# Premium Vector | White abstract background in 3d paper style | Abstract backgrounds, Abstract, Geometric backgroundA screenshot of a device Description automatically generated



Hand Book

**Data Science for Applied Sciences**

Disclaimer: The content is curated from online/offline resources and used for educational purpose only

**Table of Contents**

[Chapter 1: Data Science for Applied Sciences 4](#_Toc181000362)

[1.1 Introduction 4](#_Toc181000363)

[1.2 What is Data? 4](#_Toc181000364)

[1.3 Data Science for Applied Science 6](#_Toc181000365)

[Chapter 2: Data Visualization using MS Excel 4](#_Toc181000362)

[2.1 Data Visualization using MS Excel 11](#_Toc181000366)

[2.2 GPT for Excel 18](#_Toc181000367)

[Chapter 3: Introduction to Python 22](#_Toc181000368)

[3.1 Introduction 22](#_Toc181000369)

[3.2 Installation and Setting Up Path 26](#_Toc181000370)

[3.3 Python Variable 35](#_Toc181000371)

[3.4 Data Types 37](#_Toc181000372)

[3.5 Control Structures 65](#_Toc181000373)

[3.6 Functions in Python 70](#_Toc181000374)

[Chapter 4: Data Manipulation and Analysis - Working with Data 22](#_Toc181000368)

[4.1 Data Manipulation and Analysis: Working with Data 98](#_Toc181000376)

[4.2 Pandas Library 99](#_Toc181000377)

[4.3 NumPy Library 109](#_Toc181000378)

[4.4 Data Preprocessing 114](#_Toc181000379)

[References: 121](#_Toc181000380)

**Chapter 1: Data Science for Applied Sciences**

|  |
| --- |
| Learning Outcomes:  By the end of this chapter, students will be able to:   * Create line, bar, pie, and scatter charts in Excel to effectively visualize data trends, comparisons, proportions, and relationships. * Apply customization techniques such as adding titles, labels, and legends, and adjusting styles to enhance chart clarity and visual impact. * Utilize Excel's data visualization tools to support data analysis and communicate insights effectively. * Define Python and identify its key features and advantages. * Recall the steps required for Python installation and setting up a development environment. * Describe the purpose of variables and different data types (integers, floats, strings, booleans). * Describe functions, including lambda functions, and their use in Python. * Create and use custom functions to organize code. * Create a basic class with attributes and methods. * Identify attributes as variables belonging to a class or object. * Create and call methods within a class to perform actions on object data. * Describe inheritance as a way to create a new class from an existing class. * Explain encapsulation as bundling data and methods within a class and restricting access. * Describe polymorphism and its purpose in using a common interface for different forms. * Identify the purpose of Pandas for data manipulation with data frames and series. * Recognize NumPy as a library for handling arrays and numerical data. * Explain common techniques for handling missing values and preparing data for analysis. * Describe the function of Matplotlib in creating visualizations like plots and charts. * Explain Seaborn’s role in creating advanced visualizations and statistical plots. |

# 1.1 Introduction

Disclaimer: The content is curated from online/offline resources and used for educational purpose only

Data and Microsoft Excel play a pivotal role in the field of artificial intelligence (AI), serving as foundational tools for data analysis, manipulation, and visualization. Excel, a widely used spreadsheet application, provides an accessible platform for individuals and organizations to organize and analyse large datasets. Its robust functions and features allow users to perform complex calculations, create models, and visualize trends, making it an ideal starting point for understanding data patterns.

# 1.2 What is Data?

Since the invention of computers, people have used the term data to refer to computer information, and this information was either transmitted or stored. But that is not the only data definition; there exist other types of data as well.

A close-up of a data

Description automatically generated

Source: https://herovired.com/learning-hub/blogs/what-is-data/

So, what is the data? Data can be texts or numbers written on papers, or it can be bytes and bits inside the memory of electronic devices, or it could be facts that are stored inside a person’s mind.

Data, at its core, is any set of values or details that provide information about something. When we say "data," we're talking about information that can be observed, measured, recorded, and used to understand different aspects of the world. It's not limited to a single form; it can be as straightforward as numbers in a spreadsheet or as complex as the pixels in an image, the sound waves of music, or the words of a book. Data's versatility and potential to represent almost anything are what make it so powerful and universally applicable.

## 1.2.1 Types of Data

**Qualitative (Categorical) Data**

* This type of data describes qualities or characteristics and is non-numerical. Qualitative data is often divided into:
  + **Nominal Data**: Categories without a specific order. Examples include gender (male, female), color (red, green, blue), and types of cuisine (Italian, Chinese, Indian).
  + **Ordinal Data**: Categories with a specific order but without a standard distance between values. Examples include levels of satisfaction (satisfied, neutral, dissatisfied), and education levels (high school, undergraduate, graduate).
* **Uses**: Categorical data is often used for classification tasks and is commonly visualized using bar charts or pie charts.

**Quantitative (Numerical) Data**

* Quantitative data represents measurable quantities and is always numerical. It’s further divided into:
  + **Discrete Data**: Countable data, often whole numbers. Examples include the number of students in a class, the number of cars in a parking lot, or the number of products sold.
  + **Continuous Data**: Data that can take on any value within a range, often measured. Examples include height, weight, temperature, and time.
* **Uses**: Numerical data is crucial for regression, statistical analysis, and various machine learning algorithms. It’s often visualized using histograms, line charts, and scatter plots.

**Structured Data**

* Structured data is highly organized, typically stored in tabular formats like spreadsheets or databases, with well-defined rows and columns. Examples include:
  + Databases containing customer details (names, ages, purchase history).
  + Spreadsheets tracking inventory or sales records.
* **Uses**: Structured data is easy to analyze and visualize due to its organized nature. SQL databases, for instance, are widely used to store and query structured data.

**Unstructured Data**

* Unstructured data lacks a predefined format, making it more challenging to analyze directly. Examples include:
  + Text data (emails, social media posts, product reviews).
  + Images and videos (photos, videos from security cameras).
  + Audio data (recorded conversations, voice commands).
* **Uses**: Unstructured data is processed using Natural Language Processing (NLP) for text, computer vision techniques for images and video, and audio processing techniques for sound.

**Time-Series Data**

* Time-series data is a type of structured data where each data point is associated with a specific timestamp, representing changes over time. Examples include:
  + Stock prices over days, months, or years.
  + Temperature records taken every hour or day.
  + Website traffic data captured daily or weekly.
* **Uses**: Time-series analysis helps in forecasting and identifying trends over time. Special methods like moving averages, seasonal decomposition, and ARIMA (Auto-Regressive Integrated Moving Average) models are often applied.

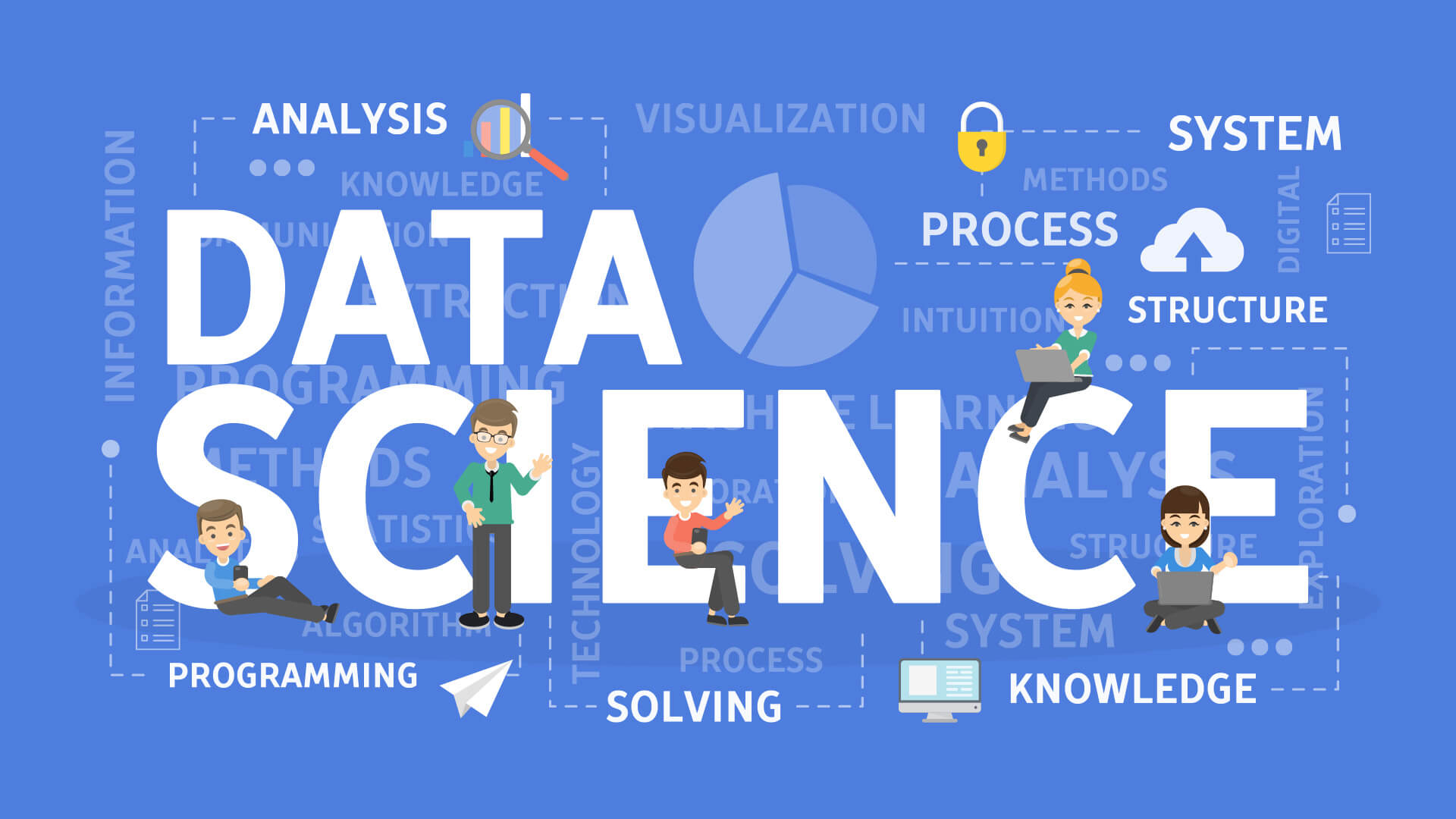
**Spatial Data**

* Spatial data, or geospatial data, includes information about physical locations and shapes. Examples include:
  + GPS coordinates of locations.
  + Maps, satellite imagery, and geographic information systems (GIS).
  + Weather data tied to specific geographic regions.
* **Uses**: Spatial data is useful in mapping, navigation, and location-based services. In data science, spatial data requires specific handling, often using GIS software or packages like Geopandas in Python.

# 1.3 Data Science for Applied Science

**What is Data Science?**

* Data science is an interdisciplinary field that combines techniques from statistics, computer science, mathematics, and domain-specific knowledge to extract insights and knowledge from both structured and unstructured data. It encompasses a range of processes and methodologies, including data collection, cleaning, analysis, visualization, and interpretation. The ultimate goal of data science is to derive meaningful insights from data that can inform decisions, drive innovations, and enhance understanding in various contexts.



Source: https://www.decube.io/post/data-science-concept

## 1.3.1 Key components of data science include:

* + **Data Collection**: Gathering data from multiple sources, including databases, sensors, online platforms, and other repositories.
  + **Data Cleaning and Preparation**: Processing raw data to remove inconsistencies, handle missing values, and format it for analysis.
  + **Data Analysis**: Using statistical methods and algorithms to explore data, identify trends, and uncover relationships.
  + **Machine Learning**: Applying algorithms that enable computers to learn from data and make predictions or decisions based on that data.
  + **Data Visualization**: Creating graphical representations of data to communicate findings effectively to stakeholders.

#### Relevance of Data Science in Applied Sciences

* Data science plays a pivotal role in applied sciences by enhancing decision-making and problem-solving across various domains.
* Supports innovation and research across various domains (e.g., healthcare, environmental science, engineering).

## 1.3.2 Applications of Data Science

* Data science is making significant strides across various fields, providing tools and techniques that drive innovation and improve outcomes. Below are key applications of data science in healthcare, environmental science, engineering, and agriculture.

**1. Healthcare**

* **Predictive Analytics for Patient Outcomes and Treatment Efficacy**:
  + Data science algorithms analyze historical patient data, including demographics, medical history, and treatment responses, to predict future health outcomes. This can help healthcare providers identify high-risk patients and tailor interventions accordingly.
  + For example, machine learning models can forecast the likelihood of readmission for patients with chronic conditions, allowing for more proactive management.
* **Analyzing Patient Data for Personalized Medicine**:
  + By leveraging genomic data, lifestyle factors, and previous treatment outcomes, data science enables personalized treatment plans. This approach can enhance the effectiveness of therapies and minimize side effects.
  + Techniques such as clustering and classification can identify patient subgroups that respond similarly to specific treatments, facilitating targeted therapies.

