```

In this example, we use the **read_csv()** function from the Pandas library to load the Titanic survival data into a DataFrame. We then select the "Age" column and remove missing values using the **dropna()** function. We use the **hist()** function to plot the histogram of the "Age" data, and specify the number of bins using the bins parameter. We add labels and a title to the plot using the **xlabel()**, **ylabel(),** and **title()** functions, and display the plot using the **show()** function.

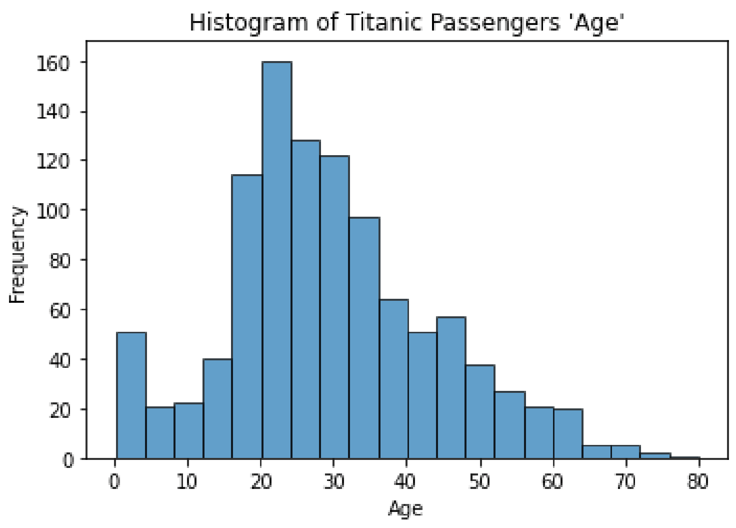


Figure 5.8*: A Histogram*

This will create a histogram of the Titanic passengers' ages with the x-axis labeled as "Age" and the y-axis labeled as "Frequency". The title of the plot will be "Histogram of Titanic Passengers' Ages".

Scatter plot

A scatter plot is a type of graphical representation used to display the relationship between two variables in a dataset. It plots individual data points as dots (or markers) on a two-dimensional coordinate system, with one variable on the x-axis and the other variable on the y-axis.

- Scatter plots are useful for exploring the relationship between two variables, including the following:
- **Positive or negative association**: A positive association between two variables means that as one variable increases, the other variable also increases. A negative association means that as one variable increases, the other variable decreases.
- **Linear or non-linear association**: A scatter plot can reveal whether the relationship between two variables is linear or non-linear. A linear relationship means that the points on the scatter plot follow a straight line, while a non-linear relationship means that the points do not follow a straight line.

- **Strength of the association**: The strength of the association between two variables can be determined by the density of the points on the scatter plot. A dense cluster of points indicates a strong association, while a spread-out distribution of points indicates a weak association.
- **Outliers**: Outliers, or data points that fall outside the expected range of values, can be easily identified on a scatter plot.

Scatter plots are a useful tool for exploring the relationship between two variables in a simple, visual way, and are often used in exploratory data analysis. They can be created using the scatter function in the Matplotlib library in Python.

Here is an example of how to create a scatter plot of the "Age" and "Fare" attributes of the Titanic survival data using Matplotlib in Python:

```
import matplotlib.pyplot as plt
import pandas as pd
 #Load the Titanic dataset
df = pd.read_csv('titanic.csv')

#Select the 'Age' and 'Fare' columns from dataset
age = df.age.fillna(df.age.max())
fare = df.fare.fillna(df.fare.max())

#Plot the Scatter Plot
plt.scatter(age, fare, alpha =0.7)

#Add labels and title
plt.xlabel('Age')
plt.ylabel('Fare')
plt.title('Scatter Plot of Titanic Passengers \'Age\' and \'Fare\'')

 #Show the plot
plt.show()
```

In this example, we use the **read_csv()** function from the Pandas library to load the Titanic survival data into a DataFrame. We then select the "Age" and "Fare" columns and remove missing values using the **dropna()** function. We use the **scatter()** function to plot the scatter plot of the "Age" and "Fare" data. We add labels and a title to the plot using the **xlabel()**, **ylabel()**, and **title()** functions, and display the plot using the **show()** function.

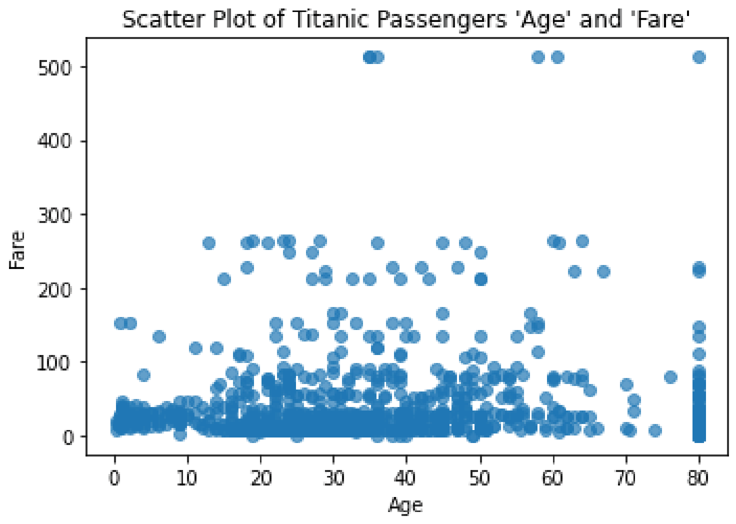


Figure 5.9: *A Scatter Plot*

This will create a scatter plot of the Titanic passengers' ages and fares with the x-axis labeled as "Age" and the y-axis labeled as "Fare". The title of the plot will be "Scatter Plot of Titanic Passengers' Age and Fare".

Pie chart

A pie chart is a type of circular statistical graphic, which is used to represent the proportional distribution of a dataset. Each segment of the pie chart represents a proportion of the total dataset, with the size of each segment proportional to the magnitude of the value it represents.

The pie chart consists of several parts:

- **Center:** The center of the pie chart is usually empty, but can be filled with a color or pattern.
- **Slices:** Each slice represents a category in the dataset, with the size of the slice proportional to the value of the category. The slices are usually

drawn as arcs that start from the center of the chart and extend outward.

- **Legend:** A legend is used to label each slice and associate it with the corresponding category.
- **Angle:** The angle of each slice is proportional to the value of the category.

Pie charts are useful for representing simple, proportional relationships in a clear and visual manner. However, they can become cluttered and difficult to interpret when the number of categories is large, or when the categories have similar sizes. In these cases, alternative visualizations such as bar charts or stacked bar charts may be more appropriate.

Here is an example of creating a pie chart using the Matplotlib library in Python to visualize the distribution of the number of survivors and casualties on the Titanic:

```
import matplotlib.pyplot as plt
import pandas as pd

#Load the Titanic dataset
df = pd.read_csv('titanic.csv')

#Calculate the proportion of Survivors
survived = df['survived'].value_counts()
proportion = survived / df['survived'].sum()

#Plot the Pie Chart
plt.pie(proportion, labels =["Died", "Survived"], autopct ='%1.1f%%')

#Add title
plt.title('Proportion of Passengers Who Survived the Titanic')

#Show the plot
plt.show()
```

This code will generate a pie chart that shows the distribution of passengers who survived and those who did not survive on the Titanic, along with the percentage values for each category. The code uses pandas to load the data into a dataframe, and then calculates the proportion of survivors and non-survivors. Finally, the pie chart is created using the pie function from Matplotlib, which takes the proportions as the input data and labels for each slice as the labels argument. The **autopct** argument is used to format the percentage values displayed on the chart.

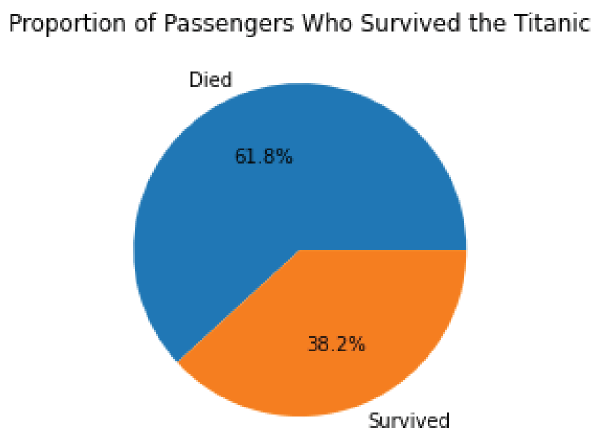


Figure 5.10: *A Pie Chart*

Area plot

An area plot in Matplotlib is a type of plot that represents the quantity for each category as a filled polygon. The area of the polygon is proportional to the quantity, making it easy to compare the relative sizes of the categories.

Here's an example of how to create an area plot using Matplotlib:

```
import matplotlib.pyplot as plt
import numpy as np

#Sample Data
categories =['Category 1', 'Category 2', 'Category 3']
quantities = [10,20,30]

#Plot the data as an Area Plot
plt.fill_between(categories, quantities, alpha=0.5)

#Add labels and title
```

```
plt.xlabel('Categories')
plt.ylabel('Quantities')
plt.title('Area plot Example')

#Show the plot
plt.show()
```

In this example, the **fill_between** function is used to create the area plot. The categories and quantities arrays provide the data for the plot, and the alpha argument is used to control the transparency of the fill. The plot can then be customized further using various other functions in Matplotlib, such as adding labels, titles, and annotations. Please refer to the following figure:

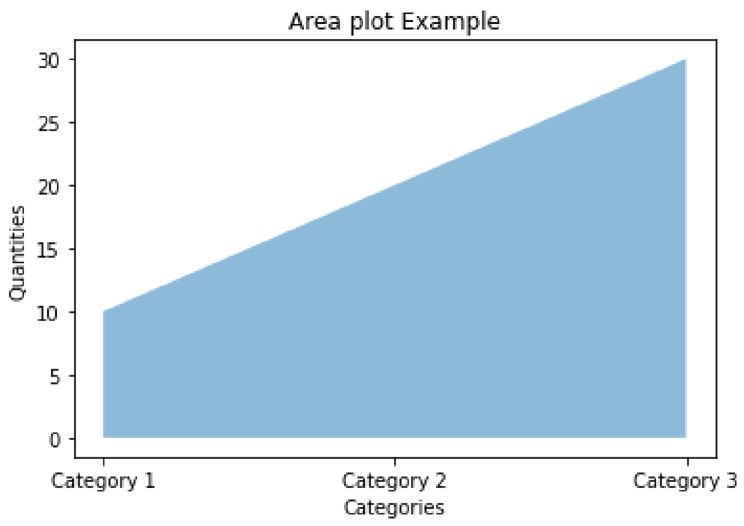


Figure 5.11: *An Area Plot*

Boxplot

A box plot, also known as a box-and-whisker plot, is a type of graph used to represent and visualize the distribution of a dataset. It is a standardized way of representing the shape of a distribution, allowing for easy comparison between multiple datasets.

A box plot consists of the following elements:

- **Box:** The box is drawn from the first quartile (25th percentile) to the third quartile (75th percentile), with a line inside the box representing the median (50th percentile). The box shows the interquartile range (IQR), which covers the middle 50% of the data and is a measure of the spread of the data.

- **Whiskers:** Whiskers are drawn from the box to the minimum and maximum values of the dataset, excluding outliers.
- **Outliers:** Outliers are plotted as individual points outside the whiskers. They represent values that fall outside the expected range for the data.
- **Mean:** The mean value of the dataset is often plotted as a symbol, such as a diamond, inside the box.

A box plot provides a quick and concise summary of the distribution of a dataset, making it easy to identify skewness, outliers, and other important features of the data. It is often used in exploratory data analysis and for comparing the distributions of multiple datasets.

```
import matplotlib.pyplot as plt
import pandas as pd

#Load the Titanic dataset
df = pd.read_csv('titanic.csv')

#Plot Box Plot on Age of Passengers, Grouped by their Survival Status
x=df[df['survived']==0]['age'].dropna()
x=df[df['survived']==1]['age'].dropna()
plt.boxplot([x,y], labels = ['Did not Survive', 'Survived'])

#Add labels and title
plt.xlabel('Survived Status')
plt.ylabel('Age')
plt.title('Box Plot of Age by Survival Status')

#Show the plot
plt.show()
```

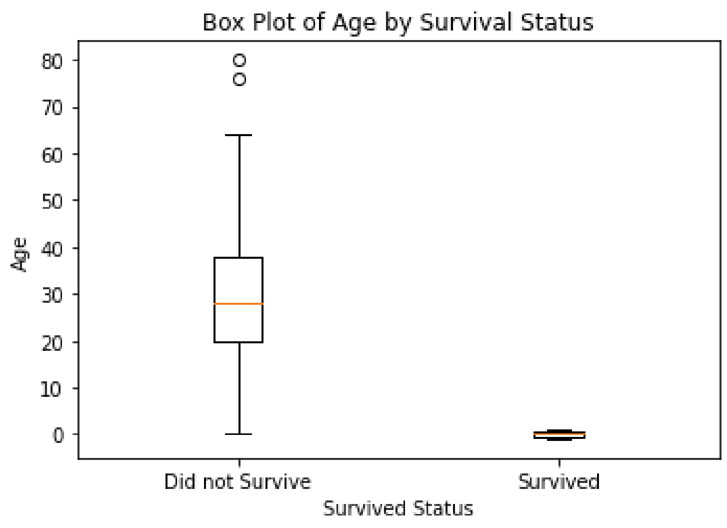


Figure 5.12: *A Box Plot*

You can always modify the appearance of the boxplot by using parameters like, **patch_artist**, sym, notch and so on. In the below code, we have given True value to notch and **patch_artist** to add a notch in the box and color it. The whisker representation symbol is changed to "+" using sym parameter.

plt.boxplot([x,y], labels = ['Did not Survive', 'Survived'], patch_artist=True, sym='+',

notch=True)

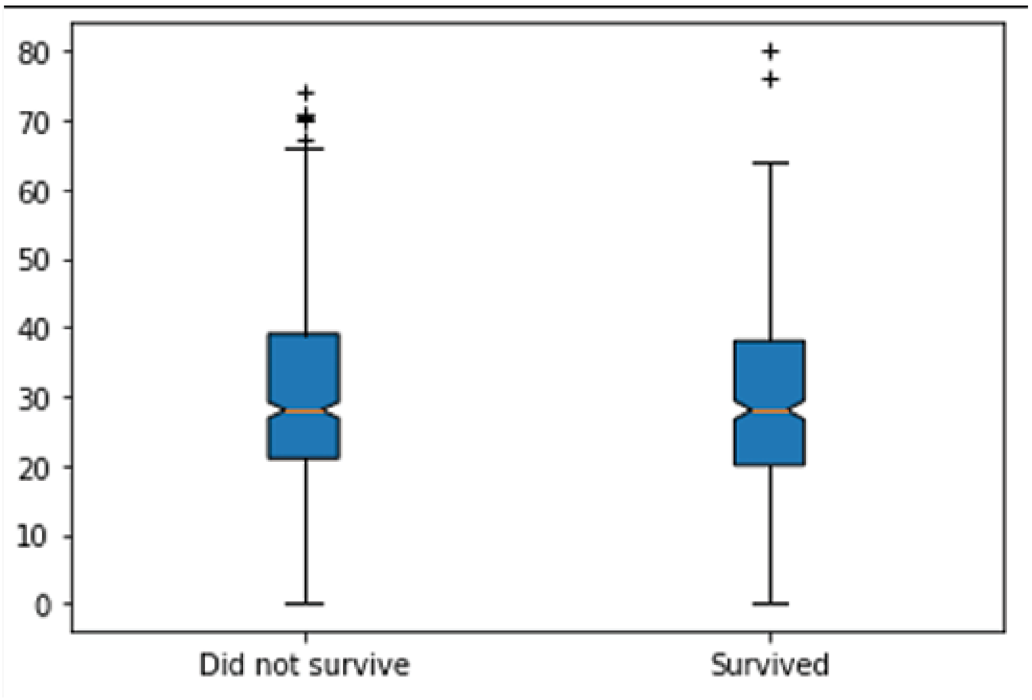


Figure 5.13: *A Box Plot with a Notch*

In this example, the Titanic dataset is loaded using the pandas library, and a box plot of the age of passengers is created, grouped by their survival status. The **plt.boxplot** function is used to create the box plot, with each sub-array representing the age data for a different passenger class. The plot is then customized using labels and a title.

Advanced plots

Matplotlib can be used to create 3D plots and visualizations. The most common way to do this is by using the mplot3d toolkit. Here's an example of how to create a simple 3D scatter plot:

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

#Sample Data
x = np.random.standard_normal(100)
y = np.random.standard_normal(100)
z = np.random.standard_normal(100)

#Plot the Data
ax.scatter(x, y, z)

#Set Labels for the axes
ax.set_xlabel('X axis')
ax.set_ylabel('Y axis')
ax.set_zlabel('Z axis')

#Show the plot
plt.show()
```

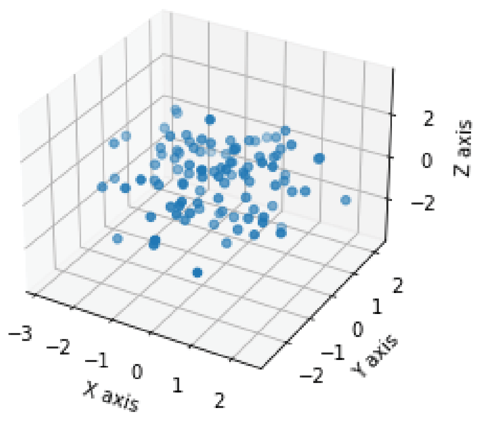


Figure 5.14: *A 3D Scatter Plot*

In addition to scatter plots, you can also create 3D line plots, bar plots, surface plots, and more. You can even plot implicit equations in 3D using the **plot_surface** method. The **mplot3d** toolkit provides a lot of options for customization and fine-tuning of your 3D plots.

A 3D surface plot can be created in Matplotlib using the **plot_surface** method of the Axes3D class. Here's an example of how to create a simple surface plot:

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

#Sample Data
x = np.linspace(-5,5,100)
y = np.linspace(-5,5,100)
x, y  = np.meshgrid(x, y)
z = np.sin(np.sqrt(x**2 + y**2))

#Plot the Surface plot
ax.plot_surface(x, y, z, cmap = 'viridis')

#Set Labels for the axes
```

```
ax.set_xlabel('X axis')
ax.set_ylabel('Y axis')
ax.set_zlabel('Z axis')

#Show the plot
plt.show()
```

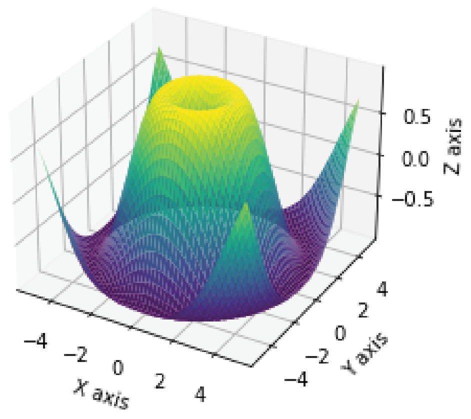


Figure 5.15: *A 3D Surface Plot*

In this example, the X and Y data are used to create a mesh grid, and the Z data is generated using the sin function. The **plot_surface** method is then used to plot the surface, and the **cmap** parameter is used to specify the colormap. You can experiment with different colormaps and other parameters to customize the appearance of your surface plot. the supported values for cmap are:

'Accent', 'Accent_r', 'Blues', 'Blues_r', 'BrBG', 'BrBG_r', 'BuGn', 'BuGn_r', 'BuPu', 'BuPu_r', 'CMRmap', 'CMRmap_r', 'Dark2', 'Dark2_r', 'GnBu', 'GnBu_r', 'Greens', 'Greens_r', 'Greys', 'Greys_r', 'OrRd', 'OrRd_r', 'Oranges', 'Oranges_r', 'PRGn', 'PRGn_r', 'Paired', 'Paired_r', 'Pastel1', 'Pastel1_r', 'Pastel2', 'Pastel2_r', 'PiYG', 'PiYG_r', 'PuBu', 'PuBuGn', 'PuBuGn_r', 'PuBu_r', 'PuOr', 'PuOr_r', 'PuRd', 'PuRd_r', 'Purples', 'Purples_r', 'RdBu', 'RdBu_r', 'RdGy', 'RdGy_r', 'RdPu', 'RdPu_r', 'RdYlBu', 'RdYlBu_r', 'RdYlGn', 'RdYlGn_r', 'Reds', 'Reds_r', 'Set1', 'Set1_r', 'Set2', 'Set2_r', 'Set3', 'Set3_r', 'Spectral', 'Spectral_r', 'Wistia', 'Wistia_r', 'YlGn', 'YlGnBu', 'YlGnBu_r', 'YlGn_r', 'YlOrBr', 'YlOrBr_r', 'YlOrRd', 'YlOrRd_r', 'afmhot', 'afmhot_r', 'autumn', 'autumn_r', 'binary', 'binary_r', 'bone', 'bone_r', 'brg', 'brg_r', 'bwr', 'bwr_r', 'cividis', 'cividis_r', 'cool', 'cool_r', 'coolwarm', 'coolwarm_r', 'copper', 'copper_r', 'cubehelix', 'cubehelix_r', 'flag', 'flag_r', 'gist_earth', 'gist_earth_r', 'gist_gray', 'gist_gray_r', 'gist_heat', 'gist_heat_r', 'gist_ncar', 'gist_ncar_r', 'gist_rainbow', 'gist_rainbow_r', 'gist_stern', 'gist_stern_r', 'gist_yarg', 'gist_yarg_r', 'gnuplot', 'gnuplot2', 'gnuplot2_r', 'gnuplot_r', 'gray', 'gray_r', 'hot', 'hot_r', 'hsv', 'hsv_r', 'inferno', 'inferno_r', 'jet', 'jet_r', 'magma', 'magma_r', 'nipy_spectral', 'nipy_spectral_r', 'ocean', 'ocean_r', 'pink', 'pink_r', 'plasma', 'plasma_r', 'prism', 'prism_r', 'rainbow', 'rainbow_r', 'seismic', 'seismic_r', 'spring', 'spring_r', 'summer', 'summer_r', 'tab10', 'tab10_r', 'tab20', 'tab20_r', 'tab20b', 'tab20b_r', 'tab20c', 'tab20c_r', 'terrain', 'terrain_r', 'turbo', 'turbo_r', 'twilight', 'twilight_r', 'twilight_shifted', 'twilight_shifted_r', 'viridis', 'viridis_r', 'winter', 'winter_r'

with **cmap ='Blues'**, the plot looks like this:

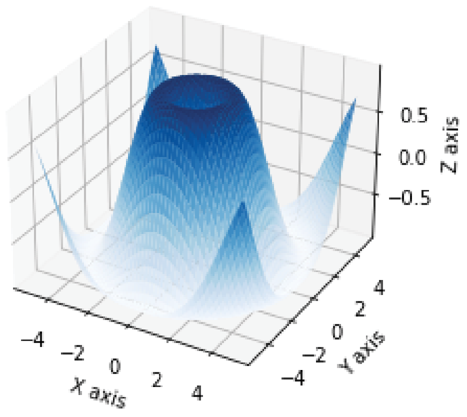


Figure 5.16: *A 3D Scatter Plot with cmap as 'Blues'*

The **np.meshgrid** function in NumPy is used to generate a 2D grid from two 1D arrays. The function takes two 1D arrays as input and returns two 2D arrays, representing a grid of points. The grid can then be used for plotting or other purposes.

Here is a simple example of how to use the **np.meshgrid** function:

```
import numpy as np

# Generate two 1D arrays
x = np.linspace(-5, 5, 10)
y = np.linspace(-5, 5, 10)

# Use np.meshgrid to generate a 2D grid
X, Y = np.meshgrid(x, y)

print(X)
print(Y)
```

The X array will contain the x-coordinates for the grid, and the Y array will contain the y-coordinates. The **np.meshgrid** function can be used to generate grids for other purposes, such as evaluating functions over a 2D domain or creating 3D plots. The function is especially useful when you want to generate a regular grid of points, as it avoids the need to write a loop to generate the grid.

Saving the plots

In Matplotlib, you can save a plot by using the **savefig** function. The basic syntax for saving a plot is as follows: This function takes a filename and optional parameters as arguments and saves the plot as an image file.

Here is the basic syntax for using the savefig function:

```
plt.savefig(filename, dpi=None, facecolor='w', edgecolor='w',
                    orientation='portrait', papertype =None, format= None,
                    transparent=False, bbox_inches=None, pad_inches=0.1,
                    frameon=None, metadata=None)
```

The **plt.savefig** function in Matplotlib has several parameters that you can use to customize the appearance of the saved plot:

filename: This is the name of the file to be saved. The format of the file is determined by the file extension (for example, ".png", ".pdf", and so on.).

dpi: This parameter specifies the number of dots per inch in the saved image. The default value is None, which means that the saved image will have a resolution of 100 DPI.

facecolor and **edgecolor:** These parameters specify the background color of the plot and the color of the plot's border, respectively. The default value for both parameters is 'w' (white). Below example will save the plot as a PNG file with the name "**example.png**" and a red background color, and a blue border color.

```
plt.plot([1,2,3,4])
plt.savefig("example.png", facecolor='red', edgecolor='blue')
```

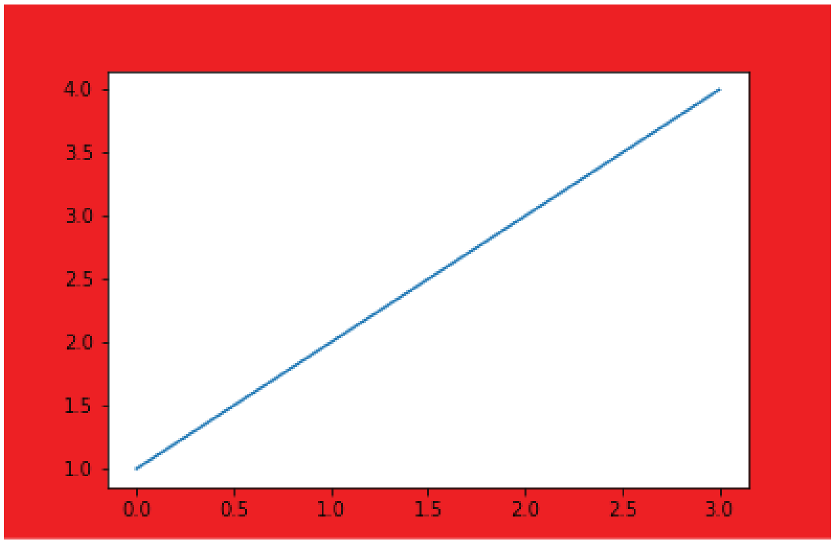


Figure 5.17: *Image saved with Red facecolor and Blue edge color of line plot*

orientation: This parameter specifies the orientation of the plot in the saved image. The default value is 'portrait', but you can also set it to 'landscape' to save the plot in landscape orientation. The below code will save the plot as a PNG file with the name "**example.png**" in landscape orientation.

```
plt.plot([1,2,3,4])
plt.savefig("example1.png", orientation='landscape', facecolor='red',
    edgecolor='blue')
```

papertype: This parameter specifies the type of paper on which the plot will be printed. The default value is None, which means that the plot will be saved on a default size paper. Below code will save the plot as a PNG file with the name "**example.png**" on legal size paper.

```
plt.plot([1,2,3,4])
plt.savefig("example2.png",papertype='legal')
```

The above code will save the plot as a PNG file with the name "example.png" with a transparent background.

format: This parameter specifies the format of the saved image. The default value is None, which means that the format will be determined by the file extension in the filename.

transparent: This parameter specifies whether the background of the plot will be transparent or not. The default value is False, which means that the background will not be transparent.

```
plt.plot([1,2,3,4])
plt.savefig("example3.png",transparent = True)
```

bbox_inches: This parameter specifies the size of the bounding box surrounding the plot in inches. The default value is None, which means that the bounding box will have the same size as the plot.

pad_inches: This parameter specifies the amount of padding around the plot in inches. The default value is 0.1, which means that there will be a 0.1-inch padding around the plot.

frameon: This parameter specifies whether the plot will have a border or not. The default value is None, which means that the plot will have a border.

metadata: This parameter specifies a dictionary of metadata that will be saved with the image file. The default value is None, which means that no metadata will be saved.

By using these parameters, you can customize the appearance of the saved plot and make sure that it meets your specific requirements.

For example, if you wanted to save a plot as a PNG image, you would use the following code:

```
plt.savefig("plot.png")
```

You can also save a plot in other formats such as JPEG, PDF, SVG, and so on. by changing the file extension in the filename. For example, to save a plot as a PDF file, you can use the following code:

```
plt.savefig("plot.pdf")
```

You can also specify other parameters when saving a plot, such as the DPI (dots per inch), the size of the plot, and the transparency. For example, the following code saves a plot as a 300 DPI PNG image with a size of 10x8 inches:

```
plt.plot([1,2,3,4])
plt.savefig("example4.png",dpi=300,bbox_inches='tight', pad_inches=0.5,transparent = True)
```

It's important to note that the **savefig** function must be called after all plot elements have been added and the plot has been displayed (using show or draw).

Features of Matplotlib library

Matplotlib is a data visualization library in Python. It provides a wide range of plotting and charting functions for creating static, animated, and interactive visualizations in Python. Matplotlib is one of the most widely used data visualization libraries in the scientific and research communities, and it's widely used for creating plots, charts, histograms, scatter plots, and other types of visualizations.

Some of the features of Matplotlib include:

- **Plotting functions:** Matplotlib has a variety of functions for creating different types of plots and charts, including line plots, bar charts, histograms, scatter plots, and more.
- **Customization options:** Matplotlib provides a wide range of customization options, such as color, line styles, labels, and annotations, that allow users to fine-tune their visualizations.
- **Interactive visualization:** Matplotlib supports interactive visualization, which allows users to zoom, pan, and explore their data visualizations in real-time.
- **Integration with other libraries:** Matplotlib can be easily integrated with other data analysis and machine learning libraries in Python, such as Pandas and Scikit-learn, to create advanced visualizations and analyses.

Matplotlib is a powerful tool for data visualization and can help in gaining insights from data and communicating results effectively. It is widely used in fields such as data science, finance, and scientific research, among others.

Conclusion

In this chapter, we explored the powerful capabilities of Matplotlib, a widely used data visualization library in Python. We discussed various aspects of data visualization, ranging from basic plotting techniques to advanced customization options. With Matplotlib, we learned to create basic plots using pyplot functions, customize colors and styles, and add labels and annotations.

We also explored working with multiple axes for side-by-side comparisons and customizing plots by adjusting limits, adding grid lines, and incorporating text. Following best practices, such as choosing appropriate plot types and ensuring readability, is essential. Matplotlib further enables advanced visualizations like 3D plots, time series analysis, and interactive visuals. Additionally, we learned how to save plots in different formats and embed them in various output formats.

In the next chapter, we will explore on plotting various charts and plots with seaborn library.

Points to remember

- Matplotlib provides functions like **xlabel()**, **ylabel()**, and **title()** to add labels to the x-axis, y-axis, and the plot's title, respectively.
- Customization options in Matplotlib include changing line colors, line styles, marker styles, and plot transparency using parameters like **color**, **linestyle**, **marker**, and **alpha**.
- Legends can be added to plots using the **legend()** function, allowing you to label different elements on the plot and provide a key for interpretation.
- Annotations can be added to highlight specific points or features on the plot using the **annotate()** function, including text, arrows, and markers.
- Matplotlib allows for the creation of subplots using the **subplots()** function or by using the **add_subplot()** method on an existing figure object, enabling the display of multiple plots in a grid layout.
- Grid lines can be added to plots using the **grid()** function, helping to visualize the underlying structure and aid in data interpretation.
- Matplotlib provides various color maps, accessible through the **cmap** parameter, allowing for the customization of colors in plots.
- Matplotlib supports saving plots in different file formats, such as PNG, JPEG, PDF, or SVG, using the **savefig()** function.

- Plots can be embedded in different output formats, including Jupyter Notebooks, HTML documents, or interactive web applications, using appropriate methods or libraries like Matplotlib's mpld3.

Questions

1. What is Matplotlib?
2. What is the purpose of Matplotlib in data visualization?
3. How can Matplotlib be installed in Python?
4. What is the role of the pyplot module in Matplotlib?
5. What are some commonly used plot types in Matplotlib?
6. How can multiple subplots be created in a single figure using Matplotlib?
7. What are some techniques for customizing plots in Matplotlib?
8. How can labels and titles be added to a Matplotlib plot?
9. What is the purpose of legends and annotations in Matplotlib?
10. How can plots be saved in different file formats using Matplotlib?
11. Given a dataset of monthly sales figures for a year, can you plot a line plot to visualize the trend over time? Monthly sales figures = [100, 150, 200, 180, 250, 300, 280, 320, 350, 400, 380, 420]. Plot the line plot and customize it by adding labels, title, and changing the line color.
12. Create a scatter plot to show the relationship between the temperature and the number of ice cream sales in different cities? Temperature (in degrees Celsius) and Ice cream sales. Data = [(28, 150), (30, 180), (25, 120), (32, 200), (27, 160)]. Plot the scatter plot and customize it by adding labels, title, and changing the marker style.
13. Create a bar plot to compare the monthly revenue generated by different products in a store. Monthly revenue for products A, B, C, D ([5000, 3500, 4500, 6000]). Plot the bar plot and customize it by adding labels, title, and changing the color of the bars.
14. Visualize the distribution of ages of participants in a survey using a histogram. Participant ages ([22, 25, 28, 30, 33, 35, 40, 45, 50, 55, 60]).

Plot the histogram and customize it by adding labels, title, and adjusting the number of bins.

15. Create a box plot to represent the test scores of students in different subjects. Test scores for subjects Math, Science, English, History ([85, 90, 75, 80]). Plot the box plot and customize it by adding labels, title and changing the color of the boxes.

Annexure Installing Python

1. Download the Python installer from the official website. I would recommend downloading Anaconda.

   https://www.anaconda.com/

   Anaconda is a popular distribution of the Python programming language that comes with a pre-installed set of packages and tools for data science and scientific computing.

   https://www.anaconda.com/

   Some of the key features of Anaconda include:

   - **Package Management:** Anaconda comes with a package manager called Conda that makes it easy to install, update, and manage packages and dependencies. This can save time and reduce the risk of compatibility issues.

   - **Pre-installed Packages:** Anaconda includes a large number of pre-installed packages for data science, scientific computing, and machine learning, including popular packages such as NumPy, Pandas, Matplotlib, and more.

   - **Jupyter Notebook:** Anaconda includes the Jupyter Notebook, a web-based interactive computing environment that is widely used for data analysis and visualization.

   - **Environment Management:** Anaconda allows you to create isolated environments for your projects, each with its own set of packages and dependencies. This makes it easy to switch between projects and ensures that each project is using the correct version of packages.

- **Cross-platform Support:** Anaconda is available for Windows, macOS, and Linux, making it a versatile choice for a wide range of users.

  Overall, Anaconda provides a comprehensive and convenient solution for data science, scientific computing, and machine learning in Python. Whether you're a beginner or an experienced user, Anaconda can help you get started quickly and easily.

2. Run the installer and follow the instructions.
3. Once the installation is complete, you can verify the installation by running the Python interpreter.
4. You can also check the version of Python installed by running the command python --version in the terminal.
5. Finally, you can start writing and running Python programs.

Annexure Pandas

Installing pandas:

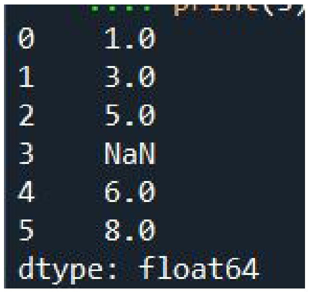
pip install pandas

Pandas is an open-source Python library for data analysis and manipulation. It provides data structures for efficiently storing large datasets and tools for working with them. The two most important classes defined by pandas are the Series and DataFrame classes.

A Series is a one-dimensional labeled array that can hold any data type. For example, the following code creates a series of integers:

```
# Create a series of integers

import pandas as pd

s= pd.Series([1, 3, 5, np.nan, 6, 8])

print(s)
```

Output:



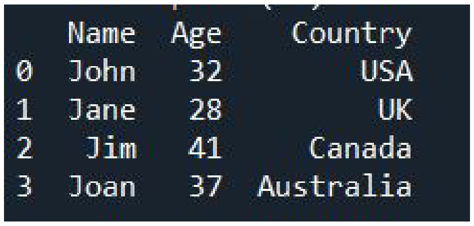
```
0    1.0
1    3.0
2    5.0
3    NaN
4    6.0
5    8.0
dtype: float64
```

Output 5.1

A DataFrame is a two-dimensional labeled data structure with columns of potentially different types. You can think of it as a spreadsheet or a SQL table. For example, the following code creates a dataframe from a dictionary:

```
import pandas as pd

#create a dictionary of data
data ={'Name' : ['John', 'Jane', 'Jim', 'Joan'],
    'Age' : [32, 28, 41, 37],
    'Country' : ['USA', 'UK', 'Canada', 'Australia']}

#creating dataframe from dictionary
df = pd.DataFrame(data)
print(df)
```

Output:



```
   Name  Age    Country
0  John   32        USA
1  Jane   28         UK
2   Jim   41     Canada
3  Joan   37  Australia
```

Output 5.2

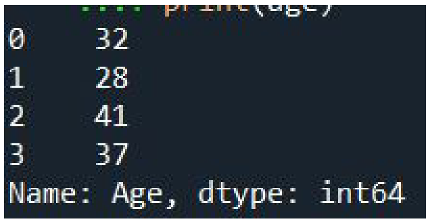
Pandas provides a variety of functions for working with both Series and DataFrames, such as indexing, slicing, reshaping, grouping, merging, and joining. Here are a few examples:

#Accessing a column from the dataframe

age = df['Age']

print(age)

Output:



```
0    32
1    28
2    41
3    37
Name: Age, dtype: int64
```

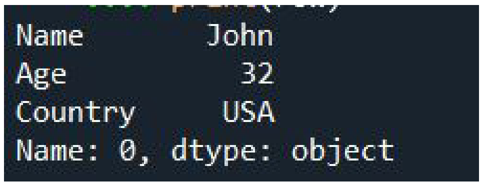
Output 5.3

#Access a row in a dataframe using indexing

row = df.loc[0]

print(row)

Output:



```
Name       John
Age          32
Country     USA
Name: 0, dtype: object
```

Output 5.4

#Filter rows in a dataframe based on a condition

filtered_df = df[df['Age']>35]

print(filtered_df)

Output:

Output 5.5

#Group data in a dataframe by a column and aggregate the results

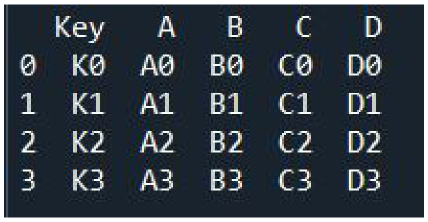
```
grouped_df = df.groupby(['Country']).mean()
print(grouped_df)
```

Output:

Output 5.6

```
#Merge two dataframes based on a common column

df1 = pd.DataFrame ({'Key' : ['K0','K1','K2','K3'],
                    'A' : ['A0','A1','A2','A3'],
                    'B' : ['B0', 'B1','B2','B3']})

df2 = pd.DataFrame ({'Key' : ['K0','K1','K2','K3'],
                    'C' : ['C0','C1','C2','C3'],
                    'D' : ['D0', 'D1','D2','D3']})

merged_df = pd.merge(df1, df2, on='Key')
print(merged_df)
```

Output:



```
  Key   A   B   C   D
0  K0  A0  B0  C0  D0
1  K1  A1  B1  C1  D1
2  K2  A2  B2  C2  D2
3  K3  A3  B3  C3  D3
```

Output 5.7

These are just a few examples of what you can do with the Pandas library

Annexure Numpy

Installing numpy:

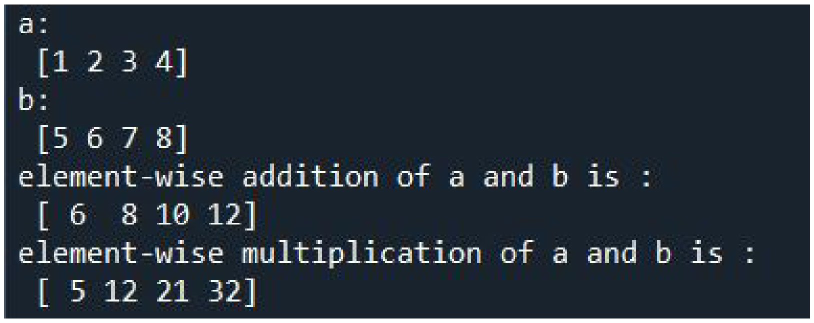
pip install numpy

NumPy is a powerful library for numerical computing in Python. It provides support for arrays, which are multi-dimensional data structures for storing large datasets. NumPy arrays are more efficient than regular Python lists for numerical operations and support a wide range of mathematical operations.

Here's an example of how you could use NumPy to perform element-wise operations on arrays:

```
import numpy as np

#creat two arrays
a = np.array([1,2,3,4])
b = np.array([5,6,7,8])

print("a:\n",a)
print("b:\n",b)
 #perform element-wise addition
c = a + b
print("element-wise addition of a and b is : \n", c)
#perform element-wise multiplication
d = a * b
print("element-wise multiplication of a and b is : \n", d)
```

Output:



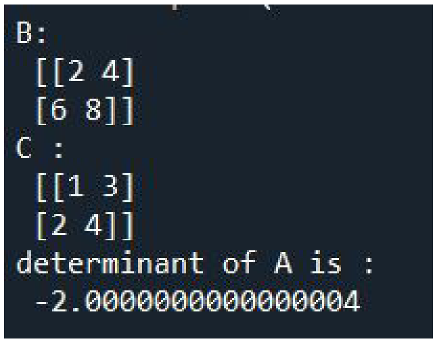
```
a:
 [1 2 3 4]
b:
 [5 6 7 8]
element-wise addition of a and b is :
 [ 6  8 10 12]
element-wise multiplication of a and b is :
 [ 5 12 21 32]
```

Output 5.8

NumPy also provides functions for performing matrix operations, such as matrix multiplication, transposition, and determinant calculation. Here's an example:

```
import numpy as np

#create a 2X2 matrix
A = np.array([[1,2],[3,4]])

#Multiply the matrix with a scalar
B = 2* A
print("B: \n", B)

# Transpose the matrix
C = A.T
print("C : \n", C)

#Calculate the determinant of the matrix
det = np.linalg.det(A)
print("determinant of A is :\n",det)
```

Output:



```
B:
 [[2 4]
 [6 8]]
C :
 [[1 3]
 [2 4]]
determinant of A is :
 -2.0000000000000004
```

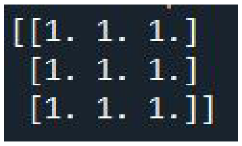
Output 5.9

NumPy also provides functions for generating arrays with specific values, such as ones, zeros, or a range of values. For example:

```
#creating an array of ones
ones = np.ones((3,3))
```

print(ones)

Output:



```
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
```

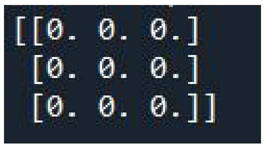
Output 5.10

#creating an array of zeros

zeros = np.zeros((3,3))

print(zeros)

Output:



```
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
```

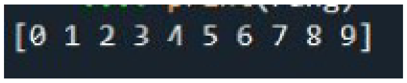
Output 5.11

#Create an array with range of values

rang = np.arange(0,10)

print(rang)

Output:



```
[0 1 2 3 4 5 6 7 8 9]
```

Output 5.12

These are just a few examples of what you can do with the NumPy library. It is a powerful tool for numerical computing, and is widely used in scientific computing, data analysis, and machine learning.

CHAPTER 6
Data Visualization with Seaborn

The goal is to turn data into information, and information into insight.
– Carly Fiorina

Seaborn is a popular Python data visualization library built on top of Matplotlib that provides a high-level interface for creating informative and attractive statistical graphics. It is particularly useful for exploratory data analysis, as it allows you to quickly create a wide range of visualizations that can reveal insights about your data.

In this chapter, you will explore the key features and functionalities of the Seaborn module and show how to create different types of visualizations using Seaborn.

Structure

In this chapter, we will discuss the following topics:

- Introduction
- Importing Seaborn and its documentation, Seaborn Datasets
- Understanding figure, axes and subplots, Palette
- Categorical plots
- Other plots
- Saving the plot

Objectives

As the book title suggests, the main objective of the book is to provide knowledge on using python libraries for data visualization. In this chapter, the main objective is to create a cohesive narrative that supports the book's overall purpose of learning visualization using the Seaborn. This chapter presents various functions available in the seaborn module of python, which are used for presenting data in graphical representations.

Introduction

Seaborn is a Python data visualization library that is built on top of Matplotlib. It provides a high-level interface for creating attractive and informative statistical graphics. Seaborn is particularly useful for visualizing complex datasets with many variables. Seaborn comes with several built-in datasets that can be used for practice and exploration. Some of the popular datasets included are tips, flights, iris, titanic, and diamonds.

Seaborn offers a range of visualization types including scatter plots, line plots, bar plots, heat maps, and distribution plots. Seaborn also makes it easy to customize the look and feel of the plots, allowing us to create beautiful and informative visuals. It provides several functions for customizing the look and feel of plots, including color palettes, grid styles, and font sizes. In addition to its visualization capabilities, Seaborn also offers several statistical functions that can be used to explore relationships between variables in the dataset. For example, Seaborn's regression plots can be used to fit and visualize linear models between variables. Seaborn is a powerful and flexible tool for data visualization and exploration in Python, making it an essential library for data scientists and analysts.

Seaborn is a versatile and powerful data visualization library in Python. Some of its main features include:

- **Built-in datasets**: Seaborn provides several built-in datasets for practice and exploration, including tips, flights, iris, titanic, and diamonds.

- **High-level interface**: Seaborn offers a high-level interface for creating complex and informative statistical graphics with minimal code. It provides easy-to-use functions for creating different types of plots, such as scatter plots, line plots, bar plots, heat maps, and distribution plots.
- **Customization options**: Seaborn provides a range of customization options for plots, including color palettes, grid styles, and font sizes. Users can also customize the appearance of plots by tweaking parameters such as figure size, axis labels, and title.
- **Statistical functions**: Seaborn offers several statistical functions that can be used to explore relationships between variables in the dataset. For example, Seaborn's regression plots can be used to fit and visualize linear models between variables.
- **Integration with Matplotlib**: Seaborn is built on top of Matplotlib, which means that users can use Matplotlib functions and objects alongside Seaborn functions.
- **Easy-to-use syntax**: Seaborn's syntax is easy to learn and use, making it accessible to both beginner and advanced users.

Wide range of applications: Seaborn can be used in various fields such as data science, machine learning, finance, healthcare, and social sciences to explore and visualize complex data sets.

Importing seaborn and its documentation

Before we dive into the different visualizations provided by Seaborn, we need to first install the module and import it into our Python environment. You can install Seaborn using pip, the Python package manager, by running the following command in your terminal or command prompt:

```
C:\Users>pip install seaborn
```

Once Seaborn is installed, you can import it into your Python script using the following code:

```
import seaborn
```

Or

import seaborn as sns

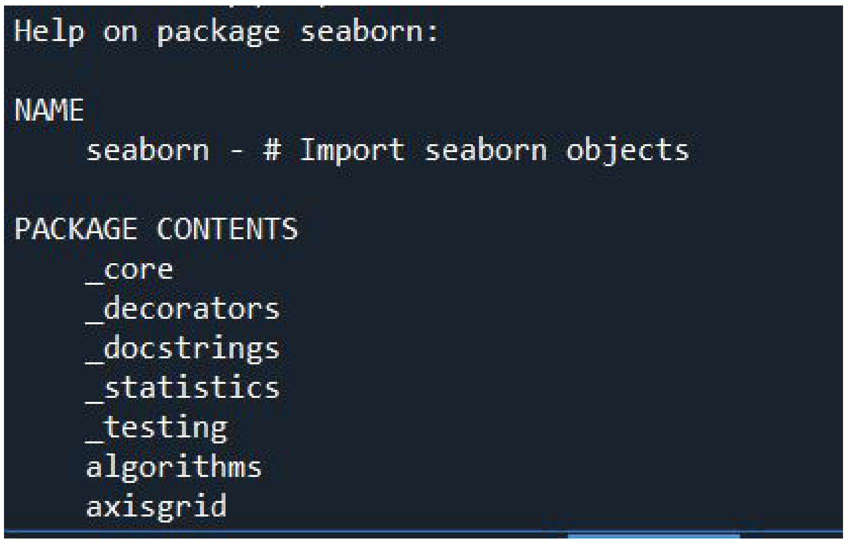
With Seaborn imported, you are now ready to explore its functionalities to create different types of plots.

To access the documentation for Seaborn, you can visit the official Seaborn website at **https://seaborn.pydata.org/.** The website provides comprehensive documentation for all the functions and features of Seaborn, as well as several examples and tutorials to help you get started with data visualization in Seaborn.

You can also use the following command in your Python script or interactive shell:

help(sns)

This will display the documentation for the entire Seaborn module, including all of its functions and classes. Please refer to the following figure:



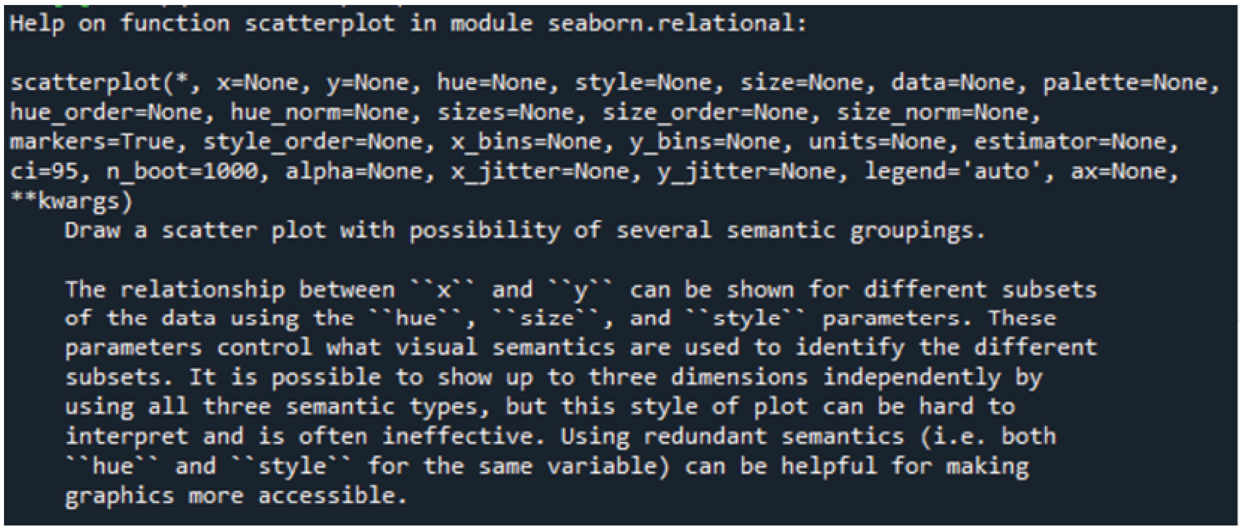
```
Help on package seaborn:

NAME
    seaborn - # Import seaborn objects

PACKAGE CONTENTS
    _core
    _decorators
    _docstrings
    _statistics
    _testing
    algorithms
    axisgrid
```

Figure 6.1: *Output of help(sns)*

You can also access the documentation for a specific function by using the help() function and passing the function name as an argument, like this:

help(sns.scatterplot)

This will display the documentation for the **scatterplot()** function, including its arguments, return value, and usage examples.



```
Help on function scatterplot in module seaborn.relational:

scatterplot(*, x=None, y=None, hue=None, style=None, size=None, data=None, palette=None,
hue_order=None, hue_norm=None, sizes=None, size_order=None, size_norm=None,
markers=True, style_order=None, x_bins=None, y_bins=None, units=None, estimator=None,
ci=95, n_boot=1000, alpha=None, x_jitter=None, y_jitter=None, legend='auto', ax=None,
**kwargs)
    Draw a scatter plot with possibility of several semantic groupings.

    The relationship between ``x`` and ``y`` can be shown for different subsets
    of the data using the ``hue``, ``size``, and ``style`` parameters. These
    parameters control what visual semantics are used to identify the different
    subsets. It is possible to show up to three dimensions independently by
    using all three semantic types, but this style of plot can be hard to
    interpret and is often ineffective. Using redundant semantics (i.e. both
    ``hue`` and ``style`` for the same variable) can be helpful for making
    graphics more accessible.
```

Figure 6.2: *Output of help(sns.scatterplot)*

You can use the documentation to learn about the different functions provided by Seaborn, their arguments and default values, and how to use them to create different types of plots.

Seaborn datasets

You can use the '**load_dataset()** function to load an example dataset from the online repository (requires internet). This function provides quick access to a small number of example datasets which you can make use of while practicing with data visualization techniques. In the code below we are trying to fetch the 'tips' dataset from example datasets available with seaborn and loading that into the tips variable. The dataset will be returned in the form of a dataframe.

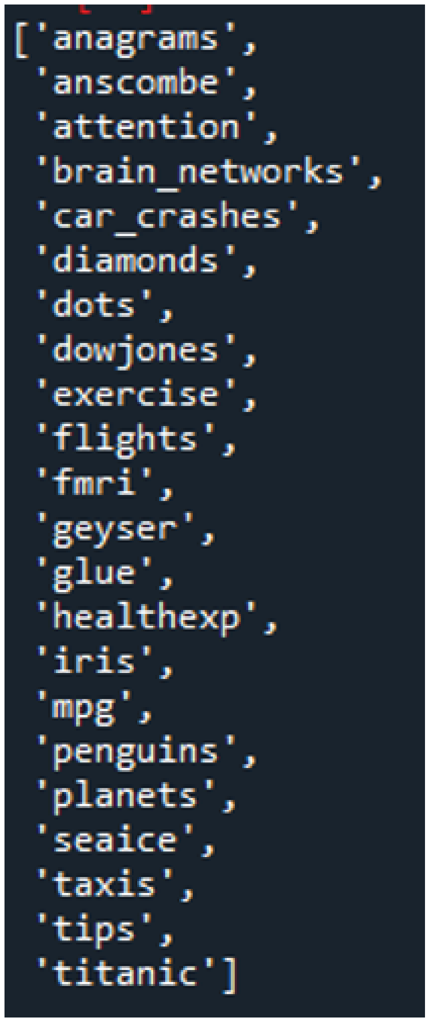
```
import seaborn as sns

#Load the tips dataset

tips = sns.load_dataset("tips")
```

You can use **get_dataset_names** to see a list of available datasets like this:

```
import seaborn as sns
```

sns.get_dataset_names()

Output:



```
['anagrams',
 'anscombe',
 'attention',
 'brain_networks',
 'car_crashes',
 'diamonds',
 'dots',
 'dowjones',
 'exercise',
 'flights',
 'fmri',
 'geyser',
 'glue',
 'healthexp',
 'iris',
 'mpg',
 'penguins',
 'planets',
 'seaice',
 'taxis',
 'tips',
 'titanic']
```

Figure 6.3: *Output of get_dataset_names*

In this chapter, we will be using ‘tips’ dataset for creating various visualizations.

Understanding figure, axes, subplots, and palette

In this section of the chapter, let us understand the basic components required to build a plot in seaborn viz. Figure, axes, subplots and palette.

Figure, axes, and subplots

Seaborn is a Python data visualization library that is built on top of Matplotlib. It provides a high-level interface for creating informative and attractive statistical graphics. Seaborn is particularly useful for visualizing complex datasets with multiple variables.

To understand Figure, Glyphs and Axes, Palettes in Seaborn, let’s start with some basic definitions:

- **Figure**: A figure in Seaborn is a container for all the elements of a plot, including the axes, titles, legends, and so on.
- **Axes and subplots**: Axes in Seaborn are the individual plots within a figure. Each set of axes represents a single plot. A figure can contain one or more axes.
- **Palettes**: A palette is a collection of colors that can be used to visually distinguish different categories or levels of a variable in a plot. Seaborn provides a number of pre-defined color palettes, or you can create your own.

In Matplotlib, Bokeh, and Seaborn, the concept of a Figure refers to a container that holds all the elements of a plot, such as axes, labels, legends, and other components. However, there are some differences in the implementation and purpose of Figures in the libraries. In Matplotlib, a Figure is the top-level container that contains one or more Axes objects, which represent individual plots or subplots within the Figure. You can create a Figure in Matplotlib using the **plt.figure()** function, which returns a Figure object that can be used to add plots and other elements to the figure. The Figure object provides methods for setting various properties of the plot, such as the title, axis labels, and legend. Matplotlib’s Figure is highly customizable and allows for fine-grained control over the plot’s appearance, as we have learned in the previous chapter on matplotlib.

Bokeh’s Figure is a high-level object that represents the entire plotting canvas. It is the primary object used to create visualizations in Bokeh. A Bokeh Figure is created using the **bokeh.plotting.figure()** function, which returns a Figure object that can be used to add glyphs, lines, and other visual elements to the plot. The Figure object provides methods for setting various properties of the plot, such as the title, axis labels, and legend. You will learn more on bokeh in the next chapter on Bokeh.

In Seaborn, a figure is a container for all the elements of a plot, including the axes, titles, legends, and so on. It is the top-level component that provides the overall structure of a plot. To create a figure in Seaborn, you can use the **plt.subplots()** function, which returns a tuple containing a figure object and one or more axes objects. For example:

```
import seaborn as sns

import matpotlib.pyplot as plt

#Create a figure and a set of subplots

fig, ax = plt.subplots()
```

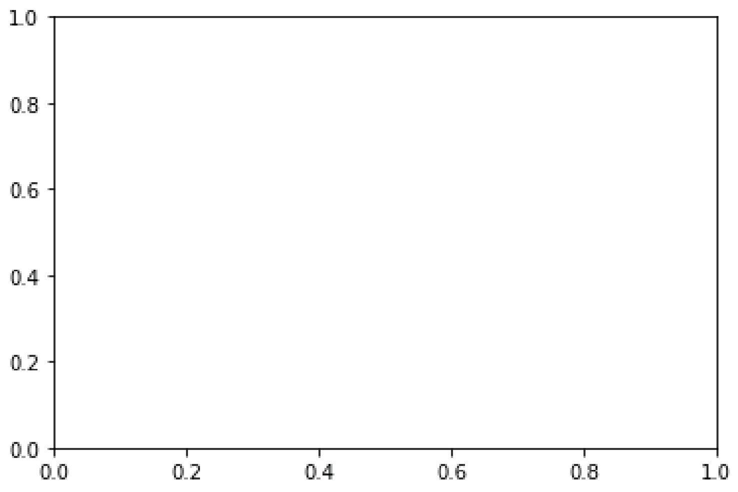


Figure 6.4: *Creating a figure and a subplot*

The Figure object is the top-level container, while the Axes objects represent the individual plots or subplots within the Figure. Seaborn builds on top of Matplotlib and provides a higher-level interface that simplifies common data visualization tasks. Seaborn provides functions that generate complex plots with a single function call, reducing the need for low-level customization. Additionally, Seaborn provides built-in support for statistical analysis, such as the ability to plot regression lines and calculate confidence intervals. Matplotlib's Figure provides a highly customizable container for creating individual plots and subplots, while Seaborn's Figure is designed to make it easy to create attractive, publication-quality plots quickly with less need for customization.

Below is an example of labelling the plot in Seaborn:

```
# Plot the data using seaborn lineplot
sns.lineplot(x='year', y='pop', data=my_data, ax=ax, palette=palette)

# Set the title of the figure
ax.set_title('Population Growth Over Time')

# Set the ticks, labels for the x and y axes
ax.set_xticks([2000, 2001, 2002, 2003, 2004])
ax.set_xlabel('Year')
ax.set_ylabel('Population (millions)')
# Display the plot
plt.show()
```

In this example, **sns.lineplot()** function is used to add a line plot to the Figure using the data in **my_data** and the **ax** object as the target. We also set the title, ticks and labels for the plot using the **ax.set_xticks(), ax.set_title()** and **ax.set_xlabel()** and **ax.set_ylabel()** methods. Please refer to the following figure:

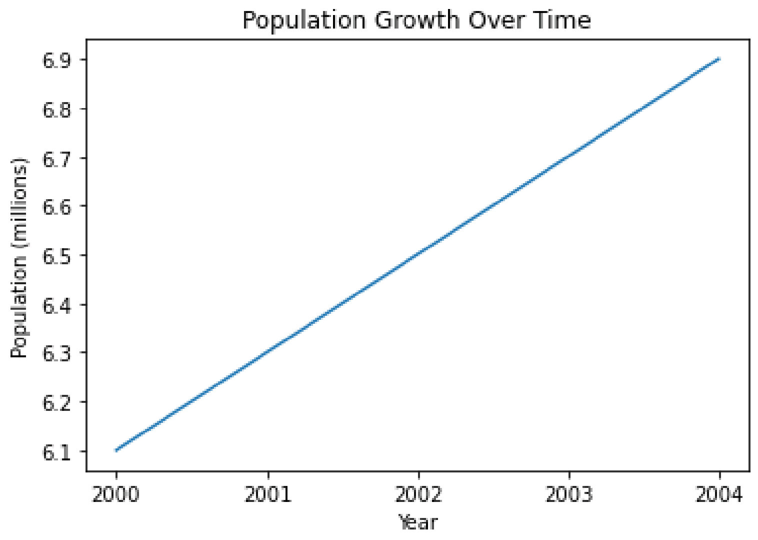


***Figure 6.5**: Labelled plot*

You can customize the Figure further by adding more Axes objects to it, adjusting the layout, and adding other plot elements. Once you have created a Figure, you can save it as an image file using the **fig.savefig()** method. Note the data is created using the following code of line:

```
#Synthetic data creation using dictionary
data_dict = {‘year’: [2000, 2001, 2002, 2003, 2004],
            ‘pop’: [6.1, 6.3, 6.5, 6.7, 6.9]}
# Creating a DataFrame from the dictionary
my_data = pd.DataFrame(data_dict)
```

Axes in Seaborn refers to the individual plot objects that are used to create a visualization. These Axes objects are similar to the matplotlib Axes objects and are created using the **seaborn.axes** module.

In Seaborn, the Axes class is a subclass of the matplotlib Axes class and provides additional methods and attributes for creating statistical plots. Each Axes object can be thought of as a single plot panel that can be customized separately. You can create any plot using the available functions and specify the **ax** parameter to specify the Axes object to draw the plot on. Similarly, you can create a figure with multiple subplots using the **plt.subplots()** function and then pass the individual Axes objects to Seaborn plotting functions to create multiple plots in the same figure. Once you have created a figure, you can add one or more axes to it using the **add_subplot()** method of the figure object. For example:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# Load the Titanic dataset
titanic = pd.read_csv('titanic.csv')
# Create a figure with two subplots
fig = plt.figure(figsize=(10, 5))
# First subplot: survival rate by sex
```

```
ax1 = fig.add_subplot(1, 2, 1)
sns.barplot(x='sex', y='survived', data=titanic, ax=ax1)
ax1.set_title('Survival Rate by Sex')
# Second subplot: survival rate by passenger class
ax2 = fig.add_subplot(1, 2, 2)
sns.barplot(x='pclass', y='survived', data=titanic, ax=ax2)
ax2.set_title('Survival Rate by Passenger Class')
# Adjust the layout
fig.tight_layout()
# Show the plot
plt.show()
```

Output:

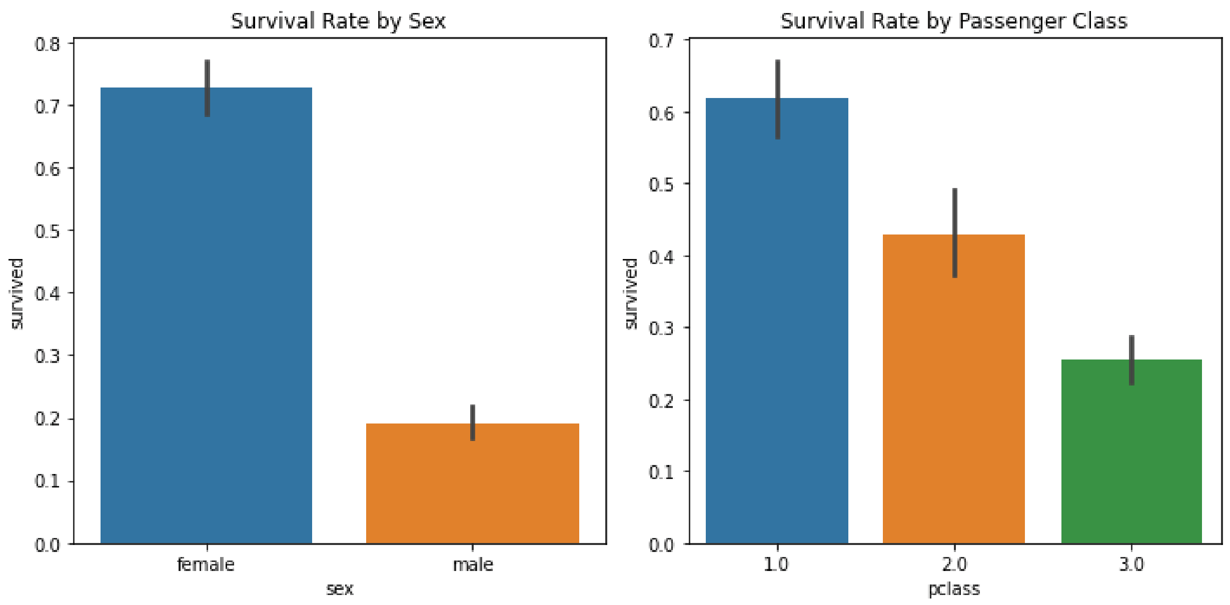


Figure 6.6: *Displaying two plots using subplot()*

A Figure object using **plt.figure()** is implemented and two subplots are included in it using **add_subplot()**. To create the bar plots in each subplot we have used the **sns.barplot()** function. The x and y parameters of **sns.barplot()** specify the columns to use for the x-axis and y-axis, respectively. We have passed the **ax** parameter to **sns.barplot()** to specify which subplot to use for each plot. Note that you can adjust the layout, and show the plot using **tight_layout()** and **show(),** respectively.

Palettes

Seaborn is a Python library for data visualization that provides a wide range of color palettes for creating visually appealing plots. Here are some of the color palettes available in Seaborn:

- deep
- muted
- bright
- pastel
- dark
- colorblind
- husl

These palettes can be accessed through the **sns.color_palette()** function by passing the name of the palette as a string. For example, to use the "muted" palette, you would use the following code:

```
import seaborn as sns
sns.set_palette("muted")
```

Additionally, Seaborn also provides a **sns.color_palette()** function that allows you to create custom color palettes by specifying a list of colors. You can also adjust the brightness and saturation of the colors using the **sns.light_palette()** and **sns.dark_palette()** functions.

Let's display a heatmap with a dark color palette:

```
import seaborn as sns

import matplotlib.pyplot as plt

# Create a matrix of values to display

matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

# Create a dark color palette
```

```
palette = sns.dark_palette("red", as_cmap=True)
# Plot the heatmap with the matrix and color palette
sns.heatmap(matrix, cmap=palette)
# Display the plot
plt.show()
```

Output:

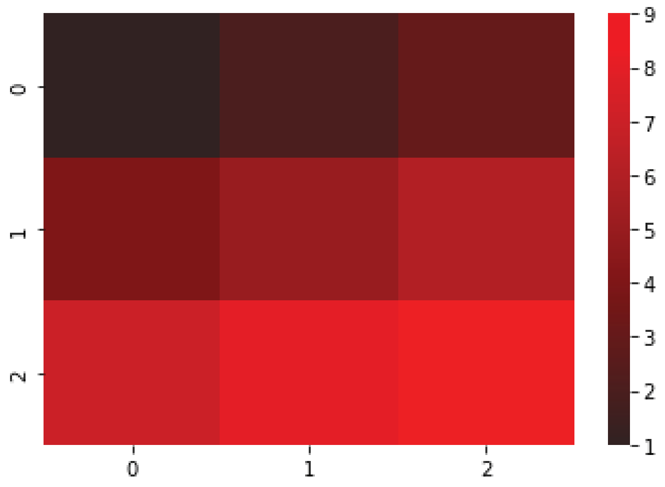


Figure 6.7: *Palette example –dark_palette*

This program creates a 3x3 matrix of values, creates a dark color palette using the **dark_palette** function from the Seaborn library, and plots the heatmap using the heatmap function from Seaborn. Finally, it displays the plot using **plt.show()**. You can modify the matrix values or the color palette to suit your needs.

Categorical plots

Seaborn is a popular Python data visualization library that can be used to create plots on **categorical** data. Seaborn is designed to work with both numerical and categorical data types, and it provides specialized functions to create plots on both types of data.

For categorical data, you do not need to convert the data into a numerical format to create plots. Seaborn functions can handle categorical data directly and can create plots that show the distribution, relationships, or comparisons of different categories.

In fact, Seaborn is particularly useful for visualizing categorical data, as it provides several specialized functions that make it easy to create visually appealing and informative plots, such as bar plots, count plots, and point plots, among others.

Some Seaborn functions, such as **lmplot()** and **regplot(),** require numerical data as inputs. In these cases, you may need to convert categorical data to numerical format before creating the plot. However, this is not always necessary, and Seaborn provides many options for working with categorical data directly. Seaborn has several specialized functions to create different types of plots specifically for categorical data, such as bar plots **(sns.barplot())**, count plots **(sns.countplot()),** and point plots **(sns.pointplot())**, among others.

These plots are useful for visualizing categorical variables, such as the distribution of data across different categories, the relationship between different categorical variables, and comparing different groups or subgroups of data.

Seaborn provides a high-level interface to create visually appealing and informative plots, and it also integrates well with Pandas data frames, making it easy to manipulate and visualize data. Additionally, Seaborn provides many customization options, such as changing the color palette, adding annotations, and adjusting the plot style, among others, to tailor the plots to specific needs.

The main categorical plots available in seaborn:

- **sns.barplot()**: A bar plot represents an estimate of the mean of a numerical variable for each category of a categorical variable. This plot can also show confidence intervals around the mean estimate.
- **sns.countplot()**: A count plot shows the count of observations in each categorical bin using bars. This plot is useful for visualizing the distribution of data across different categories.

- **sns.boxplot()**: A box plot summarizes the distribution of a numerical variable for each category of a categorical variable. This plot shows the median, quartiles, and outliers of the data.
- **sns.violinplot()**: A violin plot is similar to a box plot, but it also shows the density of the data at different values. This plot is useful for visualizing the distribution of data across different categories.
- **sns.stripplot()**: A strip plot shows the individual observations of a numerical variable for each category of a categorical variable using dots. This plot is useful for visualizing the spread of data across different categories.
- **sns.swarmplot()**: A swarm plot is similar to a strip plot, but it also avoids overlapping points by adjusting their positions. This plot is useful for visualizing the spread of data across different categories.
- **sns.catplot():** A catplot is a general categorical plotting function that can create several types of categorical plots, such as bar plots, count plots, box plots, and violin plots, among others. This function also allows for easy faceting of plots based on additional categorical variables.
- **sns.FacetGrid():** FacetGrid is particularly useful when you want to visualize the relationship between a categorical variable and one or more numerical variables.

These categorical plots are useful for visualizing and understanding the distribution of data across different categories, identifying patterns and trends in the data, and comparing different groups or subgroups of data.

Bar plot

A bar chart is a type of data visualization that uses rectangular bars to represent categorical data. It is commonly used to compare and display the values of different categories or groups. The length or height of each bar corresponds to the value or quantity associated with the category it represents. In seaborn, to create a bar chart barplot() function is used as below:-

```
import seaborn as sns
import matplotlib.pyplot as plt
# Load the tips dataset
tips = sns.load_dataset("tips")
# Create a bar plot using Seaborn
sns.barplot(x="day", y="total_bill", data=tips,
            hue="sex", ci=None, palette="Set2")
# Set the title and axis labels of the plot
plt.title("Total Bill Amount by Day of the Week and Gender")
plt.xlabel("Day of the Week")
plt.ylabel("Total Bill Amount ($)")
# Show the plot
plt.show()
```

Output:

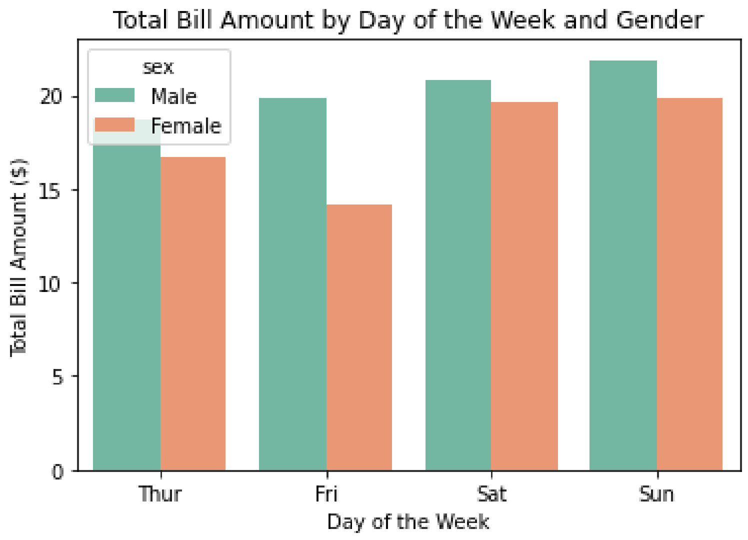


Figure 6.8: *Bar chart with barplot()*

In this example code, we create a bar plot of the total bill amount by day of the week and gender using the tips dataset. Here's an explanation of the parameters used in the **sns.barplot()** function:

- **x="day"**: This sets the x-axis of the plot to the "day" column of the tips dataset.
- **y="total_bill"**: This sets the y-axis of the plot to the "**total_bill**" column of the tips dataset.
- **data=tips**: This sets the dataset to be used for creating the plot.
- **hue="sex"**: This creates a grouping variable based on the "sex" column of the tips dataset. In this case, the bars are colored differently based on the gender of the customers.
- **ci=None**: This sets the size of the error bars to be displayed on each bar. In this case, we set it to None to remove the error bars.
- **palette="Set2"**: This sets the color palette to be used for coloring the bars based on gender.

Here are some additional parameters that can be used in the **sns.barplot()** function:

- **estimator**: This parameter specifies the statistical function used to aggregate the values of each group. For example, the default estimator is the mean function, but it can be changed to other functions like median or standard deviation.
- **order**: This parameter specifies the order of the categories on the x-axis. By default, the categories are sorted in ascending order.
- **hue_order**: This parameter specifies the order of the categories for the hue variable.
- **capsize**: This parameter sets the size of the caps at the end of the error bars.
- **dodge**: This parameter specifies whether the bars for each group should be positioned closely together or with a small gap between them.

Let's try to implement more of the parameters of **barplot()** function.

```
sns.barplot(x='day', y='total_bill', data=tips, capsize=0.5, estimator=sum,
                          order=['Thur', 'Fri', 'Sat', 'Sun'],)
```

```
# Set the title and axis labels of the plot
plt.title('Average total bill amount by day of the week')
plt.xlabel("Day of the Week")
plt.ylabel("Total Bill Amount ($)")
# Show the plot
plt.show()
```

Output:

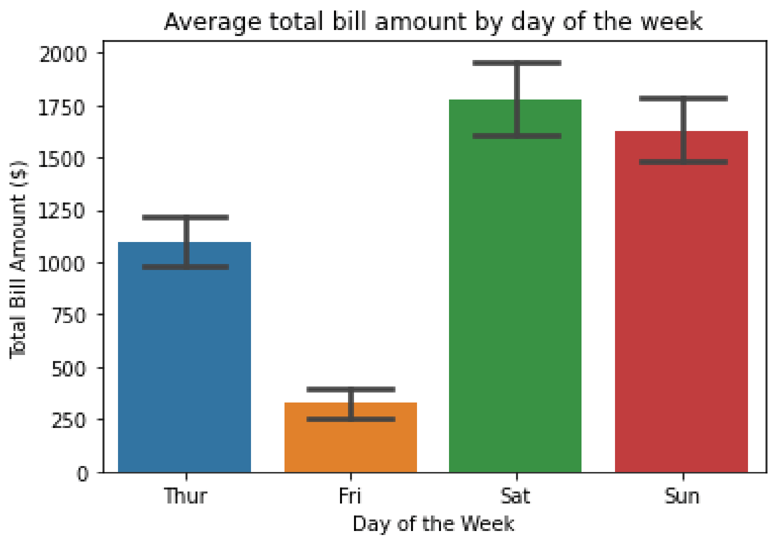


Figure 6.9: *Bar chart with estimator, order and capsize parameters*

To the **barplot()** function, we have passed the following arguments:

- **x='day'**: The column in the tips dataframe that contains the days of the week.
- **y='total_bill'**: The column in the tips dataframe that contains the total bill amount.
- **data=tips**: The dataframe containing the data.
- **capsize=0.2**: The size of the caps on the error bars.
- **estimator=sum**: The function used to aggregate the data for each category. In this case, we use the sum function to calculate the total bill amount for each day of the week.

- **order=[‘Thur’, ‘Fri’, ‘Sat’, ‘Sun’]**: The order in which the days of the week should appear on the x-axis of the plot. The resulting bar plot shows the average total bill amount for each day of the week, with error bars showing the standard deviation of the data. The caps on the error bars will be 20% of the width of the bars, and the data will be aggregated using the sum function. The days of the week will be ordered from Thursday to Sunday on the x-axis.

Count plot

A count plot is a type of categorical plot in which the count of observations in each category is represented using bars. The **sns.countplot()** function in the Seaborn library creates a count plot, which is a type of categorical plot that shows the frequency of each category in a dataset. The function has several parameters that can be used to customize the plot. Here are some of the main parameters:

- **x**: The variable to count and plot on the x-axis.
- **y**: The variable to count and plot on the y-axis.
- **data**: The dataset to use for the plot.
- **hue**: A categorical variable to separate the counts by color.
- **palette**: A color palette to use for the plot.
- **order**: The order in which to plot the categories.
- **hue_order**: The order in which to plot the categories for the hue variable.
- **orient**: Whether to plot the bars horizontally or vertically.
- **ax**: The matplotlib Axes object to draw the plot onto.
- **color**: A single color to use for all the bars.

```
import seaborn as sns

import matplotlib.pyplot as plt

# Load the Tips dataset

tips = sns.load_dataset("tips")
```

```
# Create a count plot of the number of customers by day of the week
sns.countplot(x="day", data=tips)
# Set plot title and axes labels
plt.title("Number of Customers by Day of the Week")
plt.xlabel("Day of the Week")
plt.ylabel("Number of Customers")
# Show the plot
plt.show()
```

Output:

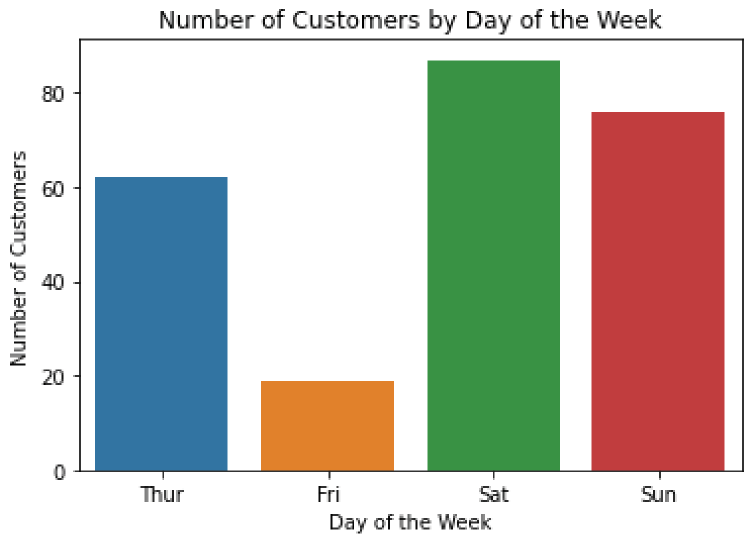


Figure 6.10: *Count plot with countplot()*

In this code, we are trying to create a count plot that shows the number of customers for each day of the week. The x parameter specifies the categorical variable to plot, and the data parameter specifies the dataset to use. Seaborn automatically counts the number of occurrences of each category and plots them using bars.

```
# Create a count plot of the number of customers by day of the week
sns.countplot(x="day", hue="sex", data=tips, palette="pastel",
order=["Thur", "Fri", "Sat", "Sun"])
```

Output:

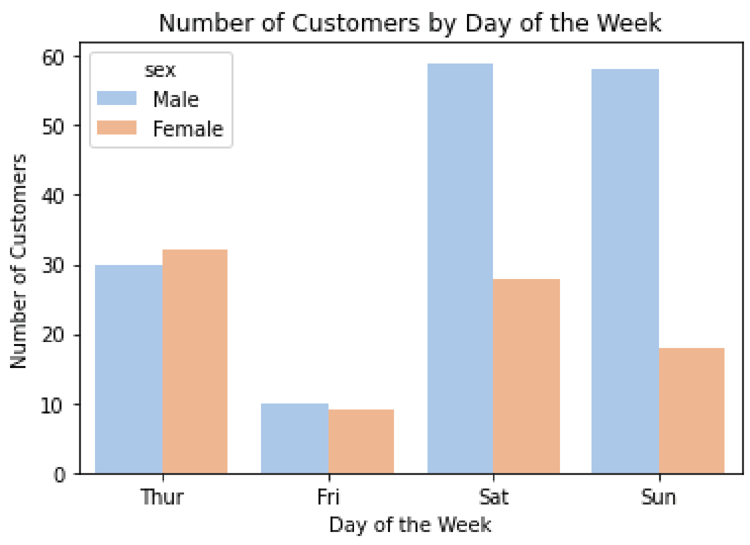


Figure 6.11: *Count plot with hue and palette parameters*

In this code, we are trying to create a customized count plot using various parameters that shows the number of male and female customers for each day of the week. The x parameter specifies the day variable to plot on the x-axis, and the hue parameter separates the counts by sex variable and assigns a different color to each category. You can also use the palette parameter to specify a pastel color palette, the order parameter to plot the categories in a specific order, and the usual axis labels and plot title to make the plot more informative.

Box plot

Creating a box plot using Seaborn's **sns.boxplot()** function:

```
import seaborn as sns
import matplotlib.pyplot as plt
# Load the tips dataset
tips = sns.load_dataset("tips")
# Create a box plot using Seaborn
sns.boxplot(x="day", y="total_bill", data=tips,
            hue="sex", palette="Set2", notch=True)

# Set the title and axis labels of the plot
```

```
plt.title("""Distribution of Total Bill Amount
          by Day of the Week and Gender""")
plt.xlabel("Day of the Week")
plt.ylabel("Total Bill Amount ($)")
# Show the plot
plt.show()
```

Output:

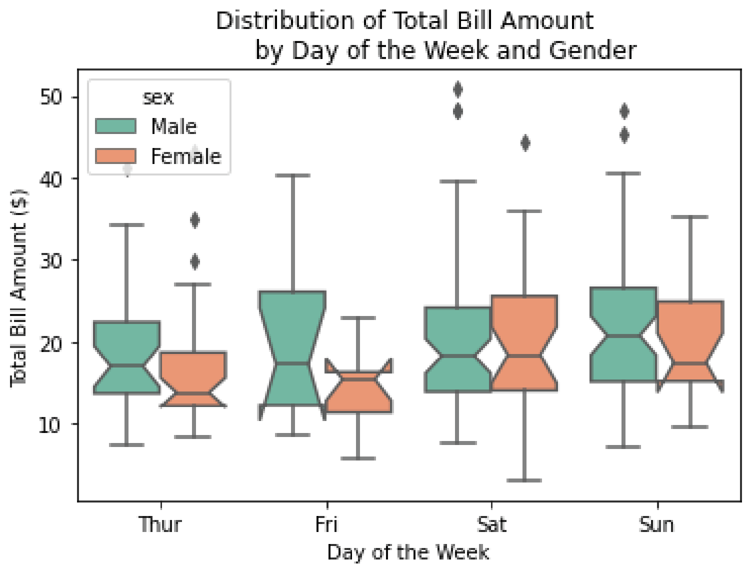


Figure 6.12: *Boxplot with notched boxes*

Here's an explanation of the parameters used in the **sns.boxplot()** function:

- **x=”day”**: This sets the x-axis of the plot to the “day” column of the tips dataset.
- **y=”total_bill”**: This sets the y-axis of the plot to the “total_bill” column of the tips dataset.
- **data=tips**: This sets the dataset to be used for creating the plot.
- **hue=”sex”**: This creates a grouping variable based on the “sex” column of the tips dataset. In this case, the boxes are colored differently based on the gender of the customers.

- **palette=”Set2”**: This sets the color palette to be used for coloring the boxes based on gender.
- **notch=True**: This sets whether to draw a notch around the median of each box, which is used to compare the medians between groups.

Here are some additional parameters that can be used in the sns.boxplot() function:

- **order**: This parameter specifies the order of the categories on the x-axis. By default, the categories are sorted in ascending order.
- **hue_order**: This parameter specifies the order of the categories for the hue variable.
- **showfliers**: This parameter controls whether to show the outliers in the box plot or not.
- **width**: This parameter sets the width of the boxes in the plot.
- **whis**: This parameter sets the proportion of the data outside the whiskers to be shown as outliers.
- **linewidth**: This parameter sets the width of the lines used to draw the boxes and whiskers.

```
# Create a box plot with custom parameters

sns.boxplot(x="day", y="total_bill", hue="sex", data=tips,

        showfliers=False, whis=[5, 95], width=0.5)
```

Output:

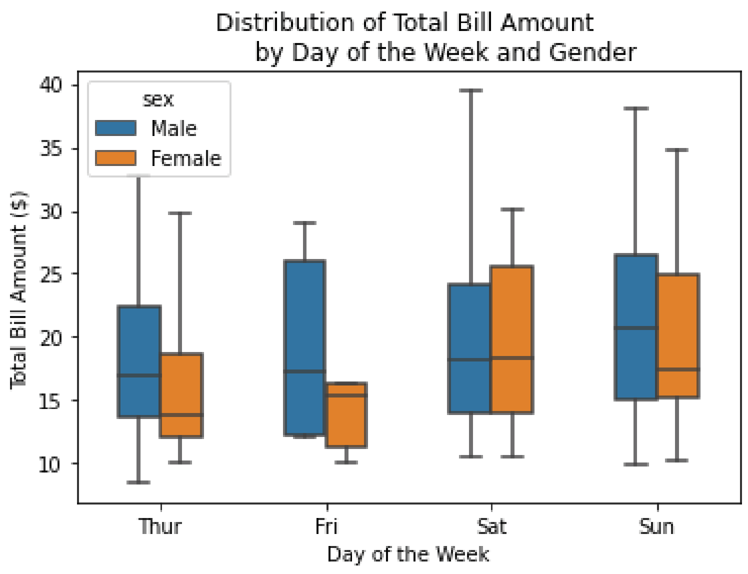


Figure 6.13: *Customized boxplot with hue, whis and showfliers*

The **showfliers** parameter controls whether to show outliers (in this case we set it to False to hide them), while the **whis** parameter sets the whiskers to a custom percentile range (here we set it to 5th and 95th percentiles). Finally, the width parameter sets the width of the boxes. Note that the notch parameter is not used in this code. Note the difference in the shape of boxes. By adjusting these parameters, you can customize the appearance of the plot to better suit our needs.