**2. Environmental Science**

* **Modeling Climate Change Impacts Using Data-Driven Simulations**:
  + Data science models simulate various climate scenarios by analyzing historical weather data, greenhouse gas emissions, and environmental changes. These simulations help predict potential future impacts on ecosystems and human societies.
  + For instance, researchers can forecast how rising temperatures and changing precipitation patterns will affect agriculture, water resources, and biodiversity.
* **Monitoring Biodiversity and Ecosystems Through Remote Sensing**:
  + Remote sensing technologies collect data from satellites and drones, which data scientists analyze to monitor habitat changes, species distributions, and ecosystem health.
  + Techniques such as image recognition and classification can be applied to satellite imagery to assess deforestation rates or identify critical habitats.

**3. Engineering**

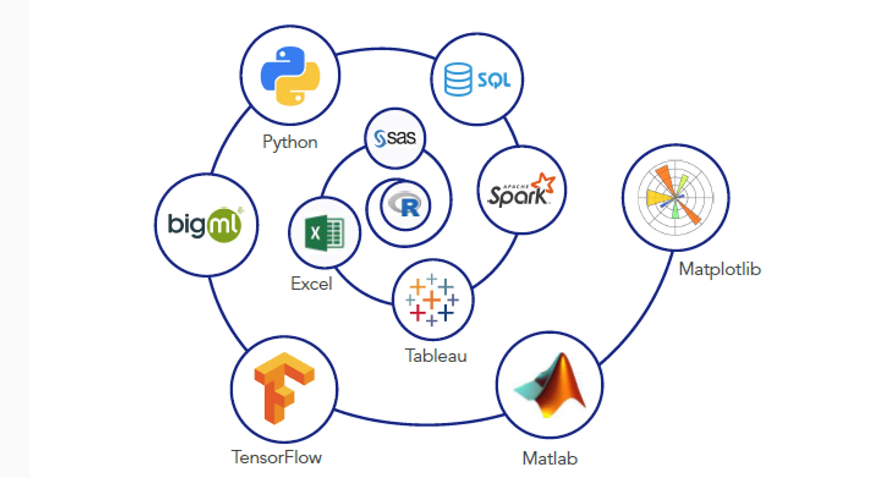
* **Optimizing Manufacturing Processes with Data Analysis**:
  + Data science helps identify inefficiencies in manufacturing processes by analyzing production data, workflow patterns, and machine performance. This can lead to improved efficiency, reduced waste, and lower costs.
  + For example, advanced analytics can optimize production schedules and inventory management, resulting in better resource allocation.
* **Predictive Maintenance of Machinery Using IoT Data**:
  + Data from Internet of Things (IoT) devices attached to machinery can be analyzed to predict when equipment is likely to fail or require maintenance. This proactive approach reduces downtime and maintenance costs.
  + Techniques such as time series analysis and anomaly detection are commonly used to monitor equipment health and performance trends.

**4. Agriculture**

* **Precision Farming Techniques for Crop Yield Optimization**:
  + Data science applications in precision agriculture involve analyzing data from sensors, drones, and satellites to optimize crop management practices. This can lead to increased yields and reduced resource consumption.
  + By assessing soil health, moisture levels, and crop growth stages, farmers can make informed decisions about irrigation, fertilization, and pest management.
* **Analyzing Soil and Weather Data for Informed Decision-Making**:
  + Data scientists analyze historical and real-time soil and weather data to provide farmers with actionable insights. This includes predicting planting times, irrigation needs, and potential pest infestations.
  + Machine learning models can analyze weather patterns to optimize planting schedules, maximizing productivity while minimizing risks associated with adverse weather conditions.

## 1.3.3 Tools and Techniques for Data Science

* Data science involves a diverse set of tools and techniques that enable practitioners to extract insights, analyze data, and develop predictive models. Here’s an overview of common tools and essential techniques used in the field.



Source: https://medium.com/javarevisited/10-essential-tools-data-scientists-should-learn-in-2022-acbae6558643

**Common Tools**

* **Programming Languages**
  + **Python**:
    - Widely used for data manipulation, analysis, and machine learning due to its simplicity and extensive libraries (e.g., Pandas, NumPy, Matplotlib, Seaborn).
    - Popular for tasks such as data cleaning, exploratory data analysis, and building machine learning models.
  + **R**:
    - A language specifically designed for statistical analysis and data visualization, favored among statisticians and data analysts.
    - Offers powerful packages like ggplot2 for visualization, dplyr for data manipulation, and caret for machine learning.
* **Data Visualization Tools**
  + **Tableau**:
    - A leading data visualization tool that allows users to create interactive and shareable dashboards. It connects to various data sources and provides a user-friendly interface for visual analysis.
  + **Power BI**:
    - Developed by Microsoft, Power BI enables users to visualize data and share insights across organizations. It integrates well with other Microsoft products and offers a robust set of features for business analytics.
* **Machine Learning Libraries**
  + **TensorFlow**:
    - An open-source library developed by Google for building and deploying machine learning models, particularly deep learning applications. It supports a wide range of neural network architectures and is highly scalable.
  + **scikit-learn**:
    - A versatile library for machine learning in Python that provides simple and efficient tools for data mining and data analysis. It includes various algorithms for classification, regression, clustering, and dimensionality reduction.

**Techniques**

* **Statistical Analysis**
  + **Descriptive Statistics**: Summarizes data characteristics through measures such as mean, median, mode, variance, and standard deviation.
  + **Inferential Statistics**: Involves making predictions or inferences about a population based on a sample. Techniques include hypothesis testing, confidence intervals, and regression analysis.
  + **Trend Analysis**: Identifies patterns over time in datasets, often using time series analysis to forecast future values based on historical data.
* **Machine Learning**
  + **Supervised Learning**: Involves training a model on labeled data to make predictions or classifications. Common algorithms include linear regression, decision trees, and support vector machines.
  + **Unsupervised Learning**: Involves finding hidden patterns or intrinsic structures in unlabeled data. Techniques include clustering (e.g., k-means, hierarchical clustering) and dimensionality reduction (e.g., PCA).
  + **Reinforcement Learning**: A type of machine learning where agents learn to make decisions by taking actions in an environment to maximize cumulative rewards.

# Chapter 2: Data Visualization Using MS Excel

Learning Outcomes:

* Create line, bar, pie, and scatter charts in Excel to effectively visualize data trends, comparisons, proportions, and relationships.
* Apply customization techniques such as adding titles, labels, and legends, and adjusting styles to enhance chart clarity and visual impact.
* Utilize Excel's data visualization tools to support data analysis and communicate insights effectively.
* Define Python and identify its key features and advantages.
* Recall the steps required for Python installation and setting up a development environment.
* Describe the purpose of variables and different data types (integers, floats, strings, booleans).
* Describe functions, including lambda functions, and their use in Python.
* Create and use custom functions to organize code.
* Create a basic class with attributes and methods.
* Identify attributes as variables belonging to a class or object.
* Create and call methods within a class to perform actions on object data.
* Describe inheritance as a way to create a new class from an existing class.
* Explain encapsulation as bundling data and methods within a class and restricting access.
* Describe polymorphism and its purpose in using a common interface for different forms.
* Identify the purpose of Pandas for data manipulation with data frames and series.
* Recognize NumPy as a library for handling arrays and numerical data.
* Explain common techniques for handling missing values and preparing data for analysis.
* Describe the function of Matplotlib in creating visualizations like plots and charts.
* Explain Seaborn’s role in creating advanced visualizations and statistical plots.

# 2.1 Data Visualization using MS Excel

**What is Data Visualization?**

* Data visualization is the graphical representation of information and data. By utilizing visual elements like charts, graphs, and maps, data visualization tools provide an accessible way to see and understand trends, patterns, and outliers in data. The key benefits of data visualization include:
  + **Simplification of Complex Data**: Visualizations help break down large datasets into digestible pieces, allowing users to grasp essential insights quickly.
  + **Identification of Trends and Patterns**: Graphical representations make it easier to spot trends over time or highlight relationships between variables.
  + **Highlighting Outliers**: Visualizations can help identify anomalies or outliers in data that may require further investigation.

A screenshot of a computer

Description automatically generated

Source: https://zebrabi.com/best-practices-for-using-excel-for-data-visualization/

## 1.1.1 Importance of Data Visualization in Excel

Excel is one of the most widely used tools for data visualization, primarily due to its accessibility and powerful features. Here are some reasons why data visualization in Excel is crucial:

* **Accessibility and Familiarity**:
  + Excel is user-friendly and widely used across industries, making it a go-to tool for many professionals. Most users are already familiar with its basic functions, allowing for a smoother learning curve when it comes to creating visualizations.
* **Diverse Visualization Options**:
  + Excel provides a variety of chart types, including bar charts, line graphs, pie charts, scatter plots, and more. Users can choose the best type of visualization based on their data and the message they want to convey.
* **Data Analysis Made Intuitive**:
  + By transforming data into visual formats, Excel enables users to analyze data more intuitively. Visuals can highlight relationships between data points, making it easier to draw conclusions without getting lost in raw numbers.
* **Enhances Presentations and Reports**:
  + Well-designed visualizations can significantly improve the quality of presentations and reports. Visuals can communicate information more effectively than text alone, capturing the audience's attention and making complex data more understandable.
* **Interactive Features**:
  + Excel offers interactive features such as slicers and pivot charts that allow users to manipulate data visualizations in real time. This interactivity helps users explore data from different angles and gain deeper insights.
* **Integration with Data Analysis Tools**:
  + Excel integrates seamlessly with various data analysis tools and functions. Users can leverage Excel’s built-in formulas and functions to analyze data and then visualize the results directly, streamlining the workflow.
* **Customization and Flexibility**:
  + Users can customize charts and graphs extensively in Excel, from colors and labels to layout and styles, ensuring that the visualizations meet their specific needs and preferences.

## 1.1.2 Types of Charts in Excel

* Excel provides a wide range of chart types, each designed to represent data visually in a way that highlights specific relationships or insights. Here’s a detailed look at some common chart types and tips for choosing the right one for your data.

A collage of graphs and charts

Description automatically generated

**Common Chart Types**

* **Column Chart**
  + This chart type uses vertical bars to represent data. Each bar's height corresponds to its value, making it easy to compare different categories.
  + Ideal for comparing values across different categories, such as sales figures for various products or survey results across demographics.
  + **Example**: Comparing monthly sales across different regions.
* **Line Chart**
  + A line chart connects individual data points with a continuous line, emphasizing trends over time.
  + Best for displaying data trends across periods, such as stock prices over months or temperature changes throughout a year.
  + **Example**: Visualizing the trend of website traffic over the last 12 months.
* **Pie Chart**
  + This circular chart is divided into slices that represent the proportion of each category to the whole. The size of each slice is proportional to its percentage of the total.
  + Effective for showing percentage shares of a whole, such as market share among competitors or budget allocation.
  + **Example**: Displaying the composition of a company’s revenue sources.
* **Bar Chart**
  + Similar to column charts, but uses horizontal bars instead of vertical ones. This format can make it easier to read category labels, especially with long names.
  + Useful for comparing values across categories, especially when category names are lengthy.
  + **Example**: Comparing the number of users across different social media platforms.
* **Scatter Plot**
  + A scatter plot uses dots to represent values for two different variables. The position of each dot indicates the relationship between the variables.
  + Best for showing correlations or relationships, such as the relationship between advertising spend and sales revenue.
  + **Example**: Analyzing the correlation between hours studied and exam scores.

**Choosing the Right Chart**

Selecting the appropriate chart type is crucial for effectively communicating your data.