Violin plot

A violin plot is a type of data visualization that is like a box plot, but it also shows the distribution of the data. In a violin plot, the shape of the "violin" represents the distribution of the data, with the width of the violin indicating the density of the data at different values.

Here's an example of how to create a violin plot using Seaborn's **sns.violinplot()** function on the tips dataset:

```
import seaborn as sns

import matplotlib.pyplot as plt

# Load the tips dataset

tips = sns.load_dataset("tips")
```

```
# Create a violin plot using Seaborn
sns.violinplot(x="day", y="total_bill", data=tips,
        hue="sex", split=True, inner="stick", palette="Set2")

# Set the title and axis labels of the plot
plt.title("Distribution of Total Bill Amount by Day of the Week and Gender")
plt.xlabel("Day of the Week")
plt.ylabel("Total Bill Amount ($)")
# Show the plot
plt.show()
```

Output:

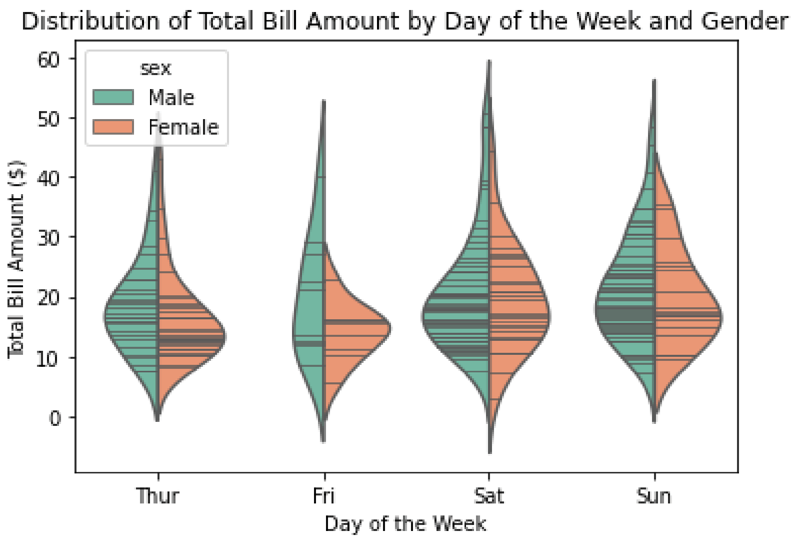


Figure 6.14: *Violin plot*

Here's an explanation of the parameters used in the **sns.violinplot()** function:

- **x=”day”**: This sets the x-axis of the plot to the “day” column of the tips dataset.
- **y=”total_bill”**: This sets the y-axis of the plot to the “total_bill” column of the tips dataset.
- **data=tips**: This sets the dataset to be used for creating the plot.

- **hue=”sex”**: This creates a grouping variable based on the “sex” column of the tips dataset. In this case, the violins are colored differently based on the gender of the customers.
- **split=True**: This splits the violins for each gender so that they are side-by-side.
- **inner=”stick”**: This adds a line to the center of each violin to show the median value.
- **palette=”Set2”**: This sets the color palette to be used for coloring the violins based on gender.

Here are some additional parameters that can be used in the **sns.violinplot()** function:

- **bw**: This parameter sets the bandwidth of the kernel density estimate used to smooth the violin shape. By default, it is set to “scott”, but it can be set to other values like “silverman” or a scalar value to adjust the bandwidth.
- **cut**: This parameter sets the extent of the violin shape beyond the minimum and maximum values of the data. By default, it is set to 2, but it can be set to other values to adjust the shape of the violin.
- **scale**: This parameter sets the method used to scale the width of the violins. By default, it is set to “area”, which scales the width of the violins based on the number of observations in each group. Other options include “width” or a scalar value to adjust the width.

```
# Create a violin plot with custom parameters

sns.violinplot(x="day", y="total_bill", hue="sex", data=tips,

               bw=0.3, cut=0, scale="width")
```

Output:

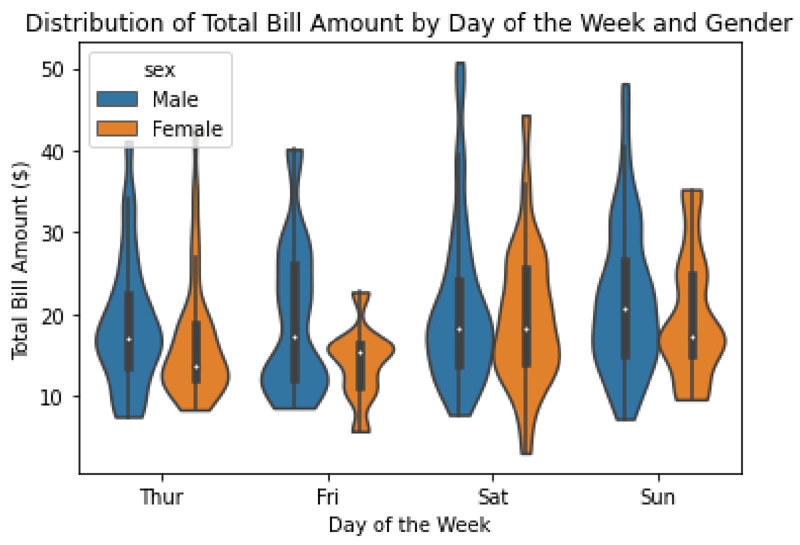


Figure 6.15: *Customized Violinplot with cut, bw and scale*

The bw parameter sets the width of the kernel used to estimate the density, while the cut parameter determines how far the violin extends past the extreme data points. Finally, the scale parameter determines how to scale the width of the violins, with “width” scaling them to the same width, and “area” scaling them to have the same area.

By adjusting these parameters, you can customize the appearance of the plot to better suit our needs.

Strip plot

A strip plot is a type of data visualization that displays the distribution of a dataset by plotting individual data points along a horizontal or vertical axis. In a strip plot, each data point is represented by a dot or a small vertical/horizontal line.

Here’s an example of how to create a strip plot using Seaborn’s **sns.stripplot()** function on the tips dataset:

```
import seaborn as sns

import matplotlib.pyplot as plt

# Load the tips dataset

tips = sns.load_dataset("tips")
```

```
# Create a strip plot using Seaborn
sns.stripplot(x="day", y="total_bill",
              data=tips, jitter=True, hue="sex", palette="Set2")

# Set the title and axis labels of the plot
plt.title("Total Bill Amount by Day of the Week and Gender")
plt.xlabel("Day of the Week")
plt.ylabel("Total Bill Amount ($)")
# Show the plot
plt.show()
```

Output:

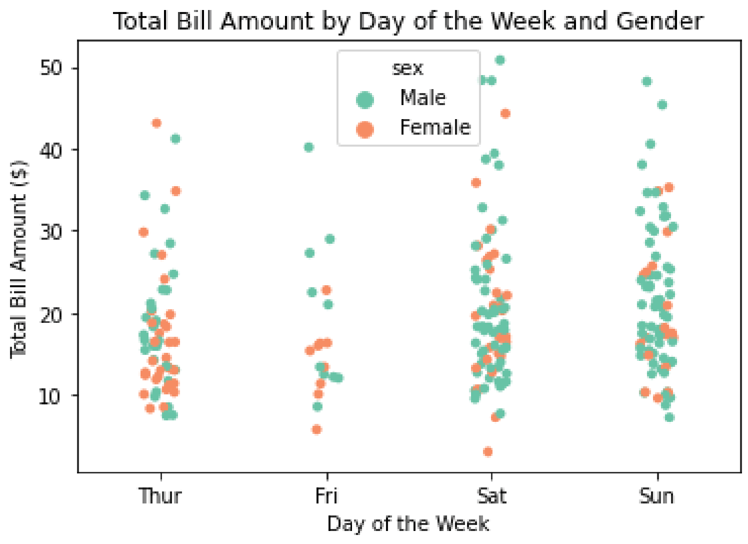


Figure 6.16: *Stripplot*

Here's an explanation of the parameters used in the **sns.stripplot()** function:

- **x="day"**: This sets the x-axis of the plot to the "day" column of the tips dataset.
- **y="total_bill"**: This sets the y-axis of the plot to the "total_bill" column of the tips dataset.
- **data=tips**: This sets the dataset to be used for creating the plot.

- **jitter=True**: This adds a small amount of random noise to the position of each data point to avoid overlapping points.
- **hue=”sex”**: This creates a grouping variable based on the “sex” column of the tips dataset. In this case, the dots are colored differently based on the gender of the customers.
- **palette=”Set2”**: This sets the color palette to be used for coloring the dots based on gender.

Here are some additional parameters that can be used in the **sns.stripplot()** function:

- **size**: This parameter sets the size of the dots representing each data point. By default, it is set to 5, but it can be set to other values to adjust the size.
- **alpha**: This parameter sets the transparency of the dots representing each data point. By default, it is set to 1, but it can be set to other values between 0 and 1 to adjust the transparency.
- **linewidth**: This parameter sets the width of the line around each dot representing each data point. By default, it is set to 0, but it can be set to other values to adjust the line width.
- **dodge**: This parameter specifies whether the dots for each group should be positioned closely together or with a small gap between them.
- **order**: This parameter specifies the order of the categories on the x-axis. By default, the categories are sorted in ascending order.

Swarm plot

A swarm plot is a type of data visualization that displays the distribution of a dataset by plotting individual data points along a horizontal or vertical axis, while avoiding overlapping points. In a swarm plot, each data point is represented by a dot or a small vertical/horizontal line, and the positions of the points are adjusted to avoid overlapping.

Here's an example of how to create a swarm plot using **Seaborn's sns.swarmplot()** function on the tips dataset:

```
import seaborn as sns

import matplotlib.pyplot as plt

# Load the tips dataset

tips = sns.load_dataset("tips")

# Create a swarm plot using Seaborn

sns.swarmplot(x="day", y="total_bill",

              data=tips, hue="sex", palette="Set2")


# Set the title and axis labels of the plot

plt.title("Total Bill Amount by Day of the Week and Gender")

plt.xlabel("Day of the Week")

plt.ylabel("Total Bill Amount ($)")

# Show the plot

plt.show()
```

Output:

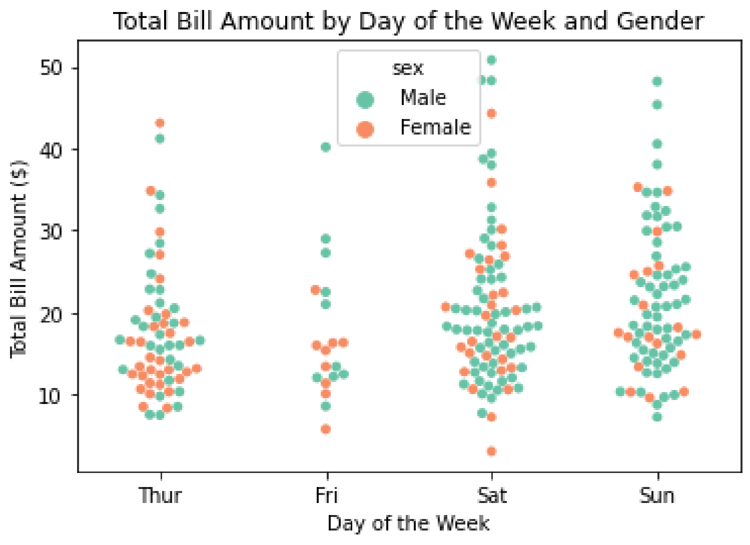


Figure 6.17: *Swarm plot with swarplot()*

Here's an explanation of the parameters used in the **sns.swarmplot()** function:

- **x=”day”**: This sets the x-axis of the plot to the “day” column of the tips dataset.
- **y=”total_bill”**: This sets the y-axis of the plot to the “total_bill” column of the tips dataset.
- **data=tips**: This sets the dataset to be used for creating the plot.
- **hue=”sex”**: This creates a grouping variable based on the “sex” column of the tips dataset. In this case, the dots are colored differently based on the gender of the customers.
- **palette=”Set2”**: This sets the color palette to be used for coloring the dots based on gender.

Here are some additional parameters that can be used in the **sns.swarmplot()** function:

- **size**: This parameter sets the size of the dots representing each data point. By default, it is set to 5, but it can be set to other values to adjust the size.
- **alpha**: This parameter sets the transparency of the dots representing each data point. By default, it is set to 1, but it can be set to other values between 0 and 1 to adjust the transparency.
- **linewidth**: This parameter sets the width of the line around each dot representing each data point. By default, it is set to 0, but it can be set to other values to adjust the line width.
- **dodge**: This parameter specifies whether the dots for each group should be positioned closely together or with a small gap between them.
- **order**: This parameter specifies the order of the categories on the x-axis. By default, the categories are sorted in ascending order.

We set the dodge parameter to True, which creates a small gap between the dots for each group (male and female). This makes it easier to see the distribution of the data for each group. We also set the order parameter to a list of the days of the week in the desired order (Sunday, Saturday, Friday, Thursday). This specifies the order in which the days of the week are displayed on the x-axis.

```
sns.swarmplot(x="day", y="total_bill", data=tips,
          hue="sex", palette="Set2", dodge=True,
          order=["Sun", "Sat", "Fri", "Thur"])
```

Output:

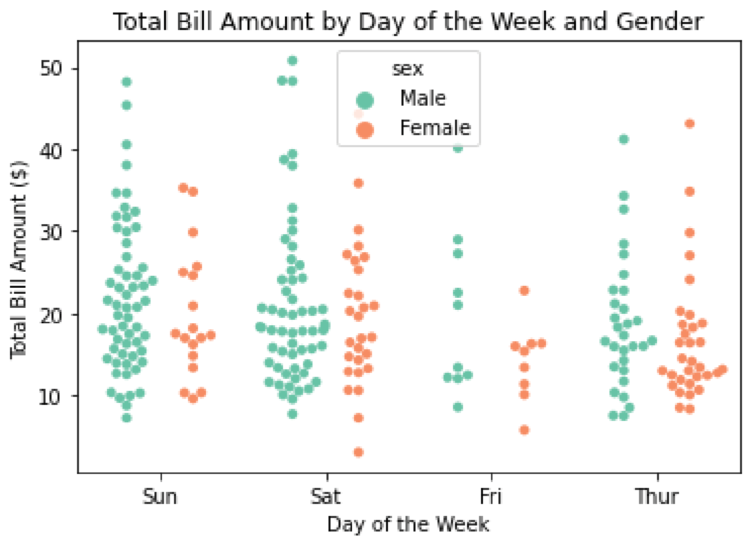


Figure 6.18: *Customized Swamp plot with dodge set as True*

You can adjust the values of dodge and order to suit your needs, depending on the structure of your data and the goals of your visualization.

Note that unlike a strip plot, a swarm plot ensures that each data point is visible and does not overlap with other data points. However, it can be computationally expensive for larger datasets, so it may not be the best choice for very large datasets.

Cat plot

Catplot() is a powerful function in Seaborn that creates various categorical plots. It can be used to create a variety of plots like scatter plots, strip plots, swarm plots, box plots, violin plots, and many others. The **catplot()** function provides a unified interface to all of these plots, allowing you to create complex visualizations with just a few lines of code.

Here's an example of how to use the **catplot()** function to create a box plot on the tips dataset:

```
import seaborn as sns

import matplotlib.pyplot as plt

# Load the tips dataset

tips = sns.load_dataset("tips")

# Create a box plot using Seaborn

sns.catplot(x="day", y="total_bill",
            hue="sex", kind="box", data=tips)

# Show the plot

plt.show()
```

Output:

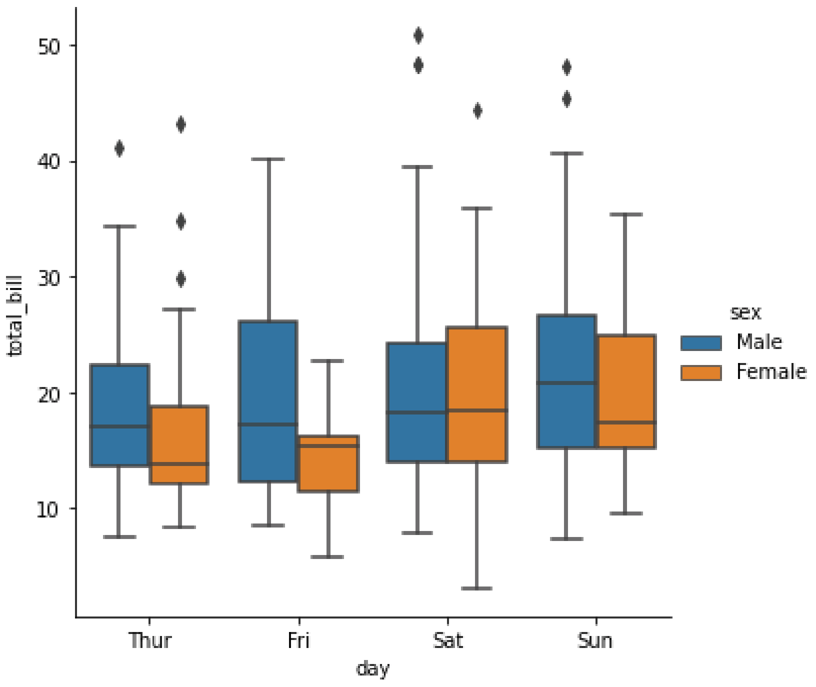


Figure 6.19: *Category plot with catplot()*

In this example, we create a box plot that shows the distribution of total bills by day of the week and gender. The x parameter specifies the variable to be plotted on the x-axis (day), and the y parameter specifies the variable to be plotted on the **y-axis (total_bill)**. The hue parameter specifies the variable to be used for grouping the data (sex). Finally, the kind parameter specifies the type of plot to be created (box).

The kind parameter in catplot is used to specify the type of categorical plot to create. The kind parameter accepts several values, including:

- “**strip**”: This creates a scatter plot with one variable categorical.
- “**swarm**”: This is similar to “strip”, but it avoids overlapping points by adjusting their positions.
- “**box**”: This creates a box plot that shows the distribution of a quantitative variable across different categories.
- “**violin**”: This creates a violin plot that combines the box plot and kernel density estimation.
- “**boxen**”: This creates a boxen plot, which is an improved version of the box plot that can show more information about the distribution of the data.
- “**point**”: This creates a scatter plot with one variable categorical and different points are joined by lines.
- “**bar**”: This creates a bar plot that shows the count of observations in each category.
- “**count**”: This is similar to “bar”, but it shows the number of occurrences of each category.

```
sns.catplot(x="day", y="total_bill", hue="sex", kind="boxen", data=tips)
```

Output:

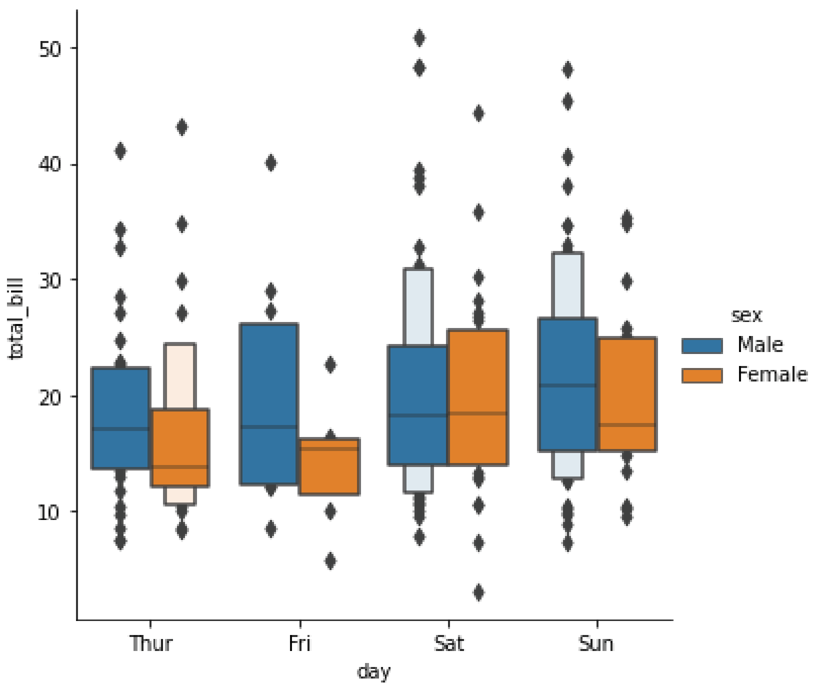


Figure 6.20: *Category plot with catplot() kind set to boxen*

Here are some additional parameters that can be used in the **catplot()** function:

- **col** and **row**: These parameters allow you to create a grid of plots based on the values of one or more variables. For example, you could create a grid of box plots for each day of the week, with separate columns for male and female customers, by setting **col=”sex”** and **row=”day”**.
- **palette**: This parameter sets the color palette to be used for the plot.
- **order** and **hue_order**: These parameters allow you to specify the order of the categories on the x-axis and the grouping variable, respectively.
- **height** and **aspect**: These parameters allow you to adjust the size and aspect ratio of the plot.
- **jitter**, **dodge**, and **bw**: These parameters control the behavior of certain types of plots, like strip plots, swarm plots, and violin plots.

Overall, the **catplot()** function provides a powerful and flexible way to create a wide range of categorical plots. By changing the kind parameter and adjusting the other parameters, you can easily create different types of plots and explore your data in different ways.

FacetGrid

Seaborn’s **FacetGrid** function is a tool used for visualizing data on multiple subsets of the dataset. **FacetGrid** is a categorical plot in Seaborn. It allows you to create a grid of plots based on the levels of one or more categorical variables. This is useful when you want to compare the distribution of a variable across different categories. It allows for the creation of multiple plots, each showing a different subset of the data, based on one or more categorical variables. The **FacetGrid** function takes in several parameters, including the dataset, the variables to use for the rows and columns of the grid, and the plot type to use for each cell of the grid. The hue parameter can also be used to create different plots for different levels of a categorical variable. Once the **FacetGrid** object is created, additional plotting functions can be applied to it to create the actual plots. This allows for the creation of complex, multi-layered plots that show the relationships between multiple variables. FacetGrid is a powerful tool that allows you to create multi-plot grids, where each subplot displays a subset of the data.

Facet_kws is a dictionary of keyword arguments that are passed to the FacetGrid object. Here are some of the most commonly used arguments you can use in this parameter:

- **margin_titles**: If set to True, adds a title to each row and column of the grid. The title is centered above or below the subplot and includes the corresponding row or column label.
- **space**: The amount of space between the subplots, in inches. By default, seaborn adjusts the spacing automatically based on the size of the plots and the number of rows and columns in the grid.
- **sharex**: If set to False, each subplot will have its own x-axis. If set to True or “col”, the subplots in each column will share the same x-axis. If set to “row”, the subplots in each row will share the same x-axis.
- **sharey**: Same as sharex, but for the y-axis.

```
rimport seaborn as sns
import matplotlib.pyplot as plt
# Load the tips dataset
```

```
tips = sns.load_dataset("tips")
# Create a grid of scatter plots, one for each day of the week
grd = sns.FacetGrid(data=tips, col="day", col_wrap=2,)
# Add the scatter plots to the grid
grd = grd.map(sns.scatterplot, "total_bill", "tip")
# Add titles to each subplot
grdr = grd.set_titles("{col_name}")
# Add margin titles to the grid
grd = grd.set_axis_labels("Total Bill", "Tip")
# Show the plot
plt.show()
```

Output:

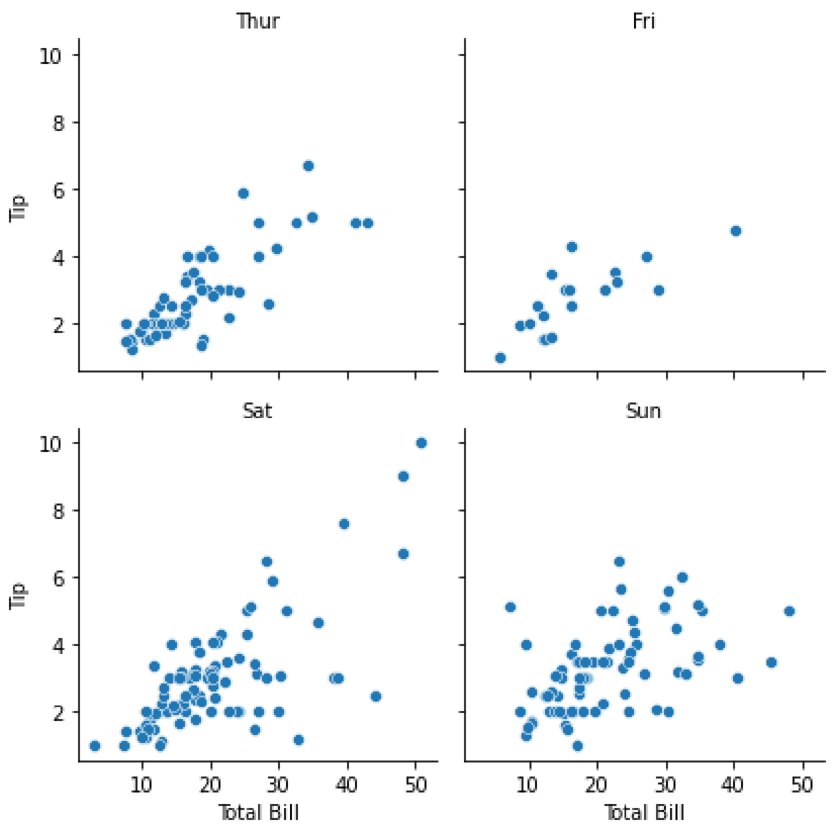


Figure 6.21: *Grid of Categorical plots with facetgrid()*

In this code, FacetGrid is used to create a grid of scatter plots, with one plot for each day of the week. You can use **col=”day”** to specify that we want to group the data by the day column in the tips dataset, and **col_wrap=2** to wrap the columns into multiple rows when there are more than two columns. Notify the use of map to add a scatter plot to each subplot in the grid, and **set_titles** to add titles to each subplot based on the corresponding column label **({col_name})**.

Other plots

Seaborn offers a wide range of plot functions beyond categorical plots. Some of the commonly used non-categorical plots functions in Seaborn include:

- **Scatter plots – sns.scatterplot()**: Used to visualize the relationship between two continuous variables.
- **Line plots – sns.lineplot()**: Used to visualize changes in a continuous variable over time or other continuous variable.
- **Heatmaps – sns.heatmap()**: Used to visualize the correlation or relationship between variables in a matrix format.
- **Joint plots – sns.jointplot()**: Used to visualize the relationship between two variables with both a scatter plot and a histogram.
- **Pair plots – sns.pairplot()**: Used to visualize the pairwise relationships between multiple variables in a dataset.
- **Distribution plots – sns.distplot()**: Used to visualize the distribution of a single variable, such as a histogram, KDE (Kernel Density Estimate), or rug plot.
- **Regression plots – sns.regplot()**: Used to create a regression plot

There are many more plot functions available in Seaborn, and the choice of which to use depends on the data and the questions you want to answer.

Each function has different parameters that you can use to customize the plot to your needs, such as the x-axis and y-axis variables, the color palette, the size of the plot, and the font size of the labels. In addition to these functions, seaborn also provides other utilities for data visualization, such as functions to customize the style of the plots **(sns.set_style())**, to add annotations to the plots **(sns.text(), sns.annotate())**, and to create faceted plots **(sns.FacetGrid(), sns.catplot(), sns.relplot())**.

Scatter plot

sns.scatterplot() is a seaborn function used to create a scatter plot. A scatter plot is a type of plot that displays the relationship between two continuous variables. Each point in the plot represents the value of the two variables for one observation in the data. It can be used to display the relationship between two numerical variables by plotting them as points on a two-dimensional plane. Here's an explanation of all the parameters that can be used with **sns.scatterplot():**

```
sns.scatterplot(
  x=None, y=None, hue=None, style=None, size=None,
  data=None, palette=None, hue_order=None, hue_norm=None,
  sizes=None, size_order=None, size_norm=None, markers=True,
  style_order=None, x_bins=None, y_bins=None, units=None,
  estimator=None, ci=95, n_boot=1000, alpha=None, x_jitter=None,
  y_jitter=None, legend='auto', ax=None, **kwargs)
```

- **x** and **y**: The variables to be plotted on the x-axis and y-axis, respectively. These can be column names or arrays of values.
- **hue**: A categorical variable that is used to color-code the points. This can be a column name or array of values.
- **style**: Another categorical variable that is used to differentiate the points by their markers. This can be a column name or array of values.
- **size**: A numerical variable that is used to adjust the size of the markers. This can be a column name or array of values.
- **data**: The DataFrame containing the variables to be plotted.
- **palette**: The color palette to use for the hue variable. This can be a string (for example, "Set1") or a list of colors.

- **hue_order**: The order in which to display the levels of the hue variable.
- **hue_norm**: The normalization to use when mapping the hue variable to colors.
- **sizes**: The sizes to use for the markers, either as a list of sizes or a tuple of min and max sizes.
- **size_order**: The order in which to display the levels of the size variable.
- **size_norm**: The normalization to use when mapping the size variable to sizes.
- **markers**: A list or dictionary of markers to use for each level of the style variable.
- **style_order**: The order in which to display the levels of the style variable.
- **x_bins** and **y_bins**: The number of bins to use for aggregating the data before plotting.
- **units**: A variable that specifies groups of observations that should be treated as separate points (for example, when plotting time series data for multiple individuals).
- **estimator**: The function to use for aggregating the data within each bin. This can be a string (for example, “mean”, “median”) or a function.
- **ci**: The size of the confidence interval to draw around the estimate.
- **n_boot**: The number of bootstrap samples to use when computing the confidence interval.
- **alpha**: The transparency of the markers.
- **x_jitter** and **y_jitter**: The amount of random noise to add to the x and y coordinates, respectively, to reduce overplotting.
- **legend**: Whether to display a legend for the hue and style variables.
- **ax**: The Axes object to draw the plot onto.
- ****kwargs**: Additional arguments to pass to the underlying matplotlib scatter plot function.

For all details, you can refer to the documentation of seaborn library. However, producing an example below for your understanding of using **sns.scatterplot()** with some useful parameters:

```
import seaborn as sns
import matplotlib.pyplot as plt
# Load the tips dataset
tips = sns.load_dataset("tips")

# Create a scatter plot of total_bill vs tip
sns.scatterplot(x="total_bill", y="tip", data=tips,
                hue="sex", style="time",
                size="size", alpha=0.8)
# Set the title and axes labels
plt.title("Total Bill vs Tip")
plt.xlabel("Total Bill")
plt.ylabel("Tip")
# Customize the legend
legend_labels = ["Male - Lunch", "Male - Dinner",
                 "Female - Lunch", "Female - Dinner"]
plt.legend(title="Legend", labels=legend_labels,
           loc="upper left", frameon=True)
# Show the plot
plt.show()
```

Output:

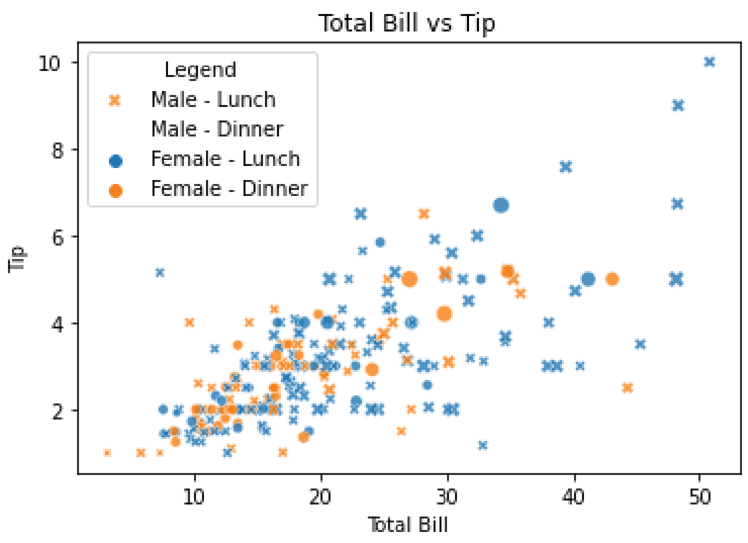


Figure 6.22: *Scatter plot on total bill vs tips from tips dataset*

In this example, we're creating a scatter plot of the '**total_bill**' and '**tip**' variables from the tips dataset, colored by 'sex' and differentiated by time. Also the size variable is used to adjust the size of the markers, and setting the transparency to 0.8 using alpha.

Also the title and axes labels are included using **plt.title()**, **plt.xlabel()**, and **plt.ylabel()**.

Finally, the legend is customized by specifying the labels for each level of the hue and style variables using the labels parameter of **plt.legend()**, and setting the location to "upper left" using loc. Also, legend title is included by using title, and setting 'frameon' to True to draw a box around the legend.

Line plot

sns.lineplot() is a function in the Seaborn data visualization library that creates a line plot. It can be used to visualize the relationship between two numerical variables by plotting them as connected line segments. Most of the parameters of **sns.lineplot()** function are similar to that of scatterplot function. The omens which are different are mentioned with red color and the explanation is provided below the syntax.

```
sns.lineplot(x=None, y=None, hue=None, size=None, style=None,
             data=None, palette=None, hue_order=None, hue_norm=None,
```

sizes=None, size_order=None, size_norm=None, dashes=True,

markers=None, style_order=None, units=None, estimator='mean',

ci=95, n_boot=1000, sort=True, err_style='band', err_kws=None,

legend='auto', ax=None, **kwargs)

- **x** and **y**: The variables to be plotted on the x-axis and y-axis, respectively. These can be column names or arrays of values.
- **dashes**: A boolean or list of dash styles to use for the lines.
- **estimator**: The function to use for aggregating the data within each group. This can be a string (for example, "mean", "median") or a function. Note the default value is ‘mean’
- **n_boot**: The number of bootstrap samples to use when computing the confidence interval.
- **sort**: Whether to sort the x-axis by the variable being plotted.
- **err_style**: The style to use for drawing the confidence interval around the estimate.
- **err_kws**: Additional keyword arguments to pass to the errorbar function when drawing the confidence interval.

Here's an example of using **sns.lineplot()** with some useful parameters:

import seaborn as **sns**

import **matplotlib.pyplot** as **plt**

```
 # Load the tips dataset
tips = sns.load_dataset("tips")

# Create a bar plot of total bill by day of the week
sns.lineplot(x="day", y="total_bill", data=tips,
            hue='sex',hue_order=(['Male','Female']),alpha=0.8)
# Add labels to the plot
plt.xlabel("Day of the Week")
plt.ylabel("Total Bill")
plt.title("Total Bill by Day of the Week")
plt.show()
```

Output:

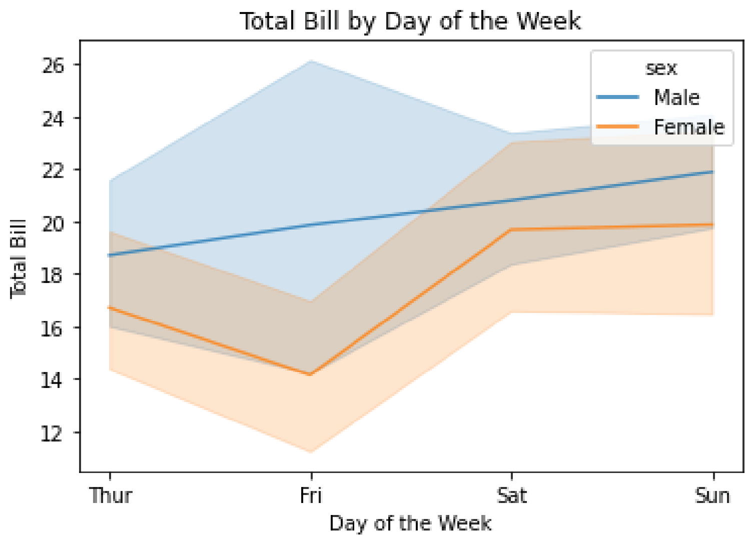


Figure 6.23: *Line plot on total bill with hue as sex hue_order set as [Male, Female]*

The code creates a line plot using the **sns.lineplot()** function. The "day" column of the "tips" dataset is used as the x-axis variable, while the "**total_bill**" column is used as the y-axis variable. Additionally, the plot is colored according to the "sex" column, and the **hue_order** argument is used to specify the order in which the different categories of the "sex" column should be plotted. The plot is labeled using the **plt.xlabel()**, **plt.ylabel()**, and **plt.title()** functions. The resulting plot shows the total bill amount for each day of the week, with different colors for male and female payers. The plot helps to identify any patterns or trends in the total bill amount based on the day of the week and sex of the payer.

In the example below, we are trying to create the line plot on **total_bill** vs tip, with separate lines for each gender (Male and Female). The **hue_order** parameter is used to specify the order in which the colors are used for the lines. The resulting plot should show the relationship between the two variables for each gender separately.

```
# Create a line plot of total_bill vs tip, with separate lines for each gender
```

```
sns.lineplot(x="total_bill", y="tip", hue="sex",
             hue_order=["Male", "Female"], data=tips)
# Add a title and axis labels
plt.title("Tip vs Total Bill by Gender")
plt.xlabel("Total Bill")
plt.ylabel("Tip")
# Show the plot
plt.show()
```

Output:

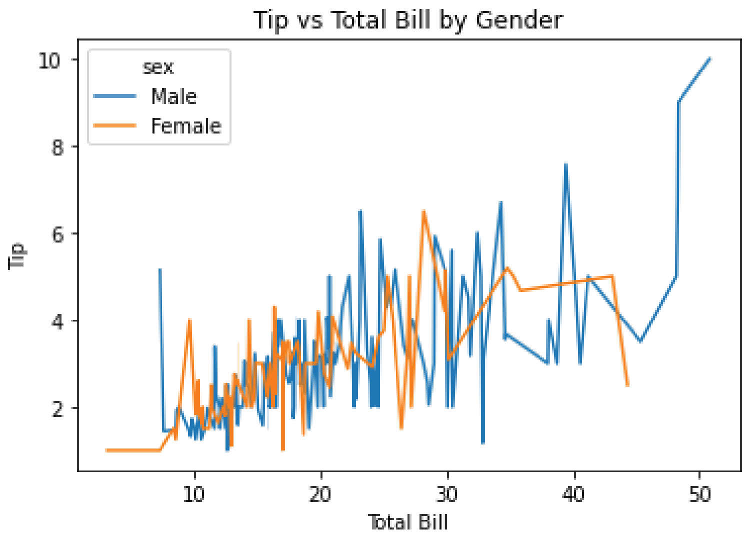


Figure 6.24: *Line plot on tips obtained on total bill amount with hue set as sex*

Heatmap

A heatmap is a graphical representation of data that uses a color-coding scheme to represent different values. Heatmaps are commonly used to display large datasets and can be used to identify patterns and trends in the data. You have seen the use of heatmap() earlier in the palette section of this chapter. Below is the example of developing the heatmap to visualize the correlation between variables in the "tips" dataset.

Note that the correlation is a statistical measure that indicates the degree of association between two variables. It is commonly used to measure the strength and direction of the relationship between two variables. You will learn more on this in the last chapter of the book. As of now, understand that the correlation values range from -1 to 1, where -1 indicates a perfect negative correlation, 1 indicates a perfect positive correlation, and 0 indicates no correlation.

```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas
#Fetching tips dataset from the seaborn
tips = sns.load_dataset("tips")
#creating correlation matrix on whole dataset
corr_matrix = tips.corr()
#Developing heatmap plot on correlation matrix
sns.heatmap(corr_matrix, annot=True, cmap="YlGnBu")
plt.show()
```

Output:

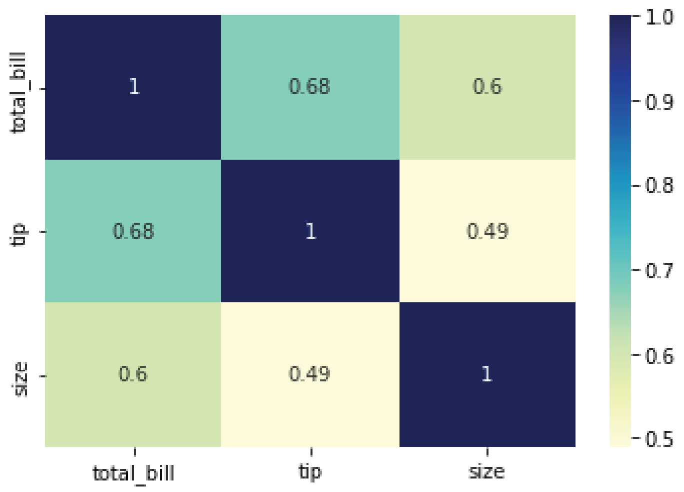


Figure 6.25: *Heatmap representing the correlation on tips dataset*

For calculating the correlation, **corr()** method of the pandas DataFrame object is used on the dataset. The resulting correlation matrix is then passed as input to the **sns.heatmap()** function to create a heatmap. The **annot=True** parameter is used to display the correlation values on the heatmap. Identify the use of the **sns.color_palette()** function to create a custom color palette and pass it as a parameter to **cmap**; the **cmap="YlGnBu"** parameter is used to specify the color map to be used for the heatmap. In this case, we are using the "**YlGnBu**" color map, which is a sequential color map that goes from yellow (low correlation) to green (high correlation) to blue (negative correlation). Finally, the **plt.show()** function is called to display the heatmap.

Note that even if the whole dataset is used to create the correlation matrix, the matrix is developed only on ‘numeric’ attributes.

Joint plot

A joint plot is a type of plot that combines two different types of plots: a scatter plot and a histogram. It is a useful tool for exploring the relationship between two variables, as it shows the distribution of each variable separately, as well as their correlation.

To create a joint plot in Python, we can use the seaborn library, which provides a **jointplot()** function. Here's an example of how to use the **jointplot()** function to create a joint plot between two variables:

```
import seaborn as sns

import matplotlib.pyplot as plt

#loading data

tips = sns.load_dataset("tips")

#creating join on total_bill and tip

sns.jointplot(x="total_bill", y="tip", data=tips)

plt.show()
```

Output:

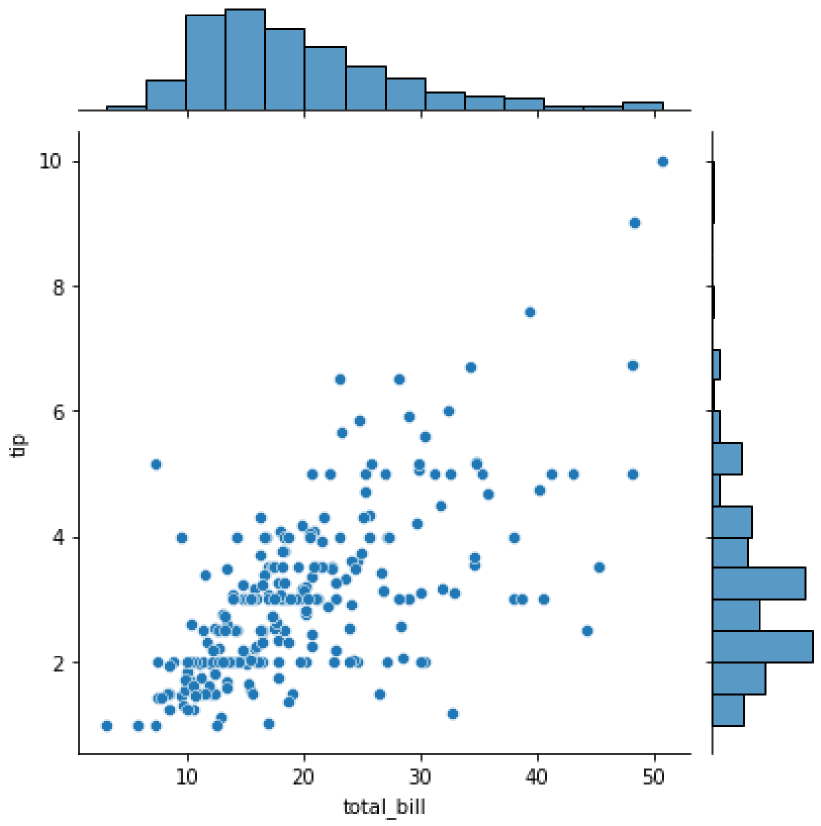


Figure 6.26: *Joint plot on tip and total bill from tips dataset*

In the example we are trying to create a joint plot between the "**total_bill**" and "**tip**" variables from the "**tips**" dataset. The x parameter specifies the variable to be plotted on the x-axis, while the y parameter specifies the variable to be plotted on the y-axis.

The resulting plot displays a scatter plot of the two variables, with the histogram of each variable shown on the side. The scatter plot shows the correlation between the two variables, while the histograms show the distribution of each variable separately. You can customize the appearance of the joint plot by specifying additional parameters to the **jointplot()** function, such as the kind parameter to specify the type of plot to be displayed (for example "**scatter**", "**kde**", "**hex**"), or the color parameter to specify the color of the plot. The code below adds a regression line to show the trend between the two variables.

```
# Create a joint plot between "total_bill" and "tip" with a regression line

sns.jointplot(data=tips, x="total_bill", y="tip",kind='reg')
```

This will create a joint plot with a regression line, which shows the trend between the "**total_bill**" and "**tip**" variables. You can use this plot to analyze how the tip amount changes as the total bill increases. Please refer to the following figure:

Output:

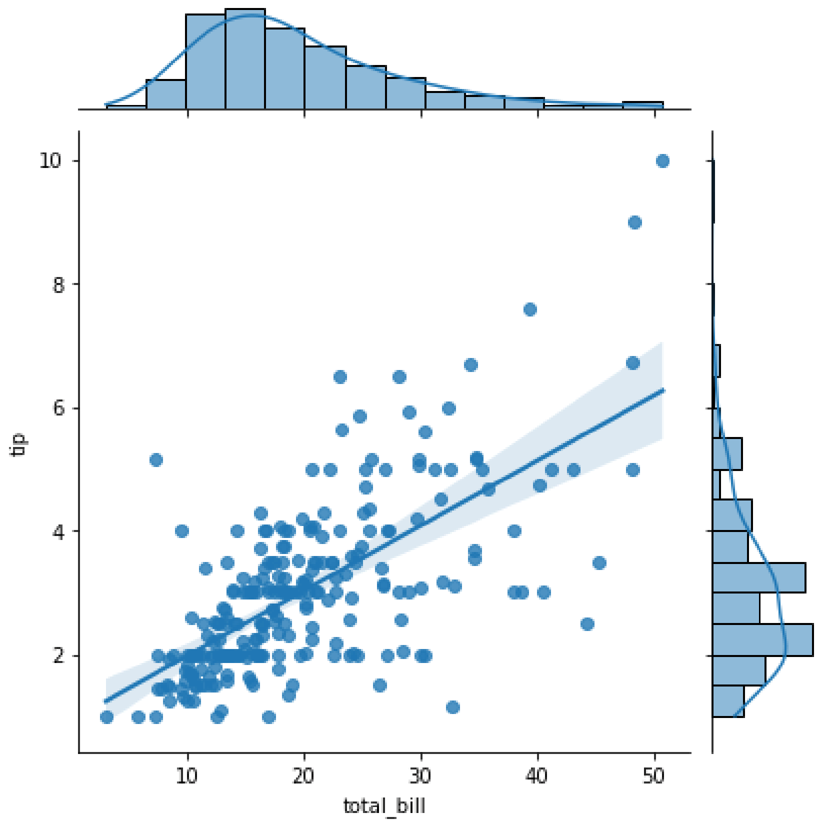


Figure 6.27: *Joint plot on tip and total bill with regression line showing the correlation between two features*

You can also specify the color of the regression line different from the other plots in joinplot, for this you need to specify the **line_kws** argument. The **line_kws** parameter accepts a dictionary of keyword arguments that will be passed to the matplotlib function used to draw the regression line, in this case, the **plt.plot()** function. Let's set the "**color**" key to "red" to change the color of the line to red. (Note that color parameter of **joinplot()** is used for specifying the color of the whole graph)

# Create a joint plot between "total_bill" and "tip"

#with a regression line and with different plot and line color

sns.jointplot(data=tips, x=”total_bill”, y=”tip”,color =’Purple’,kind=’reg’, line_kws={‘color’:’red’})

Output:

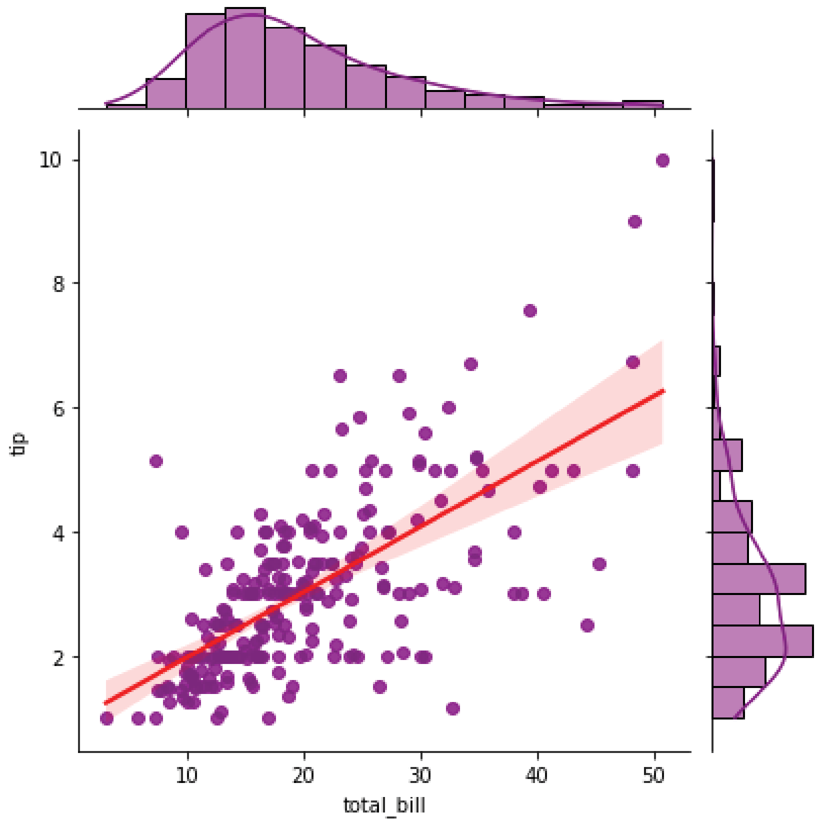


Figure 6.28*: Customized joint plot with different line and plot colors*

Furthermore, you can also specify the color for the outer histogram display in joinplot by using **marginal_kws** parameter, for instance to display it in red color mention **marginal_kws={color="red"}**

# Create a joint plot between "total_bill" and "tip"

#with a regression line and with different plot

#and line color a diffrent histogram color

sns.jointplot(data=tips, x="total_bill", y="tip",color ='Purple',kind='reg', line_kws={'color':'red'}, marginal_kws = {'color':'yellow'})

Output:

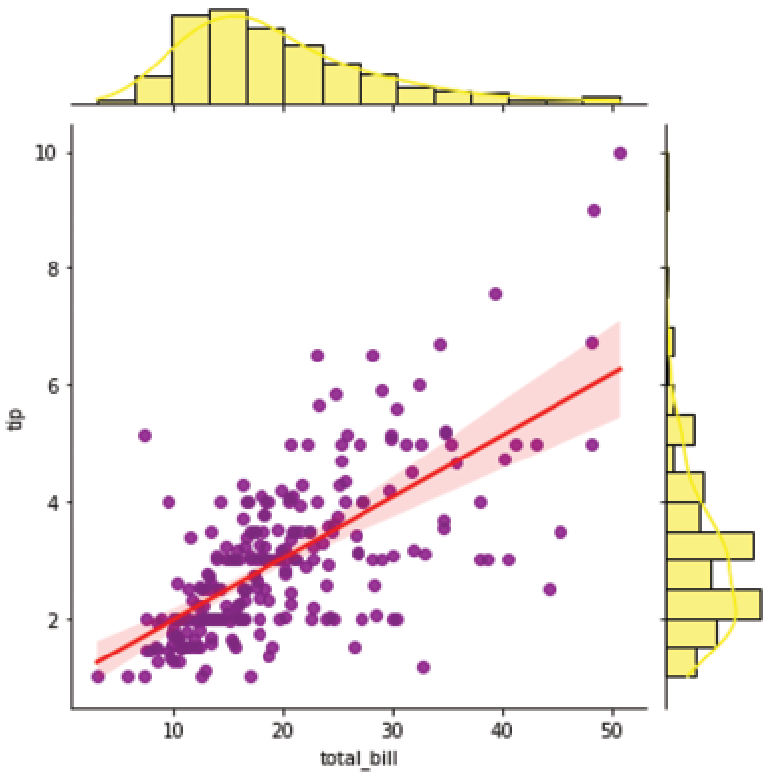


***Figure 6.29**: Customized joint plot with different plot colors – purple for scatter and yellow for histograms*

Note that you can also set some other parameters to customize the jointplot, such as height, ratio, and space.

Pair plot

Pairplot is a type of visualization in which pairwise relationships among multiple variables are shown in a grid format. It is essentially a combination of scatter plots for each pair of variables and histograms for each individual variable. It allows us to quickly identify any patterns or correlations between the variables.

```
import seaborn as sns

import matplotlib.pyplot as plt


# Load the tips dataset

tips = sns.load_dataset("tips")

# Create a pairplot
```

```
sns.pairplot(tips)
plt.show()
# Create a pairplot with KDE at diagnol
sns.pairplot(tips, diag_kind ='kde')
plt.show()
```

Output:

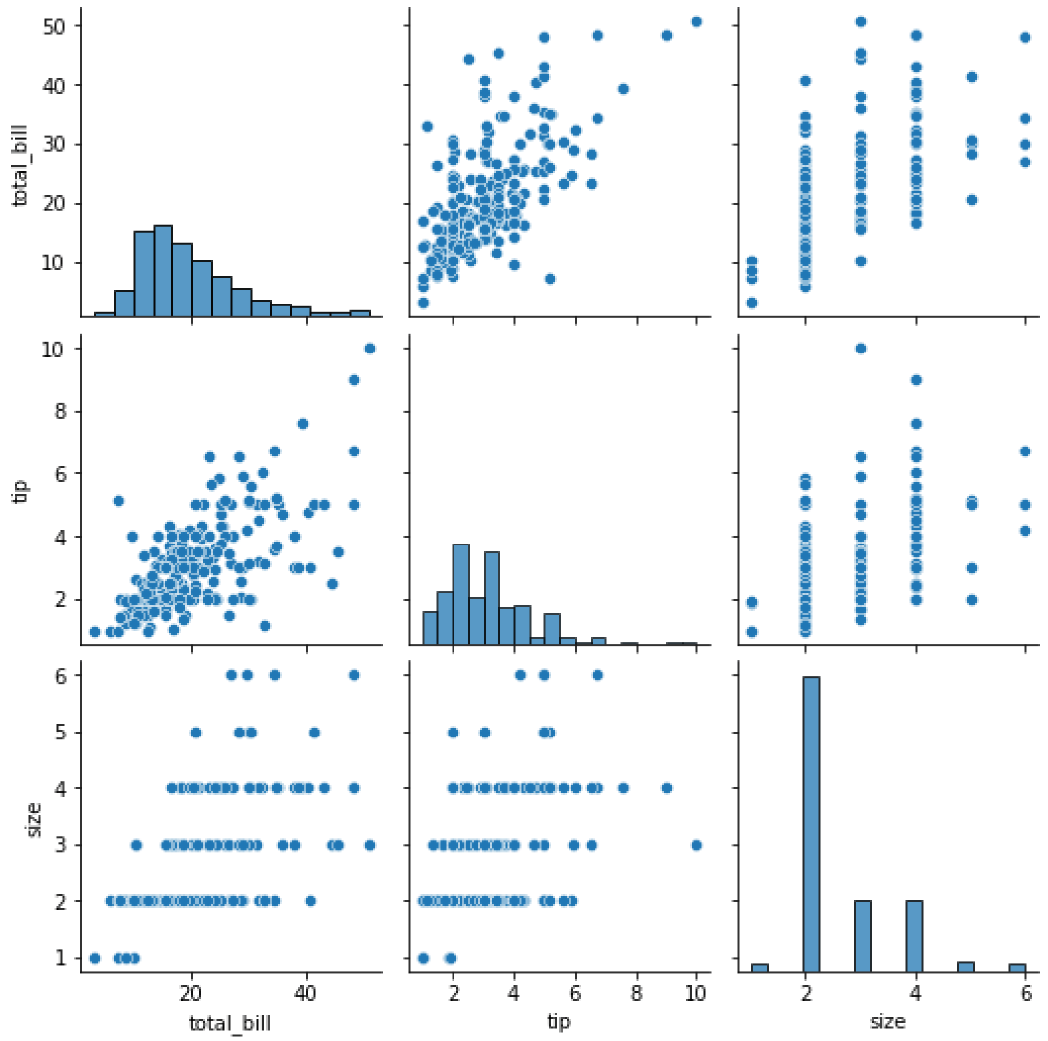


***Figure 6.30**: Pair plot*

By default, the diagonal plots show a histogram of the values of each variable. However, you can change this to a different type of plot, such as a kernel density estimate or a rug plot. You can do this by specifying the "**diag_kind**" parameter.

```
# Create a pairplot with KDE at diagnol and regression plot for others

sns.pairplot(tips, kind='reg', diag_kind ='kde')

plt.show()
```

Output:

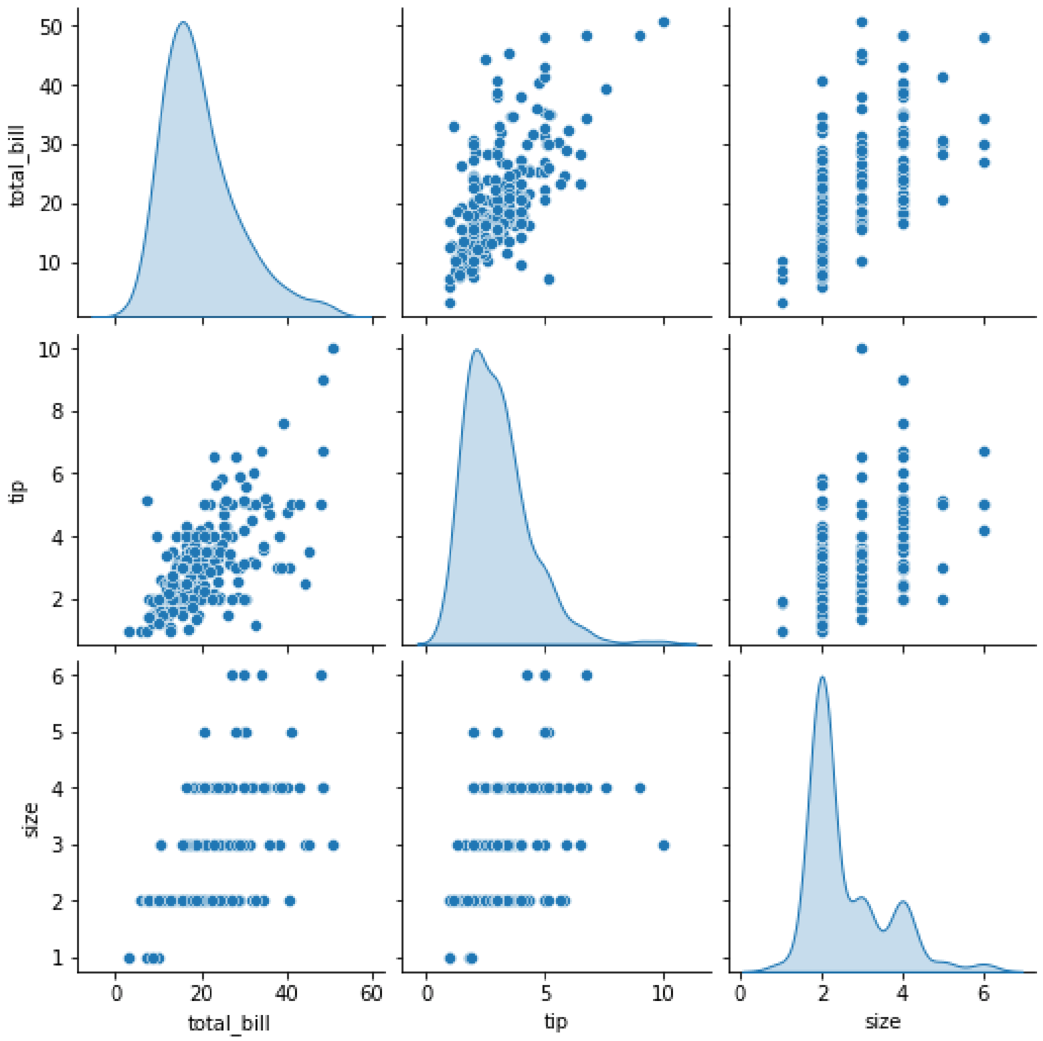


Figure 6:31*: Pair plot with diagonals as ‘kde’ plot*

By default, the off-diagonal plots show scatter plots of each combination of two numerical variables. We can change this to a different type of plot, such as a **hexbin** plot or a regression plot. You can do this by specifying the "**kind**" parameter.

```
# Create a pairplot with KDE at diagnol and regression plot for others
sns.pairplot(tips, kind='reg', diag_kind ='kde')
plt.show()
```

Output:

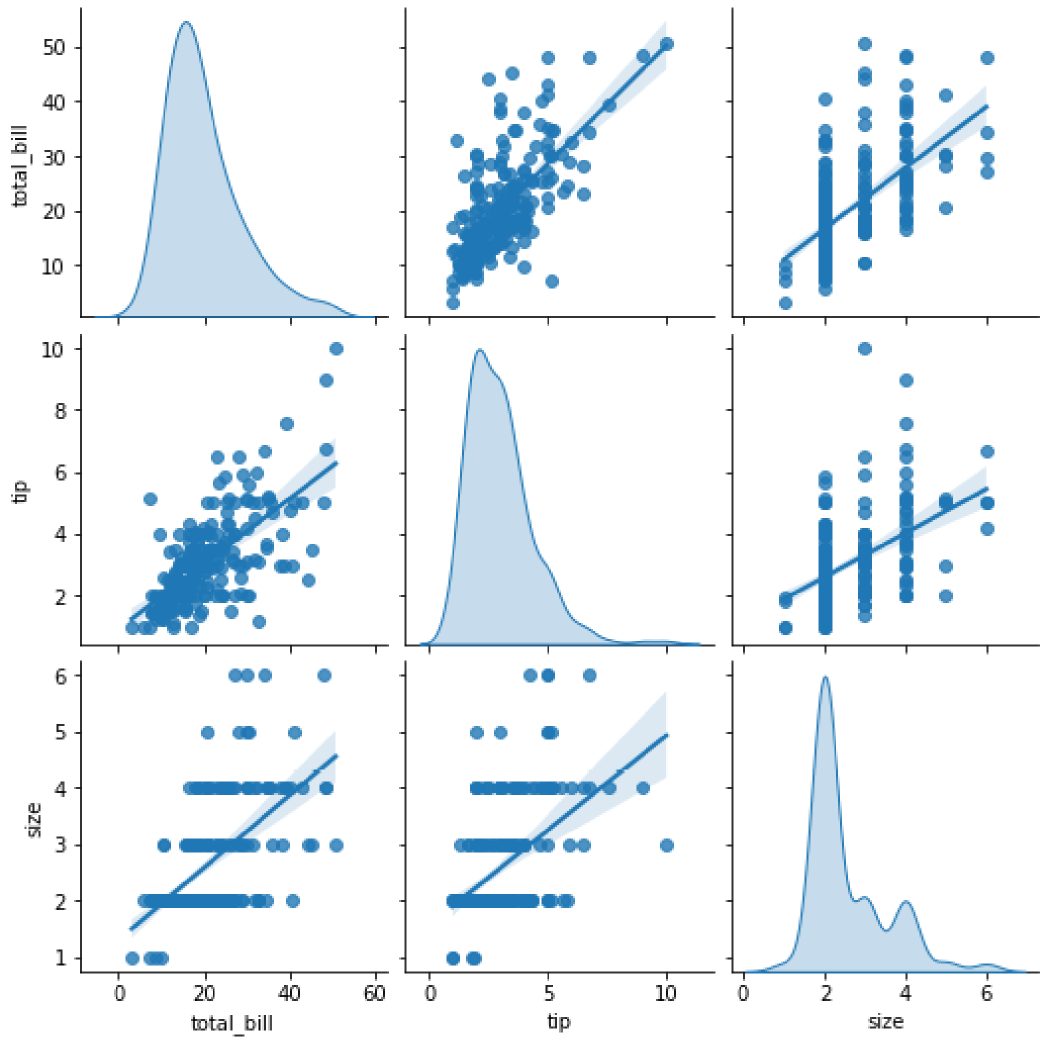


Figure 6:32: *Pair plot with regression line and 'kde' plot*

You can change the color palette of the plots by specifying a different color map using the "**palette**". You can use any color map for the plots from the palette. You can also change the labels of the x and y axes by passing a dictionary of labels to the "**vars**" parameter.

```
# Create a pairplot with vars
sns.pairplot(tips, kind = 'reg', diag_kind="kde",
        vars={"total_bill", "tip"}, x_vars=["total_bill"],
        y_vars=["tip"],plot_kws={'color':'green'})
plt.show()
```

Output:

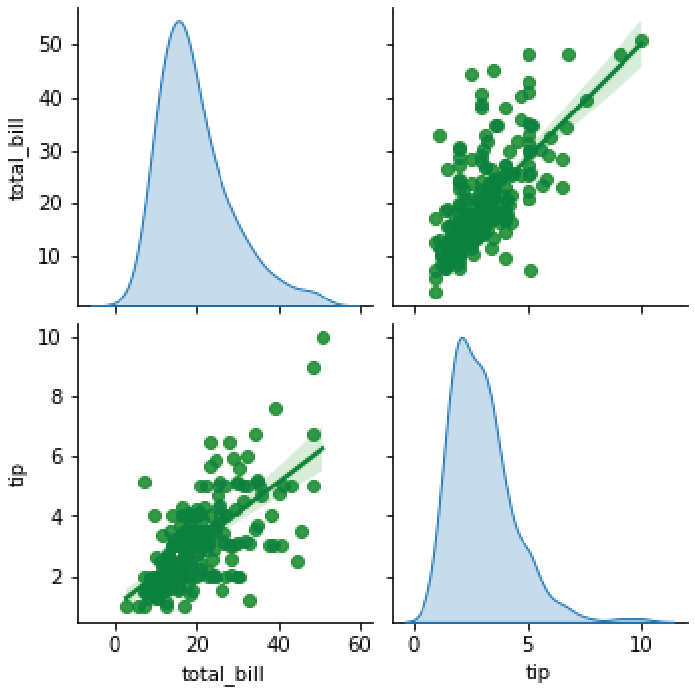


Figure 6:33*: Customized pair plot with color*

The **plot_kws** parameter is used to pass additional keyword arguments to the plotting function used to create the plots in the pairplot. For example, you can use this parameter to adjust the color, size and so on of the plots using **plot_kws** parameter. The hue parameter is used to specify a categorical variable to use for coloring the points in the pairplot. For example, if your dataset includes a categorical variable called species, you can use **hue='gender'** to create separate plots for each gender and color the points accordingly. The hue parameter can also be used with the palette parameter to specify a custom color palette to use for coloring the points. The corner parameter is used to specify if the plots should be drawn only in the upper diagonal (default), lower diagonal, or both diagonals of the pairplot. For example, **corner='lower'** will draw plots only in the lower diagonal.

```
# Create a pairplot with vars and hue_order
sns.pairplot(tips, kind = 'reg', diag_kind="kde",
            hue="sex", hue_order=['Male','Female'],
            vars={"total_bill", "tip"}, x_vars=["total_bill"],
            y_vars=["tip"],plot_kws={'color':'green'})
plt.show()
```

Output:

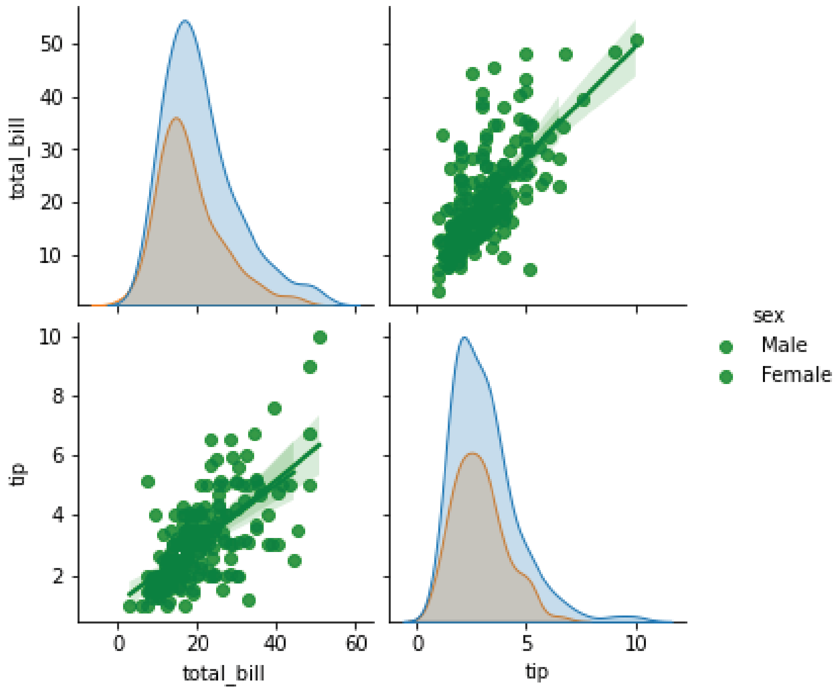


Figure 6.34: *Customized pairplot with vars and hue_order*

Distribution plot

One of the commonly used plots in Seaborn is the Distribution plot, which is used to visualize the distribution of a continuous variable. t is a univariate plot that shows the spread and shape of the data. A Distribution plot shows the distribution of a variable, along with the **Probability Density Function** (**PDF**), which represents the probability of observing a certain value for that variable. The distribution plot can also include a histogram of the data, which shows the number of observations that fall within each bin of the variable. Please refer to the following figure:

```
import seaborn as sns
import matplotlib.pyplot as plt
# load the tips dataset
tips = sns.load_dataset('tips')
# create a distribution plot of the total bill
sns.displot(tips['total_bill'], kde=True,color = 'purple',)
# show the plot
plt.show()
```

Output:

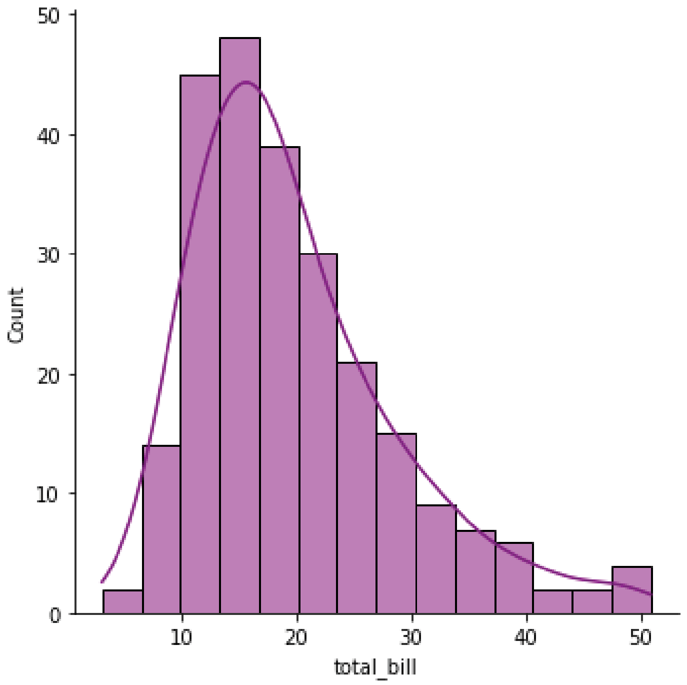


Figure 6.35*: Distribution plot on total bill from tips dataset*

In this example, first the tips dataset is loaded using **sns.load_dataset().** Then a distribution plot of the **total_bill** column is developed using the **sns.displot()**

function. Make a note here on the use of '**kde=True'** to show the probability density function along with the histogram. Finally, **plt.show()** is called to display the plot. Notice the value of the color parameter is set to 'purple'. The resulting plot shows the distribution of the **total_bill** variable, along with the PDF and histogram. You can see that the data is roughly normally distributed, with a peak around $15-20.

Let us try another code, here we are trying to create a **kernel density estimate** (**KDE**) plot of the **total_bill** column in the tips dataset, grouped by sex and day, and colored by time. you can '**log_scale=True**' to use a logarithmic scale on the x-axis, which can be useful when the data is highly skewed. Understand how **row_order** is used, with values mentioned as **["Thur", "Fri", "Sat", "Sun"]** to specify the order of the rows in the plot. You can make use of **facet_kws=dict(margin_titles=True)** to add titles to each row and column of the plot, which helps to identify the groups of data being displayed. You can also use the '**col_wrap**' parameter to wrap the columns into multiple rows when there are more than specified columns.

```
# Create a distribution plot with row_order, and log_scale
displ = sns.displot(data=tips, x="total_bill", col="sex",
    row="day", hue="time", kind="kde",log_scale=True,
    row_order=["Thur", "Fri", "Sat", "Sun"],
    facet_kws=dict(margin_titles=True),)
# Add titles and axis labels
displ.fig.suptitle("Distribution of Total Bill by Day and Sex")
displ.set_xlabels("Total Bill")
displ.set_ylabels("Density")
# Show the plot
plt.show()
```

Output:

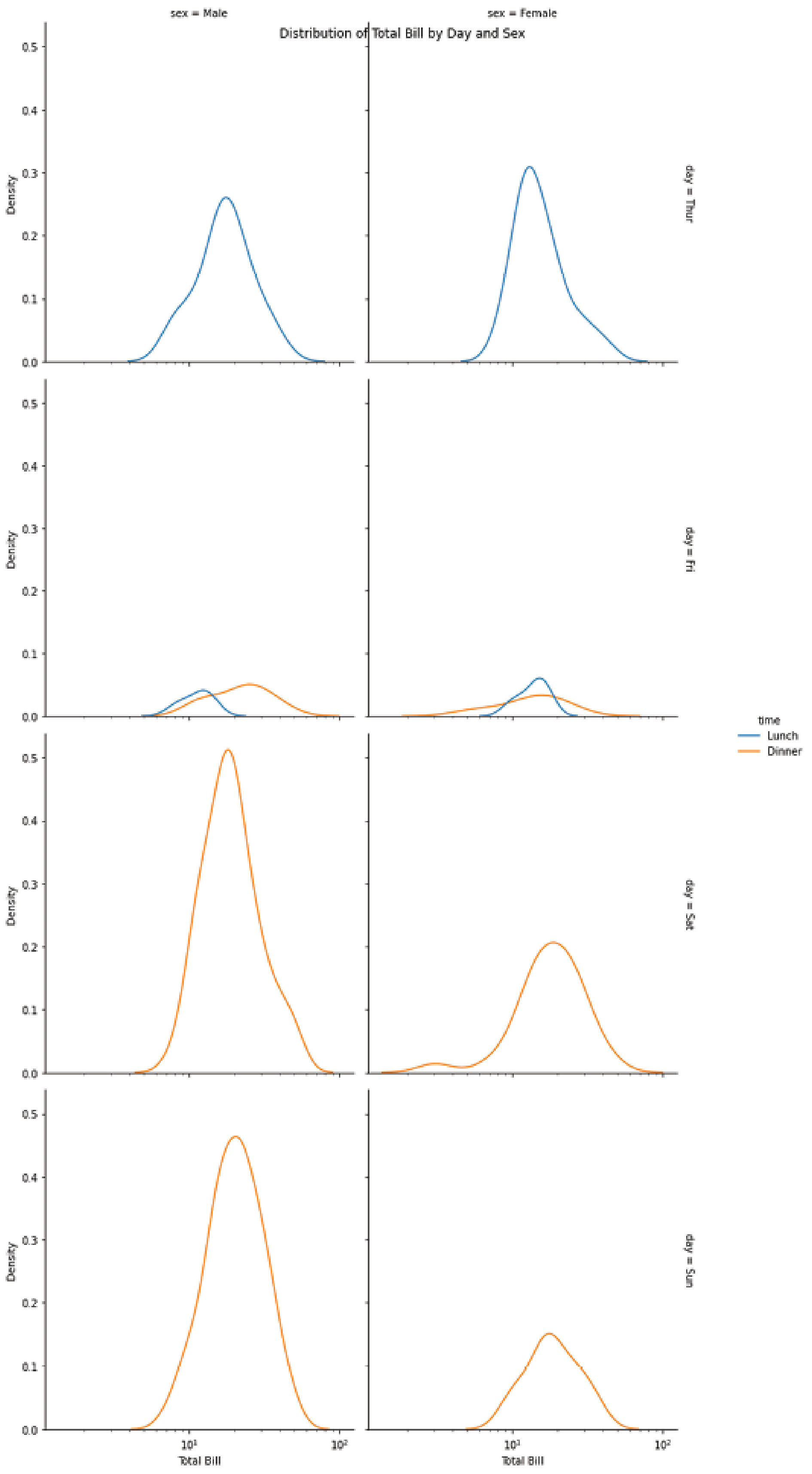

sex = Male
sex = Female
Distribution of Total Bill by Day and Sex
Density
0.0
0.1
0.2
0.3
0.4
0.5
day = Thur
day = Fri
day = Sat
day = Sun
time
Lunch
Dinner
10^1
10^2
Total Bill

***Figure 6.36**: Distribution plot on total bill with row_order, and log_scale*

Note that **facet_kws** is a parameter in seaborn plotting functions that allows you to pass additional arguments to the **FacetGrid object** that is created behind the scenes.

Regression plot

Regression plot is a type of plot in Seaborn that allows you to visualize the relationship between two variables by fitting a linear regression model to the data. It is a useful tool for exploring the correlation between variables and for making predictions based on the relationship between them.

```
import seaborn as sns
import matplotlib.pyplot as plt
# load the tips dataset
tips = sns.load_dataset("tips")
# create a regression plot using the total bill
#as the x-variable and the tip amount as the y-variable
sns.regplot(x="total_bill", y="tip", data=tips,
            fit_reg=True, ci=95, scatter_kws={"s": 50, "alpha": 0.5},
            line_kws={"color": "red", "linewidth": 3})

# set plot title and axes labels
plt.title("Regression plot for tips dataset")
plt.xlabel("Total bill")
plt.ylabel("Tip amount")
# show the plot
plt.show()
```

Output:

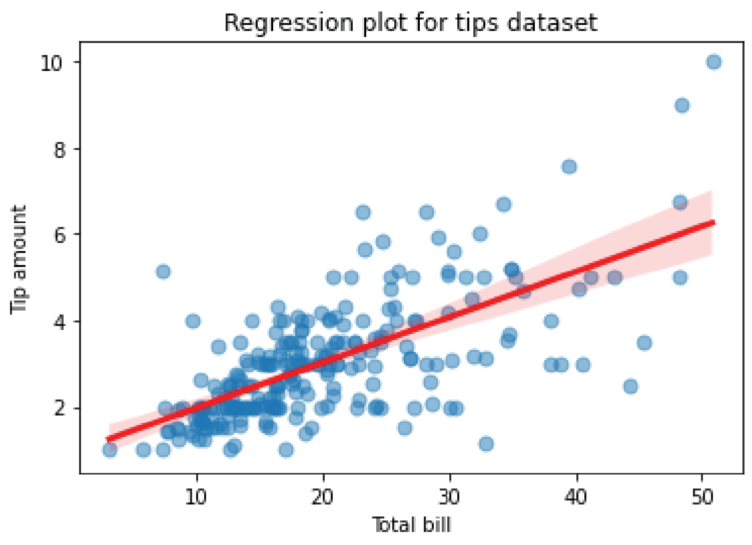


Figure 6.37*: Regression plot on tips obtained per total bill*

The **regplot()** function to create a regression plot with the following parameters:

The various parameters used in this code in **regplot()** are as below:

- **fit_reg**: A Boolean value indicating whether or not to fit a regression line to the data. If set to True, a line will be fitted and drawn on the plot.
- **ci:** The confidence interval to use for the regression line. This is expressed as a percentage (for example, 95 for a 95% confidence interval). The ci parameter is set to 95, which is the default value. This means that the shaded area around the regression line covers 95% of the expected values of the regression line at each point, indicating the uncertainty of the estimate. It controls the width of the shaded area around the regression line, indicating the uncertainty of the estimate.
- **scatter_kws:** A dictionary of keyword arguments to pass to the scatter plot function. In our example we have set the size of the markers(s) and the transparency of the markers (alpha).
- **line_kws:** A dictionary of keyword arguments to pass to the regression line function. In the code mentioned we have set the color of the line (color) and the width of the line (linewidth).

You can also set the plot title and axis labels using **plt.title()**, **plt.xlabel()**, and **plt.ylabel()**, and then show the plot using **plt.show().**

Saving the plot

Seaborn provides a simple way to save plots to a file. You can use the **savefig()** function from the **matplotlib.pyplot** module to save a plot generated with Seaborn to a file in a variety of formats such as PNG, PDF, SVG, and so on. It is better to refer to the chapter on matplotlib for all details on **savefig()** function parameters. Here's an example of how to save a Seaborn plot to a file:

```
import seaborn as sns
import matplotlib.pyplot as plt

# Load the tips dataset
tips = sns.load_dataset("tips")

# Create a bar plot of total bill by day of the week
sns.barplot(x="day", y="total_bill", data=tips)

# Add labels to the plot
plt.xlabel("Day of the Week")
plt.ylabel("Total Bill")
plt.title("Total Bill by Day of the Week")

# Save the plot to a file
plt.savefig("total_bill_by_day.png")
```

In this example, first create a bar plot of the total bill by day of the week using Seaborn's **barplot()** function. Then, add labels to the plot using **xlabel()**, **ylabel()**, and **title()** functions from **matplotlib.pyplot**. Finally, save the plot to a PNG file using the **savefig()** function and providing the file name as an argument.

You can also specify additional parameters to the **savefig()** function to control the resolution, size, and other aspects of the saved file. For example:

```
# Save the plot to a PDF file with a higher resolution
plt.savefig("total_bill_by_day.pdf", dpi=300)
```

This will save the plot to a PDF file with a resolution of 300 dots per inch.

Conclusion

In this chapter, we have learned to use seaborn functionalities for data visualization. Seaborn is a powerful and popular Python data visualization library that provides a high-level interface for creating informative and beautiful plots. It is built on top of Matplotlib and integrates well with other data analysis libraries such as Pandas and NumPy.

Seaborn's strengths lie in its ability to easily create complex visualizations with minimal code, including scatterplots, line plots, bar charts, histograms, heatmaps, and more. It also offers a range of customization options, such as changing color palettes, adding annotations, and modifying axis labels and tick marks. One of Seaborn's unique features is its ability to create statistical visualizations such as distribution plots, box plots, and violin plots, which can help reveal insights about the underlying data distribution and patterns. Additionally, Seaborn provides tools for visualizing relationships between variables, including correlation matrices, regression plots, and pair plots.

While Seaborn is a versatile and useful library, it is not perfect. One limitation is that it can be more resource-intensive than other data visualization libraries, which may cause performance issues when working with large datasets. Additionally, Seaborn may not always offer the level of customization and flexibility that some users require for their specific needs. Overall, Seaborn is a valuable addition to the data scientist's toolkit, particularly for creating informative and aesthetically pleasing visualizations for exploratory data analysis and data communication. With its broad range of visualization types, customization options, and statistical capabilities, Seaborn can help users uncover insights and communicate findings in a compelling way. Practice creating different types of visualizations in seaborn to become comfortable with the library and its functions. This will help you create effective and informative plots in the future.

Points to remember

- Seaborn is built on top of Matplotlib, so it is important to import both libraries at the beginning of your code. You may also need to import Pandas or NumPy if you are working with data in those formats.
- You need to provide the data to seaborn for visualization. The data can be in various forms like a pandas dataframe, numpy array, list or series.
- Seaborn offers a wide range of plot types, each with its own strengths and limitations. Choose the plot type that best represents the data you are working with and the message you want to convey.
- Seaborn offers a range of color palettes to choose from, each with its own set of colors and tones. Choose a palette that is appropriate for the data you are working with and the visualization you are creating.
- Seaborn offers a range of customization options, including changing axis labels, modifying tick marks, and adding annotations. You can further customize your plot by setting additional parameters like title, axis labels, legend, and so on.
- Seaborn offers a range of statistical visualizations that can help you understand the underlying data distribution and patterns. Make sure you understand the meaning and interpretation of these visualizations before using them.
- FacetGrid function is a tool used for visualizing data on multiple subsets of the dataset. It allows for the creation of multiple plots, each showing a different subset of the data, based on one or more categorical variables.
- A heatmap is a graphical representation of a matrix in which the individual values are represented as colors. The color intensity of each cell in the matrix represents the value of the corresponding variable.

- A clustermap is a type of heatmap that also shows hierarchical clustering of variables and/or observations.
- Seaborn can be resource-intensive, especially when working with large datasets. Be mindful of performance issues and use appropriate techniques such as subsetting or aggregating data to improve performance.
- Seaborn code can become complex, especially when creating more complex visualizations. Keep your code organized and well-commented to make it easier to understand and modify in the future.

Questions

1. What are the main features of the Seaborn library?
2. Create a bar chart of the number of passengers in each class of the Titanic dataset using Seaborn?
3. With the help of a snippet code, elaborate how to set the x-axis and y-axis labels in a Seaborn figure?
4. How do you create a scatter plot with Seaborn on the age and height of 10 persons? (create lists beforehand)
5. Create a boxplot of the fares paid by passengers in different classes using Seaborn in the Titanic dataset?
6. With the help of a code, visualize the distribution of a continuous variable using Seaborn in the Titanic dataset?
7. With the help of a code, demonstrate the purpose of using the hue parameter in Seaborn's FacetGrid function when visualizing the Titanic dataset?
8. What is the purpose of using the swarmplot function?
9. What is the difference between a heatmap and a clustermap in Seaborn, and how can they be used to analyze the Titanic dataset?