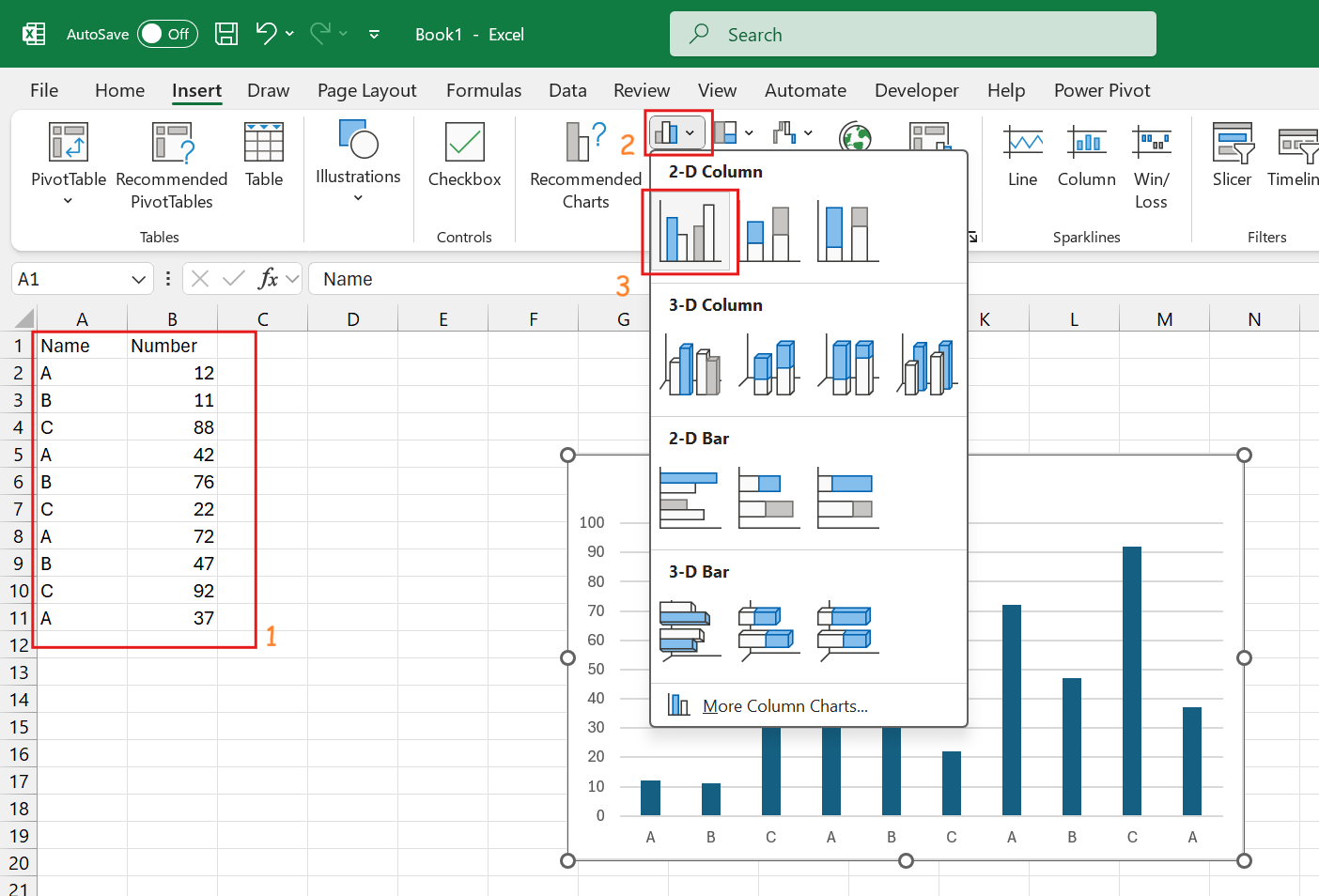
Here are some considerations to guide your decision:

* **Consider the Data Type**:
  + **Categorical Data**: Use column or bar charts to compare categories.
  + **Continuous Data**: Line charts are ideal for showing trends in continuous data over time.
  + **Proportional Data**: Pie charts are suitable for illustrating parts of a whole, while bar charts can show proportions if there are many categories.
* **Understand the Audience**:
  + **Complexity**: For a general audience, simpler charts (like column or pie charts) may be more effective. For a more technical audience, scatter plots or more complex visualizations can convey detailed relationships.
  + **Context**: Consider what insights the audience is seeking. Choose a chart type that will highlight those insights clearly.
* **Keep Clarity and Simplicity in Mind**:
  + Avoid clutter. Too many data points or excessive detail can overwhelm viewers. Choose a chart type that presents the data clearly without unnecessary complexity.
  + Use labels and legends appropriately. Ensure that viewers can easily understand what each part of the chart represents.
* **Test Different Options**:
  + Sometimes, creating multiple chart types for the same dataset can help you find the most effective way to communicate your message. Excel allows you to easily switch between different chart types, so experiment to see what works best.

#### Creating a simple Chart

**Step-by-Step Guide:**

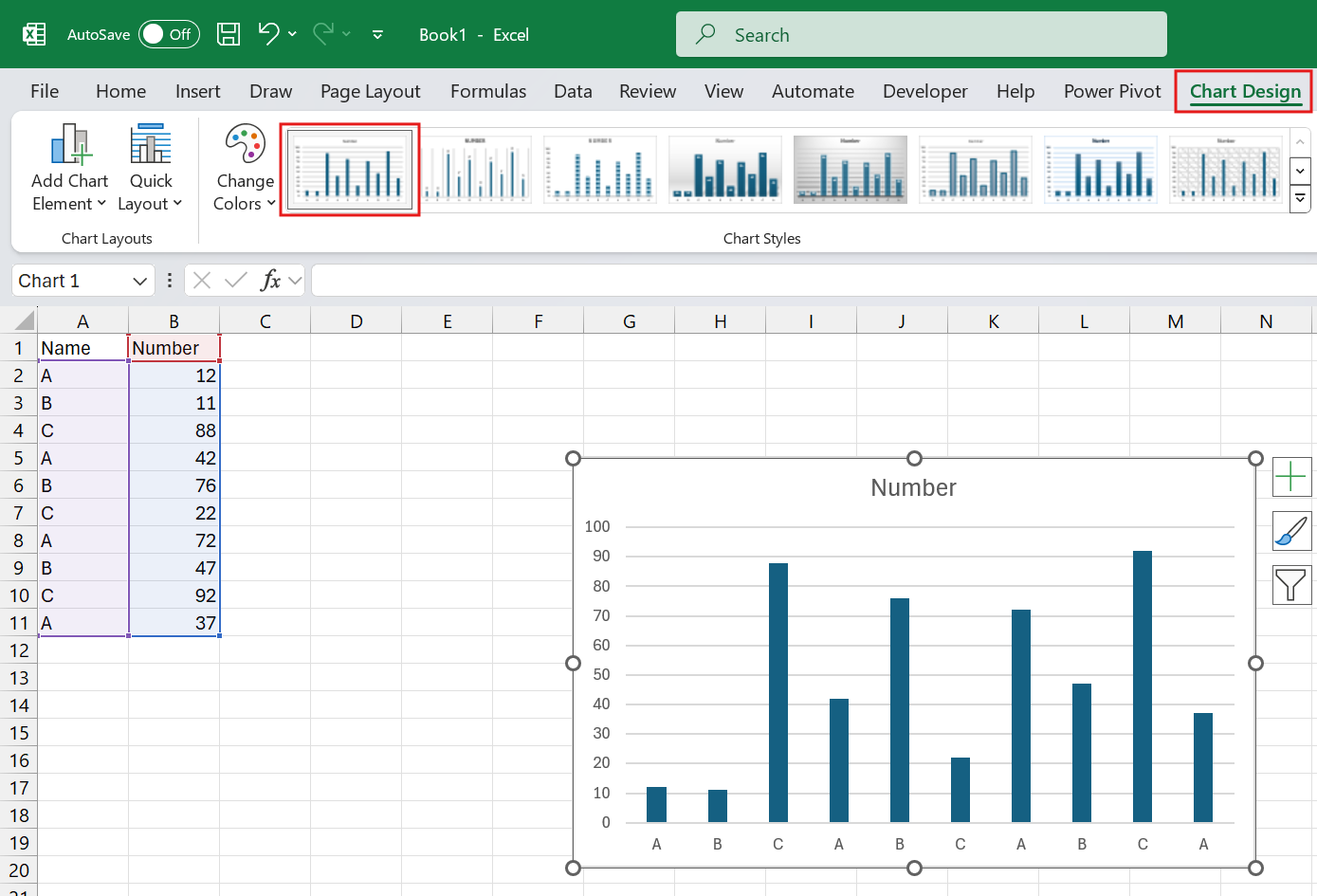
* + **Select Data:**Highlight the data range for the chart.
  + **Insert Chart:**Go to the Insert tab > Choose Chart type.
  + **Customize Chart:**Use Chart Tools to format title, legend, and axes.
  + **Final Touches:**Add data labels and adjust colors as needed.



**Customizing Your Charts**

**Using Chart Tools**

* **Design Tab:**Change chart styles, colors, and layout.
* **Format Tab:**Modify individual chart elements (e.g., fonts, colors).
* **Adding Elements:**Titles, axis labels, data labels, gridlines.
* **Using Templates:**Save custom chart formats for future use.



## 1.1.3 Best Practices for Data Visualization

* Creating effective data visualizations requires attention to detail and an understanding of how viewers will interpret the information presented. Following best practices can significantly enhance the clarity and impact of your visualizations. Here are some key guidelines to consider:

**1. Keep It Simple**

* + **Avoid Clutter**: Limit the number of data points and visual elements to focus on the most important information. Too much detail can overwhelm viewers and obscure key messages.
  + **Highlight Key Data Points**: Use emphasis (such as bolding or color contrast) to draw attention to critical insights. Focus on the main takeaway rather than trying to show every detail.
  + **Choose Clear Visuals**: Select chart types that best represent your data without unnecessary embellishments. Simplicity helps your audience understand the message quickly.

**2. Be Consistent**

* + **Uniform Color Scheme**: Stick to a consistent color palette throughout your visualizations. This helps viewers quickly recognize related information and improves the overall aesthetic.
  + **Consistent Font Style**: Use the same font type and size for titles, labels, and legends across all charts. Consistency in typography enhances readability and professionalism.
  + **Standardized Layouts**: Keep the layout of your charts and graphs similar across different visualizations. This makes it easier for your audience to interpret data without having to adjust to new formats.



Source: https://thenewstack.io/7-best-practices-for-data-visualization/

**3. Make It Accessible**

* + **Consider Color Blindness**: Approximately 1 in 12 men and 1 in 200 women have some form of color blindness. Use color combinations that are distinguishable for everyone. Tools like color blindness simulators can help ensure accessibility.
  + **Ensure Readability**: Choose font sizes that are easily legible from a distance, especially for presentations. Avoid overly decorative fonts that may distract from the data.
  + **Alternative Text Descriptions**: For digital formats, provide alternative text descriptions for charts and graphs to assist those using screen readers.

**4. Provide Context**

* + **Include Titles and Labels**: Every chart should have a clear title that summarizes what the data represents. Axis labels and data point labels should be informative to provide context at a glance.
  + **Use Legends**: If your visualization has multiple data series, include a legend to explain what each series represents. Make sure the legend is easily identifiable and does not clutter the visualization.
  + **Add Annotations**: If specific data points are particularly significant, consider adding annotations to highlight them. This can provide additional context and guide the viewer's attention.

# 2.2 GPT for Excel

**What is GPT?**

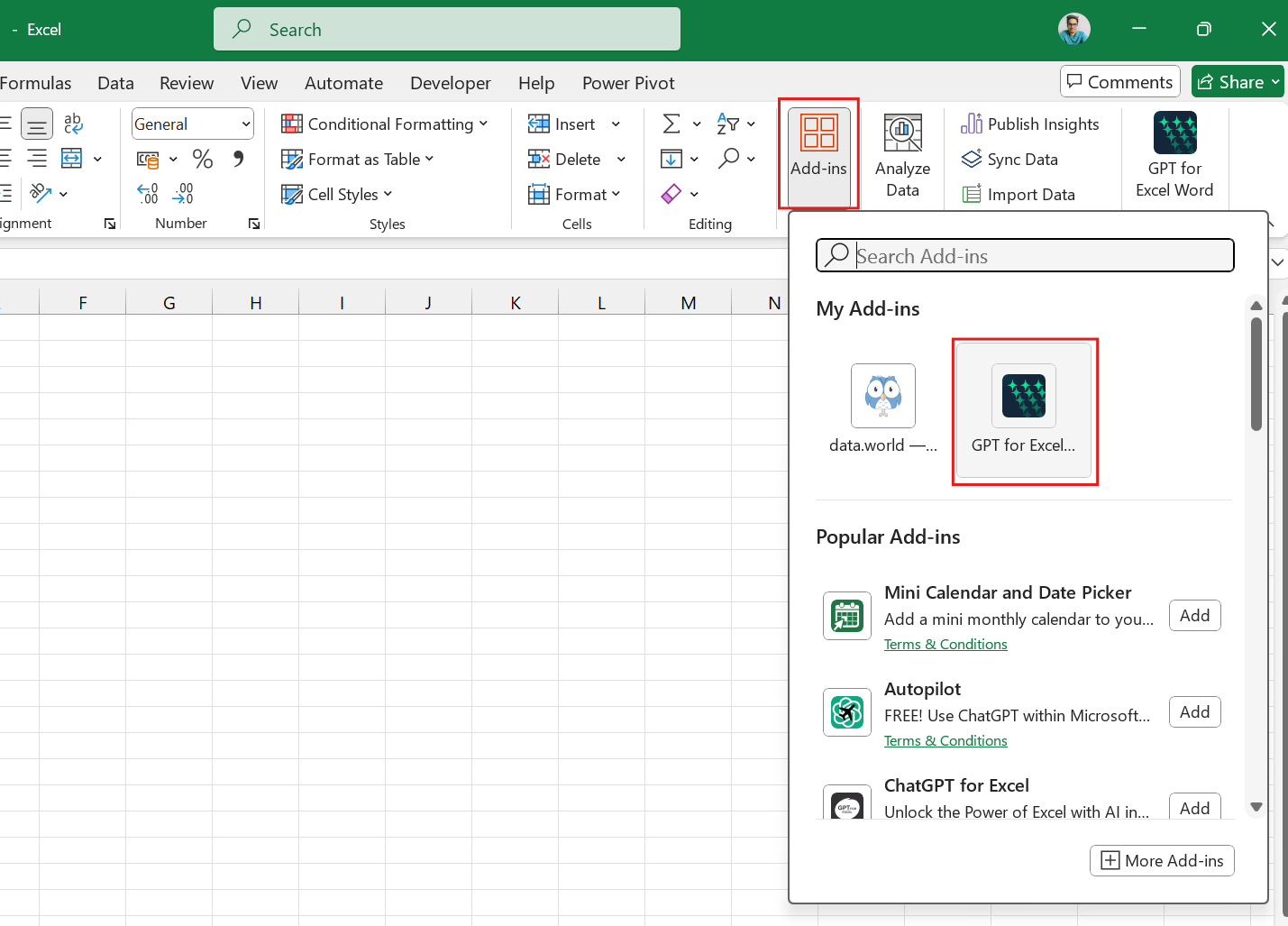
* **GPT (Generative Pre-trained Transformer)** is a powerful AI language model developed by OpenAI. It is designed to understand and generate human-like text based on the input it receives. GPT can perform a variety of tasks, including:
  + **Text Generation**: Producing coherent and contextually relevant text based on prompts.
  + **Question Answering**: Responding to inquiries with informative answers derived from its training data.
  + **Language Translation**: Converting text from one language to another while maintaining meaning.
  + **Summarization**: Condensing lengthy texts into shorter summaries without losing key information.
* The model's ability to understand context and generate nuanced responses makes it a versatile tool for many applications.