Join our book’s Discord space

CHAPTER 7
Data Visualization with Bokeh

By visualizing information, we turn it into a landscape that you can explore with your eyes. A sort of information map. And when you're lost in information, an information map is kind of useful.

– David McCandless

As the book title suggests, the main objective of the book is to provide knowledge on using Python libraries for data visualization. This chapter presents various functions available in the Bokeh module of Python, which are used for presenting data in graphical representations.

Structure

In this chapter, we will discuss the following topics:

- Introduction
- Importing Bokeh and its documentation
- Understanding figures, glyphs, and axes
- Palettes and Colors
- Creating Plots
- Creating Interactive plots
- Creating multiple plots
- Saving the plot

Objectives

In this chapter, the main objective is to create a cohesive narrative that supports the book's overall purpose of learning visualization using Python libraries. In this chapter, the focus is on the Bokeh module and understanding various functions and methods available in Bokeh for data visualization.

Introduction

Bokeh is a Python data visualization library that provides interactive and browser-based visualizations for modern web browsers. It allows you to create interactive visualizations, dashboards, and applications that can be easily shared with others.

Bokeh makes it easy to create complex visualizations by providing a high-level interface for creating interactive plots, as well as low-level tools for customizing and extending plots. It includes a variety of chart types, including scatter plots, line charts, bar charts, histograms, and heatmaps, as well as support for streaming and real-time data.

Some key features of Bokeh include:

- **Interactive plots**: Bokeh allows you to create interactive plots that can be zoomed, panned, and selected, as well as linked together to create complex visualizations.
- **Customizable visualizations**: Bokeh provides a range of tools for customizing and extending plots, including support for custom JavaScript callbacks and widgets.
- **Flexible output options**: Bokeh supports a range of output options, including HTML, standalone documents, and Bokeh server applications.
- **Integration with other Python libraries**: Bokeh can be easily integrated with other Python libraries, such as Pandas and NumPy, as well as with popular data science frameworks, such as Jupyter Notebook and Flask.

Bokeh is widely used in data science, finance, and other industries for creating interactive visualizations and dashboards. It is well-documented and has an active community of developers, making it a popular choice for data visualization in Python.

Bokeh is a powerful data visualization library that offers a wide range of interactive features for creating dynamic, responsive visualizations that can be easily shared and explored in web browsers.

Some of the key interactive features of Bokeh plots include:

- **Hover Tooltips:** Hover tooltips provide a way to display additional information about a data point when the user hovers over it with their mouse. To add hover tooltips to a Bokeh plot, you can use the **HoverTool** from the **bokeh.models** module.
- **Pan and Zoom:** Bokeh provides built-in tools that allow users to pan and zoom in and out of a plot. These tools can be added to a plot using the tools argument of the figure function.
- **Widgets:** Bokeh provides a range of interactive widgets, such as sliders, dropdown menus, and buttons, that allow users to change the parameters of a plot or explore different aspects of the data. Widgets can be added to a plot using the **bokeh.models.widgets** module.
- **Linked interactions:** Bokeh allows multiple plots to be linked together so that interactions in one plot affect the others. This can be useful for exploring multi-dimensional data and comparing different views of the same data. Linked interactions can be created using the **bokeh.models** module to define the data sources for each plot.
- **Custom JavaScript callbacks:** Bokeh allows users to define custom JavaScript callbacks that can be triggered by user interactions or changes to the data. This provides a high level of flexibility for creating complex and highly interactive visualizations.

These are just a few examples of the interactive features of Bokeh plots. Bokeh provides many other tools and options for creating dynamic and responsive visualizations that can be easily explored and shared in web browsers.

Bokeh provides a variety of plots that can be used for data visualization, including:

- **Scatter Plot:** A scatter plot is a two-dimensional plot that displays data points as dots. It is useful for visualizing the relationship between two continuous variables.
- **Line Plot:** A line plot is a plot of data points connected by a line. It is useful for visualizing trends over time or other ordered data.
- **Bar Plot:** A bar plot is a plot that displays data using rectangular bars. It is useful for visualizing discrete data or comparing multiple categories.

- **Heatmap:** A heatmap is a plot that displays data as a grid of colored rectangles, with the color representing the magnitude of the data value. It is useful for visualizing patterns in two-dimensional data.
- **Histogram:** A histogram is a plot that displays the distribution of a single variable. It is useful for visualizing the frequency of values within a range.
- **Box Plot:** A box plot is a plot that displays the distribution of a single variable using a box-and-whisker diagram. It is useful for visualizing the spread and skewness of data.

Importing Bokeh and its documentation

To import Bokeh, you can use the following command:

```
import bokeh
```

To access the documentation for Bokeh, you can either visit the Bokeh website at https://docs.bokeh.org/en/latest/index.html, or you can use the help function in Python to display the documentation for a specific function or module. For example, to display the documentation for the plotting module in Bokeh, you can use the following command:

```
help(“bokeh.plotting”)
```

This will display the documentation for the figure function in the console, which includes a description of all the functions, their parameters, and their return value, as well as examples of how to use it.

For beginners, check and refer to the documentation available on the official website of bokeh. **https://docs.bokeh.org/en/latest/docs/first_steps.html**.

Understanding figures, glyphs, axes, and palettes

Bokeh is a visualization library that provides tools for creating interactive plots, visualizations, and applications in web browsers. The three main components of a Bokeh plot are the Figure, the glyphs, and the axes.

Figure

The Figure is the central object in a Bokeh plot. It provides a canvas on which to draw the glyphs, as well as the overall layout and formatting of the plot. You can create a new Figure object by calling the figure function in the **bokeh.plotting** module. The figure function takes a number of arguments that allow you to customize the appearance of the plot, including the size, background color, and font properties.

The **figure()** function in Bokeh creates a new plotting canvas or figure. It takes several arguments that allow you to customize the appearance of the plot, such as the size and color of the plot background, the range of values for the x- and y-axes, and the appearance of the x- and y-axis labels.

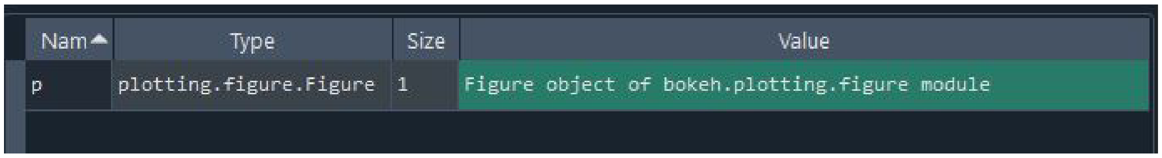
Here is a list of the most common parameters for the **figure()** function in Bokeh:

- **plot_width** and **plot_height**: The width and height of the plot, in pixels. The default values are 600 and 400, respectively.
- **title**: The title of the plot. You can set this to a string value.
- **x_axis_label** and **y_axis_label**: The labels for the x- and y-axes, respectively.
- **x_range** and **y_range**: The range of values to display on the x- and y-axes, respectively. These can be specified as a tuple or a list, with the first value indicating the minimum value and the second value indicating the maximum value. If not specified, the range is automatically determined based on the data.
- **x_axis_type** and **y_axis_type**: The type of scaling to use for the x- and y-axes, respectively. The default is "auto", but you can also use "linear", "log", or "datetime".
- **background_fill_color** and **border_fill_color**: The colors to use for the plot background and border, respectively.
- **tools**: A list of tools to include in the plot. The default value is "**pan,box_zoom,wheel_zoom,reset,save**", but you can also include other tools such as "**hover**" or "**lasso_select**".
- **toolbar_location**: The location of the toolbar, which contains the plot tools. The default is "above", but you can also use "below", "left", or "right".

Here is an example of using the **figure()** function with several of these parameters:

```
from bokeh.plotting import figure

p = figure(plot_width=800, plot_height=600, title="My Plot",

    x_axis_label="X Axis Label", y_axis_label="Y Axis Label",

    x_range=(0, 10), y_range=(0, 1),
    x_axis_type="linear", y_axis_type="log",
    background_fill_color="#f5f5f5", border_fill_color="#cccccc",
    tools=["pan", "box_zoom", "wheel_zoom", "reset", "save", "hover"],
    toolbar_location="above")
```

In this example, we create a new plot with a width of 800 pixels and a height of 600 pixels. We set the title to "My Plot" and add labels for the x- and y-axes. We also set the range of the x-axis to be from 0 to 10 and the range of the y-axis to be from 0 to 1. We specify that the x-axis should use linear scaling and the y-axis should use logarithmic scaling. We set the background and border colors to light gray and gray, respectively. We include several tools in the plot, including "**pan**", "**box_zoom**", "**wheel_zoom**", "**reset**", "**save**", and "**hover**", and we place the toolbar above the plot. You can verify in the variable explorer a figure object is created. See its '**type**'.



Nam	Type	Size	Value
p	plotting.figure.Figure	1	Figure object of bokeh.plotting.figure module

Figure 7.1: *Display of type of figure created from VariableExplorer*

Glyphs

Glyphs are the visual objects that are displayed on the Figure. They can be used to represent different types of data, such as lines, bars, circles, or text. You can add glyphs to a Figure using various methods, such as line, circle, or text, depending on the type of glyph you want to add. Each glyph has its own set of properties, such as the x and y coordinates, size, color, and alpha (transparency). You can set these properties using keyword arguments when you create the glyph, or by accessing the glyph's properties directly. In Bokeh, a glyph is a basic visual building block used to create plots, and it represents the visual mark that is placed at specific coordinates in a plot. Glyphs are used to represent different types of data, such as lines, circles, rectangles, polygons, and more.

A glyph is defined by specifying the type of glyph, along with the data that the glyph represents, and any visual properties of the glyph. For example, a line glyph would be defined by specifying the type as "line", and providing the x and y coordinates of the points that make up the line.

Some common glyph properties that can be set in Bokeh include the color, size, alpha (transparency), and line width. Glyphs can also be styled based on data values using a color mapper, which maps data values to specific colors.

In Bokeh, glyphs are typically created and added to a plot using a figure object, which provides a canvas for creating visualizations. Once the figure is created, glyphs can be added to it by calling the appropriate glyph method on the figure, such as **line()** for adding a line glyph, **circle()** for adding a circle glyph, or **rect()** for adding a rectangle glyph. Bokeh provides several different types of glyphs, which are visual markers used to represent data points in a plot. Here's a list of some of the most commonly used glyphs in Bokeh, along with a brief description of their use:

- **Annulus:** A ring-shaped glyph with an outer and inner radius. Often used to represent a data point with a distinct inner and outer value.
- **Arc:** A portion of a circle or annulus, specified by a start angle, end angle, and radius. Useful for creating pie charts or other circular visualizations.
- **Circle:** A simple circle glyph, often used to represent individual data points.
- **Ellipse:** An oval-shaped glyph, often used to represent a data point with an x and y value.
- **Image:** A rectangular image glyph, used to represent images in a plot.
- **Line:** A line glyph, used to represent a line or curve connecting data points.
- **MultiLine:** A multi-line glyph, used to represent multiple lines or curves connecting multiple sets of data points.
- **Patch:** A filled polygonal glyph, used to represent data points with multiple x and y values.
- **Quad:** A rectangular glyph, used to represent data points with a fixed x and y value, and a width and height.
- **Ray:** A line glyph starting at a given point and extending indefinitely in a specified direction. Useful for creating directional visualizations.
- **Rect:** A rectangular glyph with a fixed width and height, often used to represent individual data points.
- **Segment:** A line glyph connecting two points, often used to create segmented lines or curves.

- **Text:** A text glyph, used to add text to a plot.
- **Triangle:** A triangular glyph, used to represent data points with three values.
- **Wedge:** A portion of a circular or annular glyph, specified by a start angle, end angle, and radius. Similar to the Arc glyph, but with a fixed inner radius and filled color.

These glyphs can be customized with various visual properties such as color, size, alpha (transparency), line width, fill color, and more, to create rich and informative visualizations.

Here's an example of each type of glyph in Bokeh, using the Iris dataset as an example:

```
import bokeh

from bokeh.plotting import figure, show

from bokeh.sampledata.iris import flowers

# Create a ColumnDataSource from the Iris dataset

source = bokeh.plotting.ColumnDataSource(flowers)

# Create a figure with a title and x/y axis labels

p = figure(title="Iris Dataset", x_axis_label="Petal Length", y_axis_label="Petal Width")

# Add glyphs for each type of glyph to the first five rows of the iris dataset

p.circle(x=flowers["sepal_length"][:5], y=flowers["sepal_width"][:5], size=10, color="blue")

# Show the plot

show(p)

p.square(x=flowers["sepal_length"][:5], y=flowers["sepal_width"][:5], size=10, color="red")

# Show the plot

show(p)

p.triangle(x=flowers["sepal_length"][:5], y=flowers["sepal_width"][:5], size=10, color="green")

# Show the plot

show(p)

p.diamond(x=flowers["sepal_length"][:5], y=flowers["sepal_width"][:5], size=10, color="purple")

# Show the plot
```

```
show(p)
p.cross(x=flowers["sepal_length"][:5], y=flowers["sepal_width"][:5], size=10,
color="orange")
# Show the plot
show(p)
```

Output of circle glyph:

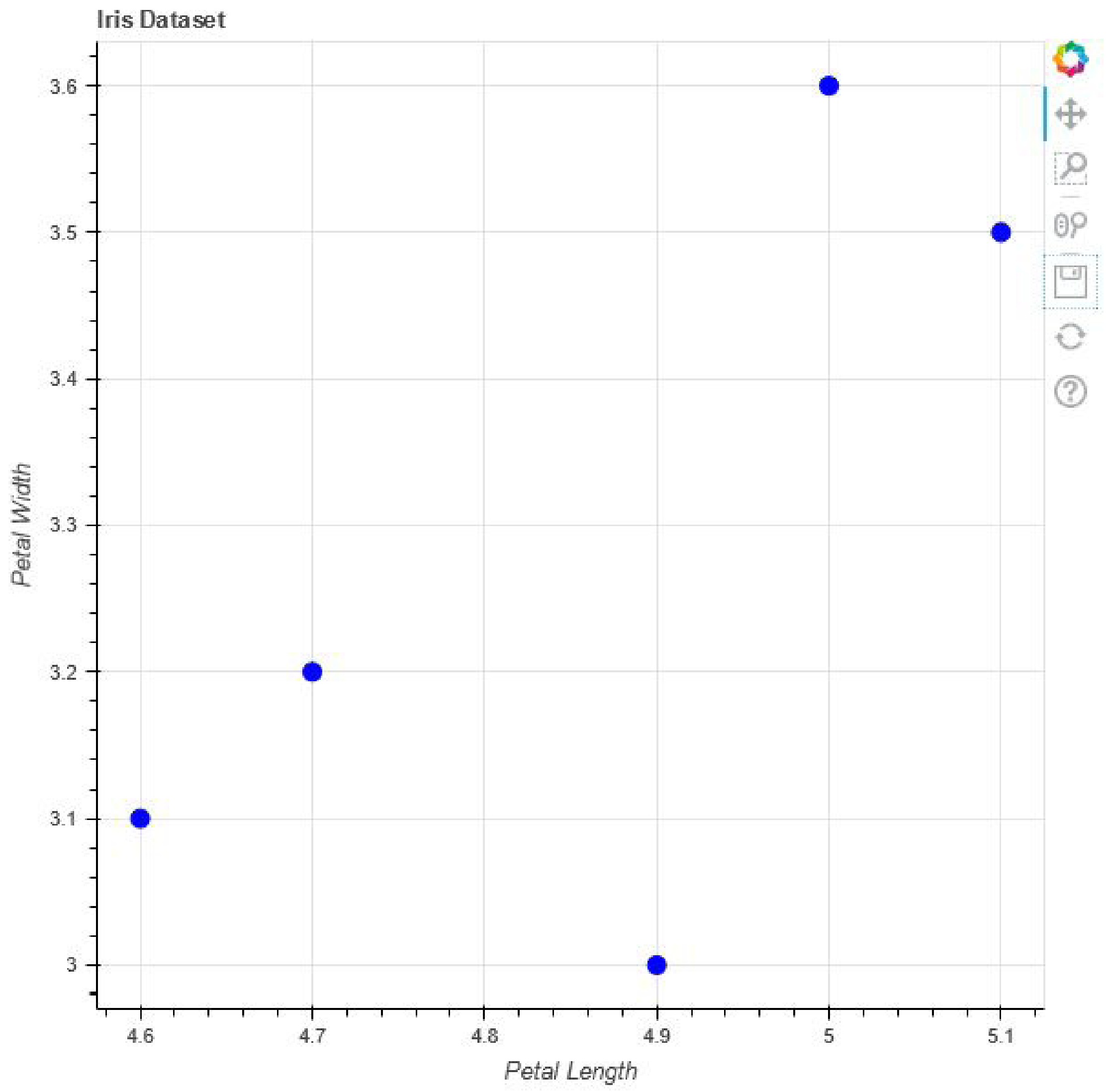


Figure 7.2: *Representation of Circle Glyph*

This code will create a single plot with each type of glyph included, using different visual properties for each to help differentiate between them. Notice that the plots will open in web browser. Note that not all glyphs are appropriate for all types of data, so it's important to choose the right glyph for your specific use case.

Axes

Axes provide the scales and labels for the x and y axes of the Figure. You can add axes to a Figure using various methods, such as xaxis and yaxis, and customize their appearance using various properties, such as the font size, tick labels, and tick mark locations. Bokeh also provides additional tools for interacting with the axes, such as zooming, panning, and selecting data points. here are a few examples of how you can customize the appearance of axes in Bokeh:

Changing the axis range

You can set the range of an axis using the x_range and y_range properties of the Figure object. For example, to set the x-axis range to be from 0 to 10:

```
p = figure(x_range=(0, 10), ...)
```

Adjusting the tick marks and labels

You can control the tick marks and labels on an axis using the **major_tick_in**, **major_tick_out**, **minor_tick_in**, **minor_tick_out**, **major_label_text_font_size**, and other similar properties. For example, to change the font size of the x-axis labels:

```
p.xaxis.major_label_text_font_size = "16pt"
```

Changing the axis line color and width

You can customize the appearance of the axis line using the **axis_line_color** and **axis_line_width** properties. For example, to change the y-axis line color to red and increase its width:

```
p.yaxis.axis_line_color = "red"

p.yaxis.axis_line_width = 3
```

Formatting tick labels

You can format the tick labels on an axis using the formatter property. For example, to format the y-axis tick labels as percentages:

```
from bokeh.models import NumeralTickFormatter

p.yaxis.formatter = NumeralTickFormatter(format="0%")
```

These are just a few examples of how you can customize the appearance of axes in Bokeh. There are many other properties and options available, so I recommend checking out the Bokeh documentation for more information.

In summary, the Figure provides the overall structure and layout of the plot, the glyphs provide the visual representation of the data, and the axes provide the scales and labels for the x and y axes. Together, these components allow you to create rich and interactive visualizations in Bokeh.

Palettes and colours

Bokeh is a powerful Python library for creating interactive visualizations. It provides various ways to control palettes and colors in plots. You can control palettes and colors in plots using the palette parameter and the color property.

Palettes

The **bokeh.palettes** module is a part of the Bokeh Python library, which is used for creating interactive visualizations in web browsers. The module contains a collection of color palettes that can be used to create visually appealing plots and charts.

A color palette is a collection of colors that are designed to work well together. Bokeh provides a number of pre-defined palettes that can be used directly, or as a starting point for creating custom palettes.

Here's an overview of some of the most commonly used palettes in the **bokeh.palettes** module:

- **Category10**: A palette of 10 colors that can be used for categorical data.
- **Category20**: A palette of 20 colors that can be used for categorical data.
- **Viridis**: A palette of colors that is designed to be perceptually uniform and work well for continuous data.
- **Inferno**: A palette of colors that is similar to Viridis but has a different color scheme.
- **Magma:** A palette of colors that is similar to Inferno but has a different color scheme.
- **Plasma**: A palette of colors that is similar to Inferno and Magma but has a different color scheme.

Turbo

A palette of colors that is similar to Viridis but with a higher contrast and better visibility for small differences.

To use a palette, you can call the palette function with the number of colors you want in the palette as an argument. For example, to create a palette of 5 colors using the Viridis palette, you can use the following code:

from bokeh.palettes import Viridis

my_palette = Viridis[5]

This will create a list of 5 colors that you can use in your plot. You can also use the **inferno()**, **magma()**, **plasma()**, and **turbo()** functions to generate palettes directly.

In addition to these predefined palettes, you can also create custom palettes by combining colors or interpolating between colors. The **linear_palette()** and **log_palette()** functions can be used to generate linear and logarithmic color palettes, respectively. The Color class can be used to define custom colors in various ways, including specifying RGB or HSL values.

Overall, the **bokeh.palettes** module provides a useful set of tools for creating color palettes that can be used in a variety of plots and visualizations. Whether you're working with categorical or continuous data, there's likely a palette that will work well for your needs.

Colors

The **bokeh.colors** module is a part of the Bokeh Python library, which provides tools for creating interactive visualizations in web browsers. This module provides a set of functions and classes for working with colors in Bokeh.

One of the most basic functions in the **bokeh.colors** module is **color()**, which takes a string representing a color and returns an RGB tuple of integers. For example, **color("red")** would **return (255, 0, 0)**.

Another function in the **bokeh.colors** module is **RGB()**, which takes three integers representing the red, green, and blue values of a color and returns an RGB tuple. For example, **RGB(255, 0, 0)** would **return (255, 0, 0)**.

The **Color** class in the **bokeh.colors** module provides a more object-oriented way to work with colors. You can create a Color object by passing in a color string, an RGB tuple, or a hex string. For example, **Color("red")** would create a Color object representing the color red. You can also access the red, green, and blue components of a Color object using the red, green, and blue attributes.

The HSV class in the **bokeh.colors** module provides a way to work with colors in the HSV (hue, saturation, value) color space. You can create an HSV object by passing in the hue, saturation, and value components as floats between 0 and 1. For example, HSV(0.0, 1.0, 1.0) would create an HSV object representing pure red.

The **CategoricalColorMapper** class in the **bokeh.colors** module provides a way to map categorical values to colors. You can create a **CategoricalColorMapper** object by passing in a list of categories and a list of colors. The **CategoricalColorMapper** object will then map each category to a color in the list. For example, **CategoricalColorMapper(factors=["foo", "bar", "baz"], palette=["red", "green", "blue"])** would create a **CategoricalColorMapper** object that maps the category "**foo**" to red, "**bar**" to green, and "**baz**" to blue.

In summary, the **bokeh.colors** module provides a set of tools for working with colors in Bokeh, including functions for converting between color representations and classes for working with colors in different color spaces and for mapping categorical values to colors.

Creating plots

Bokeh is a powerful library for creating various types of interactive plots. In this section of the chapter, you will learn to develop various plots using the bokeh library of the Python.

Scatter plot

Here is an example of how to create a scatter plot using Bokeh:

```
from bokeh.plotting import figure, output_file, show
from bokeh.models import ColumnDataSource

#Create sample data
X = [1, 2, 3, 4, 5]
Y = [6, 7, 2, 4, 5]

#Create a ColumnDataSource Object
```

```
source = ColumnDataSource(data = dict(x=X,y=Y))

# Create a new plot with a title and axis labels
p = figure(title ="Scatter Plot", x_axis_label='X - Axis', y_axis_label='Y - Axis')

# Add a circle glyph to the plot using the data from ColumnDataSource
p.circle('x','y', size=10, source=source)

#Save th eplot to an HTML file and display it
output_file("scatter.html")
show(p)
```

Output:

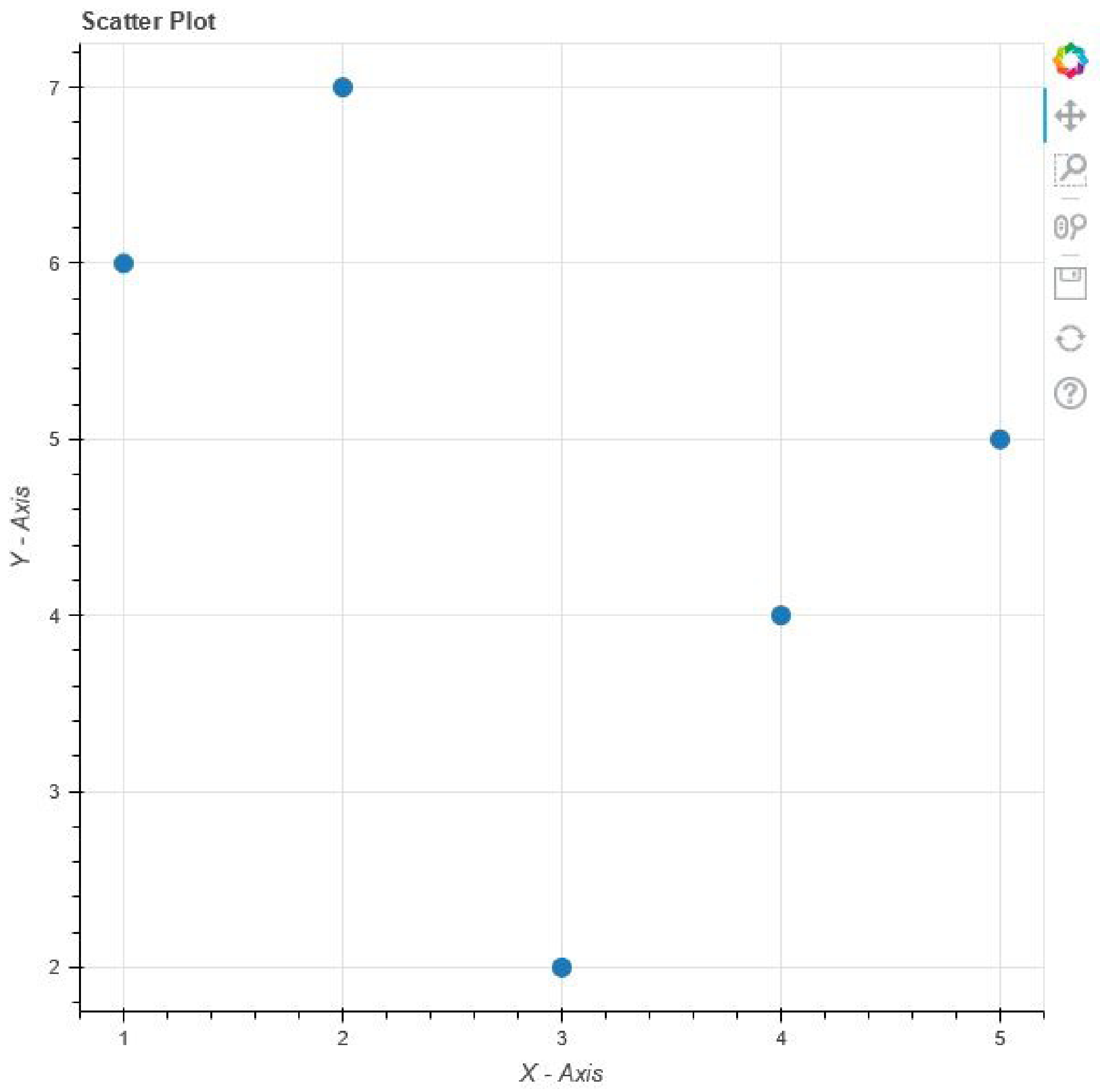


Figure 7.3: *Scatter Plot*

This code creates a scatter plot with a title and axis labels, and adds a circle glyph to the plot using the data from the **ColumnDataSource** object. The resulting plot is saved to an HTML file and displayed using the show function.

In Bokeh, a **ColumnDataSource** is a fundamental data structure used to hold data that will be plotted on a figure. It is essentially a dictionary-like object that maps string column names to sequences of values. The sequences can be lists, arrays, or Pandas series.

Using a **ColumnDataSource** can simplify the process of creating interactive plots, because it allows Bokeh to update the plot in response to user interactions such as panning, zooming, or hovering over glyphs. It can also be used to share data between multiple plots in a dashboard, for example.

Here is an example of creating a **ColumnDataSource** in Bokeh:

```
from bokeh.plotting import figure, output_file, show
from bokeh.models import ColumnDataSource
import numpy as np

#Sample Data
x=np.linspace(0,10,100)
y=np.sin(x)

#Create a ColumnDataSource Object
source = ColumnDataSource(data = dict(x=X,y=Y))
```

In this example, we create some data to plot, in this case x and y arrays containing 100 points along a sine wave. We then create a **ColumnDataSource** object, passing it a dictionary with keys 'x' and 'y', and the x and y arrays as values.

Line plot

An example of how to create a simple line plot using Bokeh:

```
from bokeh.plotting import figure, output_file, show
#Data
X = [1, 2, 3, 4, 5]
Y = [6, 7, 2, 4, 5]

# Create a new plot with a title and axis labels
p = figure(title ="Line Plot", x_axis_label='X - Axis', y_axis_label='Y - Axis')

# Add a line glyph to the plot using the data from ColumnDataSource
p.line(X,Y, line_width=2)

#Save the plot to an HTML file and display it
output_file("line.html")
show(p)
```

Output:

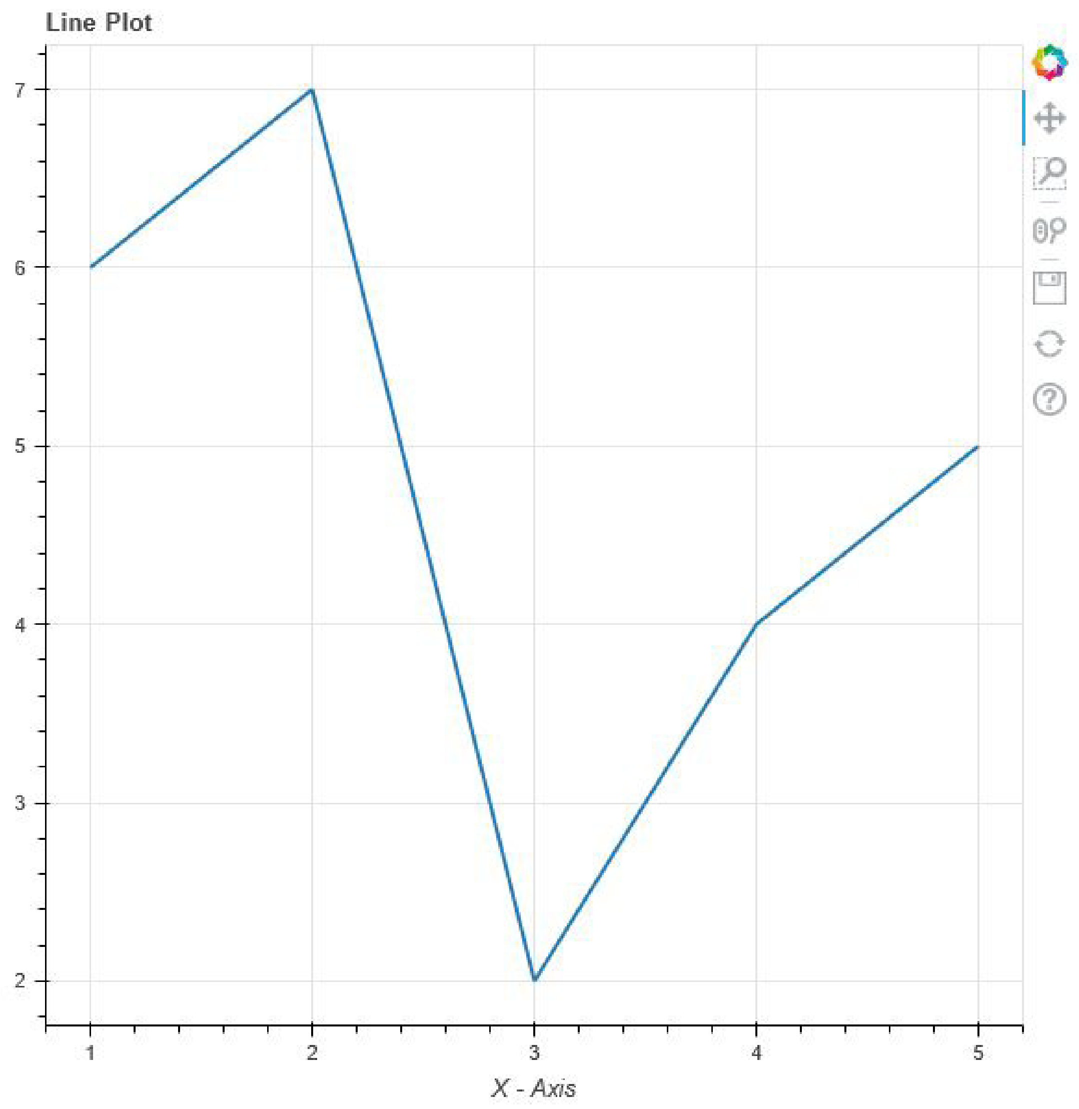


Figure 7.4: *Line Plot*

This code creates a line plot with a title and axis labels, and adds a line glyph to the plot using the data from the x and y lists. The **line_width** argument sets the width of the line. The resulting plot is saved to an HTML file and displayed using the show function.

You can customize the appearance of the plot further using additional arguments to the figure and line functions. For example, you can set the color of the line using the **line_color** argument, or add markers to the line using the **line_dash** and **line_dash_offset** arguments.

Bar plot

An example of how to create a simple bar plot using Bokeh:

```
from bokeh.plotting import figure, output_file, show

# Data
categories = ['Apple', 'Orange', 'Banana']
Values = [5, 3, 4]

# Create a new plot with a title and axis labels
p = figure(title ="Bar Plot", x_axis_label='Category', y_axis_label='Value',
        x_range=categories)

# Add a vbar glyph to the plot using the data from ColumnDataSource
p.vbar(categories,top=Values,bottom=0, width =0.6)

#Save th eplot to an HTML file and display it
output_file("bar.html")
show(p)
```

Output:

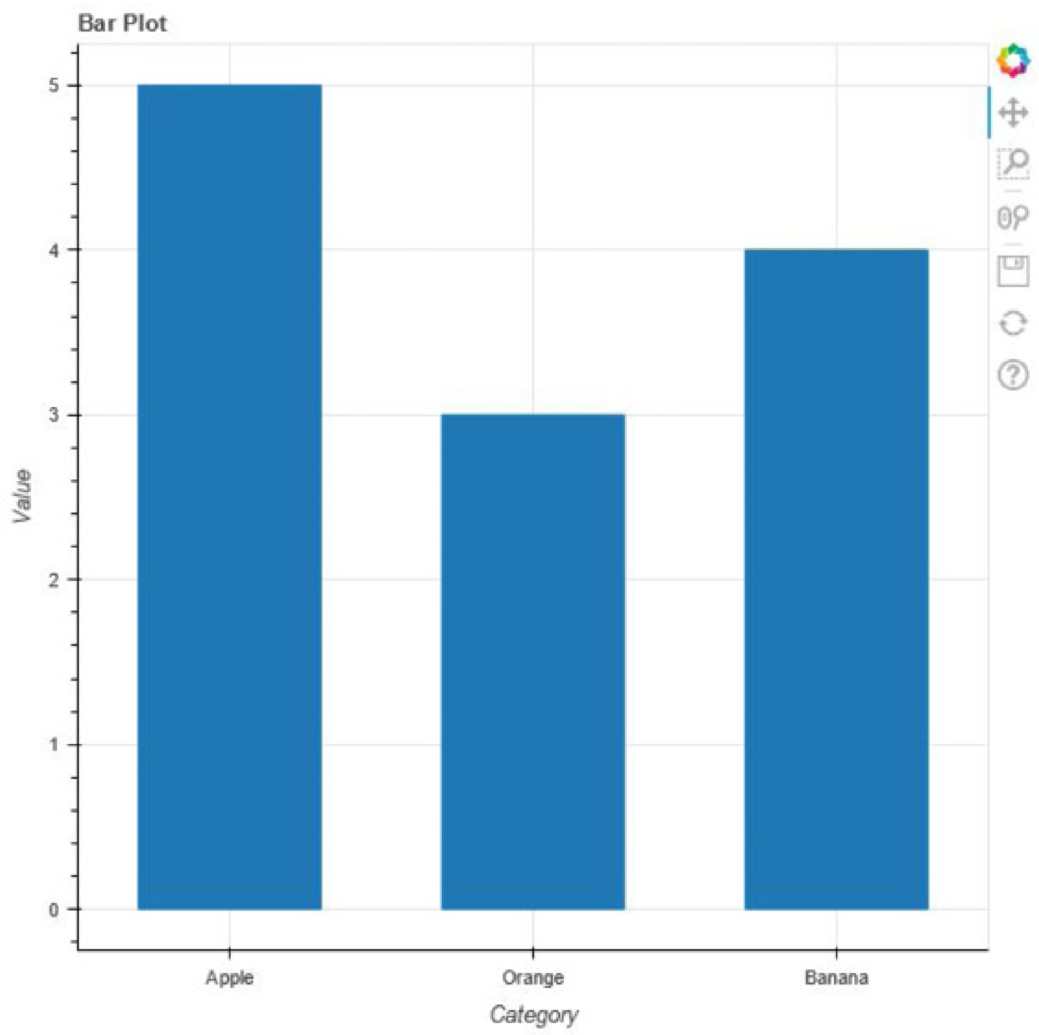


Figure 7.5: *Bar Plot*

This code creates a bar plot with a title and axis labels, and adds a vbar glyph to the plot using the data from the categories and values lists. The **x_range** argument sets the range of values for the x-axis. The width argument sets the width of the bars. The resulting plot is saved to an HTML file and displayed using the show function.

You can customize the appearance of the plot further using additional arguments to the figure and vbar functions. For example, you can set the color of the bars using the color argument, or add a legend to the plot using the **legend_label** argument.

HeatMap

Bokeh does not have a built-in heatmap chart type, but you can create a heatmap using a **Rect** glyph and a **ColorMapper** to map the values to colors. Here's an example of how to create a simple heatmap using Bokeh:

```
from bokeh.plotting import figure, output_file, show
from bokeh.models import ColumnDataSource, LinearColorMapper
from bokeh.transform import transform
# Create some data
x = ['Mon', 'Tue', 'Wed', 'Thu', 'Fri']
y = ['Morning', 'Afternoon', 'Evening']
data = {'x': x*len(y), 'y': [i for i in y for _ in x],
        'values': [5, 3, 2, 7, 6, 4, 8, 1, 9, 2, 5, 7, 6, 2, 3]}
# Create a new plot with a title and axis labels
p = figure(title="Heatmap", x_range=x, y_range=y)
# Create a color mapper to map the values to colors
color_mapper = LinearColorMapper(palette="Viridis256",
                                 low=min(data['values']), high=max(data['values']))
# Add a Rect glyph to the plot using the data and color mapper
source = ColumnDataSource(data)
p.rect(x='x', y='y', width=1, height=1, source=source,
       line_color=None, fill_color=transform('values', color_mapper))
# Add a color bar to the plot
color_bar = bokeh.models.ColorBar(color_mapper=color_mapper, location=(0, 0))
p.add_layout(color_bar, 'right')
# Save the plot to an HTML file and display it
output_file("heatmap.html")
show(p)
```

Output:

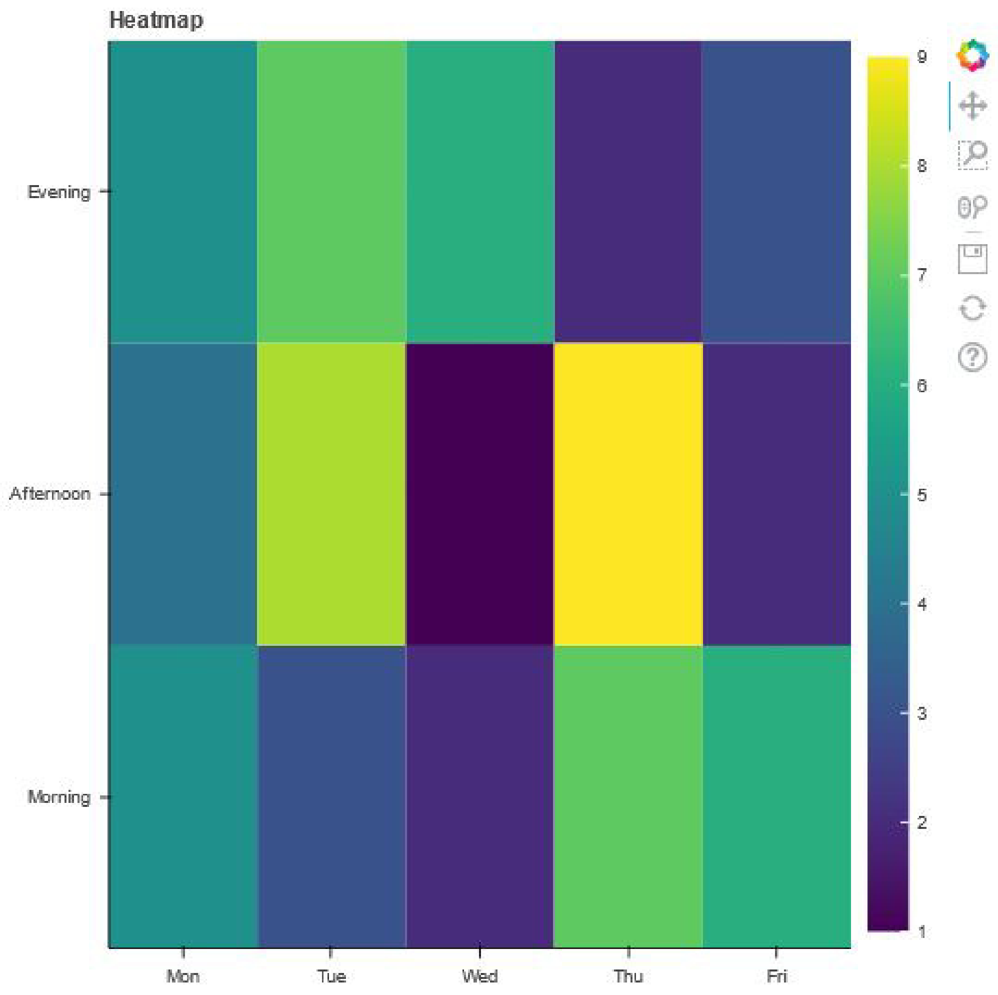


Figure 7.6: *Heatmap*

This code creates a heatmap with a title and axis labels, and adds a Rect glyph to the plot using the data from the x, y, and values lists. The width and height arguments set the size of each rectangle. The **line_color** argument is set to None to remove the borders around the rectangles. The **fill_color** argument specifies the color to fill the rectangle with. In this case, the **transform()** function is used to apply a **color_mapper** to the values column of the source data source, which suggests that the fill color may be based on some kind of data mapping or normalization. The resulting plot is saved to an HTML file and displayed using the show function.

You can customize the appearance of the plot further using additional arguments to the figure and rect functions. For example, you can set the size and location of the color bar using the width, height, and location arguments of the **ColorBar** constructor.

Histogram

Bokeh provides the histogram function to create histograms. Here's an example of how to create a simple histogram using Bokeh:

```
from bokeh.plotting import figure, output_file, show
from bokeh.sampledata.iris import flowers
import numpy as np
 # Create a new plot with a title and axis labels
p = figure(title="Histogram", x_axis_label="Sepal Length", y_axis_label="Count")
 # Get the data
values, edges = np.histogram(flowers['sepal_length'], bins=20)
 # Add the histogram to the plot
p.quad(top=values, bottom=0, left=edges[:-1], right=edges[1:],
       fill_color="navy", line_color="white", alpha=0.5)
 # Save the plot to an HTML file and display it
output_file("histogram.html")
show(p)
```

Output:

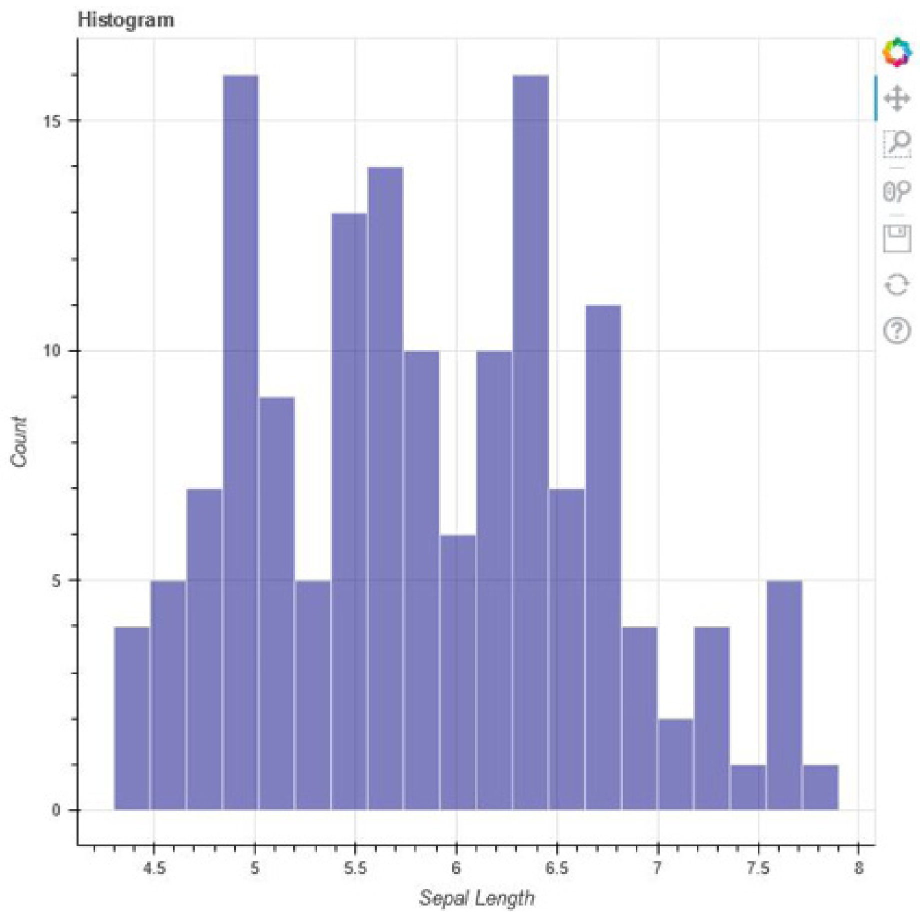


***Figure 7.7**: Histogram*

This code creates a histogram of the sepal length of the flowers in the Iris dataset. The values and edges arrays are obtained using the **numpy.histogram** function, which bins the data into 20 equally spaced intervals.

The quad function is used to create a rectangle for each bin, with the top of the rectangle set to the number of data points in the bin. This line of code, creates a new glyph using **p.quad()** and adds it to the p plot object. It takes several arguments:

- **top=values**: This argument sets the top of each rectangle to the corresponding value in the values array, which represents the height of each bin in the histogram.
- **bottom=0**: This argument sets the bottom of each rectangle to 0, which ensures that the rectangles start from the x-axis.
- **left=edges[:-1]**: This argument sets the left edge of each rectangle to the corresponding value in the edges array, excluding the last value. This

ensures that each rectangle starts at the beginning of each bin.

- **right=edges[1:]**: This argument sets the right edge of each rectangle to the corresponding value in the edges array, excluding the first value. This ensures that each rectangle ends at the end of each bin.
- **fill_color="navy"**: This argument sets the fill color of each rectangle to "navy".
- **line_color="white"**: This argument sets the color of the rectangle's border to "white".
- **alpha=0.5**: This argument sets the opacity of each rectangle to 0.5, making them partially transparent. The resulting plot is saved to an HTML file and displayed using the show function.

You can customize the appearance of the plot further using additional arguments to the figure and quad functions. For example, you can set the color and line width of the rectangles using the **fill_color** and **line_width** arguments of the quad function, or add a legend using the **legend_label** argument of the quad function.

Patch plot

A Patch glyph in Bokeh is used to draw arbitrary closed polygons on a plot. The Patch glyph can be used to highlight specific regions of a plot or to create custom shapes.

Here's an example of creating a Patch plot in Bokeh:

```
from bokeh.plotting import figure, show
from bokeh.models import ColumnDataSource
# create a sample polygon
x = [1, 2, 3, 4, 5]
y = [1, 3, 4, 2, 1]
polygon = [(xi, yi) for xi, yi in zip(x, y)]
# create a data source for the polygon
source = ColumnDataSource(data=dict(x=x, y=y))
# create a figure and add a patch glyph
p1 = figure(title="Patch plot")
p1.patch(x='x', y='y', fill_alpha=0.4, line_width=2, source=source)
```

show(p1)

Output:

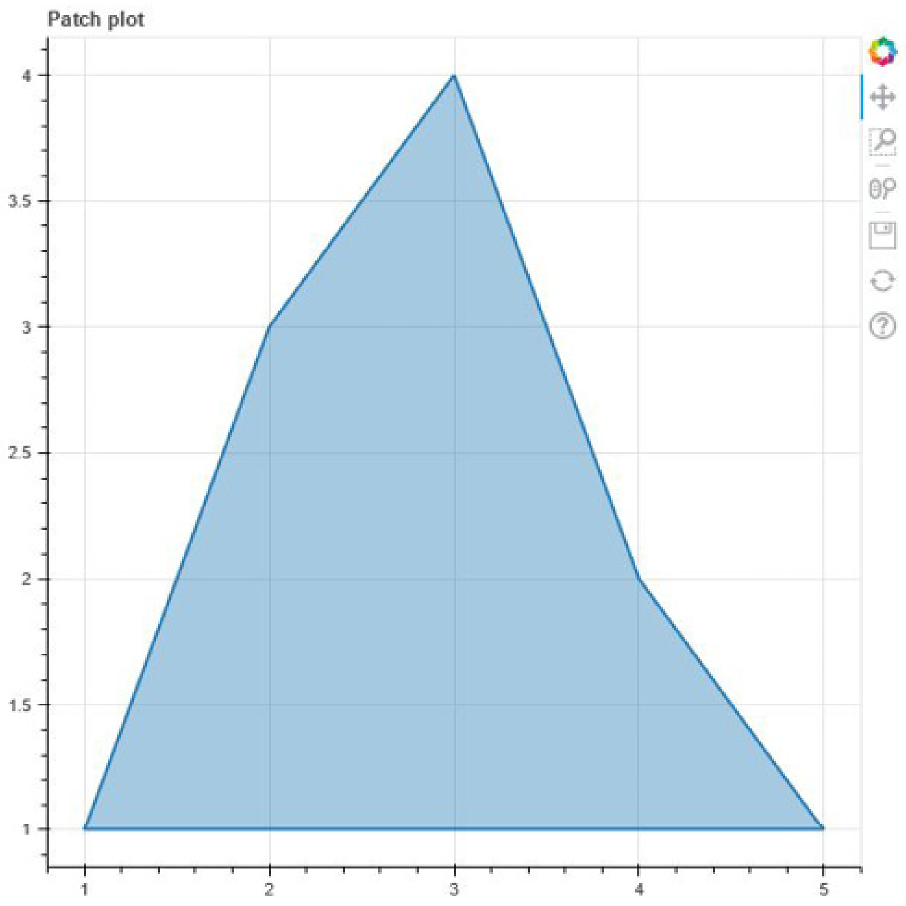


Figure 7.8: *Patch Plot*

In this example, we first define a sample polygon with five vertices. We then create a **ColumnDataSource** object to store the data for the polygon. We add the polygon to the data source using a dictionary that maps column names to data arrays.

We then create a figure object and add a Patch glyph to it using the patch method. We pass the column names of the data source to the x and y parameters of the patch method to specify the coordinates of the polygon. We also specify the **fill_alpha** and **line_width** parameters to control the appearance of the polygon.

Finally, we call the show function to display the plot in the default Bokeh output interface. The resulting plot will show the polygon defined by the x and y coordinates.

Area plots

Area plots are a type of chart that display quantitative data by plotting the cumulative values of a variable over time or another independent variable. In Bokeh, area plots can be created using the varea or harea methods of the Figure class, depending on whether you want a vertical or horizontal area plot.

Below is an example of how to create a basic vertical area plot using Bokeh:

```
from bokeh.plotting import figure, show
x = [1, 2, 3, 4, 5]
y1 = [2, 4, 3, 6, 4]
y2 = [1, 3, 2, 4, 3]

p = figure(title="Area Plot", x_axis_label="X-axis", y_axis_label="Y-axis")
p.varea(x=x, y1=y1, y2=y2, fill_color="blue")
Show(p)
```

Output:

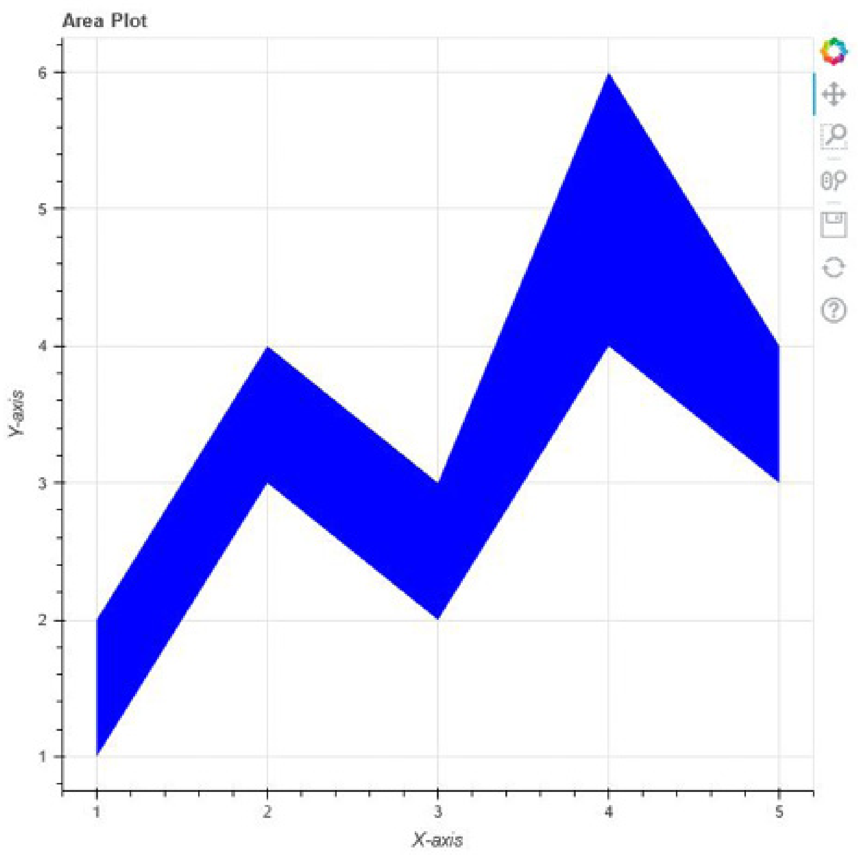


Figure 7.9: *Area Plot*

In this example, the varea method is used to create a vertical area plot with x as the independent variable and y1 and y2 as the cumulative variables. The **fill_color** argument is used to set the color of the filled area between the two lines.

You can also customize the appearance of the area plot using various properties and options, such as the **line_width**, **line_color**, **line_alpha**, and **fill_alpha** properties. For example, to make the pattern in the area and more transparent, and to increase the transparency of the filled area:

```
# add the vertical area plot with a hatching pattern and color

p.varea(x=x, y1=y1, y2=y2, fill_color="blue", hatch_color="red", hatch_pattern="dot")

show(p)
```

In this example, we add a green hatching pattern to the blue fill color using the **hatch_color** parameter set to "red” and the **hatch_pattern** parameter set to "dot".

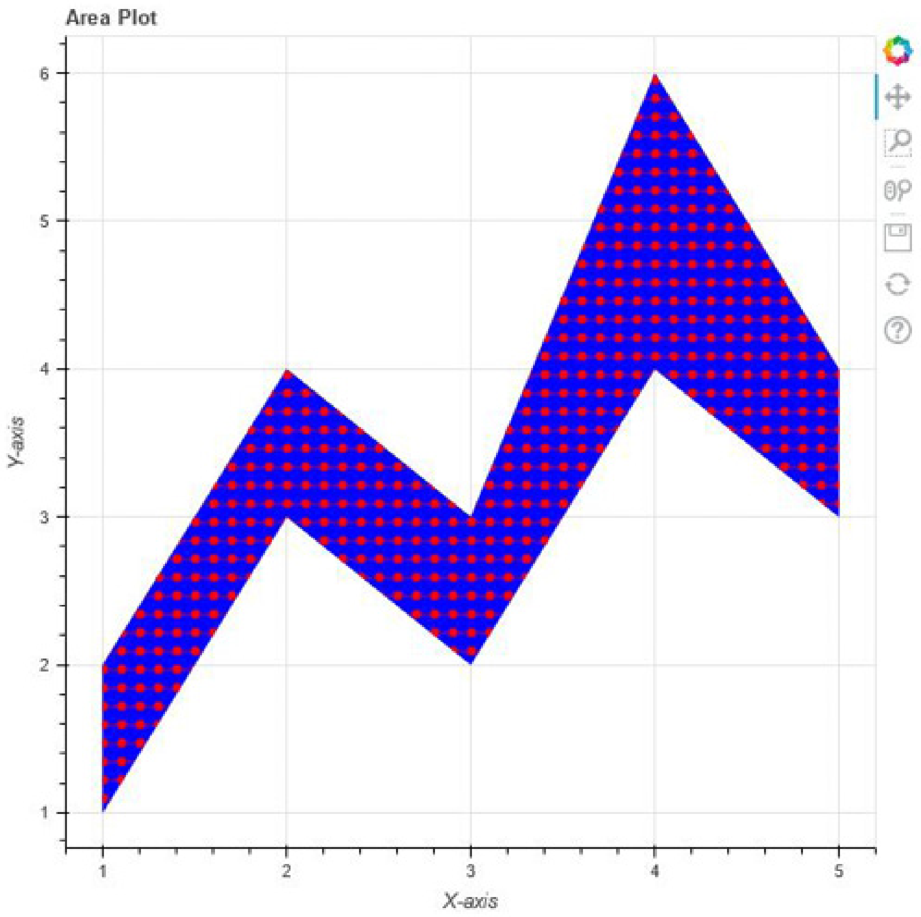


Figure 7.10: *Area plot customized with hatch_color and hatch_pattern*

You can also stack multiple area plots on top of each other to create a stacked area plot. For example, to stack two vertical area plots:

```
source = ColumnDataSource(data=dict(x=x,y1=y1,y2=y2))
p.varea_stack(['y1', 'y2'], x='x',
              fill_color=["lightblue", "lightgreen"],
              source=source, legend_label=["Area 1", "Area 2"])
show(p)
```

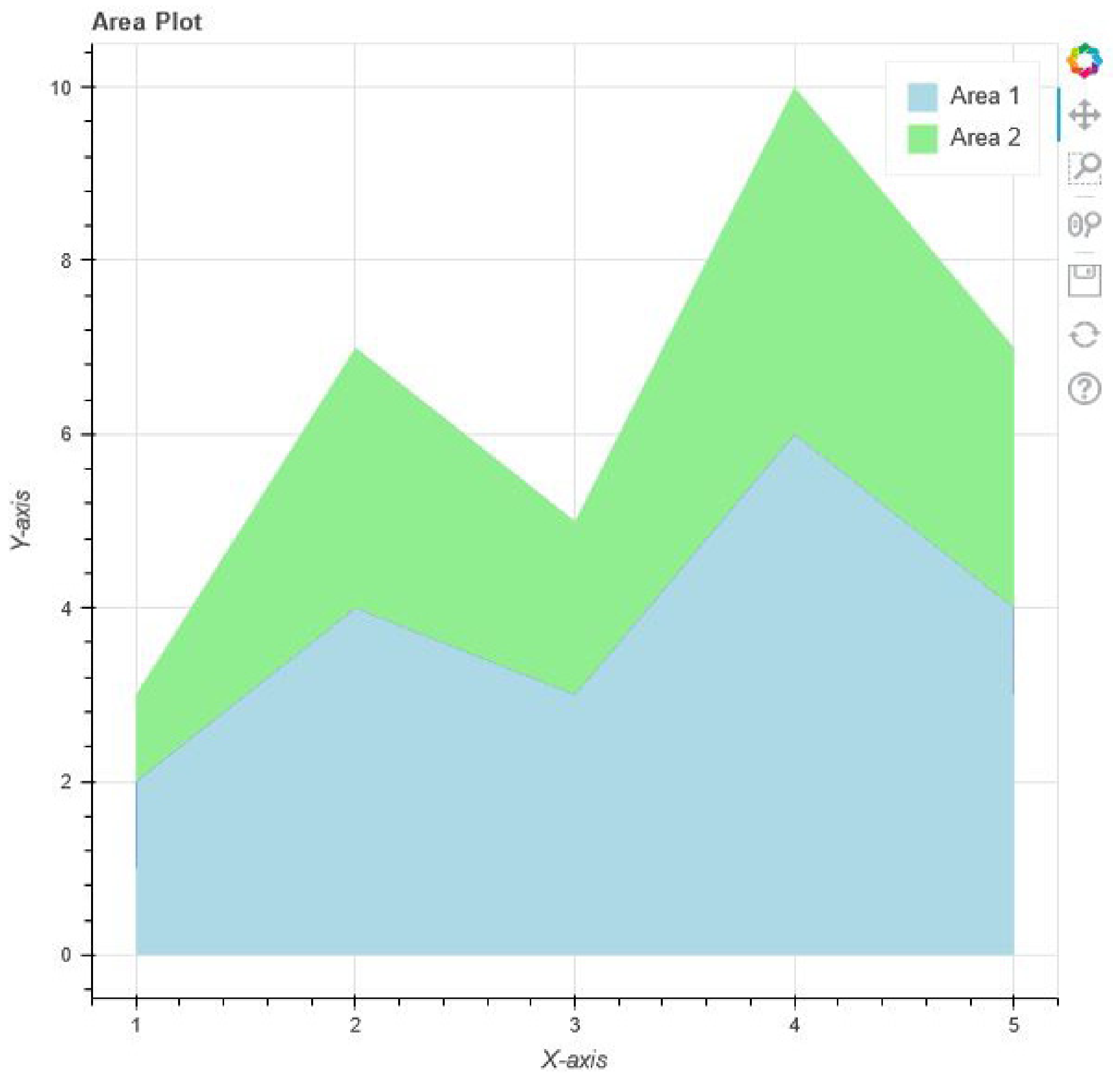


Figure 7.11: *Area Plot Customized with fill_color*

In this example, the **varea_stack** method is used to stack y1 and y2 on top of each other, with the **fill_color** argument setting the color of each filled area, and the **legend_label** argument setting the label for each area plot in the legend.

These are just a few examples of how to create and customize area plots in Bokeh. The library provides many other options and properties to create a wide range of area plots.

Stacked bar plot

A stacked bar plot is a type of chart that shows the distribution of a categorical variable by stacking the values of each category on top of each other. In Bokeh, stacked bar plots can be created using the **vbar_stack** or **hbar_stack** methods of the

Figure class, depending on whether you want a vertical or horizontal stacked bar plot.

Here is an example of how to create a basic vertical stacked bar plot using Bokeh:

```
from bokeh.plotting import figure, show
from bokeh.palettes import Spectral6
x = ["Books", "Magazine", "Posters"]
data = {'Product': x,
        '2020': [20, 30, 40],
        '2021': [10, 15, 20],
        '2022': [5,10,15]}
color=Spectral6[0:3]
# Convert the data to a ColumnDataSource
source = ColumnDataSource(data=data)
p = figure(x_range=x, plot_height=350, title="Product Sales by Year",
           toolbar_location=None, tools="")

p.vbar_stack(['2020','2021','2022'], x='Product', color=color,
             source=source)

show(p)
```

Output:

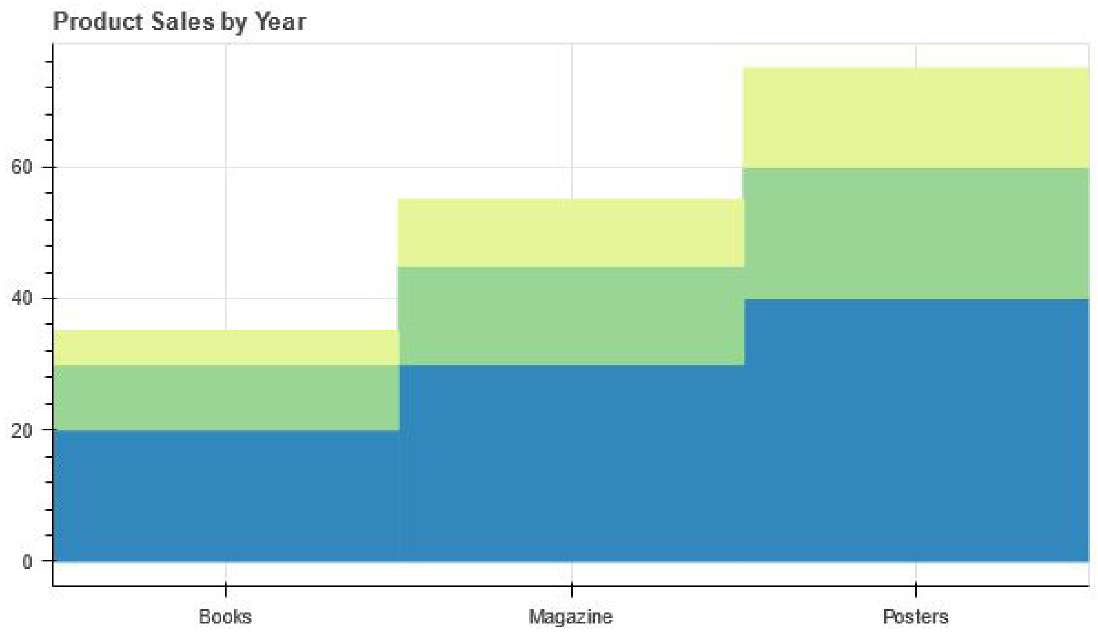


Figure 7.12: *Stacked Bar Plot*

In this example, the plot displays stacked vertical bars to represent the sales of different products over the years 2020, 2021, and 2022. The x axis of the plot shows the three different product types: "Books", "Magazine", and "Posters". The y axis of the plot shows the total sales for each product type for the years 2020, 2021, and 2022.

The Spectral6 palette from the **bokeh.palettes** module is used to define the colors of the bars. The Spectral6 palette contains six colors, and in this code we use the first three colors (using Spectral6[0:3]) to represent the three years.

The **vbar_stack** function is used to create the stacked vertical bars. The first parameter is a list of the keys from the data dictionary that correspond to the years that we want to stack. The x parameter is set to "Product" to indicate that we want to stack the bars for each product type. The color parameter is set to the Spectral6[0:3] list to define the colors of the bars. Finally, the source parameter is set to the **ColumnDataSource** object that we created earlier to provide the data for the plot. The **legend_label** argument is used to set the label for each stack in the legend.

You can also customize the appearance of the stacked bar plot using various properties and options, such as the **line_width**, **line_color**, **line_alpha**, and **fill_alpha** properties.

You can also create a horizontal stacked bar plot using the **hbar_stack** method, which works similarly to the **vbar_stack** method, but with the orientation of the plot and the dimensions of the bars flipped.

The library provides many other options and properties to create a wide range of stacked bar plots.

Creating interactive plots

Creating an interactive plot is a simple process that involves creating a Bokeh figure, defining the scatter plot data, and adding interactive features to the plot.

Here is an example of an interactive scatter plot in Bokeh:

```
from bokeh.plotting import figure, show, output_file
from bokeh.models import HoverTool
import pandas as pd

#Data
df = pd.DataFrame({
    'x':[1,2,3,4,5],
    'y':[4,7,1,6,3],
    'size':[10,20,30,40,50],
    'color':['red','green','blue','orange','purple']})
#create Bokeh figure and add scatter plot
p=figure(title='Interactive Scatter Plot',
         tools='box_select,lasso_select,reset')
p.scatter('x','y',size='size',color='color',alpha=0.5, source=df)
#Add hover tooltip
hover = HoverTool(tooltips=[('x', '@x'),('y','@y')])
p.add_tools(hover)
#Show plot in output file or in notebook
output_file('interactive_Scatter.html')
show(p)
```

Output:

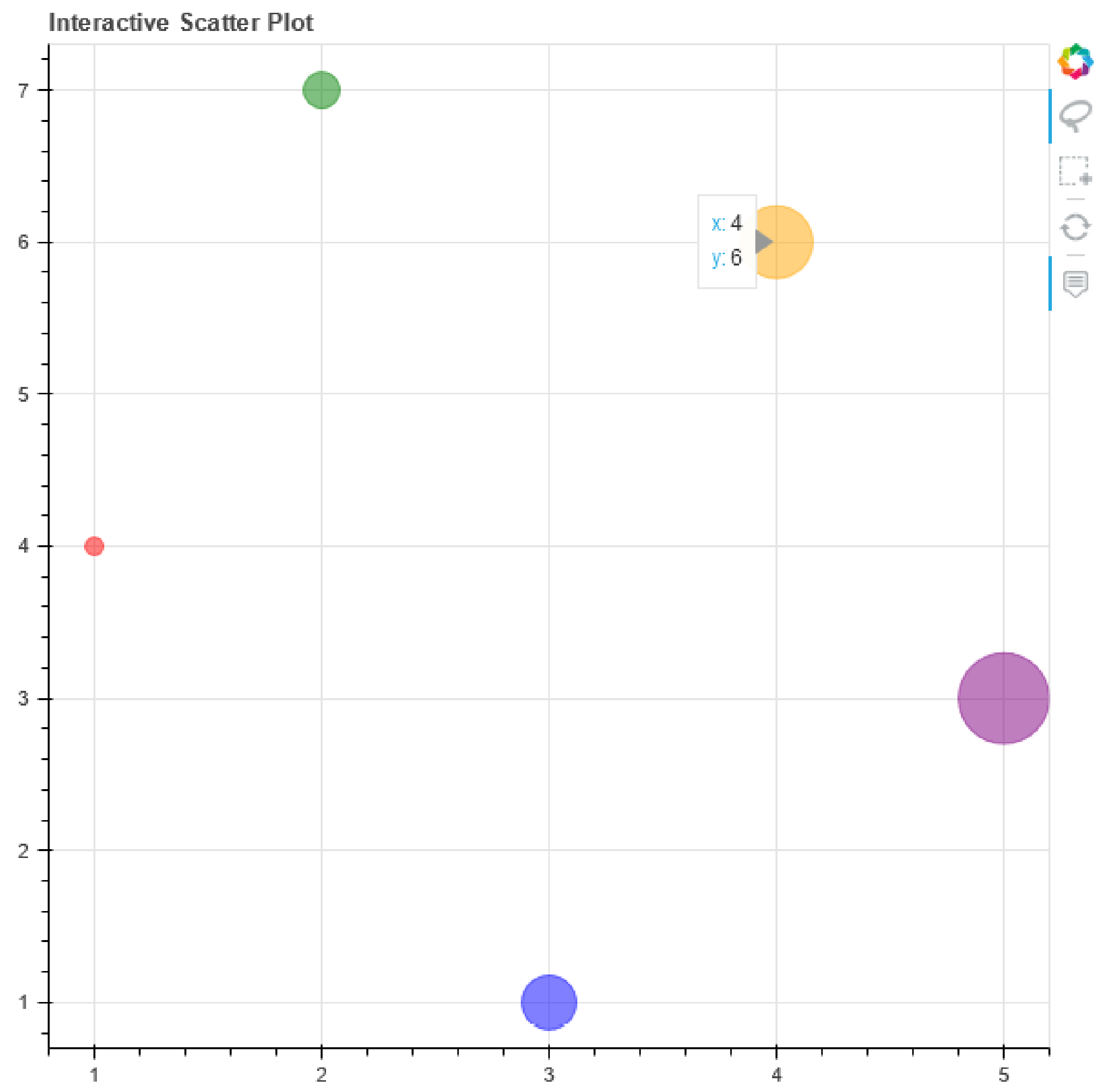


Figure 7.13: *Interactive Scatter Plot*

In this example, we first create a sample data frame with some random data. We then create a Bokeh figure using the figure function and add the scatter plot data using the scatter function. The size and color arguments are used to set the size and color of each scatter point based on the size and color columns in the data frame. The alpha argument is used to set the transparency of each scatter point.

Next, we add a hover tooltip to the scatter plot using the HoverTool from the **bokeh.models** module. The tooltips argument is used to define the information to be displayed in the tooltip when the user hovers over a scatter point.

Finally, we show the Bokeh plot in the output file or in the notebook using the **output_file** and show functions. The tools argument is used to add interactive tools to the plot, such as box select and lasso select, which allow users to select and highlight data points in the scatter plot.

This is just a simple example of how to create an interactive scatter plot in Bokeh. Bokeh provides many other tools and options for creating dynamic and responsive scatter plots, including zoom and pan, linked interactions, and custom JavaScript callbacks.

Creating multiple plots

Bokeh provides various methods to create multiple plots and combine them into a single layout. Here are some ways to create multiple plots using Bokeh:

1. **Using gridplot():** You can use the **gridplot()** function in the **bokeh.layouts** module to create a grid of plots. The **gridplot()** function takes a list of lists of plots, where each inner list represents a row of plots. Here's an example:

    ```
    from bokeh.layouts import gridplot
    from bokeh.plotting import figure, show

    #create two figures
    p1=figure(title="Plot 1",plot_width=250, plot_height=250)
    p1.circle([1,2,3],[4,5,6])

    p2=figure(title="Plot 2",plot_width=250, plot_height=250)
    p2.square([1,2,3],[4,5,6])

    #create grid og plots
    grid=gridplot([[p1,p2]])

    show(grid)
    ```

    In this example, we create two figures, add some glyphs to them, and then pass them to the **gridplot()** function to create a grid of plots.

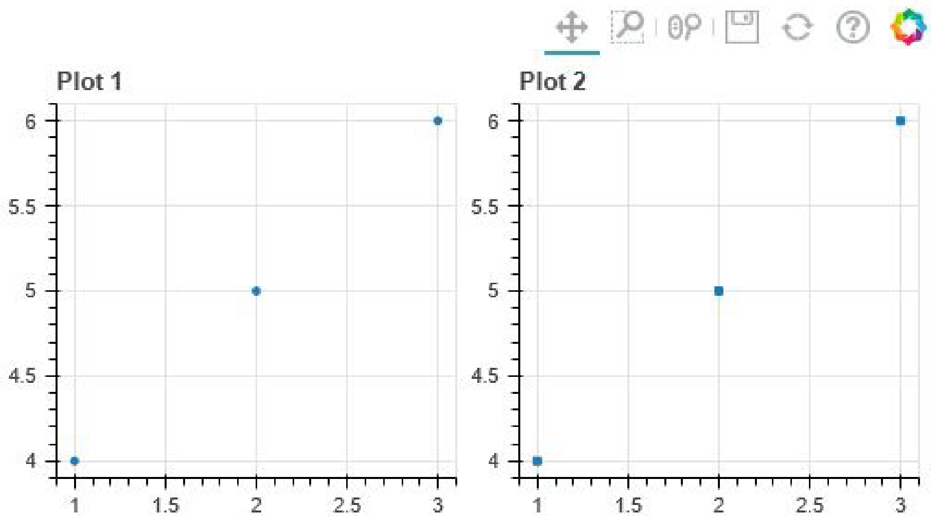


Figure 7.14: *Grid of Plots*

The **gridplot()** function in Bokeh is used to arrange multiple figure objects in a grid layout. Here are the parameters that can be passed to the **gridplot()** function:

- **children (required)**: A list of lists of Figure objects to be arranged in a grid. Each row in the grid is represented by a list of Figure objects, and the rows are arranged vertically in the grid.
- **plot_width (optional)**: The width of each plot in the grid, in pixels.
- **plot_height (optional)**: The height of each plot in the grid, in pixels.
- **toolbar_location (optional)**: The location of the toolbar for each plot in the grid. This can be "above", "below", "left", or "right".
- **toolbar_options (optional)**: A dictionary of options to pass to the toolbar. Possible options include "**logo**", "**help**", "**save**", and "**pan**". For example, **toolbar_options={"logo": None, "help": None}** will disable the logo and help buttons on the toolbar.
- **merge_tools (optional)**: If True, the toolbar for each plot in the grid will be merged into a single toolbar. The default is False.
- **sizing_mode (optional)**: Determines how the size of the grid will be calculated. Possible values include "**fixed**", "**stretch_both**", "**scale_width**", and "**scale_height**". The default is "fixed", which sets the size of the grid based on the **plot_width** and **plot_height** parameters.

2. **Using row() and column():** You can use the **row()** and **column()** functions in the **bokeh.layouts** module to create rows or columns of plots. The **'row(*children, sizing_mode='stretch_width')** takes a list of Bokeh plots or layout objects as arguments and arranges them horizontally in a row. The **sizing_mode** parameter controls how the layout should resize with respect to the parent container. The default value of **sizing_mode** is '**stretch_width**', which means that the width of the layout should stretch to match the width of the parent container.

   The '**column (*children, sizing_mode='stretch_height')** takes a list of Bokeh plots or layout objects as arguments and arranges them vertically in a column. The **sizing_mode** parameter controls how the layout should resize with respect to the parent container. The default value of **sizing_mode** is '**stretch_height**', which means that the height of the layout should stretch to match the height of the parent container.

   Here's an example:

   Creating rows/columns to display previously created plots - patch and **varea_stack**

```
from bokeh.plotting import figure, show
from bokeh.models import ColumnDataSource
# create a sample data
x = [1, 2, 3, 4, 5]
y = [1, 3, 4, 2, 1]
y1 = [2, 4, 3, 6, 4]
y2 = [1, 3, 2, 4, 3]
#create polygon with x and y
polygon = [(xi, yi) for xi, yi in zip(x, y)]
# create a First plot as patch glyph
source = ColumnDataSource(data=dict(x=x, y=y))
p1 = figure(title="Patch plot")
p1.patch(x='x', y='y', fill_alpha=0.4, line_width=2, source=source)
# create a Second plot as varea
p2 = figure(title="Area Plot", x_axis_label="X-axis", y_axis_label="Y-axis")
source = ColumnDataSource(data=dict(x=x,y1=y1,y2=y2))
```

```
p2.varea_stack(['y1', 'y2'], x='x', fill_color=["lightblue", "lightgreen"],
          source=source, legend_label=["Area 1", "Area 2"])
#Create Row of Plots
from bokeh.layouts import row
row=row(p1,p2)
show(row)
```

Output:

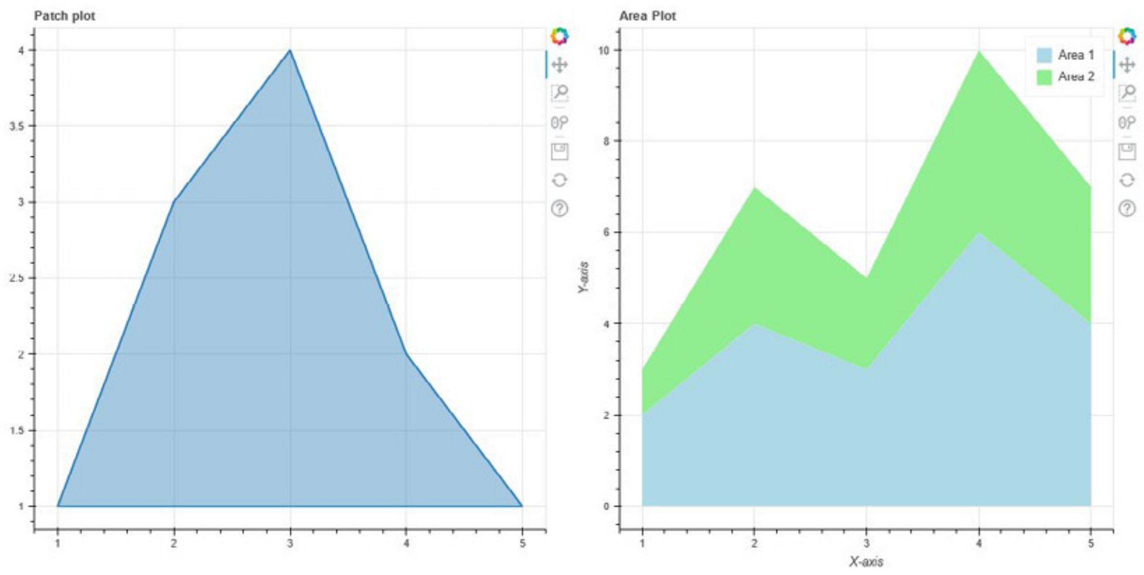


Figure 7.15: *Plotting in a row*

In this example, we create two figures, add some glyphs to them, and then pass them to the **row()** and **column()** functions to create a row or column of plots.

```
# create a row of plots
row = row(p1, p2)
show(row)
# create a column of plots
column = column(p1, p2)
```

3. **Using tabs()**: You can use the **Tabs()** class in the **bokeh.models** module to create a set of tabs, each containing a different plot. In Bokeh, Tabs is a layout container that displays one or more panels with tabs to switch

between them. The Tabs constructor takes a list of Tab objects as arguments.

Tab is a class that represents a single panel in a Tabs layout. It takes two arguments:

- **child:** a Bokeh layout object, such as a figure, column, row, or gridplot, that will be displayed in the panel.

- **title:** a string that will be displayed as the title of the panel's tab.

The Tabs constructor has several parameters that can be used to customize the appearance and behavior of the tabs layout:

- **tabs:** a list of Tab objects to display in the tabs layout. This parameter is required.

- **active:** an integer that specifies the index of the active tab when the tabs layout is first displayed. The default value is 0, which means that the first tab will be active.

- **sizing_mode:** a string that specifies how the tabs layout should resize with respect to the parent container. The default value is '**stretch_both**', which means that the layout should stretch both horizontally and vertically to match the size of the parent container.

- **width:** an integer that specifies the width of the tabs layout in pixels. If not specified, the layout will fill the available width of the parent container.

- **height:** an integer that specifies the height of the tabs layout in pixels. If not specified, the layout will fill the available height of the parent container.

- **tabs_location:** a string that specifies the location of the tabs with respect to the panel content. The allowed values are 'above', 'below', 'left', and 'right'. The default value is 'above'.

- **orientation:** a string that specifies the orientation of the tabs with respect to the panel content. The allowed values are 'horizontal' and 'vertical'. The default value is 'horizontal'.

- **css_classes:** a list of strings that specifies additional CSS classes to add to the tabs layout.

Here's an example:

Creating tabs to display previously created plots - patch and varea_stack

```
from bokeh.plotting import figure, show
from bokeh.models import ColumnDataSource

# create a sample data
x = [1, 2, 3, 4, 5]
y = [1, 3, 4, 2, 1]
y1 = [2, 4, 3, 6, 4]
y2 = [1, 3, 2, 4, 3]

#create polygon with x and y
polygon = [(xi, yi) for xi, yi in zip(x, y)]

# create a First plot as patch glyph
source = ColumnDataSource(data=dict(x=x, y=y))
p1 = figure(title="Patch plot")
p1.patch(x='x', y='y', fill_alpha=0.4, line_width=2, source=source)

# create a Second plot as varea
p2 = figure(title="Area Plot",
        x_axis_label="X-axis", y_axis_label="Y-axis")
source = ColumnDataSource(data=dict(x=x,y1=y1,y2=y2))
p2.varea_stack(['y1', 'y2'], x='x',
         fill_color=["lightblue", "lightgreen"],
         source=source, legend_label=["Area 1", "Area 2"])

#Create tabs
from bokeh.models import Panel, Tabs
```

```
tab1 = Panel(child=p1,title="PATCH PLOT")
tab2=Panel(child=p2,title="VAREA PLOT")
tabs=Tabs(tabs=[tab1,tab2])
output_file("Plots_in_tab.html", tabs)
show(tabs)
```

Output:

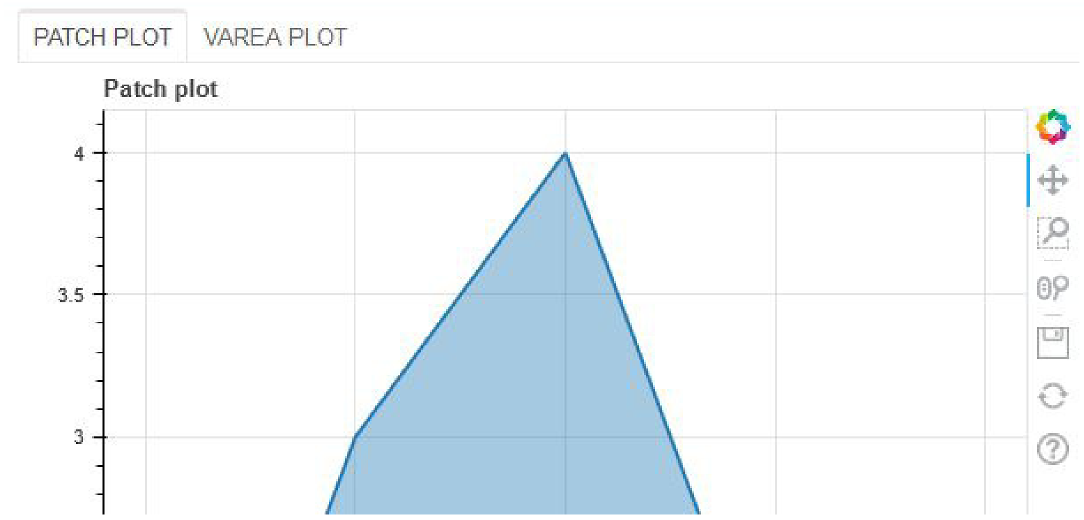


Figure 7.16: Creating Tabs for plot display

In this example, we create two figures, add some glyphs to them, and then create two Panel objects, each containing a plot and a title.