**Role of Excel**

* **Microsoft Excel** is a widely used spreadsheet application that facilitates:
  + **Data Analysis**: Excel offers a robust suite of analytical tools, including functions, formulas, pivot tables, and data visualization capabilities.
  + **Data Visualization**: Users can create various charts and graphs to represent data visually, making it easier to identify trends and patterns.
  + **Data Management**: Excel provides features for organizing, filtering, and sorting data, allowing users to manage large datasets effectively.
* Excel is widely utilized across industries for tasks ranging from simple calculations to complex financial modeling and statistical analysis.

**Combining GPT with Excel**

Enhances data analysis, automates repetitive tasks, and provides insights through natural language processing.



## 2.2.1 Features of GPT for Excel

* Integrating GPT with Excel enhances the functionality of the spreadsheet application, making data analysis and management more intuitive and efficient. Here are some of the key features that GPT brings to Excel:

A screenshot of a computer

Description automatically generated

**1. Data Analysis and Insights**

* **Generate Summaries and Insights from Large Datasets Quickly**:
  + GPT can process and summarize extensive datasets, providing concise insights that help users grasp key information at a glance. For example, it can highlight trends, averages, or outliers without the user having to sift through all the data manually.
* **Answer Complex Questions About Data Trends and Patterns**:
  + Users can pose complex questions about their data (e.g., "What factors contributed to the sales increase last quarter?"). GPT can analyze the data and generate responses that clarify trends and relationships, enabling more informed decision-making.

**2. Natural Language Queries**

* **Allow Users to Interact with Excel Using Plain Language**:
  + GPT enables users to communicate with Excel using everyday language. Instead of needing to know specific formulas or commands, users can ask questions or make requests in plain English (e.g., "Show me the total expenses for this month"), and GPT will generate the corresponding actions or formulas automatically.
* **Simplify Data Manipulation and Analysis**:
  + This feature reduces the learning curve for new Excel users and makes data manipulation more accessible to those without extensive technical knowledge. Users can focus on what they want to achieve rather than how to achieve it with complex formulas.

**3. Automated Reporting**

* **Create Reports and Dashboards Automatically Based on Data Inputs**:
  + GPT can automate the generation of reports and dashboards by analyzing the data and creating visualizations or textual summaries that reflect the data's current state. This saves users time and effort, allowing them to focus on interpreting the results instead of compiling them.
* **Tailored Reporting Based on User Preferences**:
  + Users can set preferences for how they want reports formatted or the types of insights they are interested in, and GPT can adjust the reports accordingly. This personalization enhances the relevance of the information provided.

## 2.2.2 Use Cases of GPT in Excel

* The integration of GPT with Excel offers various innovative use cases across different domains, enhancing data analysis, reporting, and decision-making. Here are some key use cases:

**1. Financial Analysis**

* **Generate Financial Forecasts**:
  + Users can input historical financial data and ask GPT to generate forecasts for future revenue, expenses, or cash flows. For example, a user might type, "What are the projected sales for the next quarter based on the last three years of data?" GPT will analyze the provided data and produce a forecast.
* **Analyze Budget Data Using Natural Language**:
  + GPT can help users understand budget performance by answering queries like, "How does this quarter's spending compare to the budget?" or "What areas exceeded budget expectations?" This simplifies the financial review process, enabling quick insights without complex formula creation.

**2. Marketing Analytics**

* **Summarize Customer Feedback**:
  + By analyzing survey responses or customer reviews, GPT can summarize sentiment and key themes. For instance, users can ask, "What are the main points of feedback from our last marketing campaign?" GPT can quickly extract and present the most relevant insights, aiding in decision-making.
* **Sentiment Analysis from Survey Data**:
  + GPT can process qualitative data to provide sentiment scores or categorize feedback into positive, negative, or neutral sentiments. This allows marketers to gauge overall customer satisfaction and identify areas for improvement.

**3. Data Cleaning**

* **Suggest Cleaning Methods**:
  + When users upload messy or inconsistent datasets, GPT can propose specific cleaning methods. For example, it might suggest removing duplicates, filling in missing values, or standardizing formats. Users can receive tailored recommendations based on the nature of their data.
* **Highlight Inconsistencies in Datasets**:
  + GPT can analyze the dataset and identify anomalies, such as outliers or inconsistencies in data entry. For instance, if a user has entered inconsistent date formats, GPT could flag these issues, allowing for quicker resolution.

**4. Personalized Dashboards**

* **Create Tailored Dashboards**:
  + Users can describe their specific reporting needs and objectives (e.g., "Create a sales dashboard that tracks monthly performance against targets"). GPT can generate customized dashboards that include relevant metrics, charts, and visualizations tailored to the user's preferences.
* **Dynamic Updates Based on User-Defined Criteria**:
  + GPT can automate the updating of dashboards based on defined criteria, such as changes in key performance indicators (KPIs) or triggers based on specific thresholds. This ensures users have the most current and relevant information available at their fingertips.

**Chapter 3: Introduction to Python**

|  |
| --- |
| Learning Outcomes:  By the end of this chapter, students will be able to:   * Define Python and identify its key features and advantages. * Recall the steps required for Python installation and setting up a development environment. * Describe the purpose of variables and different data types (integers, floats, strings, booleans). * Describe functions, including lambda functions, and their use in Python. * Create and use custom functions to organize code. * Create a basic class with attributes and methods. * Identify attributes as variables belonging to a class or object. * Create and call methods within a class to perform actions on object data. * Describe inheritance as a way to create a new class from an existing class. * Explain encapsulation as bundling data and methods within a class and restricting access. * Describe polymorphism and its purpose in using a common interface for different forms. * Identify the purpose of Pandas for data manipulation with data frames and series. * Recognize NumPy as a library for handling arrays and numerical data. * Explain common techniques for handling missing values and preparing data for analysis. * Describe the function of Matplotlib in creating visualizations like plots and charts. * Explain Seaborn’s role in creating advanced visualizations and statistical plots. |

# 3.1 Introduction

Disclaimer: The content is curated from online/offline resources and used for educational purpose only

Python is a high-level, interpreted, interactive, and object-oriented scripting language designed with readability and simplicity in mind. Unlike many programming languages that rely heavily on punctuation, Python uses English keywords more frequently, resulting in fewer syntactic constructions and a more intuitive coding experience.

## 3.1.1 Key Features of Python

#### Interpreted

Python code is processed at runtime by the interpreter, eliminating the need for a separate compilation step before execution. This feature is similar to languages such as Perl and PHP.

#### Interactive

Python allows for direct interaction with the interpreter, which facilitates real-time code writing and testing. This interactive nature is valuable for rapid prototyping and experimentation.

#### Beginner-Friendly

Python is renowned for being an excellent choice for beginners. Its simple syntax and readability support the development of various applications, from basic text processing scripts to complex web browsers and games.

## Features of Python



Source: https://unstop.com/blog/features-of-python

**Easy to** Cod**e and Understand**

Python's high-level nature and straightforward syntax make it much easier to learn and use compared to other languages like C, C#, JavaScript, and Java. Beginners can often grasp the fundamentals within a few hours or days, thanks to Python's emphasis on readability and simplicity.

**Expres**siv**e Language**

Python is highly expressive, meaning that a single line of Python code can accomplish what might require multiple lines in other languages. This feature enhances productivity and makes the codebase more readable and maintainable.

#### Free **and** Open Source

Python is free to use and can be easily installed on any system. As an open-source language, Python's source code is available for anyone to modify and distribute, encouraging community contributions and fostering a robust ecosystem.

#### Interp**reted** Language

Unlike compiled languages, Python executes code line by line. This approach simplifies debugging by providing immediate feedback on errors, which is particularly useful for identifying and fixing issues in a complex codebase.

#### Object-Oriented Language

Python supports object-oriented programming (OOP), enabling developers to create and utilize classes and objects. Key OOP principles such as inheritance, polymorphism, data abstraction, and encapsulation are all well-supported in Python.

#### Dynamically Typed Programming Language

In Python, the type of a variable is determined at runtime, allowing for greater flexibility in coding. This dynamic typing facilitates rapid development without the need to specify data types explicitly.

#### Cross-Platform Language

Python is designed to be cross-platform, meaning that code written on one operating system can run on another without modification. This feature simplifies development and testing across different environments.

#### Large Standard Library

Python comes with a comprehensive standard library that includes pre-written code for many common tasks such as file I/O, unit testing, web browsing, database access, CGI, and image manipulation. This extensive library accelerates development by reducing the need to write code from scratch.

#### Extensible Language

Python can be extended by integrating with other languages such as C++ or Java. This extensibility allows developers to optimize performance and add functionalities beyond the core Python language.

#### Platform Independent

Python code is platform-independent, which means it can be executed on various operating systems like Windows, Unix, Linux, and macOS without requiring modifications. This feature ensures seamless development across diverse platforms.

#### Simple Syntax

Python’s syntax is designed to be clear and readable. The language emphasizes simplicity and minimalism, which helps reduce the learning curve for new programmers and facilitates easier code maintenance.

#### High-Level Language

As a high-level language, Python abstracts away much of the complexity associated with system architecture and memory management. This abstraction allows developers to focus on writing code rather than handling low-level system details.

#### Rich Library Support

Python boasts an extensive collection of libraries and frameworks, which provide pre-written solutions for a wide range of tasks. This rich library support helps accelerate development and minimizes the need for custom implementations.

## History of Python

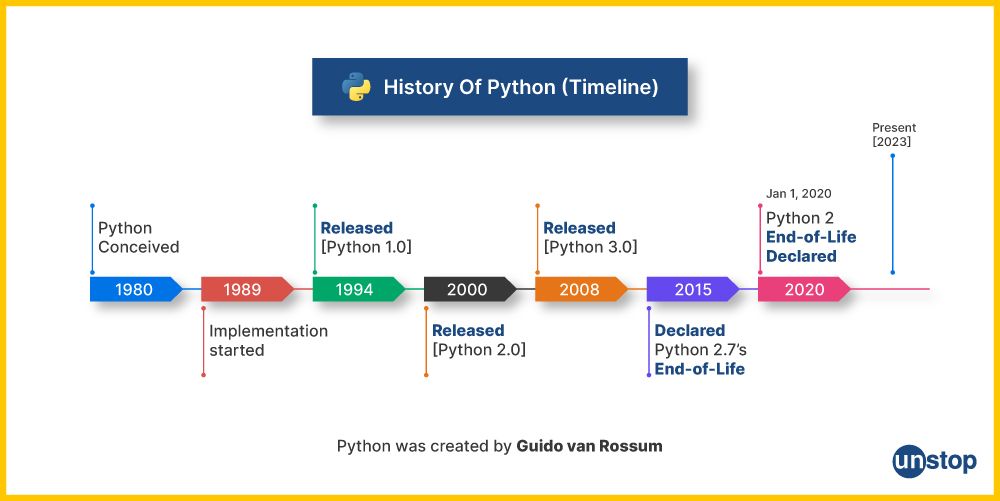
#### Python Timeline/History and IEEE Rankings

Python is a widely-used general-purpose, high-level programming language with a rich history of development and evolution.

#### Key Milestones in Python’s History

* **1980**: Python was conceived.
* **1989**: Implementation of Python began.
* **1991**: Guido van Rossum published the first version of Python (0.9.0) in February at CWI (Centrum Wiskunde & Informatica) in Amsterdam, Netherlands.
* **1994**: Python 1.0 was officially released.
* **2000**: Python 2.0 was released.
* **2008**: Python 3.0 was introduced.
* **2020**: Python 2.7 reached its end-of-life.
* **2024**: Present year.

Python’s development has witnessed several key milestones, each marking significant advancements in its features and usability.



Source: https://unstop.com/blog/what-is-python

#### Notable Facts

* **ABC Influence**: Python draws inspiration from the ABC Programming Language, which was developed at CWI.
* **Naming**: Guido van Rossum chose the name “Python” in honor of his admiration for Monty Python’s Flying Circus.
* **Development Team**: Python is maintained by a core development team, with Guido van Rossum continuing to influence its direction.

#### IEEE Rankings

The latest rankings from IEEE Spectrum reflect Python’s strong presence across various domains. The report analyzes data from diverse sources to assess programming language popularity based on different use cases, including web development, enterprise applications, mobile development, and embedded systems.

#### Ranking Metrics

* **Social Channels**: The rankings consider the language’s popularity on platforms like Twitter, StackOverflow, GitHub, and Reddit.
* **Job Market**: The rankings also account for the demand for languages in the job market.

Python consistently ranks highly in these metrics, underscoring its widespread adoption and ongoing relevance in the programming world.

# 3.2 Installation and Setting Up Path

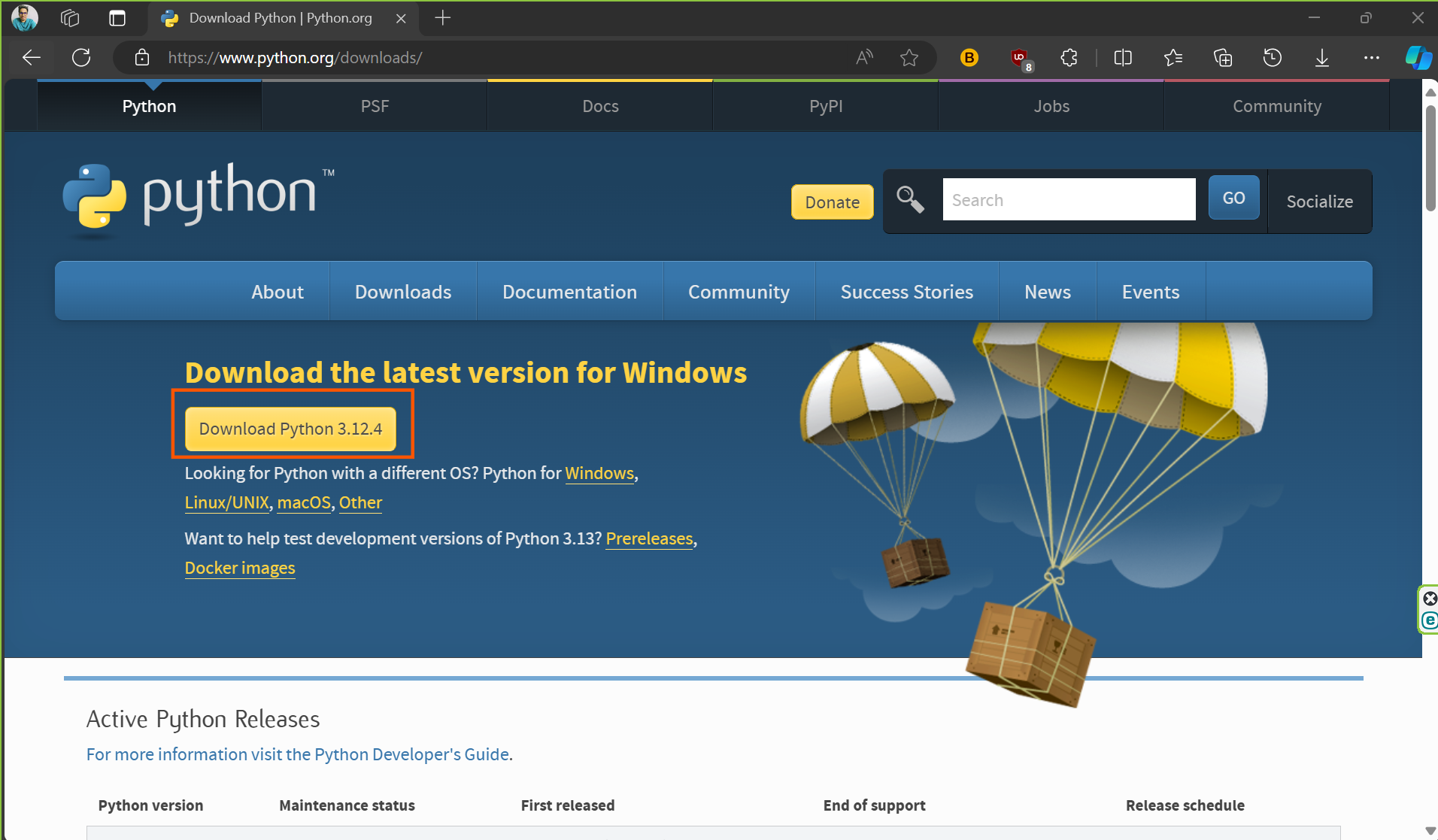
## 3.2.1 Python Installation for Windows

#### List of Hardware/Software Requirements

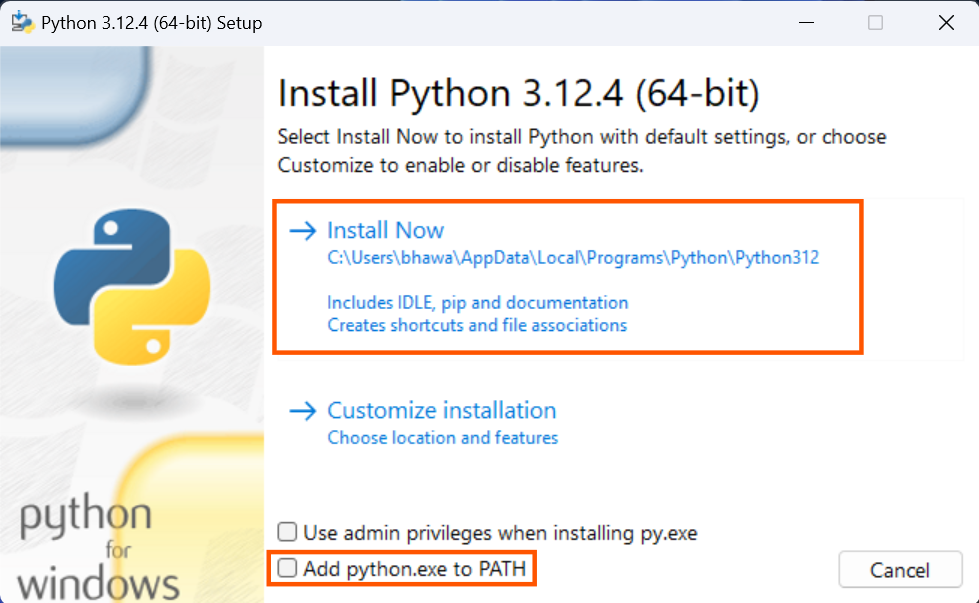
1. Laptop/Computer with Windows OS
2. Python Software Installation

#### Steps: Install Python Software on Windows

1. Open your web browser and navigate to [python.org/downloads](https://www.python.org/downloads).



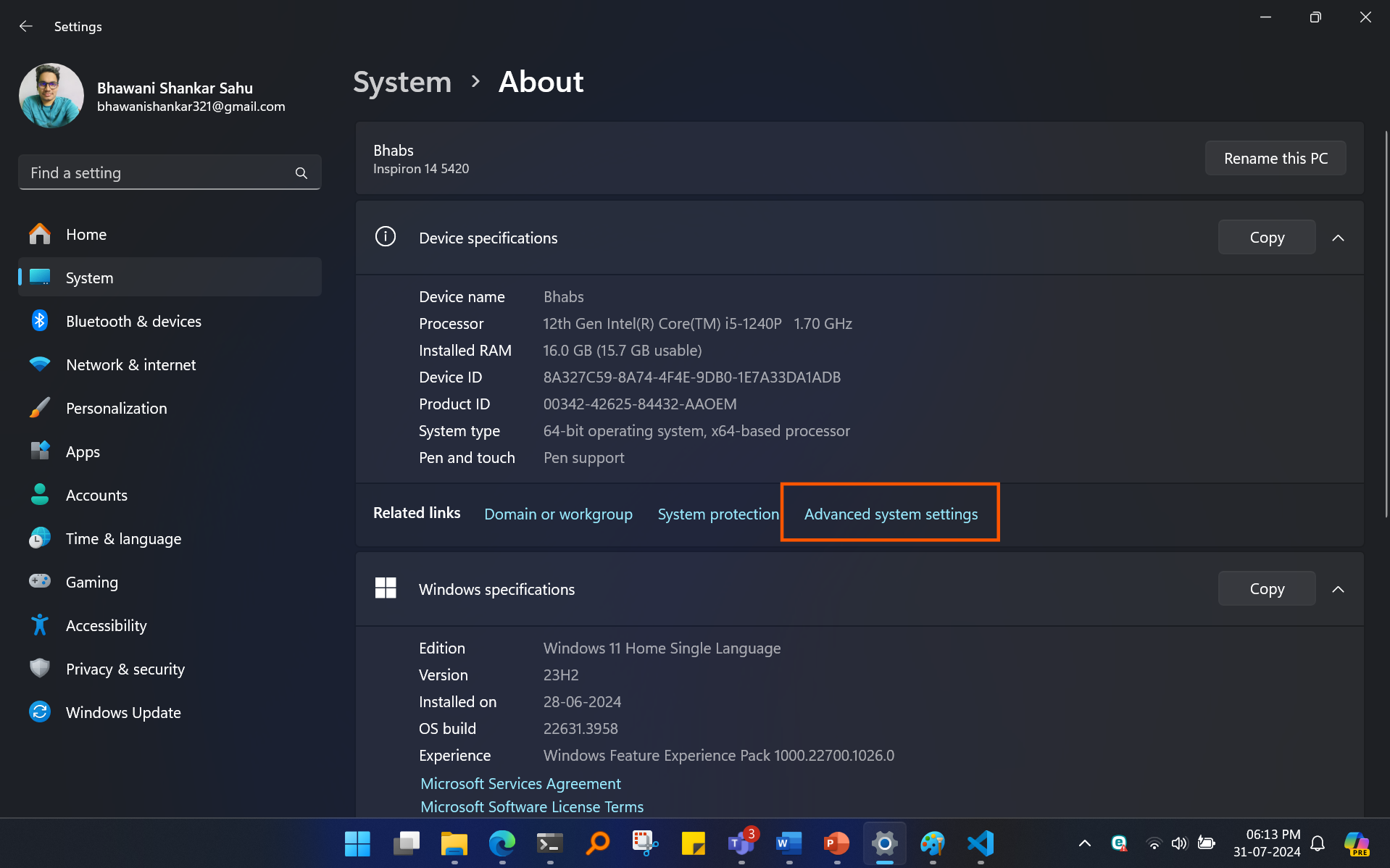
1. Click on the download link suitable for your system:
   * Choose either Windows x86-64 executable installer for 64-bit or Windows x86 executable installer for 32-bit.
2. After downloading the file, launch the setup and click on “Add python.exe to Path“ and then click on “Install Now”.



#### How to Set Python Path in Windows

To permanently modify the default environment variables:

1. **Right-click** 'My Computer' (or 'This PC' in newer versions).
2. Select **'Properties'** from the context menu.
3. Click on **'Advanced system settings'** on the left side.



1. In the Advanced tab, click **'Environment Variables...'**.

A screenshot of a computer program

Description automatically generated

1. Under 'System Variables', find and select the **'Path'** variable, then click **'Edit'**.

A screenshot of a computer

Description automatically generated

1. Add Python's path to the end of the list (paths are separated by semicolons ;).

A screenshot of a computer

Description automatically generated

## 3.2.2 IDEs and Code Editors for Python

When developing Python applications, choosing the right development environment can significantly impact productivity and ease of coding. Both IDEs and code editors serve to write and manage code, but they offer different features and levels of complexity.

A group of logos with different colors

Description automatically generated

Source: https://medium.com/@talktorahul.b/top-10-free-code-editor-software-a063f89fbc86

### Integrated Development Environments (IDEs)

IDEs are comprehensive tools that provide everything needed for software development in one place. They typically include a code editor, a debugger, build automation tools, and other features.