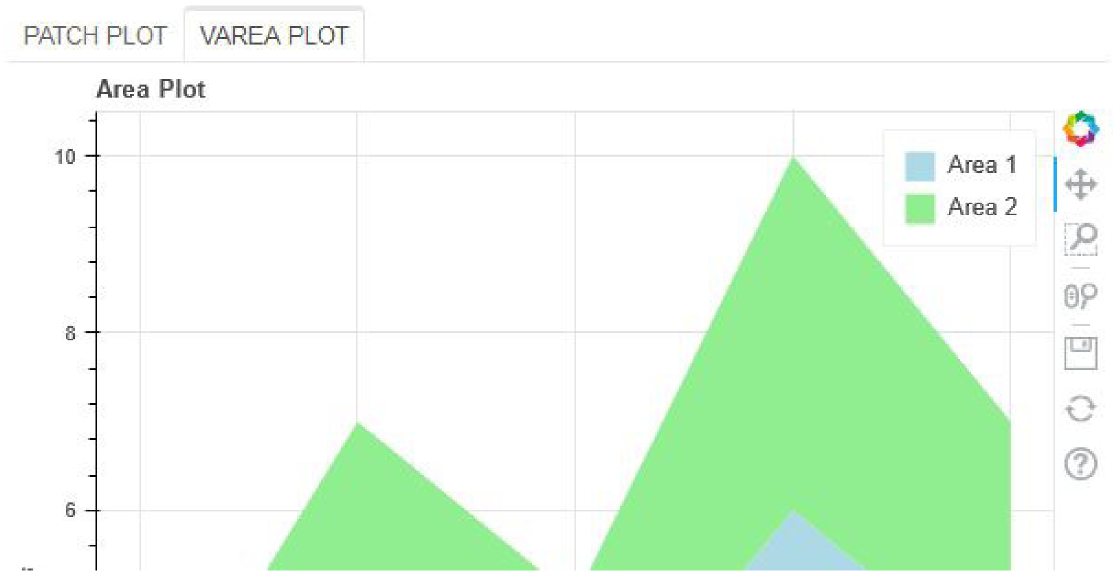


Figure 7.17: *Displaying Tab 2 from the plot*

We then pass these panels to the **Tabs()** class to create a set of tabs, each containing a different plot.

Saving the plot

You can save a Bokeh plot to an HTML file using the **save()** function. The save() function takes a **bokeh.plotting.figure.Figure** object and a filename. Here's an example:

```
from bokeh.plotting import figure, output_file, save
# create a figure
p = figure()
# add some glyphs to the figure
p.circle([1, 2, 3], [4, 5, 6])
# specify the output file name
output_file("my_plot.html")
# save the plot
save(p)
```

In this example, we create a Figure object, add some glyphs to it, and specify the output file name using the **output_file()** function. We then call the **save()** function to save the plot to the specified file.

By default, the **save()** function creates an HTML file that includes all the necessary JavaScript and CSS files. You can also save a standalone HTML file that does not require an internet connection or Bokeh server by passing mode="inline" to the **save()** function:
save(p, filename="my_plot.html", title="My Plot", mode="inline")

This will create a standalone HTML file with the given title that includes all the necessary JavaScript and CSS code.

Here are the parameters that can be passed to the **save()** function:

- **fig** (required): The Figure object to be saved.
- **filename** (optional): The name of the output file. If this is not provided, a default file name will be used.

- **title** (optional): The title of the HTML document.
- **resources** (optional): Controls how BokehJS resources (for example JavaScript, CSS) are included in the HTML document. The default is "inline", which embeds all resources in the HTML file. Other options include "**cdn**" (which loads resources from a content delivery network) and "relative" (which creates relative links to resources).
- **mode** (optional): Determines how the output file will be embedded in the HTML document. The default is "**cdn**", which includes the BokehJS JavaScript code from a content delivery network. Other options include "inline", which includes the BokehJS code in the HTML file, and "absolute", which creates an absolute URL for the BokehJS code.
- **root_dir** (optional): If resources are being included using "relative" mode, this parameter specifies the root directory to use for relative URLs.
- **template** (optional): The Jinja2 template to use for generating the HTML document. If not provided, the default Bokeh template will be used.

Here's an example of how to use some of these parameters:

```
from bokeh.plotting import figure, save

p = figure(title="My Plot")

# Add some glyphs to the figure...

save(p, filename="my_plot.html", title="My Plot", resources="cdn")
```

In this example, we create a **Figure** object and add some glyphs to it. Call function **save()** with the filename and title parameters to specify the output file name and document title. The "**cdn**" for the resources parameter is mentioned, which will load BokehJS resources from a content delivery network.

To save a Bokeh plot to PNG or SVG format, use the **export_png()** or **export_svg()** functions, respectively. Below is an example code snippet demonstrating how to use the **export_png()** and **export_svg()** functions in Bokeh to save a plot to a PNG or SVG file:

```
from bokeh.plotting import figure, output_file, show

from bokeh.io import export_png, export_svg

# Define the plot
```

```
p = figure(title='Example Plot', x_axis_label='X Axis', y_axis_label='Y Axis')
p.line(x=[1, 2, 3, 4, 5], y=[2, 4, 6, 8, 10], line_width=2)
# Save the plot to PNG format
export_png(p, filename='example_plot.png')
# Save the plot to SVG format
export_svg(p, filename='example_plot.svg')
```

In this example, we first define a simple line plot using the **figure()** function from Bokeh, with a title and labels for the x and y axes. We then use the **line()** method to add a line glyph to the plot.

Next, we use the **export_png()** function to save the plot to a PNG file, with the filename **example_plot.png**. Similarly, we use the **export_svg()** function to save the plot to an SVG file, with the filename **example_plot.svg.**

Note that we also need to import the **export_png()** and **export_svg()** functions from the bokeh.io module. Additionally, we can use the **output_file()** function to specify an output file for the plot if we want to save it to HTML format instead. Finally, we use the **show()** function to display the plot in the Jupyter notebook or web browser.

Conclusion

In this chapter, we have learned that Bokeh is a powerful and flexible data visualization library that provides a wide range of tools for creating interactive, web-based visualizations in Python. It offers a high-level interface for creating common chart types, such as scatter plots, line charts, and bar charts, as well as more advanced visualizations, such as heatmaps, parallel coordinate plots, and choropleth maps. Bokeh also supports various data sources, including Pandas DataFrames, NumPy arrays, and JSON data.

Bokeh's key strength is its interactivity, allowing users to create dynamic, responsive visualizations that can be easily shared and explored in web browsers. This makes it a popular choice for applications such as dashboards, data exploration tools, and scientific and engineering visualizations.

Bokeh is an excellent choice for creating high-quality, interactive visualizations in Python, and its extensive documentation and large community make it easy to learn and use. Remember to experiment with different shapes and colors, and don't be afraid to get creative with your bokeh shots. In the next chapter, you will learn conducting EDA, empowering you to extract meaningful information from your datasets. By utilizing various visualizations and statistical methods, you will uncover relationships, detect outliers, explore distributions, and ultimately derive valuable conclusions that can drive informed decision-making. EDA acts as a foundation for further data exploration and model building, making it an indispensable skill for any data scientist or analyst.

Points to remember

- A figure is the overall window or container that holds a visualization, including any axes, legends, and glyphs.
- Glyphs are the visual elements used to represent data points, such as points, lines, bars, or text.
- Axes are the visual elements that define the scale and orientation of the visualization, including the x-axis, y-axis, and any secondary axes.
- The x-axis is typically used to represent the independent variable or time, while the y-axis represents the dependent variable.
- Axes can be customized to include ticks, labels, grid lines, and other visual elements that help to interpret the data.
- Glyphs can be customized to include color, size, shape, and other visual elements that help to distinguish between different data points.
- Understanding the relationship between glyphs and axes is important for creating effective visualizations that communicate the intended message.
- Different types of visualizations, such as scatterplots, line charts, and bar charts, require different combinations of glyphs and axes to effectively communicate the data.
- Bokeh allows for a high degree of interactivity, with features such as hover tooltips, zooming, panning, and selection.
- Customizing the appearance of a plot involves setting properties such as background color, font size, and axis labels.
- Bokeh supports a range of output formats, including HTML, PNG, and SVG.
- Bokeh can be integrated with other Python libraries, such as NumPy, Pandas, and Scikit-learn, for data analysis and machine learning.

- To create multiple plots in Bokeh, use the **gridplot()** function to arrange individual plots in a grid layout.
- To link multiple plots together, use the **link()**, **Range1d()**, and **ColumnDataSource()** functions to synchronize data ranges and data sources.
- To save a Bokeh plot to HTML format, use the **output_file()** function and specify the filename and location for the output file.
- To save a Bokeh plot to PNG or SVG format, use the **export_png()** or **export_svg()** functions, respectively.
- When saving a Bokeh plot, it is important to consider the size and resolution of the output file, as well as any additional styling or formatting that may be required.

Questions

1. What are the main features of Bokeh library?
2. Elaborate the functionality of **bokeh.palettes** module
3. With the help of a snippet code, elaborate how to set the x-axis and y-axis labels in a Bokeh figure?
4. How do you create a scatter plot with Bokeh on the following data points:

   x = [1, 2, 3, 4, 5]

   y = [6, 7, 2, 4, 5]
5. How do you change the font size of the x-axis and y-axis labels in a Bokeh figure for the scatter plot created in the previous question?
6. With the help of a code, demonstrate adding a tooltip to a Bokeh plot?
7. With the help of a code, demonstrate how to create a dropdown menu to filter data to a Bokeh plot?

Join our book’s Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

https://discord.bpbonline.com

CHAPTER 8
Exploratory Data Analysis

This is my favorite part about analytics: Taking boring, flat data and bringing it to life through visualization.

– John W. Tukey

Data visualization can be used to clean data, explore data structure, detect outliers and unusual groups, identify trends and clusters, spot local patterns, evaluate modeling output, and present results. It is critical for exploratory data analysis and data mining to check data quality and assist analysts in becoming familiar with the structure and features of the data in front of them. This is an aspect of data analysis that is overlooked in textbooks but is prevalent in real-world investigations. The chapter is on exploratory data analysis, its types. This chapter also presents the basic steps followed in exploratory data analysis. Examples using standard datasets are presented for better understanding of **Exploratory Data Analysis (EDA)**.

Structure

In this chapter, we will discuss the following topics:

- Introduction
- Types of EDA
- Steps to perform EDA
- EDA using dataset in python

Objectives

In this chapter, the main objective is to walk you through the exploratory data analysis. EDA is an iterative process and may require multiple rounds of analysis to gain a complete understanding of the data. The insights gained from EDA can be used to formulate hypotheses, which can then be tested using more formal statistical methods. Overall, EDA is an important step in the data analysis process, as it helps you understand your data and prepares you for more in-depth analysis.

Introduction

Exploratory data analysis (**EDA**) is a process of analyzing data sets to summarize their main characteristics, often with visual methods. A primary goal of EDA is to uncover relationships among different variables. Python is a great language for performing EDA because of its strong data science libraries and tools. Python's pandas library is a powerful tool for manipulating data sets and performing exploratory data analysis. It provides a wide range of functions for exploring and analyzing data, including plotting, summary statistics, and data cleaning. Additionally, Python's matplotlib and seaborn libraries provide powerful tools for visualizing data. EDA is a process of analyzing and summarizing the main characteristics of a data set, often with visual methods. The goal of EDA is to gain a better understanding of the data and identify patterns, relationships, and trends that can be used to guide further analysis.

Types of EDA

There are three main types of EDA that can be performed on a dataset:

- **Univariate Analysis**: This involves examining each variable in the dataset individually. The main goal is to summarize the main characteristics of the variable and to identify any patterns or outliers. Common techniques used in univariate analysis include histograms, box plots, and summary statistics such as mean, median, mode, and standard deviation.

- **Bivariate Analysis**: This involves examining the relationship between two variables in the dataset. The main goal is to identify any patterns or correlations between the variables. Common techniques used in bivariate analysis include scatter plots, line graphs, and correlation analysis.
- **Multivariate Analysis**: This involves examining the relationship between three or more variables in the dataset. The main goal is to identify any complex patterns or relationships that may exist between the variables. Common techniques used in multivariate analysis include multiple regression analysis, factor analysis, and cluster analysis.

Each type of EDA provides different insights into the dataset and can help guide further analysis or modeling. It's important to use a combination of all three types of EDA to get a comprehensive understanding of the data.

Univariate analysis

The main purpose of univariate analysis is to explore the properties and characteristics of a single variable in a dataset. Univariate analysis can help in:

- **Identifying outliers or unusual observations**: Univariate analysis can help identify any unusual or outlier values within a variable. These values may indicate errors in the data collection process or provide insight into unique observations within the dataset.
- **Understanding the central tendency of the variable**: Univariate analysis can help identify the central tendency of a variable, such as its mean, median, or mode. This can provide insight into the most common value or typical value of the variable.
- **Describing the spread or dispersion of the variable**: Univariate analysis can help identify the range, variance, or standard deviation of the variable. This can provide insight into the spread of the variable, or how widely the values are distributed.

- **Identifying the shape of the variable's distribution**: Univariate analysis can help identify the shape of the distribution of the variable. This can provide insight into whether the data is normally distributed, skewed to one side, or has multiple peaks.

By understanding the properties and characteristics of a single variable, univariate analysis can help identify potential issues with the data, guide further analysis or modeling, and provide insights into the underlying processes or phenomena being measured by the variable.

Suppose you have a dataset of the heights of 100 people. The variable of interest is "height". You want to explore the properties and characteristics of this variable using univariate analysis.

Identifying outliers or unusual observations: You can create a box plot or histogram to identify any unusual or outlier values within the height variable. For example, you may notice that one person in the dataset is 8 feet tall, which is a highly unusual observation.

Understanding the central tendency of the variable: You can calculate the mean, median, or mode of the height variable to understand its central tendency. For example, you may find that the mean height in the dataset is 5 feet 7 inches.

Describing the spread or dispersion of the variable: You can calculate the range, variance, or standard deviation of the height variable to understand its spread or dispersion. For example, you may find that the standard deviation of the height variable is 3 inches, which indicates that the values are closely clustered around the mean.

Identifying the shape of the variable's distribution: You can create a histogram to identify the shape of the distribution of the height variable. For example, you may find that the distribution of heights is approximately normal, with a bell-shaped curve.

By performing univariate analysis on the height variable, you can gain insight into its properties and characteristics, which can guide further analysis or modeling.

Bivariate analysis

The main purpose of bivariate analysis is to explore the relationship between two variables in a dataset. Bivariate analysis involves examining the distribution of each variable separately, as well as analyzing the relationship between the two variables.

The main purposes of bivariate analysis include:

- **Identifying patterns and trends**: Bivariate analysis can help identify patterns and trends that may exist between two variables. For example, it can help determine whether an increase in one variable is associated with an increase or decrease in the other variable.
- **Measuring the strength and direction of the relationship**: Bivariate analysis can help quantify the strength and direction of the relationship between two variables. For example, it can help determine whether the relationship is positive, negative, or nonexistent, as well as the degree of correlation between the two variables.
- **Testing hypotheses**: Bivariate analysis can be used to test hypotheses about the relationship between two variables. For example, it can help determine whether there is a statistically significant relationship between two variables, and whether the relationship is causal or merely correlational.
- **Guiding further analysis**: Bivariate analysis can guide further analysis or modeling by providing insights into the relationships and patterns that exist between two variables.

Bivariate analysis is an important technique in data analysis and is often used in fields such as economics, social sciences, and business analytics to explore the relationships between different variables. However, it is important to also perform multivariate analysis to understand the relationships and patterns that may exist between three or more variables.

Suppose you have a dataset that contains information about the age and income of 1000 individuals. The two variables of interest are "age" and "income". You want to explore the relationship between these two variables using bivariate analysis. You can perform it in several ways as follows:

Identifying patterns and trends: You can create a scatter plot to identify any patterns or trends that may exist between age and income. For example, you may notice that as age increases, income tends to increase as well.

Measuring the strength and direction of the relationship: You can calculate the correlation coefficient between age and income to measure the strength and direction of the relationship. For example, you may find that there is a positive correlation between age and income, with a correlation coefficient of 0.5. This indicates that as age increases, income tends to increase as well, but the relationship is not perfect.

Testing hypotheses: You can use statistical tests, such as a t-test or ANOVA, to test hypotheses about the relationship between age and income. For example, you may want to test whether the mean income of individuals over 50 years old is significantly different from the mean income of individuals under 50 years old.

Guiding further analysis: Bivariate analysis can guide further analysis or modeling by providing insights into the relationships and patterns that exist between two variables. For example, you may want to include age as a predictor variable in a linear regression model to predict income.

By performing bivariate analysis on the age and income variables, you can gain insights into their relationship, which can guide further analysis or modeling.

Multivariate analysis

The main purpose of multivariate analysis is to examine the relationship between three or more variables in a dataset. Multivariate analysis involves exploring the relationships between multiple variables simultaneously to gain a more comprehensive understanding of complex phenomena. The main purposes of multivariate analysis include:

- **Identifying patterns and trends**: Multivariate analysis can help identify complex patterns and trends that may exist among multiple variables. For example, it can help determine whether multiple variables are associated with one another in a specific way.

- **Understanding the interrelationships between variables**: Multivariate analysis can help understand how variables are interrelated, and how changes in one variable may influence changes in other variables. For example, it can help determine whether a change in one variable is associated with a change in another variable, after controlling for other variables.
- **Exploring the joint effects of variables**: Multivariate analysis can help explore the joint effects of multiple variables on an outcome. For example, it can help determine whether multiple variables interact with one another to influence an outcome.
- **Testing hypotheses**: Multivariate analysis can be used to test hypotheses about the relationships between multiple variables. For example, it can help determine whether there is a statistically significant relationship between three or more variables, and whether the relationships are causal or merely correlational.
- **Guiding further analysis**: Multivariate analysis can guide further analysis or modeling by providing insights into the relationships and patterns that exist among multiple variables.

Multivariate analysis is an important technique in data analysis and is often used in fields such as psychology, sociology, marketing, and environmental science to understand complex phenomena involving multiple variables. Suppose you have a dataset that contains information about the age, income, and education level of 1000 individuals. The three variables of interest are "age", "income", and "education". You want to explore the relationships between these three variables using multivariate analysis.

Identifying patterns and trends: You can create a three-dimensional scatter plot to identify any patterns or trends that may exist among age, income, and education. For example, you may notice that as age increases, income tends to increase as well, but the relationship is stronger for individuals with higher levels of education.

Understanding the interrelationships between variables: You can calculate the correlation coefficients between age, income, and education to understand how these variables are interrelated. For example, you may find that income and education are positively correlated, but age is negatively correlated with both income and education.

Exploring the joint effects of variables: You can conduct a multiple regression analysis to explore the joint effects of age, income, and education on an outcome variable, such as job satisfaction. For example, you may find that income and education have a significant positive effect on job satisfaction, but age has a significant negative effect.

Testing hypotheses: You can use statistical tests, such as a multivariate ANOVA, to test hypotheses about the relationships between age, income, and education. For example, you may want to test whether there is a significant difference in job satisfaction among individuals with different levels of education, after controlling for age and income.

Guiding further analysis: Multivariate analysis can guide further analysis or modeling by providing insights into the relationships and patterns that exist among multiple variables. For example, you may want to include age, income, and education as predictor variables in a linear regression model to predict job satisfaction.

By performing multivariate analysis on the age, income, and education variables, you can gain insights into their complex relationships, which can guide further analysis or modeling.

Steps to perform EDA

EDA is an iterative process and may require multiple rounds of analysis to gain a complete understanding of the data. The insights gained from EDA can be used to formulate hypotheses, which can then be tested using more formal statistical methods. Overall, EDA is an important step in the data analysis process, as it helps you understand your data and prepares you for more in-depth analysis. EDA typically involves several steps, including:

1. **Data preparation:** This involves cleaning the data and transforming it into a format suitable for analysis.
2. **Univariate analysis:** This involves analyzing each variable separately to get a better understanding of their distributions, missing values, and outliers.
3. **Bivariate analysis:** This involves analyzing the relationship between two variables. This can be done using statistical tests, visualization techniques, or both.
4. **Multivariate analysis:** This involves analyzing the relationship between multiple variables. This can include techniques such as factor analysis, clustering, and regression analysis.
5. **Data visualization:** Data visualization is a key component of EDA, as it allows you to quickly see patterns and relationships in the data. Common visualization techniques include histograms, box plots, scatter plots, and heat maps.

From a programming point of view, you will be following the following step for performing EDA on any dataset. With these tools, you can quickly explore data sets and uncover relationships between variables.

1. **Import Libraries**: Start by importing the necessary libraries such as pandas, numpy, matplotlib, seaborn, and so on.
2. **Load Data**: Load the data into a pandas dataframe.
3. **Data Preprocessing**:
    a. **Descriptive Statistics**: Generate descriptive statistics such as mean, median, mode, standard deviation, and so on.
    b. **Data Cleaning**: Perform any necessary data cleaning such as dealing with missing values, dealing with outliers, and so on.
    c. **Data Visualization**: Create visualizations such as histograms, boxplots, scatterplots, and so on. to better understand the data.
    d. **Feature Engineering**: Perform any necessary feature engineering such as creating new features, transforming existing features, and so on.

4. **Correlation analysis:** Perform correlation analysis to understand the relationship between different features.
5. **Model building:** Build predictive models such as linear regression, logistic regression, decision trees, and so on. to predict the target variable.

Statistical analysis

EDA is a crucial step in the machine learning process, as it helps you to understand your data and identify patterns, relationships, and insights that can inform the development of your machine learning model. Statistical analysis is an important tool for EDA, as it provides a way to summarize, visualize, and make inferences about your data.

Statistical measures are mathematical quantities used to summarize and describe the characteristics of a set of data. There are several common statistical measures, including:

- **Mean:** The mean, or average, of a set of values is calculated by adding up all the values and dividing by the number of values in the set. The mean is a measure of central tendency that gives a rough idea of the typical value in a set.
- **Median:** The median of a set of values is the middle value when the set is ordered from least to greatest. The median is a measure of central tendency that is less sensitive to outliers than the mean.
- **Mode:** The mode of a set of values is the value that appears most frequently in the set. If there is more than one mode, the set is said to be bimodal or multimodal.
- **Standard Deviation:** The standard deviation is a measure of the spread of values in a set. A low standard deviation indicates that the values are tightly clustered around the mean, while a high standard deviation indicates that the values are more spread out.
- **Variance:** The variance is another measure of the spread of values in a set. It is calculated as the average of the squared differences between each value and the mean of the set.

- **Quartiles:** Quartiles divide a set of values into four equal parts. The first quartile, Q1, is the 25th percentile of the data and separates the lowest 25% of the values from the rest. The second quartile, Q2, is the median, or 50th percentile, and separates the lowest 50% of the values from the highest 50%. The third quartile, Q3, is the 75th percentile of the data and separates the lowest 75% of the values from the highest 25%.
- **Percentiles:** Percentiles divide a set of values into 100 equal parts. Percentiles are useful for comparing the distribution of values in different sets of data.
- **Z-score:** The Z-score is a statistical measure used to determine whether a value is significantly different from the mean of a set of values. The Z-score is calculated as the difference between a value and the mean of the set, divided by the standard deviation of the set. For example, consider a set of values with a mean of 100 and a standard deviation of 10. If a value of 120 is found in the set, the Z-score for that value would be *(120-100)/10 = 2*. A positive Z-score indicates that a value is above the mean, and a negative Z-score indicates that a value is below the mean. A Z-score of 0 indicates that the value is exactly equal to the mean. Z-scores can be used to identify outliers in a dataset, as outliers are typically defined as values that are significantly different from the mean. For example, a common definition of outliers is values with a Z-score greater than 3 or less than -3, as these values are more than three standard deviations away from the mean. By calculating Z-scores, you can identify values that are significantly different from the mean and take appropriate action, such as removing them from the dataset, imputing missing values, or transforming the data.

Here are some common statistical analysis techniques used in EDA:

- **Descriptive statistics:** This involves calculating summary statistics such as mean, median, mode, standard deviation, and quantiles for each column in your dataset. These statistics can help you to understand the central tendency, spread, and shape of your data.

- **Distribution plots:** Distribution plots, such as histograms, kernel density plots, and box plots, can be used to visualize the distribution of values in each column in your dataset. These plots can help you to identify patterns, outliers, and skewness in your data.
- **Correlation analysis**: Correlation analysis can be used to measure the strength and direction of the relationship between two variables. Correlation coefficients, such as Pearson's and Spearman's, can be calculated to measure the linear association between two variables.
- **Hypothesis testing:** Hypothesis testing is a statistical method used to test whether a sample of data is significantly different from a population. For example, you might use hypothesis testing to test whether the mean age of a sample of survivors in the Titanic dataset is significantly different from the mean age of a sample of non-survivors.
- **Principal Component Analysis (PCA):** PCA is a dimensionality reduction technique that can be used to reduce the number of variables in your dataset while retaining as much information as possible. This can be useful for visualizing patterns in high-dimensional datasets and for removing redundant variables.

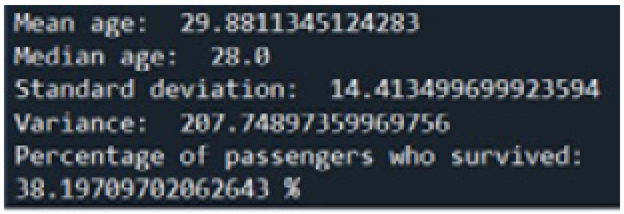
By using these statistical analysis techniques in EDA, you can gain a deeper understanding of your data and make informed decisions about how to proceed with building your machine learning model.

Below is the reference on descriptive statistical analysis using pandas on titanic survival dataset:

```
import pandas as pd

#Load the Titanic dataset

df = pd.read_csv("titanic.csv")

#Calculate the mean age of the passengers

mean_age = df['age'].mean()

print("Mean age:", mean_age)

#Calculate the median of the age of the passengers
```

```
median_age= df['age'].median()
print("Median age:", median_age)
#Calculate the standard deviation of the age of the passengers
std_dev = df['age'].std()
print("Standard deviation:", std_dev)
#Calculate the variance of the age of the passengers
var = df['age'].var()
print("Variance:", var)
#Calculate the percentage of passengers who survived
percentage_survived = df['survived'].mean()*100
print("Percentage of passengers who survived:", percentage_survived)
```

Output:



```
Mean age:  29.8811345124283
Median age:  28.0
Standard deviation:  14.413499699923594
Variance:  207.74897359969756
Percentage of passengers who survived:
38.19709702062643 %
```

Figure 8.1: *Descriptive Statistical Analysis using Pandas*

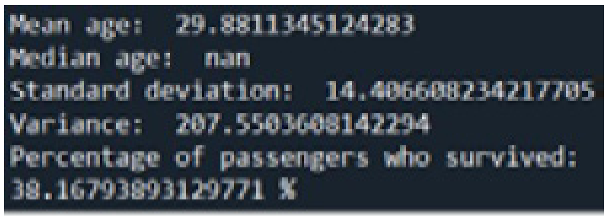
In this example, first load the Titanic dataset into a Pandas DataFrame, and then use Pandas methods to perform various statistical calculations on the "Age" and "Survived" columns. The mean age, median age, standard deviation, and variance of the ages of the passengers are calculated, as well as the percentage of passengers who survived.

Note that this is just one example of the kind of statistical analysis you can perform using Pandas. Pandas provides many convenient functions for exploring and analyzing data, such as grouping data by one or more variables, calculating aggregates, and handling missing values.

Below is the reference on descriptive statistical analysis using numpy on titanic survival dataset:

```
import numpy as np
import pandas as pd
#Load the Titanic dataset
df = pd.read_csv("titanic.csv")
#Calculate the mean age of the passengers
mean_age = np.mean(df['age'])
print("Mean age:", mean_age)
#Calculate the median of the age of the passengers
median_age= np.median(df['age'])
print("Median age:", median_age)
#Calculate the standard deviation of the age of the passengers
std_dev = np.std(df['age'])
print("Standard deviation:", std_dev)
#Calculate the variance of the age of the passengers
var = np.var(df['age'])
print("Variance:", var)
#Calculate the percentage of passengers who survived
num_survived = np.sum(df['survived']==1)
total = df.shape[0]
percentage_survived = (num_survived / total)*100
print("Percentage of passengers who survived:", percentage_survived)
```

Output:



```
Mean age:  29.8811345124283
Median age:  nan
Standard deviation:  14.406608234217705
Variance:  207.5503608142294
Percentage of passengers who survived:
38.16793893129771 %
```

Figure 8.2: *Descriptive Statistical Analysis using Numpy*

In this example, first the Titanic dataset is loaded into a Pandas DataFrame, and then use NumPy to perform various statistical calculations on the "Age" and "Survived" columns. The mean age, median age, standard deviation, and variance of the ages of the passengers are calculated, as well as the percentage of passengers who survived.

Note here the median is ‘**nan**’. This means the **np.median()** function includes ‘**NaN**’ values while calculating the median of the data. So, if your data contains nan values, the median calculated using **np.median()** will always be ‘**NaN**’. While in pandas, the median function ignores the nan values in the data while calculating median.

Note that this is just one example of the kind of statistical analysis you can perform using NumPy. You can also use NumPy to perform more complex calculations, such as calculating the correlation between variables or performing hypothesis testing.

Data cleaning

Data cleaning is an important step in the machine learning process as it can greatly impact the quality of the results produced by a machine learning model. Here are some common steps involved in the data cleaning process:

- **Handling missing values**: This involves identifying and dealing with missing or incomplete values in the dataset. Common methods for handling missing values include imputing missing values with the mean, median, or mode of the column, or simply dropping rows with missing values.
- **Handling outliers**: Outliers are data points that are significantly different from other values in the same column. Identifying and dealing with outliers is important as they can have a negative impact on the results produced by a machine learning model. Common methods for handling outliers include removing them, transforming them, or imputing them with the mean or median of the column.
- **Handling inconsistent values**: This involves identifying and correcting values that are inconsistent or do not conform to a specified format. For example, converting all values in a column to a

consistent format, such as converting all dates to a standard date format.

- By following these steps, you can clean and pre-process your data to ensure that it is ready for use in a machine learning model. Note that the specific steps involved in the data cleaning process will depend on the dataset and the machine learning problem you are working on, so it's important to be mindful of the trade-offs involved in each step and choose the appropriate methods for your specific use case.

Handling missing values

Handling missing values in a dataset is a common task in data analysis, and NumPy provides several ways to do this. Here is an example of how you can handle missing values in the Titanic survival dataset using NumPy:

```
import numpy as np
import pandas as pd

#Load the Titanic dataset
df = pd.read_csv("titanic.csv")

#Replacing missing values in 'age' with the mean of 'age' column
mean_age = np.mean(df['age'])
df['age'].fillna(mean_age, inplace =True)

#Alternatively, you can replace missing values with the median of the age
median_age= np.median(df['age'])
df['age'].fillna(median_age, inplace =True)
#Another option is to drop all rows containing missing values
df.dropna(inplace=True)
```

In this example, the Titanic dataset is loaded into a Pandas DataFrame. Then, using the fillna() method, missing values in the "Age" column are replaced with the mean age. Alternatively, you can replace missing values with the median age. Finally, you can drop all rows that have missing values using the **dropna()** method.

It's important to note that the approach you choose for handling missing values will depend on the specific dataset and analysis you're working on. There are trade-offs to each approach, so it's important to consider the implications of each method carefully before deciding which one to use.

Handling outliers

Handling outliers in the Titanic survival dataset using pandas involves identifying and dealing with data points that are significantly different from other values in the same column. Here's an example of how you could handle outliers in the dataset:

- **Identify outliers:** To identify outliers, you can use visualization techniques such as box plots, scatter plots, or histograms. Another option is to use statistical methods such as the Z-score method or the **Interquartile Range** (**IQR**) method.
- **Decide on a strategy:** After identifying the outliers, you need to decide on a strategy for dealing with them. Common methods include removing the outliers, transforming the outliers, or imputing the outliers with the mean or median of the column.
- **Implement the strategy:** Here's an example of how you could remove outliers using pandas:

There are several statistical measures that can be used to identify outliers in a dataset:

1. **Z-score**: The Z-score is a measure of how many standard deviations a value is away from the mean. Values with a Z-score greater than 3 or less than -3 are commonly considered outliers, as they are more than three standard deviations away from the mean. The Z-score (also known as the standard score) is a measure of how many standard deviations a value is away from the mean of a dataset. The formula for the Z-score is defined as:

$$Z = (x - \mu) / \sigma$$

Where x is the value, μ is the mean, and σ is the standard deviation.

The Z-score is a standardization of the original value, with respect to the mean and standard deviation of the dataset. By using the Z-score, you can compare values across datasets with different scales, and determine if a value is an outlier. Values with a Z-score greater than 3 or less than -3 are commonly considered outliers, as they are more than three standard deviations away from the mean.

2. **Interquartile range (IQR)**: The IQR is the range between the first and third quartiles of a dataset. Outliers can be identified as values that are outside the range defined by Q1 - 1.5*IQR and Q3 + 1.5*IQR, where Q1 and Q3 are the first and third quartiles, respectively.
3. **Boxplot**: A boxplot is a graphical representation of a dataset that can be used to identify outliers. In a boxplot, outliers are typically represented as individual points outside the main box., You have seen in previous chapters how can you plot boxplots and how can you identify the presence of outlier
4. **Mahalanobis Distance**: The Mahalanobis distance is a measure of the distance between a point and a mean vector in multivariate data. It takes into account the correlations between variables and can be used to identify outliers in multivariate datasets.

$$\text{Mahalanobis distance} = \sqrt{(x - \mu)^T \Sigma^{-1} (x - \mu)}$$

Where x is a vector representing an observation, μ is the mean vector, Σ is the covariance matrix, and Σ^{-1} is the inverse covariance matrix.

The Mahalanobis distance measures the distance between a vector and a mean vector, taking into account the covariance between the variables. The larger the Mahalanobis distance, the farther the vector is from the mean in terms of the covariance structure of the data.

5. **Cook's Distance**: Cook's distance is a measure of the influence of a data point on the regression model. High values of Cook's distance

indicate outliers, as they have a large impact on the regression model. Cook's distance is a measure of the influence of a single observation on the fitted regression model. It is defined as:

Cook's distance = (residual^2 / leverage) / (number of parameters + 1)

Where residual^2 is the residual sum of squares for the observation, leverage is a measure of how far the observation is from the mean of all the independent variables, and number of parameters + 1 is the number of parameters estimated in the regression model plus one.

Cook's distance provides information on the influence of each observation on the regression model. Observations with a high Cook's distance are considered to have a large influence on the model, and may need to be re-evaluated or removed from the dataset. Typically, observations with a Cook's distance greater than 4/(number of observations - number of parameters - 1) are considered to have a great influence.

These measures can be used individually or in combination to identify outliers in a dataset. By identifying outliers, you can take appropriate action, such as removing them from the dataset, imputing missing values, or transforming the data.

```
import numpy as np

import pandas as pd

import scipy.stats as stats


#Load the Titanic dataset

df = pd.read_csv("titanic.csv")


#Calculate the Z-score for each value in 'age' column

z = np.abs(stats.zscore(df['age']))


#Remove rows with z-score greater than 3
```

df = df[z < 3]

In this example, the Z-score is calculated for each value in the 'Age' column. Rows with a Z-score greater than 3 are removed, as these values are considered outliers.

By handling outliers, you can ensure that your machine learning model is trained on high-quality data, which can improve the accuracy of the results produced by the model.

In the example below a dataset on height and weight of the persons, the outlier detection and removal using quantile method is demonstrated and later with the help of boxplot the attribute is displayed with and without outlier:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
 df = pd.read_csv("w-h.csv")
df.describe()
#Visualizing the data attribute height
plt.hist(df.Height)
#Visualizing using boxplot for outlier detection
plt.boxplot(df.Height)
#Setting up the limits to remove the outlier using quantile
max_height = df.Height.quantile(0.95)
min_height = df.Height.quantile(0.05)
#Displaying rows in dataset beyond the set limits
print(df.Height[df.Height>max_height].count())
print(df.Height[df.Height<min_height].count())
#removing the outliers in the dataset
df_Height_without_outlier = df[(df.Height<max_height) & (df.Height>min_height)]
#Displaiing the datset attribute with and without outlier using boxplot
```

```
plt.subplot(1,2,1)
plt.boxplot(df.Height)
plt.title("With Outliers")
plt.subplot(1,2,2)
plt.boxplot(df_Height_without_outlier.Height)
plt.title("Without Outliers")
plt.show()
```

Output:

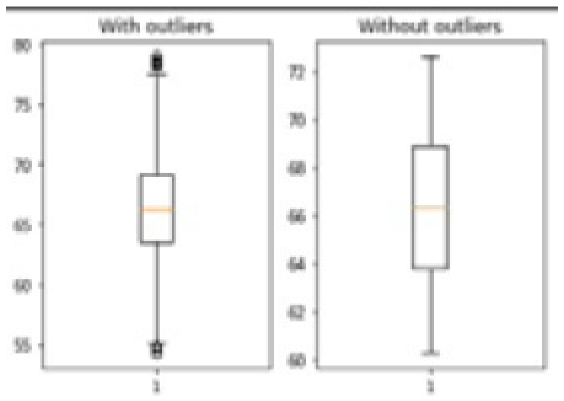


Figure 8.3: *Outlier Detection with Box Plot*

Missing inconsistent values

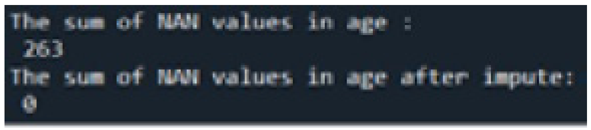
In the Titanic dataset, the age column has some missing values. One way to handle missing values is to impute them with the mean or median value of the column. However, you can also see from the dataset that some ages are recorded in fractions, such as 0.83, which represents a child under one year old. Therefore, you may want to impute the missing ages with the median value of the corresponding age group, rather than the overall median.

Here's an example of how to handle inconsistent values in the age column of the Titanic dataset using this approach:

```
import pandas as pd
#Load the Titanic dataset
titanic = pd.read_csv("titanic.csv")
```

```
#Verify with nan values in age
print("The sum of NAN values in age : \n", titanic.age.isna().sum())
#Define a function to impute missing ages with
#the median age of the corresponding age group
def impute_age(df):
    for i in range(1,4):
        median_age = df[df['pclass'] ==i]['age'].median()
        df.loc[(df['pclass']==i) & (df['age'].isnull()),'age'] = median_age
    return df
#Apply the function to the titanic dataset to  impute missing ages
titanic = impute_age(titanic)
#Verify with nan values in age
print("The sum of NAN values in age : \n", titanic.age.isna().sum())
```

Output:



```
The sum of NAN values in age :
 263
The sum of NAN values in age after impute:
 0
```

Figure 8.4: *Output representing the removal of NAN with impute method*

In this example, a function called **impute_age** is defined that takes a dataframe as input, and imputes missing ages with the median age of the corresponding age group. Here the pclass column is used to define the age groups, since we expect the age distribution to differ across passenger classes. Then the function is applied to the Titanic dataset to impute missing ages. The isnull() method is used to identify missing ages, and the loc method is used to assign the median age to the corresponding rows.

After running this code, the age column in the Titanic dataset should no longer have missing values, and the inconsistent age values (that is, fractions) should be replaced with more appropriate values, check in the output, where using isna() function along with sum() the count of nan values is printed.

Data visualization

Data visualization is a crucial component of EDA, which is the process of examining and analyzing data to summarize its main characteristics and gain insights. Data visualization involves using graphical representations of the data to help identify patterns, trends, and outliers that may be difficult to see in raw data.

Here are some common types of data visualizations used in EDA:

- **Histograms**: Histograms are a type of bar chart that shows the distribution of a continuous variable by dividing the data into bins and counting the number of observations in each bin. Histograms are useful for identifying the shape of the distribution, such as whether it is skewed or normal.
- **Scatter plots**: Scatter plots are used to visualize the relationship between two continuous variables. Each observation is represented as a point on the graph, with one variable on the x-axis and the other on the y-axis. Scatter plots can help identify patterns, such as whether there is a positive or negative correlation between the variables.
- **Box plots**: Box plots are used to summarize the distribution of a continuous variable and identify outliers. The box in the middle of the plot represents the IQR, which contains the middle 50% of the data. The line inside the box represents the median, and the whiskers represent the range of the data, excluding outliers.
- **Heatmaps**: Heatmaps are used to visualize the relationship between two categorical variables by representing the frequency or proportion of each combination of categories as a color-coded grid. Heatmaps can help identify patterns and relationships between variables that may be difficult to see in a table of numbers.
- **Line charts**: Line charts are used to visualize changes in a continuous variable over time. Each observation is represented as a point on the graph, with the x-axis representing time and the y-axis representing the value of the variable. Line charts can help identify trends, seasonality, and outliers over time.

- **Bar charts**: Bar charts are used to visualize the distribution of a categorical variable by representing the frequency or proportion of each category as a bar. Bar charts can help identify the most common categories and the relative frequencies of each category.

These are just a few examples of the many types of data visualizations that can be used in EDA. The choice of visualization depends on the nature of the data and the questions being asked. Good data visualizations should be clear, easy to understand, and visually appealing, and should help convey insights and patterns in the data.

We have covered all these plots and visualization techniques in the previous chapter with libraries ‘Matplotlib’ ‘Seaborn’, ‘Bokeh’. In this section of this chapter, you will explore more on which plot to use in which technique of EDA, instead of how to plot various visualizations. Although not all the kinds of plots have been discussed in previous chapters, for your immediate knowledge, you will have all which is required to know about visualizations to be used at univariate and multivariate analysis of EDA.

Visualizations for Univariate EDA

Univariate EDA refers to the analysis of a single variable in a dataset. The primary goal of univariate EDA is to understand the distribution, central tendency, and variability of the variable. Here are some common plots used in univariate EDA:

- **Histograms**: Histograms are used to visualize the distribution of a continuous variable. The variable is divided into bins, and the frequency or density of observations in each bin is represented by a bar. Histograms can help identify the shape of the distribution, such as whether it is skewed or normal.
- **Box plots**: Box plots are used to summarize the distribution of a continuous variable and identify outliers. The box in the middle of the plot represents the IQR, which contains the middle 50% of the data. The line inside the box represents the median, and the whiskers represent the range of the data, excluding outliers.

- **Density plots**: Density plots are similar to histograms, but instead of using bins to divide the variable, a smooth curve is used to represent the distribution. Density plots can help identify the shape of the distribution, such as whether it is skewed or normal.
- **Bar plots**: Bar plots are used to visualize the distribution of a categorical variable. The frequency or proportion of each category is represented by a bar. Bar plots can help identify the most common categories and the relative frequencies of each category.
- **Pie charts**: Pie charts are used to visualize the distribution of a categorical variable. The frequency or proportion of each category is represented by a wedge of a circle. Pie charts can help identify the most common categories and the relative frequencies of each category, but are less commonly used in data analysis due to their limited ability to show detailed information.
- **Violin plots**: Violin plots are similar to box plots, but also show the density of the variable at different values. The width of the violin represents the density of the data at that value. Violin plots can help identify the shape of the distribution, the variability of the data, and the presence of multiple modes.

These are just a few examples of the many plots that can be used in univariate EDA. The choice of plot depends on the nature of the variable and the questions being asked. Good plots should be clear, easy to understand, and visually appealing, and should help convey insights and patterns in the data.

Visualizations for Bivariate EDA

Bivariate analysis involves examining the relationship between two variables in a dataset. The primary goal of bivariate analysis is to identify patterns and relationships between the two variables. Here are some common visualizations used in bivariate analysis:

- **Scatter plots**: Scatter plots are used to visualize the relationship between two continuous variables. Each observation is represented as a point on the graph, with one variable on the x-axis and the other on the y-axis. Scatter plots can help identify patterns, such as whether there is a positive or negative correlation between the variables.

- **Line plots**: Line plots are used to visualize the relationship between two continuous variables over time or another continuous dimension. The x-axis represents the continuous dimension, and the y-axis represents the value of the variable. Line plots can help identify trends and patterns over time or another continuous dimension.
- **Bar plots**: Bar plots are used to visualize the relationship between a continuous variable and a categorical variable. The categorical variable is represented on the x-axis, and the frequency or mean value of the continuous variable is represented on the y-axis. Bar plots can help identify the differences in the values of the continuous variable between categories.
- **Box plots**: Box plots are used to summarize the relationship between a continuous variable and a categorical variable. The categorical variable is represented on the x-axis, and the distribution of the continuous variable is represented using the IQR and outliers. Box plots can help identify the differences in the distribution of the continuous variable between categories.
- **Heatmaps**: Heatmaps are used to visualize the relationship between two categorical variables by representing the frequency or proportion of each combination of categories as a color-coded grid. Heatmaps can help identify patterns and relationships between variables that may be difficult to see in a table of numbers.

These are just a few examples of the many visualizations that can be used in bivariate analysis. The choice of visualization depends on the nature of the data and the questions being asked. Good visualizations should be clear, easy to understand, and visually appealing, and should help convey insights and patterns in the data.

Visualizations for Multivariate EDA

Multivariate analysis involves examining the relationships between multiple variables in a dataset. The primary goal of multivariate analysis is to identify patterns and relationships between variables. Here are some common visualizations used in multivariate analysis:

- **Scatter plots**: Scatter plots are used to visualize the relationship between two continuous variables. Each observation is represented as a point on the graph, with one variable on the x-axis and the other on the y-axis. Scatter plots can help identify patterns, such as whether there is a positive or negative correlation between the variables.
- **Heatmaps**: Heatmaps are used to visualize the relationship between two categorical variables by representing the frequency or proportion of each combination of categories as a color-coded grid. Heatmaps can help identify patterns and relationships between variables that may be difficult to see in a table of numbers.
- **Pair plots**: Pair plots show the relationships between multiple pairs of variables in a dataset. Each plot in the matrix shows the relationship between two variables, with the diagonal plots showing the distribution of each variable. Pair plots can help identify patterns and relationships between variables, and can be used to identify potential interactions between variables.
- **Bubble charts**: Bubble charts are used to visualize the relationship between three continuous variables. Each observation is represented as a bubble on the graph, with the x-axis representing one variable, the y-axis representing another variable, and the size of the bubble representing a third variable. Bubble charts can help identify patterns and relationships between variables in three dimensions.
- **3D plots**: 3D plots are used to visualize the relationship between three continuous variables. The x, y, and z axes represent the values of the three variables, and the plot shows the relationship between them in three dimensions. 3D plots can help identify patterns and relationships between variables in three dimensions.

These are just a few examples of the many visualizations that can be used in multivariate analysis. The choice of visualization depends on the nature of the data and the questions being asked. Good visualizations should be clear, easy to understand, and visually appealing, and should help convey insights and patterns in the data.

Feature engineering

Feature engineering is the process of creating new features or transforming existing features in a dataset to improve the performance of a machine learning model. Feature engineering is a critical step in the machine learning pipeline, as the quality of the features can greatly affect the performance of the model.

Here are some examples of feature engineering techniques:

- **Handling categorical values**: Converting categorical data into numerical format is an important step in EDA, as many machine learning algorithms require numerical input data. For example, if You have a categorical variable called "color" with three possible values (red, green, and blue), You can create three binary features called "**color_red**", "**color_green**", and "**color_blue**", where each feature indicates whether the color is red, green, or blue. There are several techniques for converting categorical data into numerical format, including:
    - **Label Encoding**: Assigning each unique category in a categorical variable with an integer value, as I explained in my previous answer.
    - **One-Hot Encoding**: Creating a new binary column for each unique category in a categorical variable, where each column indicates the presence or absence of the category in a given row, as I explained in my previous answer.
    - **Count Encoding**: Replacing each category with the count of its occurrences in the dataset. For example, if a categorical variable has the category "apple" occurring 10 times in the dataset, the count encoding for "apple" would be 10.
    - **Target Encoding**: Replacing each category with the mean (or median) of the target variable for that category. This technique can

    be useful for predictive modeling, but it requires having a target variable that is already in numerical format.

    - **Binary Encoding**: This encoding method is a hybrid between one-hot encoding and label encoding. It works by first assigning each unique category a number, and then converting each number to a binary representation. The binary representation is used to create new columns for each category, resulting in fewer columns than one-hot encoding.

The choice of which encoding method to use will depend on the nature of the data and the requirements of the machine learning algorithm being used. EDA can help in deciding which method is appropriate for the data at hand.

- **Feature Scaling**: Feature scaling is the process of scaling numerical features to a specific range to ensure that each feature contributes equally to the model. Common feature scaling techniques include standardization, which scales the features to have zero mean and unit variance, and normalization, which scales the features to a range between 0 and 1.
- **Feature Extraction**: Feature extraction involves creating new features from existing features in a dataset. This can be done using techniques such as **Principal Component Analysis** (**PCA**), which extracts the most important features from a dataset, or using domain-specific knowledge to create new features that are relevant to the problem being solved.
- **Feature selection**: Feature selection is the process of selecting a subset of the most relevant and informative features for use in a machine learning model. This can be done using methods such as feature importance, which ranks features based on their contribution to the model's performance, or feature correlation, which removes features that are highly correlated with each other.
- **Time-Based Features**: Time-based features are features that capture the time component of a dataset. This can be done by creating features such as day of the week, month, year, or by creating features that capture trends or seasonality in the data.

- **Text-Based Features**: Text-based features are features that are extracted from text data. This can be done using techniques such as bag-of-words or TF-IDF, which extract features based on the frequency of words in a document, or by using more advanced techniques such as word embeddings, which represent words as vectors in a high-dimensional space.
- **Interaction Features**: Interaction features are features that are created by combining two or more features in a dataset. For example, if You have two features called "age" and "gender", You could create an interaction feature called "**age_gender**", which captures the combined effect of age and gender on the outcome variable.

These are just a few examples of the many techniques that can be used for feature engineering. The choice of feature engineering technique depends on the nature of the data and the problem being solved, and often requires some experimentation to determine which techniques work best for a particular problem.

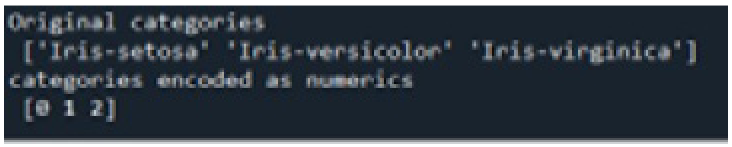
Handling categorical values

Here's an example code for each of the encoding methods you asked for using Python:

Label Encoding:

```
import pandas as pd

df= pd.read_excel('Iris1.xls')

from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

df['Species_encoded'] = le.fit_transform(df['Species'])

print("Original categories \n", df.Species.unique())

print("Categories encoded as numerics \n", df.Species_encoded.unique())
```

Output:



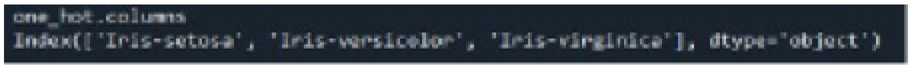
```
Original categories
 ['Iris-setosa' 'Iris-versicolor' 'Iris-virginica']
categories encoded as numerics
 [0 1 2]
```

Figure 8.5 *Output representing categories encoded with LabelEncoder*

One-Hot Encoding:

```
import pandas as pd
df= pd.read_excel('Iris1.xls')
one_hot = pd.get_dummies(df['Species'])
df= pd.concat([df, one_hot], axis = 1)
print("One-hot encoded columns \n", one_hot.columns)
```

Output:



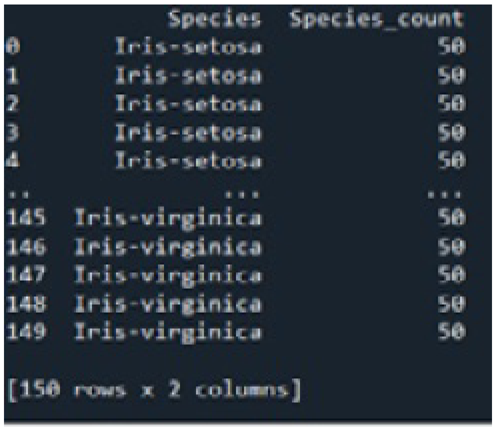
```
one_hot.columns
Index(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], dtype='object')
```

Figure 8.6: *Output representing categories encoded with get_dummies method for One-Hot Encoding*

Count encoding:

```
import pandas as pd
df= pd.read_excel('Iris1.xls')
count_map = df['Species'].value_counts().to_dict()
df['Species_count'] = df['Species'].map(count_map)
df[['Species', 'Species_count']]
```

Output:



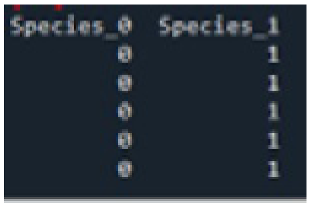
```
           Species  Species_count
0      Iris-setosa             50
1      Iris-setosa             50
2      Iris-setosa             50
3      Iris-setosa             50
4      Iris-setosa             50
..             ...            ...
145 Iris-virginica             50
146 Iris-virginica             50
147 Iris-virginica             50
148 Iris-virginica             50
149 Iris-virginica             50

[150 rows x 2 columns]
```

Figure 8.7: *Output representing categorical data encoded with count encoding*

Binary encoding:

```
import pandas as pd
df= pd.read_excel('Iris1.xls')
import category_encoders as ce
encoder = ce.BinaryEncoder(cols = ['Species'])
df_encoded  = encoder.fit_transform(df.Species)
df_encoded.head()
```

Output:



```
Species_0  Species_1
        0          1
        0          1
        0          1
        0          1
        0          1
```

Figure 8.8: *Output representing categorical data encoded with Binary encoding*

Note that in each case, df is the DataFrame containing the categorical variable 'Species', and the new encoded variables are added to the same DataFrame. Also, make sure to import the necessary libraries before running the code.

Feature scaling

Feature scaling is a technique used in machine learning to standardize the range of values of input variables, also known as features, used in a model. The goal of feature scaling is to make the input variables more comparable and easier for the machine learning algorithm to process, as well as to improve the performance of the model.

The most common techniques for feature scaling include:

- **Standardization**: In this technique, the input features are rescaled so that they have a mean of zero and a standard deviation of one. This is done by subtracting the mean from each value and then dividing by the standard deviation.
- **Min-Max scaling**: In this technique, the input features are rescaled to have values between 0 and 1. This is done by subtracting the minimum value and then dividing by the range (that is, the difference between the maximum and minimum values).
- **Normalization**: In this technique, the input features are rescaled to have a range between -1 and 1. This is done by subtracting the minimum value and then dividing by the range, and then multiplying by 2 and subtracting 1.

The choice of which technique to use depends on the characteristics of the data and the requirements of the machine learning algorithm being used. It is important to note that feature scaling does not always improve the performance of a model, and sometimes it can even make the results worse. Therefore, it is important to experiment with different scaling techniques and evaluate the performance of the model on a validation set to determine whether scaling is necessary or not.

Standardization example on 'titanic_survival' dataset

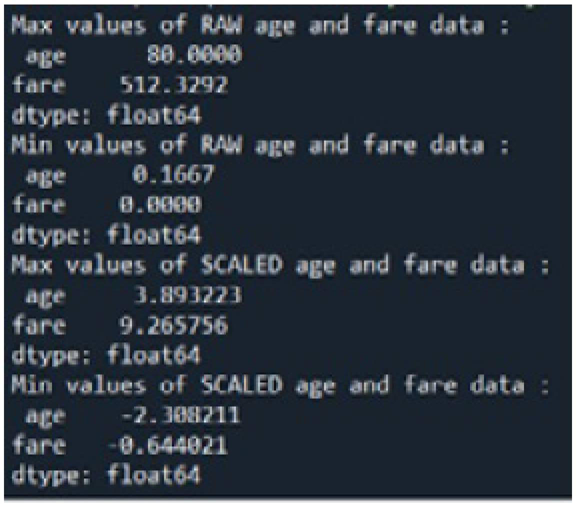
In the sklearn (Scikit learn) library in python, there are inbuilt scalers, You can import them using from - import commands. In the code example below, note that **StanderScaler** is imported from **sklearn.preprocessing** module. Make sure you use capital 'S' while importing this scaler.

```
from sklearn.preprocessing import StandardScaler
```

```
import pandas as pd
titanic = pd.read_csv('titanic.csv')
num_cols = titanic[['age','fare']]
num_cols.age = num_cols.age.fillna(num_cols.age.mean())
num_cols.fare = num_cols.fare.fillna(num_cols.fare.mean())

scaler = StandardScaler()
num_cols_scaled = pd.DataFrame(scaler.fit_transform(num_cols), columns=
['age','fare'])

print("Max values of RAW age and fare data : \n",num_cols.max())
print("Min values of RAW age and fare data : \n",num_cols.min())
print("Max values of SCALED age and fare data : \n",num_cols_scaled.max())
print("Min values of SCALED age and fare data : \n",num_cols_scaled.min())
```

After loading the data, Next, You can select the numerical columns that You want to standardize. For this example, let's select the age and fare columns. This will standardize the age and fare columns in the Titanic dataset, so that they have a mean of zero and a standard deviation of one. The **fit_transform()** method of the **StandardScaler** class fits the scaler to the data and then transforms the selected columns in place.

Also note the use of **fillna()** function to fill the nan values with the mean of corresponding attributes.



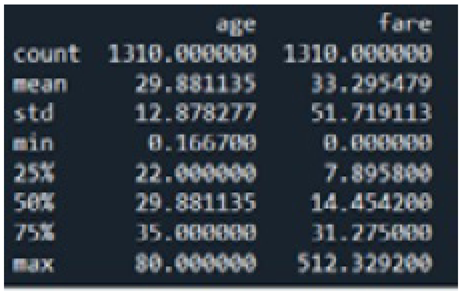
```
Max values of RAW age and fare data :
 age      80.0000
fare     512.3292
dtype: float64
Min values of RAW age and fare data :
 age      0.1667
fare     0.0000
dtype: float64
Max values of SCALED age and fare data :
 age      3.893223
fare     9.265756
dtype: float64
Min values of SCALED age and fare data :
 age     -2.308211
fare    -0.644021
dtype: float64
```

Figure 8.9: *Output representing the effect of Standard Scaling*

After standardization, the age and fare columns will have new values that are standardized. You can check the new mean and standard deviation of these columns using the **describe()** method also.

#num_cols statistic description

num_cols.describe()

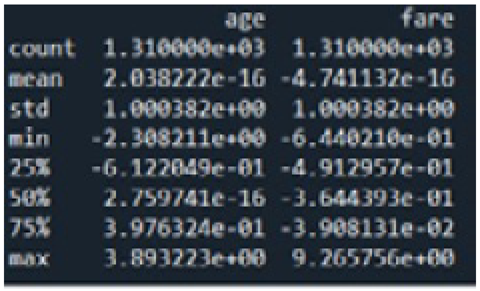


```
               age          fare
count  1310.000000   1310.000000
mean     29.881135     33.295479
std      12.878277     51.719113
min       0.166700      0.000000
25%      22.000000      7.895800
50%      29.881135     14.454200
75%      35.000000     31.275000
max      80.000000    512.329200
```

Figure 8.10: *Output representing num_cols Statistic Description*

#num_cols_scaled statistic description

num_cols_scaled.describe()



```
             age          fare
count  1.310000e+03  1.310000e+03
mean   2.038222e-16 -4.741132e-16
std    1.000382e+00  1.000382e+00
min   -2.308211e+00 -6.440210e-01
25%   -6.122049e-01 -4.912957e-01
50%    2.759741e-16 -3.644393e-01
75%    3.976324e-01 -3.908131e-02
max    3.893223e+00  9.265756e+00
```

Figure 8.11: *Output representing num_cols_scaled Statistic Description*

You should see that the mean of each column is approximately zero, and the standard deviation is approximately one.

Min-Max Scaling example on 'titanic_survival' dataset

As standardscaler, you can also import **MinMaxScaler** from the **sklearn.preprocessing**.

from sklearn.preprocessing import MinMaxScaler

scaler_mm = MinMaxScaler()

num_cols_scaled = pd.DataFrame(scaler_mm.fit_transform(num_cols), columns=['age','fare'])

#num_cols statistic description

print("Describe on raw data")

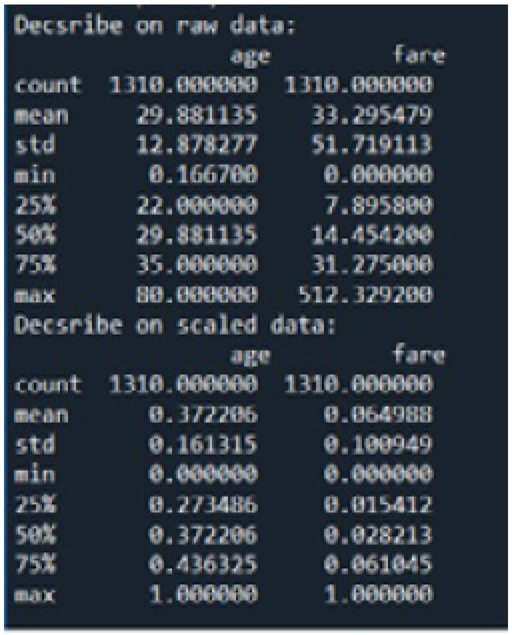
print(num_cols.describe())

#num_cols_scaled statistic description

print("Describe on scaled data")

print(num_cols_scaled.describe())

Output:



```
Decsribe on raw data:
               age         fare
count  1310.000000  1310.000000
mean     29.881135    33.295479
std      12.878277    51.719113
min       0.166700     0.000000
25%      22.000000     7.895800
50%      29.881135    14.454200
75%      35.000000    31.275000
max      80.000000   512.329200
Decsribe on scaled data:
               age         fare
count  1310.000000  1310.000000
mean      0.372206     0.064988
std       0.161315     0.100949
min       0.000000     0.000000
25%       0.273486     0.015412
50%       0.372206     0.028213
75%       0.436325     0.061045
max       1.000000     1.000000
```

Figure 8.12: *Output representing the effect of Min Max Scaler*

You should see that the minimum value of each column is approximately zero, and the maximum value is one.

Normalization example on 'titanic_survival' dataset

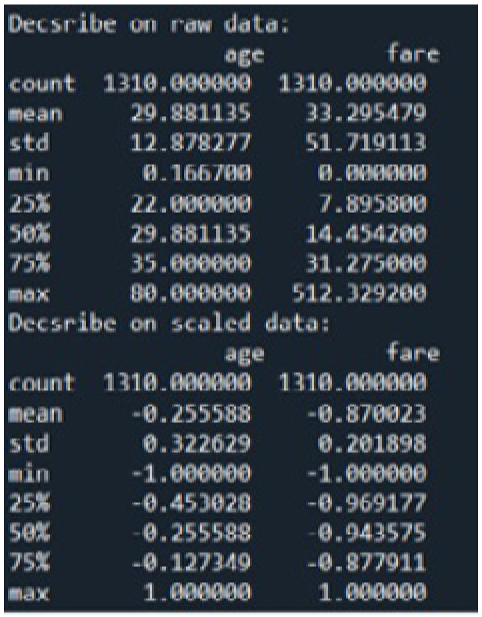
You can use the **MinMaxScaler** class from the **sklearn** library to normalize the selected columns. Here's how to do it: Make note of using **feature_range=(-1, 1)** parameter within the **MinMaxScaler** function.

```
from sklearn.preprocessing import MinMaxScaler

scaler_mm = MinMaxScaler(feature_range=(-1,1))

num_cols_scaled = pd.DataFrame(scaler_mm.fit_transform(num_cols), columns=
['age','fare'])

#num_cols statistic description

print("Describe on raw data")

print(num_cols.describe())

#num_cols_scaled statistic description

print("Describe on scaled data")

print(num_cols_scaled.describe())
```

This will normalize the age and fare columns in the Titanic dataset, so that they have values between -1 and 1. The **fit_transform()** method of the **MinMaxScaler** class fits the scaler to the data and then transforms the selected columns in place.

After normalization, the age and fare columns will have new values that are normalized. You can check the new minimum and maximum values of these columns using the **describe()** method.

Output:



```
Decsribe on raw data:
                age          fare
count  1310.000000  1310.000000
mean     29.881135    33.295479
std      12.878277    51.719113
min       0.166700     0.000000
25%      22.000000     7.895800
50%      29.881135    14.454200
75%      35.000000    31.275000
max      80.000000   512.329200
Decsribe on scaled data:
                age          fare
count  1310.000000  1310.000000
mean     -0.255588    -0.870023
std       0.322629     0.201898
min      -1.000000    -1.000000
25%      -0.453028    -0.969177
50%      -0.255588    -0.943575
75%      -0.127349    -0.877911
max       1.000000     1.000000
```

Figure 8.13: *Output representing the effect of Normalization*

This will output the summary statistics of the age and fare columns after normalization. You should see that the minimum value of each column is approximately -1, and the maximum value is approximately 1.

Correlation analysis

Correlation analysis is an important technique in EDA that is used to explore the relationships between variables in a dataset. Correlation analysis is used to determine the strength and direction of the relationship between two variables.

Correlation analysis involves calculating a correlation coefficient, which is a numerical value that indicates the strength and direction of the relationship between two variables. There are different types of correlation coefficients, such as Pearson's correlation coefficient, Spearman's correlation coefficient, and Kendall's correlation coefficient. The choice of correlation coefficient depends on the type of data being analyzed and the research question being addressed.

The correlation coefficient ranges from -1 to +1. A correlation coefficient of -1 indicates a perfect negative correlation, which means that as one variable increases, the other variable decreases. A correlation coefficient of +1 indicates a perfect positive correlation, which means that as one variable increases, the other variable also increases. A correlation coefficient of 0 indicates no correlation between the two variables.

Correlation analysis can be visualized using a scatter plot, where each point represents a pair of values for the two variables being analyzed. The scatter plot can provide a visual representation of the relationship between the variables.

Correlation analysis is an important tool in EDA because it can help identify variables that are strongly related and may be useful in predicting or explaining outcomes of interest. However, it is important to note that correlation does not imply causation. Therefore, further analysis is required to establish the causal relationship between variables. Additionally, correlation analysis assumes a linear relationship between variables and may not be appropriate for non-linear relationships.

The formula for calculating the correlation coefficient is different depending on the type of correlation being used. Here are the formulas for three common types of correlation coefficients:

- **Pearson's correlation coefficient (r):** Pearson's correlation coefficient ranges from -1 to 1, where a value of -1 indicates a perfectly negative linear relationship, a value of 0 indicates no linear relationship, and a value of 1 indicates a perfectly positive linear relationship. A correlation coefficient of 0 means there is no linear association between the two variables, but there may be other types of relationships between them.

The formula for Pearson's correlation coefficient is:

$$r = (n\Sigma xy - \Sigma x\Sigma y) / sqrt[(n\Sigma x^2 - (\Sigma x)^2)(n\Sigma y^2 - (\Sigma y)^2)]$$

where:

- *r* is the correlation coefficient
- *n* is the number of data points
- Σxy is the sum of the product of the x and y values
- Σx and Σy are the sums of the x and y values, respectively
- Σx^٢ and Σy^٢ are the sums of the squares of the x and y values, respectively

- **Spearman's correlation coefficient** (ρ): Spearman's correlation coefficient ranges from -1 to 1, where a value of -1 indicates a perfectly negative monotonic relationship, a value of 0 indicates no monotonic relationship, and a value of 1 indicates a perfectly positive monotonic relationship. A correlation coefficient of 0 means there is no monotonic association between the two variables.

  The formula for Spearman's correlation coefficient is:

  ρ = ٦) - ١ Σd^٢) / (*n(n^2 - 1))*

  where:

  - ρ is the correlation coefficient
  - Σd^٢ is the sum of the squared differences between the ranks of the x and y values
  - *n* is the number of data points

- **Kendall's correlation coefficient** (τ): Kendall's τ ranges from -1 to 1, where a value of -1 indicates a perfectly negative monotonic relationship, a value of 0 indicates no monotonic relationship, and a value of 1 indicates a perfectly positive monotonic relationship.

The formula for Kendall's τ is:

$$\tau = (n_c - n_d) / (1/2\, n\, (n-1))$$

where:

- τ is the correlation coefficient
- *n* is the number of data points
- *n_c* is the number of concordant pairs of observations (that is, pairs where the ranks of both variables agree)
- *n_d* is the number of discordant pairs of observations (that is, pairs where the ranks of the variables disagree)

Here's an example of calculating Pearson's correlation coefficient for two variables using Python:

```
import numpy as np
from scipy.stats import pearsonr
#Generate some sample data
x = np.array([1,2,3,4,5])
y = np.array([2,4,6,8,10])

#Calculate Pearson's cprrelation coefficient
corr, _ = pearsonr(x, y)
#Print the correlation coefficient
print('Correlation coefficient: ', corr)

#correlation with numpy and pandas
corr = np.corrcoef(x,y)
print('Correlation coefficient: ', corr)
```

Output:

In this example, some sample data for two variables (x and y) using numpy is generated. Then the **pearsonr()** function from **scipy.stats** is used to calculate Pearson's correlation coefficient between the two variables. The function returns two values: the correlation coefficient and the p-value. Then the correlation coefficient is assigned to the variable **corr** and ignores the p-value using the underscore (_). Finally, **print()** is used to display the correlation coefficient to the console.

However you can find the correlation in numpy and pandas as well, as below:

```
corr = np.corrcoef(x,y)
print('Correlation coefficient: ', corr)
 import pandas as pd
df=pd.DataFrame({'x':x,'y':y})
corr=df.corr(method = 'pearson')
print('Correlation coefficient: ', corr)
```

Note that in this example, the correlation coefficient is 1, which indicates a perfect positive correlation between the two variables. This is because the values in y are exactly twice the values in x. In practice, it is more common to see correlation coefficients that are not as extreme, and that may be positive or negative depending on the nature of the relationship between the variables.

EDA using dataset in Python

Exploratory data analysis (**EDA**) is an important step in any data analysis project. EDA allows you to gain a better understanding of your data, identify patterns, detect anomalies, and inform the choice of appropriate data preprocessing and modeling techniques. In this response, we will provide a brief overview of the steps involved in performing EDA using Python and some commonly used libraries.

1. **Importing libraries and dataset:**

   The first step in EDA is to import the necessary libraries and the dataset into your Python environment. The most commonly used libraries for EDA in Python are pandas, numpy, matplotlib, and seaborn.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
#Load the dataset
df = pd.read_csv('path/to/dataset.csv')
```

2. **Data inspection**

   The next step is to inspect the dataset to get an idea of its structure, features, and overall quality. Some important aspects to check are:

```
#check the shape of dataset
print("Data shape :\n", df.shape)
# view the first few rows of the dataset
print("Data's first few rows :\n", df.head())
#check the data types of each feature
print("Data type of the attributes :\n", df.dtypes)
#check for missing values
print("Checking for missing values :\n", df.isnull().sum())
```

3. **Data cleaning**

   Based on the inspection, you may need to perform some data cleaning tasks such as:

```
#drop duplicates
df.drop_duplicates(inplace=True)
#drop irrelevant columns
df.drop(['col1', 'col2'], axis=1, inplace=True)
#handle missing values
df.fillna(df.mean(), inplace=True)
```

4. **Data visualization**

   The next step is to use visualizations to gain insights into the dataset. Some commonly used plots are:

```
#histogram of a feature
sns.histplot(df['col1'])
#scatter plot of two features
sns.scatterplot(x='col1', y='col2', data=df)
#heatmap of correlation matrix
sns.heatmap(df.corr())
```

5. **Feature engineering:**

   Based on the insights gained from EDA, you may need to engineer new features or transform existing ones to improve model performance. Some common techniques are:

```
#create a new feature based on existing ones
df['new_feature']=df['col1']+ df['col2']
# normaliza a feature
df['mormalized_col1']=(df['col1'] - df['col1'].mean())/df['col1'].std()
# encode categorical variables
df = pd.get_dummies(df, columns=['cat_col'])
```

   You can use the scalers from the sklearn library as well, as per the requirement based on the dataset.

6. **Summary statistics:**

   Finally, You can use summary statistics to quantify the distribution of each feature, detect outliers, and identify any other anomalies.

```
#Summary statistics
print(df.describe())
#boxplot of a feature
sns.boxplot(df['col'])
```

These are some of the key steps involved in performing EDA in Python. However, the exact steps may vary depending on the dataset and the analysis goals.

Example 1

Here's an example of how to perform EDA on the **'boston_housing_price'** dataset in Python using the popular libraries pandas, numpy, matplotlib, and seaborn:

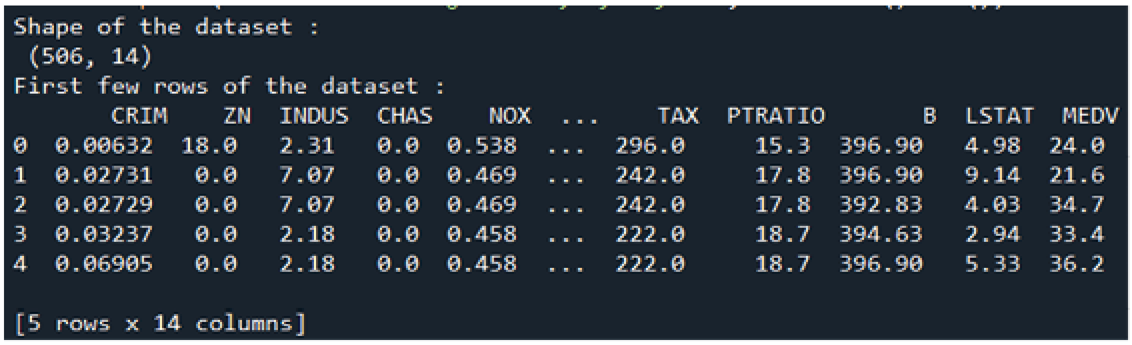
1. Import the necessary libraries and load the dataset:

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.datasets import load_boston

#load the datset

boston = load_boston()

df = pd.DataFrame(boston.data, columns=boston.feature_names)

df['MEDV'] = boston.target
```

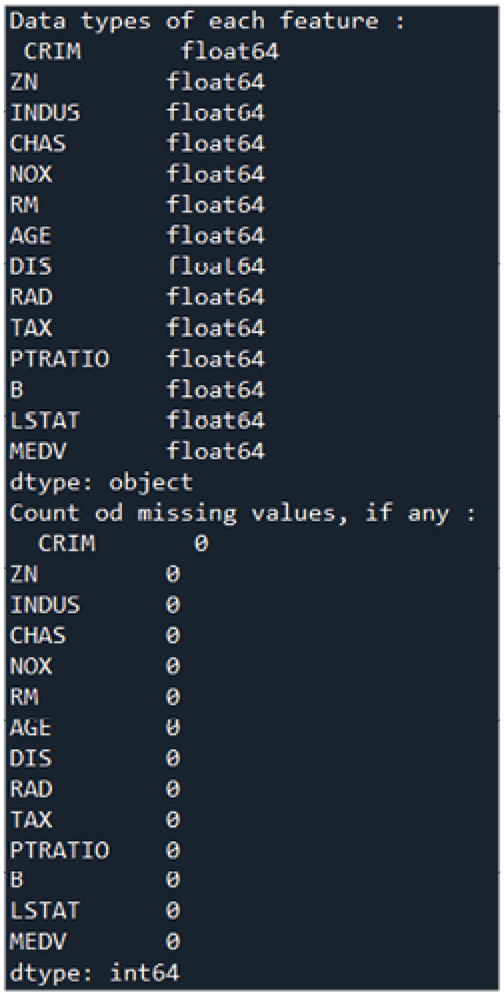
2. Inspect the dataset to get an idea of its structure and quality:

```
#check the shape of dataset

print("Data shape :\n", df.shape)

# view the first few rows of the dataset

print("Data's first few rows :\n", df.head())

#check the data types of each feature

print("Data type of the attributes :\n", df.dtypes)

#check for missing values

print("Checking for missing values :\n", df.isnull().sum())
```

Output:



```
Shape of the dataset :
 (506, 14)
First few rows of the dataset :
      CRIM    ZN  INDUS  CHAS    NOX  ...    TAX  PTRATIO       B  LSTAT  MEDV
0  0.00632  18.0   2.31   0.0  0.538  ...  296.0     15.3  396.90   4.98  24.0
1  0.02731   0.0   7.07   0.0  0.469  ...  242.0     17.8  396.90   9.14  21.6
2  0.02729   0.0   7.07   0.0  0.469  ...  242.0     17.8  392.83   4.03  34.7
3  0.03237   0.0   2.18   0.0  0.458  ...  222.0     18.7  394.63   2.94  33.4
4  0.06905   0.0   2.18   0.0  0.458  ...  222.0     18.7  396.90   5.33  36.2

[5 rows x 14 columns]
```



```
Data types of each feature :
 CRIM       float64
ZN         float64
INDUS      float64
CHAS       float64
NOX        float64
RM         float64
AGE        float64
DIS        float64
RAD        float64
TAX        float64
PTRATIO    float64
B          float64
LSTAT      float64
MEDV       float64
dtype: object
Count od missing values, if any :
  CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
MEDV       0
dtype: int64
```

Figure 8.14: *Output representing the structure and quality of the dataset as per the code snippet*

3. Use visualizations to gain insights into the dataset:

#histogram of the target variable MEDV

sns.histplot(df['MEDV'])

Output:

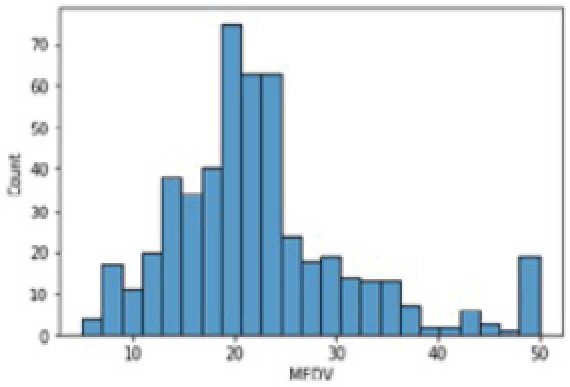


Figure 8.15: *Histogram on MEDV target variable*

#scatter plots of various features vs. MEDV

sns.scatterplot(x='RM', y='MEDV', data =df)

sns.scatterplot(x='LSTAT', y='MEDV', data =df)

sns.scatterplot(x='DIS', y='MEDV', data =df)

Output:

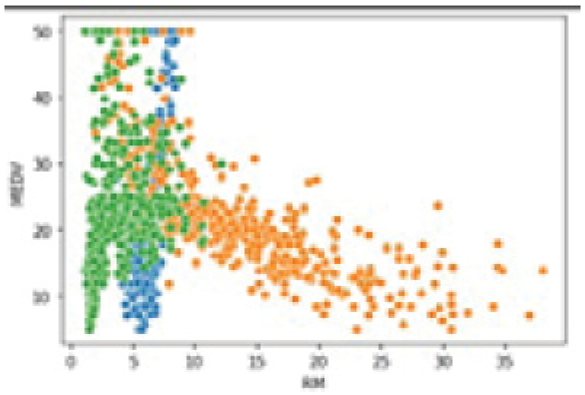


Figure 8.16: *Scatter Plot on RM vs. MDEV*

#heatmap of correlation matrix

sns.heatmap(df.corr())

Output:

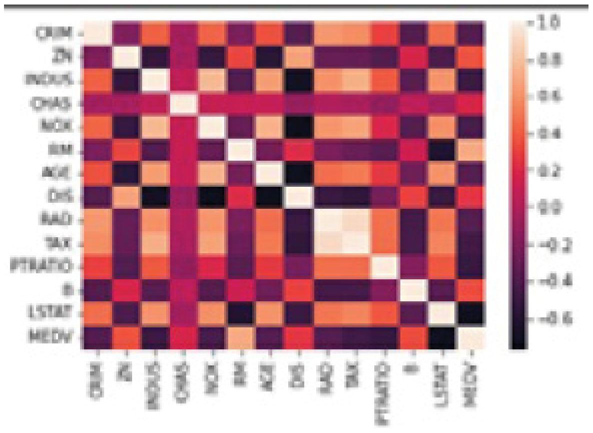


Figure 8.17: *Heatmap representing the correlation on the dataset*

4. Engineer new features or transform existing ones:

   #create a new feature based on existing ones

   df['TAXRM'] = df['TAX'] / df['RM']

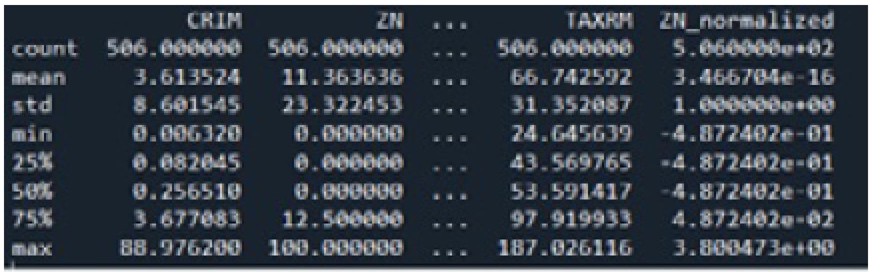
   #normalize a feature

   df['ZN_normalized'] = (df['ZN'] - df['ZN'].mean()) / df['ZN'].std()

5. Use summary statistics to quantify the distribution of each feature:

   print(df.describe())

Output:



```
             CRIM          ZN  ...       TAXRM  ZN_normalized
count  506.000000  506.000000  ...  506.000000   5.060000e+02
mean     3.613524   11.363636  ...   66.742592   3.466704e-16
std      8.601545   23.322453  ...   31.352087   1.000000e+00
min      0.006320    0.000000  ...   24.645639  -4.872402e-01
25%      0.082045    0.000000  ...   43.569765  -4.872402e-01
50%      0.256510    0.000000  ...   53.591417  -4.872402e-01
75%      3.677083   12.500000  ...   97.919933   4.872402e-02
max     88.976200  100.000000  ...  187.026116   3.800473e+00
```

Figure 8.18: *Statistic Summary of the data using describe method*

#boxplot of the target variable MEDV

sns.boxplot(df['MEDV'])

Output:

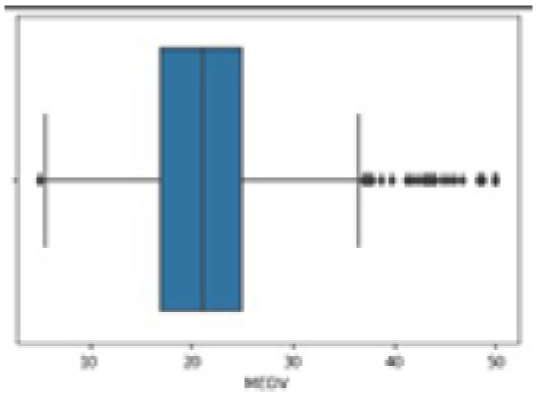


Figure 8.19: *Box Plot on MDEV target variable*

These are some basic steps you can take for EDA on the '**boston_housing_price**' dataset. However, there are many more techniques and visualizations you can use depending on your specific analysis goals.

Conclusion

In this chapter, we embarked on a journey of Exploratory Data Analysis (EDA), a fundamental step in the data analysis process. We learned invaluable techniques and tools that enable us to dive deep into datasets, unraveling insights and uncovering patterns that might otherwise remain hidden. EDA equips us with a comprehensive understanding of the data, laying a strong foundation for subsequent analysis and decision-making.

We began our exploration by harnessing the power of visualizations. Visualizations serve as our eyes, enabling us to grasp the overall characteristics of the data at a glance. Through scatter plots, bar charts, histograms, and box plots, we could visualize the relationships between variables, detect anomalies, and identify the distributional properties of the data.

Moving beyond visualizations, we delved into statistical measures and summary statistics. By calculating measures such as mean, median, variance, and standard deviation, we gained insights into the central tendencies, spread, and variability of our data. We also explored correlation matrices and covariance, allowing us to quantify the relationships between variables and identify potential dependencies.

An essential aspect of EDA involves handling missing data and outliers. We learned how to detect and deal with missing values, employing techniques such as imputation or removal based on context and domain knowledge. Furthermore, we discussed the significance of feature engineering and data transformation. By creating new variables, combining existing ones, or applying mathematical functions, we could extract more meaningful insights from our data. Techniques like normalization, and standardization aided in ensuring the robustness and interpretability of our analysis.

With a thorough understanding of EDA techniques, you have acquired the skills necessary to extract valuable information from our datasets. Armed with visualizations and statistical measures, you can confidently navigate the intricacies of the data, uncover hidden trends, and make data-driven decisions.

Points to remember

- **Exploratory Data Analysis** (**EDA**) is a vital step in the data analysis process.
- EDA involves understanding and analyzing data to gain insights and identify patterns.
- Visualizations play a crucial role in EDA, helping us grasp data characteristics quickly.
- Statistical measures provide insights into central tendencies, spread, and variability.
- Handling missing data and outliers is important in EDA.
- Feature engineering and data transformation enhance the interpretability of analysis.
- Data preparation is a process of cleaning and transforming the data to make it suitable for analysis.
- In univariate analysis, you analyze variables individually to understand their distributions, missing values, and outliers.

- In bivariate analysis, you explore the relationship between two variables using statistical tests and visualization techniques.
- In multivariate analysis, you analyze the relationship between multiple variables using techniques like factor analysis, clustering, and regression analysis.
- In standardization, you rescale input features to have a mean of zero and a standard deviation of one.
- In Min-Max scaling, you rescale input features to have values between 0 and 1.
- In Normalization, you rescale input features to have a range between -1 and 1.
- Normalization transforms data to a standard range for better interpretation.
- Standardization and scaling techniques improve model performance.
- Standardization is commonly used in algorithms like regression and clustering.
- Min-Max scaling is useful when features have bounded or limited ranges.
- Normalization is effective when features have varying scales and distributions.
- **Label encoding**: Assigning a unique numeric label to each category or class in a categorical variable.
- **One-Hot encoding**: Represent categorical variables as binary vectors, where each category is encoded as a separate binary feature.
- **Binary encoding**: Representing categorical variables as binary strings, where each category is assigned a unique binary code.
- **Count encoding**: Replacing categories with their respective frequency or count in the dataset.
- Label encoding helps convert categorical variables into a numeric format, but it may introduce an arbitrary order. It is suitable for

ordinal variables where the order of categories matters.

- One-Hot encoding creates additional binary features, each representing a category, enabling machine learning algorithms to handle categorical variables.
- Binary encoding is a space-efficient encoding technique that reduces the number of features compared to one-hot encoding; it strikes a balance between dimensionality reduction and capturing categorical information.
- Count encoding captures the frequency of categories, providing useful information about their prevalence in the dataset it is effective for high-cardinality categorical variables where one-hot encoding may lead to a large number of features.

Questions

1. What is the purpose of Exploratory Data Analysis (EDA) in the data analysis process?
2. What are some common visualization techniques used in EDA?
3. How can summary statistics help in understanding the characteristics of data during EDA?
4. What techniques can be applied to handle missing data and outliers in EDA?
5. How does bivariate analysis differ from univariate analysis in EDA?
6. What are some statistical tests commonly used in EDA to analyze relationships between variables?
7. How can data transformation and feature engineering contribute to the effectiveness of EDA?
8. What role does data visualization play in EDA and why is it important?
9. What are the steps involved in the data preparation phase of EDA?
10. How can EDA help in identifying patterns and trends in the data?

11. How does multivariate analysis differ from bivariate analysis in EDA?
12. What are some techniques used in EDA to explore and analyze categorical variables?

Index