### Features of IDEs

* **Code Editing:** Advanced text editing features like syntax highlighting, code completion, and intelligent code suggestions.
* **Debugger:** Tools for stepping through code, setting breakpoints, and inspecting variables.
* **Build and Run Tools:** Integrated support for running and building your code.
* **Version Control Integration:** Built-in support for Git, Mercurial, or other version control systems.
* **Project Management:** Tools for managing files and directories within projects.
* **Plugins and Extensions:** Support for adding extra functionalities and integrations.

### Popular Python IDEs

#### 1. PyCharm

* Developed by JetBrains, PyCharm is a powerful IDE tailored specifically for Python. It offers advanced features such as code analysis, a robust debugger, and support for web development frameworks like Django and Flask.
* Key Features:
  + Smart code completion
  + Advanced refactoring
  + Integrated testing tools
  + Git integration
  + Python-specific project templates

#### 2. Visual Studio Code (VS Code)

* VS Code is a lightweight, open-source code editor developed by Microsoft. With the Python extension, it can provide many features typical of full-fledged IDEs.
* Key Features:
  + IntelliSense code completion
  + Debugging support
  + Integrated terminal
  + Extensions marketplace
  + Git integration

#### 3. Spyder

* Spyder is an open-source IDE designed for scientific computing and data analysis. It integrates well with scientific libraries such as NumPy, SciPy, and Matplotlib.
* Key Features:
  + Variable explorer
  + Integrated IPython console
  + Scientific libraries support
  + Code analysis tools
  + Interactive data visualization

#### 4. Thonny

* Thonny is a beginner-friendly IDE developed for learning and teaching Python. It offers a simple interface and basic debugging tools.
* Key Features:
  + Simple user interface
  + Built-in debugger
  + Easy package management
  + Step-through debugging

### Code Editors

Code editors are simpler tools focused on editing code rather than providing a full development environment. They are usually more lightweight compared to IDEs.

### Features of Code Editors

* **Syntax Highlighting:** Basic color coding to differentiate code elements.
* **Code Autocompletion:** Suggestions for completing code as you type.
* **File Management:** Tools for managing and navigating code files.
* **Customization:** Support for themes and plugins to enhance functionality.
* **Integration with External Tools:** Limited but useful integration with external tools and terminal.

### Popular Python Code Editors

#### 1. Sublime Text

* Sublime Text is a fast, cross-platform code editor known for its speed and simplicity. It supports Python through plugins and packages.
* Key Features:
  + Multiple selections and editing
  + Command palette for quick access
  + Customizable via plugins
  + Syntax highlighting for Python

#### 2. Atom

* Atom, developed by GitHub, is a customizable and open-source code editor. It supports Python through community packages.
* Key Features:
  + Built-in Git and GitHub integration
  + Customizable through packages and themes
  + Real-time collaborative editing with Teletype
  + Flexible and extensible

#### 3. Notepad++

* Notepad++ is a free source code editor that supports multiple languages including Python. It is lightweight and fast.
* Key Features:
  + Syntax highlighting and folding
  + Plugin support for additional features
  + Multi-document editing
  + Regular expression search and replace

#### 4. Jupyter Notebook

* Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations, and narrative text.
* Key Features:
  + Interactive code execution
  + Integration with data visualization libraries
  + Markdown support for rich text
  + Integration with various data science tools

#### 4. Choosing Between IDEs and Code Editors

* **Complexity and Features:** IDEs are suitable for complex projects requiring advanced features and integrations. Code editors are ideal for simpler tasks or when a lightweight tool is preferred.
* **Performance:** Code editors generally perform faster and use less memory, making them suitable for quick edits and smaller projects.
* **Learning Curve:** IDEs often have a steeper learning curve due to their comprehensive feature sets, while code editors are more straightforward and easier to pick up for beginners.

### Jupyter Notebook

Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations, and narrative text. It is widely used for data analysis, scientific computing, and machine learning. Anaconda is a popular distribution of Python and R for scientific computing and data science, which includes Jupyter Notebook along with other useful tools and libraries.

### Features of Jupyter Notebook

Jupyter Notebook provides an interactive computing environment where you can write and execute code in real-time.

#### Key features include:

* **Interactive Code Execution:** Run code in small chunks (cells) and see the output immediately.
* **Rich Text Support:** Combine code with narrative text, equations, and visualizations using Markdown and LaTeX.
* **Data Visualization:** Integrate with libraries like Matplotlib, Seaborn, and Plotly to create visualizations.
* **Sharing and Collaboration:** Easily share notebooks via GitHub, email, or JupyterHub.

### Setting Up Jupyter Notebook Using Anaconda

#### Installing Anaconda

1. **Download Anaconda:** Visit the Anaconda Distribution website and download the installer for your operating system (Windows, macOS, or Linux). (Link: [Download Anaconda Distribution | Anaconda](https://www.anaconda.com/download) )

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

1. **Install Anaconda:** Run the installer and follow the on-screen instructions. Make sure to check the option to add Anaconda to your PATH environment variable.

#### Launching Jupyter Notebook

1. **Open Anaconda Navigator:** Launch Anaconda Navigator from your start menu or applications folder.
2. **Launch Jupyter Notebook:** In Anaconda Navigator, click the Launch button under the Jupyter Notebook section. This will open a new tab in your default web browser with the Jupyter Notebook interface.

#### Creating a New Notebook

1. **Navigate to the Directory:** Use the Jupyter Notebook file browser to navigate to the directory where you want to create a new notebook.
2. **Create New Notebook:** Click the New button on the right-hand side and select Python 3 (or your preferred environment) from the drop-down menu. This will open a new notebook with an empty code cell.

### Using Jupyter Notebook

#### Notebook Interface

1. **Cells:** The main building blocks of a notebook, where you write code, text, or equations.
   * **Code Cells:** Execute Python code and display the output directly below the cell.
   * **Markdown Cells:** Write formatted text using Markdown syntax.

A screenshot of a computer

Description automatically generated

1. **Toolbar:** Contains buttons for common actions such as running cells, stopping execution, and saving the notebook.
2. **Menu Bar:** Provides access to various features and functions, such as creating new cells, running the entire notebook, and exporting the notebook.

#### Running Code

1. **Write Code:** Type your Python code into a code cell.
2. **Execute Code:** Press Shift + Enter to run the code in the current cell and move to the next cell, or click the Run button in the toolbar.

#### Writing Markdown

1. **Switch to Markdown Cell:** Click on a cell and change the cell type to Markdown from the drop-down menu in the toolbar or press Esc followed by M.
2. **Write Text:** Use Markdown syntax to format text, create headings, lists, links, images, and more.
3. **Render Markdown:** Press Shift + Enter to render the Markdown cell.

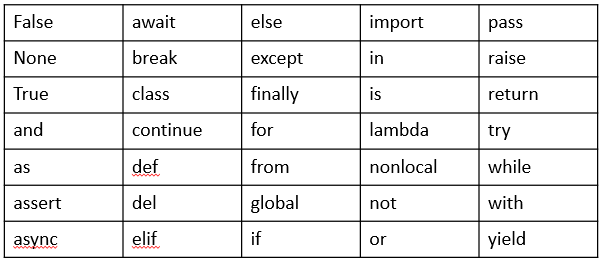
# 2.3 Python Variable and Identifiers

Identifiers are names given to variables, functions, classes, and other elements in Python. Understanding naming conventions is crucial for writing readable and maintainable code.

Rules for Naming Identifiers

* **First Character**: Must be an alphabet (uppercase or lowercase) or an underscore (\_).
* **Subsequent Characters**: Can be alphabets, digits, or underscores.
* **No Special Characters**: Characters like !, @, #, $, %, and punctuation symbols are not allowed.
* **No Successive Underscores**: Avoid using multiple consecutive underscores.
* **Keywords**: Reserved words in Python cannot be used as identifiers.

**Note:** Python is case-sensitive, so Variable and variable are considered different identifiers.



### Lines and Indentation

Indentation in Python is used to define blocks of code. Unlike other languages that use curly braces {}, Python relies on consistent spacing to denote code blocks.

#### Importance of Indentation

* **Syntax Requirement**: Python requires indentation to define blocks of code.
* **Block Definition**: Indentation indicates the start and end of a block.
* **Error Prevention**: Incorrect indentation results in syntax errors.

#### Example:

In this example, statement1, statement2, and statement4 are part of the main block, while statement3 is part of a nested block.

A computer screen shot of a code

Description automatically generated

## Input From the User

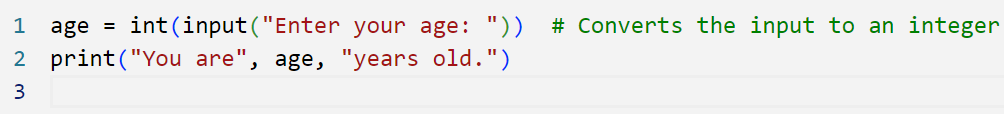
In Python, you can get input from the user using the input() function. This function reads a line of text from the user and returns it as a string.

A close up of a computer screen

Description automatically generated

* input("Enter something: ") prompts the user with the message "Enter something: " and waits for them to type something.
* The entered text is returned as a string and stored in the variable user\_input.
* print("You entered:", user\_input) displays the input text.

**Converting Input Types**: By default, the input() function returns the user input as a string. If you need a different type (e.g., an integer), you’ll need to convert it:



If the user inputs something that cannot be converted to an integer, this will raise a ValueError.

**Handling Input Errors**: It’s good practice to handle potential errors when converting input. For example:

# 3.4 Data Types and Operators

A diagram of a data type

Description automatically generated

Source: <https://www.theengineeringprojects.com/2020/06/how-to-use-data-types-in-python.html>

## Numeric Data

Numeric types represent numbers and include integers, floating-point numbers, and complex numbers.

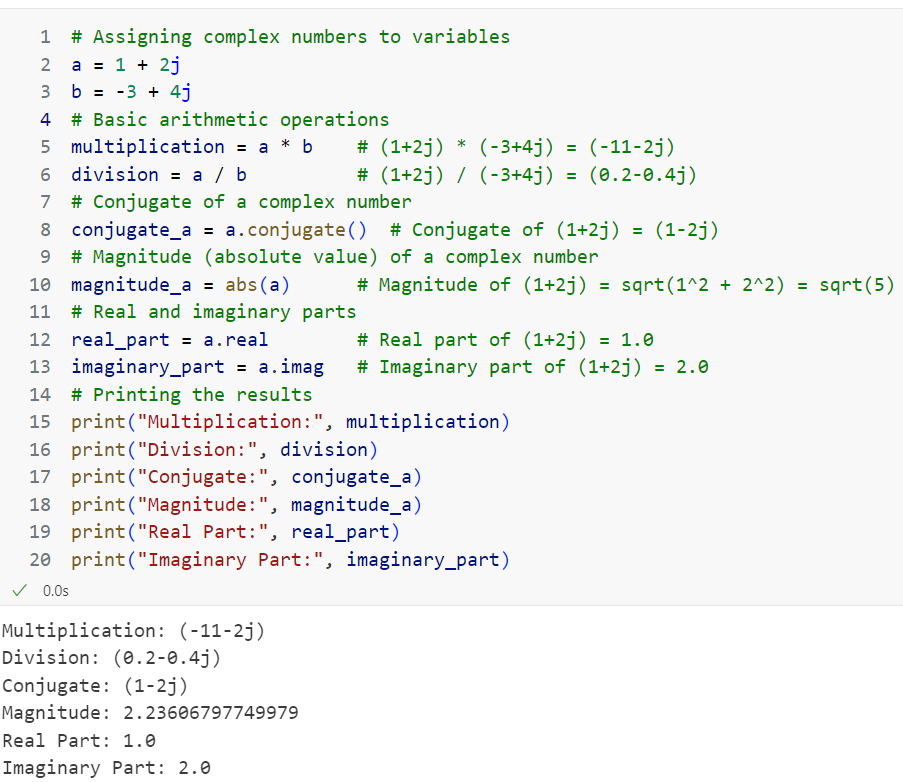
**int**: Integer numbers, e.g., 42, -3, 0



**float**: Floating-point numbers (decimal numbers), e.g., 3.14, -2.7



**complex**: Complex numbers, e.g., 1+2j, -3+4j



#### Strings

A string is just a sequence of characters. It is among the most popular data types in Python. It can be created simply by enclosing characters in quotes.

An immutable data sequence is known as a string in Python. In simple words, as discussed in the case of a crossword, a Python string is nothing but an array of characters, but a computer does not understand characters. It only understands the language of 0’s and 1’s. So these characters are converted to a number so the computer can understand them clearly. This conversion is known as encoding. ASCII and Unicode are some of the popular encodings used.

#### Creating a String in Python

A string in Python can be easily created using single, double, or even triple quotes. The characters that form the string are enclosed within any of these quotes.

**Note:** Triple quotes are generally used when we are working with multiline strings in Python and docstring in Python.

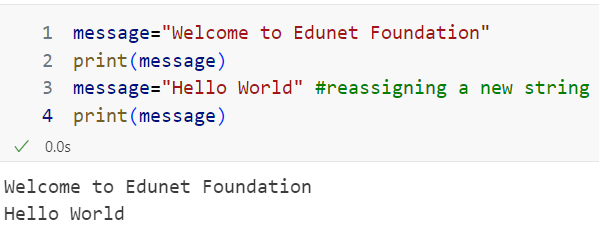
Let’s take a look at an example where we will create strings using the three types of quotes,

A screenshot of a computer program

Description automatically generated

#### Reassigning Strings in Python

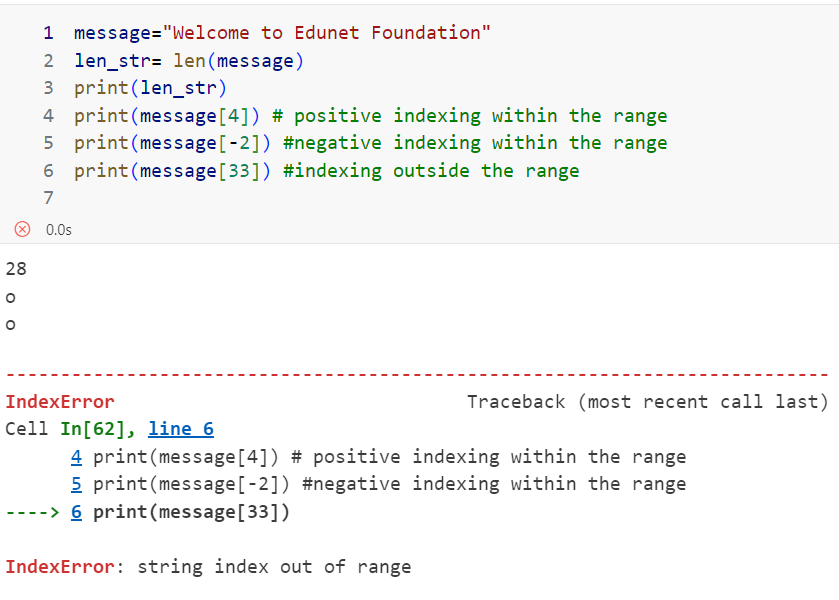
Since strings in Python are immutable, we can update the values of strings simply by reassigning these strings with new values. The strings cannot be partially replaced, so it is always replaced completely with a new string. Let’s see how that’s done



#### Access Characters in a Python string

Indexing: The length of a string can be easily calculated using the len() function. The length of a string provides us with a range of the string. In python, we can perform indexing on any string as long as it is within the range. The index is always referred to as an integer. Using any other data type would give rise to a TypeError.

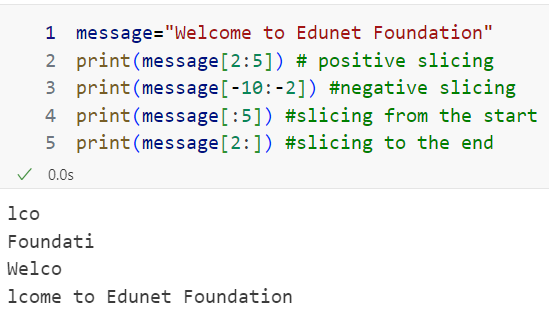
Indexing can either be positive or negative. Positive indexing starts from 0 and indexing from the beginning of the string. On the other hand, negative indexing starts from -1, and indexing starts from the end of the string.



Using the len() function, we calculated the length of the string variable message. We performed positive and negative indexing on this Python string. The thing to note is that we got an IndexError when the index value was way beyond the range of the string.

#### Slicing

Unlike indexing, slicing returns a range of characters using the slicing syntax. While performing slicing operations, one thing to remember is that the start index value is always included while that of the last index is always excluded.

 4.5 Control Structures

## 2.5.1 Conditional Statements

Decision-making (conditional) statements are essential to programming because they enable programs to execute discrete code blocks in accordance with preset circumstances. Among the Python decision-making statements are the if, elif, and if-else statements. By employing these statements, programmers can specify how a program runs based on a range of criteria.

### Types of conditional statements in python

There are four types of decision-making statements in Python language, those are

1. If statements in python
2. If else statement in python
3. Nested if else statement in python
4. if-elif-else Ladder in python

### If statements in python

The if statement in Python is used for decision-making. It allows you to execute a block of code only if a certain condition is met.

#### Syntax of if Statements

The basic syntax of an if statement is:

if condition:

# Block of code to be executed if the condition is true

Flow Chart

|  |  |
| --- | --- |
| If Statement in Python | * The simplest structure for making decisions is an if statement, which only permits the execution of a code block if a predetermined condition is met. * The if keyword is followed by an indented block of statements that are executed if the condition is assessed and found to be true. * If not, the code skips the if block. |

Source: https://www.scholarhat.com/tutorial/python/decision-making-statements-if-else-nested-if-else

#### Example of If Statement

A screenshot of a phone

Description automatically generated

### If else statement in python

The if-else statement in Python is used to execute one block of code if a condition is true, and another block of code if the condition is false. This allows you to handle different outcomes based on the evaluation of a condition.

#### Syntax of if-else Statements

if condition:

# Code to execute if the condition is true

else:

# Code to execute if the condition is false

Flow Chart

|  |  |
| --- | --- |
| How to use IF Else in Python - The Engineering Projects | * Expression or condition: An expression that evaluates to True or False. * The block of code within the if statement is executed if the condition is true. * The block of code within the else statement is executed if the condition is false. |

Source: https://www.theengineeringprojects.com/2020/06/how-to-use-if-else-in-python.html

#### Example of an if-else Statement

A screenshot of a phone

Description automatically generated

## 4.5.2 Looping Statements

Looping statements in Python are used to repeatedly execute a block of code as long as a condition is met. They are fundamental for iterating over data, automating repetitive tasks, and controlling program flow.

Python provides two main looping constructs: for and while loops. The for loop iterates over sequences like lists, tuples, strings, or ranges, executing a block of code for each item. The while loop repeatedly executes a block of code as long as a specified condition remains True.

#### Types of Looping Statements

* for loop
* while loop

### for Loop in Python

The for loop in Python is used to iterate over a sequence of elements, such as lists, tuples, strings, or ranges. It allows you to execute a block of code for each item in the sequence.

Syntax of for loop

for variable in sequence:

# Code to execute for each item in the sequence

* **variable**: The name that will be assigned to the current item in the sequence.
* **sequence**: The iterable object you want to loop through, such as a list, tuple, string, or range.

#### Examples of for loop

Iterating Over a List

A screenshot of a computer code

Description automatically generated

#### Iterating Over a String

A screenshot of a computer screen

Description automatically generated

#### Using range() Function

The range() function generates a sequence of numbers, which can be used to iterate a specific number of times.

A screenshot of a computer

Description automatically generated

#### Nested for Loops

You can use nested for loops to iterate over multi-dimensional sequences.

A screen shot of a computer code

Description automatically generated

### while Loop in Python

The while loop in Python is used to repeatedly execute a block of code as long as a specified condition remains True. It is useful for scenarios where the number of iterations is not known in advance and depends on dynamic conditions.

#### Syntax of while loop

while condition:

# Code to execute as long as the condition is true

The **condition**: An expression that is evaluated before each iteration. If it evaluates to True, the code block inside the while loop executes. If it evaluates to False, the loop terminates.

#### Simple While Loop

A screenshot of a computer code

Description automatically generated

#### Using While with a Condition

This example demonstrates using while to validate user input until it's correct.

A computer code with red and blue text

Description automatically generated

# 2.6 Functions in Python

In Python, a function is a block of reusable code that performs a specific task. Functions help to make code more modular, readable, and maintainable. By defining a function, you can encapsulate a set of instructions and execute them whenever needed without rewriting the same code.

## 2.6.1 Defining a Function

To define a function in Python, you use the **def** keyword followed by the function name, parentheses, and a colon. The code block within the function is indented.

def function\_name(parameters):  
 # Code block  
 statement

Example:

A screenshot of a computer

Description automatically generated

In this example, greet is the function name, and name is a parameter.

## 2.6.2 Calling a Function

Once a function is defined, you can call it by using its name followed by parentheses. If the function requires arguments, you need to pass them within the parentheses.

Example:

A screenshot of a computer code

Description automatically generated

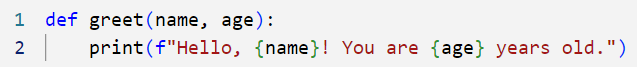
### Parameters and Arguments

In programming, **parameters** and **arguments** are fundamental concepts used in the definition and invocation of functions. Understanding the difference between them is crucial for effective function usage and programming.

#### Parameters

Parameters are variables specified in the function definition. They serve as placeholders that define what kind of data the function expects to receive. When you define a function, list these parameters inside the parentheses following the function's name.

Example:



In this function definition, name and age are parameters. They act as placeholders for the actual values that will be provided when the function is called. Parameters define what inputs the function can accept and how it will process them.

#### Arguments

Arguments are the actual values or data you provide to a function when you call it. They replace the parameters in the function definition and are used to perform operations defined by the function.

Example:



In this function call, "Alice" and 30 are arguments. They correspond to the name and age parameters, respectively. When the greet function is invoked with these arguments, it will replace the parameters with these values and execute the function’s code, resulting in the output: Hello, Alice! You are 30 years old.

A screenshot of a computer code

Description automatically generated

#### Default Parameters

Default parameters in programming allow you to specify default values for function parameters. This feature is particularly useful when you want to provide default behaviour for a function if no specific value is supplied during the function call. Default parameters make functions more flexible and can simplify code by reducing the need for multiple function overloads or additional logic to handle missing arguments.

Example:

A screenshot of a computer code

Description automatically generated

The greet function in Python showcases the use of default parameters and their behavior when called with or without arguments. In its definition, the function is specified as def greet(name="World"):, where name is a parameter with a default value of "World". This means that if no value is provided for name when the function is called, it will automatically use "World" as the default.

When calling greet() without any arguments, the function utilizes this default value, resulting in the output: Hello, World!. This illustrates how default parameters provide a fallback value, simplifying function calls when specific values are not necessary.

In contrast, when greet("Alice") is called, the argument "Alice" overrides the default parameter. Consequently, the output is Hello, Alice!

#### Keyword Arguments

Keyword arguments in programming allow you to specify arguments by name when calling a function. This approach enhances code clarity and flexibility by explicitly identifying which parameter each argument corresponds to, rather than relying solely on the order of arguments.

#### What Are Keyword Arguments?

When you use keyword arguments, you provide the name of the parameter followed by an equal sign and the value you want to pass. This method allows you to bypass the order of parameters and makes your function calls more readable, especially when dealing with functions that have many parameters or default values.

#### Example:

A computer code with black text

Description automatically generated

The describe\_pet function exemplifies the use of keyword arguments to enhance clarity and precision in function calls. Defined as def describe\_pet(animal\_type, pet\_name):, this function takes two parameters: animal\_type and pet\_name. These parameters are placeholders for the type of animal and the pet’s name, respectively. When the function is called with describe\_pet(animal\_type="hamster", pet\_name="Harry"), keyword arguments are used to specify the values directly. This means animal\_type is set to "hamster" and pet\_name is set to "Harry". As a result, the function outputs: I have a hamster named Harry.



#### Return Statement

Functions can return values using the return statement. If no return statement is provided, the function returns None by default.

Example:

A screenshot of a computer

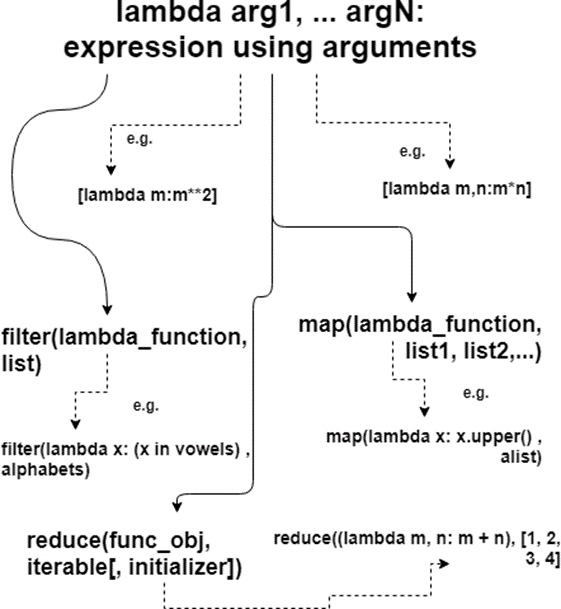
Description automatically generated

The square function is designed to compute the square of a given number. Defined as def square(number):, this function takes one parameter, number, and returns its square by multiplying the number by itself. For example, when the function is called with the argument 4, as shown in result = square(4), it calculates 4×44 \times 44×4 which equals 16. The result of this calculation is stored in the variable result. The subsequent print(result) statement then outputs the value of result, which is 16.

## 2.6.3 Lambda Functions

Lambda functions in Python are concise, anonymous functions defined using the lambda keyword. Unlike regular functions created with the def keyword, lambda functions are designed to be small and are typically used for short-term operations where a full function definition might be excessive.

Lambda is an unnamed function. It provides an expression form that generates function objects. This expression form creates a function and returns its object for calling it later.



Source: <https://techbeamers.com/python-lambda/>

How to create a lambda function?

#### Syntax:

lambda arguments: expression

Lambda arg1, arg2, ... argN: expression using arguments

The body of a lambda function is akin to what you put in a def body’s return statement. The difference here is that the result is a typed expression, instead of explicitly returning it. The lambda statement can appear in places where the def is not allowed. For example – inside a list literal or a function call’s arguments, etc.

Note: Lambda function can’t include any statements. It only returns a function object which you can assign to any variable.

#### Example:

A close-up of a white background

Description automatically generated

In the example, a lambda function named square using the syntax lambda x: x \* x. This lambda function takes a single argument x and returns its square by computing x\*x. When print(square(4)) is executed, the lambda function is called with the argument 4. The function calculates the square of 4, which is 16, and the print statement outputs this result. Consequently, the final output of this code is 16.

#### Properties of Python Lambda Functions

* Anonymous functions created using the lambda keyword can have any number of arguments, but they are syntactically restricted to just one expression, that is, they can have only one expression.
* Lambda function in Python can be used wherever a function object is required.
* Lambda functions do not require any return statement; they always return a value obtained by evaluating the lambda expression in Python.
* Python Lambda functions are widely used with some Python built-in functions such as map(), reduce(), etc. Extending Python lambda functions

# Chapter 4: Data Manipulation and Analysis - Working with Data

# 4.1 Pandas Library: Introduction to data frames, series, and basic operations.

Manipulating and analysing data forms the core of data science. These skills are essential for bringing insights, building predictive models, and visualizing trends out of data. Manipulating data generally involves various operations such as cleaning the data, transforming it to other forms, aggregating the data, and possibly visualization. These operations are vital in understanding patterns or spotting anomalies and preparing your data for more complex analysis or machine learning workflows.

Python is a versatile and powerful language with lots of libraries that make manipulating and analyzing data smooth sailing. In the following chapters, we briefly introduce a few of the most fundamental libraries used when working with data.

### Pandas:

* Pandas is primarily used to manipulate and analyze data.
* Key Features: It contains data structures, DataFrames and Series which simplify the manipulation of structured data. It has capabilities in powerful data loading, cleaning, merging, and filtering.
* Usual applications include data cleaning, exploration, and transformation.

# 4.1.1 Pandas Library

Pandas is an open-source library, designed especially to manipulate and analyze data, and widely used within programs of Python. It is one of the best tools which makes it easier for data scientists and analysts to work with large data sets and also brings vast functionality in the field of data handling and cleaning with analysis.



Source: https://www.scaler.com/topics/architecture-of-data-warehouse/

### Foundations on Numpy

Pandas uses another core library in Python called NumPy as its base. This core library supports operations on large, multi-dimensional arrays and matrices that one can use for high performance numerical operations. Using Pandas this way, you can leverage the capabilities of NumPy for large scale data manipulation without going through tough sessions of data operations.

### Key Data Structures

Pandas uses two main data structures, and they enable working with data in all formats:

**Series:**

A Series is a one-dimensional labeled array capable of holding any data type, including integers, strings, floats, and much more. It can be thought of as a single column in a spreadsheet or a database table.

**DataFrame:**

A DataFrame is a two-dimensional, size-mutable, and heterogeneous tabular data structure. It can be considered as a complete spreadsheet or a SQL table with labeled axes (rows and columns). DataFrames can hold different data types across columns, making them versatile for various types of data analysis.

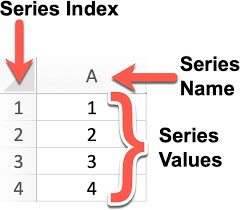
### Key Features of Pandas

Pandas has several prominent features that make data manipulation and analysis easy.

* **High-Level Data Manipulation Tools:**  
  Pandas provides extensive libraries and tools for reshaping, sorting, filtering, and aggregation of data. Such features make challenging tasks of data analysis easy and let the user dedicate their time to getting insights.
* **Data Cleaning, Transformation, and Aggregation:**  
  Missing data can be handled easily with Pandas; calculations can be made on the data; formats can be changed; and aggregation can be done on data. This is necessary for preparation of datasets before any kind of analysis, to make sure that the data is relevant and correct.
* **Merging and Joining Datasets**:  
  Pandas allows data from multiple sources to be merged and joined easily. In real-life analysis scenarios, data could originate from different files, databases, or APIs, making this feature necessary.

## 4.1.2 Pandas Series

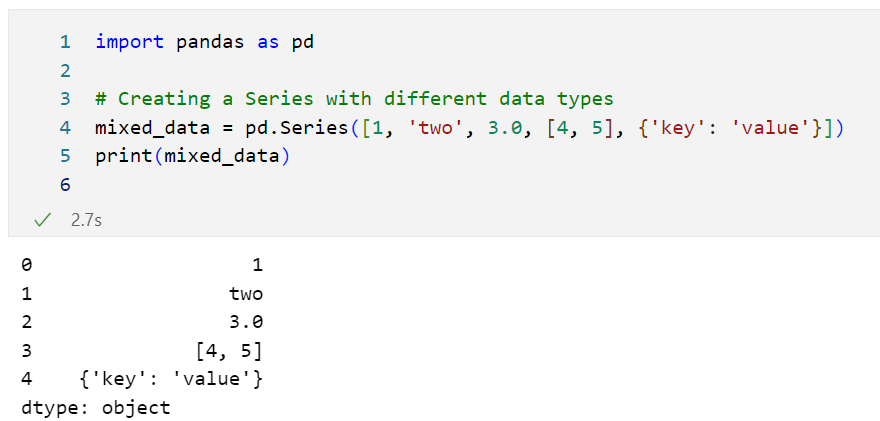
Pandas Series is a labelled one-dimensional array of almost any data type (in other words, it may contain integers, strings, floats or even complex Python objects). It is precisely the same thing as a single column of an Excel sheet. You can easily organize and address data because values come attached with labels or indexes to every entry.



Source: https://codechalleng.es/bites/251/

### Important Points about Pandas Series

* Flexible Data Types:
  + It means that a Series can accommodate many different types of data in one column, thus being flexible for many different types of data analysis and manipulation. For example, you can have integers, floats, strings, and even custom objects all within the same Series.



* Intuitive Labelling
  + Every Series has an index label. Indexing allows for fast access, manipulation, and retrieval of specific elements in the Series. The index can be default numerical indices or custom labels assigned by the user.

A screenshot of a computer code

Description automatically generated

### Creating a Pandas Series with the Series() Method

We can easily convert other data structures, such as lists, tuples, or dictionaries, into a Series object with the Series() method:

1. From a List or Tuple:
   * If you build a Series from a list or tuple, the elements are the values in the Series and will also by default acquire numeric indexing.

A screenshot of a computer code

Description automatically generated

1. From a Dictionary:
   * If you are building a Series from a dictionary, then the keys of the dictionary become index labels of the Series, while the values become the entries in the Series.

A screen shot of a computer

Description automatically generated

### Accessing and Modifying Series:

* Accessing Values:
  + You can obtain values out of a Series via their index labels or integer locations.

A computer screen shot of a code

Description automatically generated

* Updating Values:
  + Values in a Series may be updated quite easily. You could assign new values to any desired indices for updating.

A screenshot of a computer code

Description automatically generated

## 4.1.3 Panda Data Frame

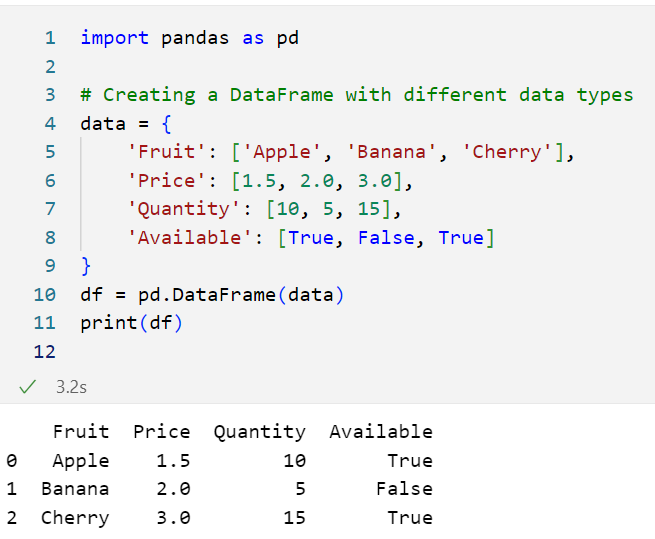
A Pandas DataFrame is the central data structure within the Pandas library and one of the most versatile tools that are found in data handling with Python. It's a two-dimensional data structure with labeled rows and columns, so it's very much like a table in any SQL database, an Excel spreadsheet, or a Google Sheet.

### Key Characteristics of a DataFrame

* Two-Dimensional Format:
  + DataFrames are structured with rows and columns. They provide a structured method for users to store and manipulate large complex datasets. A row is an observation or record, and each column comprises a variable or feature. One can easily access certain points of data from DataFrames.
* Ease of intuitive resemblance with general data format:
  + The frame closely resembles ordinary data formats, such as a spreadsheet or SQL table, and understanding this becomes easy for users accustomed to those tools. Familiarity with this makes it easy to learn and use Pandas to manipulate and analyse data.

### Three Major Parts of a DataFrame

1. **Data:**
   * The actual data contained within the DataFrame, that can be more than one kind of data type—string, integer, float and even more complex data types like a list or dictionary.



1. **Index:**
   * Actual labels assigned to rows and help in uniquely identifying that row. Indexing allows flexibility for quick retrieval of data access. Pandas assigns default integer-based indexing but the user has options to change these indexes

A screenshot of a computer

Description automatically generated

1. **Columns:**
   * Each column has a unique label permitting easy reference and filtering as well as sorting by the actual data points. You can access, modify or delete columns in a DataFrame using these labels.

A screenshot of a computer code

Description automatically generated

### Creating and Working with DataFrames

DataFrames can be created from lists of lists, dictionaries of lists, external csv files, SQL databases, or even directly from other DataFrames. Such flexibility makes DataFrames great for exploratory data analysis, data manipulation, and exploration.

**From a List of Lists:**

You can read a DataFrame directly from a list of lists and provide column names.

A screenshot of a computer code

Description automatically generated

## 4.2.5 Basic Operations with Pandas

* Pandas is a powerful data manipulation library in Python that provides a wide range of functions to work with structured data. Below are some basic operations you can perform using Pandas, including loading data, inspecting it, and selecting specific data subsets.

**1. Loading Data**

Pandas allows you to load data from various formats, making it easy to analyze and manipulate datasets. Here are some common methods:

* **Read CSV Files**:

import pandas as pd

# Load a CSV file into a DataFrame

df = pd.read\_csv('data.csv’)

* **Read Excel Files**:

# Load an Excel file into a DataFrame

df = pd.read\_excel('data.xlsx', sheet\_name='Sheet1') # Specify sheet name if needed

* **Read SQL Databases**:

from sqlalchemy import create\_engine

# Create a database connection

engine = create\_engine('sqlite:///database.db') # Example with SQLite

# Load data from a SQL query into a DataFrame

df = pd.read\_sql\_query('SELECT \* FROM table\_name', engine)

**2. Data Inspection**

Once you have loaded your data, it's essential to inspect it to understand its structure and contents.

* **View the First Few Rows**:

# Display the first 5 rows of the DataFrame

print(df.head())

* **Summary Statistics**:

# Get summary statistics for numerical columns

df.describe()

* **View DataFrame Information**:

# Display information about the DataFrame, including data types and non-null counts

print(df.info())

**3. Data Selection**

Pandas makes it easy to select specific rows and columns from a DataFrame.

* **Selecting Columns**:

# Select a single column

column\_data = df['column\_name’]

# Select multiple columns

subset\_df = df[['column1', 'column2’]]

* **Selecting Rows**:

# Select rows by index (e.g., first 10 rows)

first\_ten\_rows = df.iloc[:10]

# Select specific rows based on a condition

filtered\_df = df[df['column\_name'] > value] # Replace 'value' with your condition

* **Using .loc[] for Label-based Selection**:

# Select rows and columns by labels

selected\_data = df.loc[5:10, ['column1', 'column2']] # Rows 5 to 10 for specific columns

* **Using .iloc[] for Position-based Selection**:

# Select rows and columns by index positions

selected\_data = df.iloc[0:5, 0:2] # First 5 rows and first 2 columns

### Data Manipulation Techniques in Pandas

* Pandas provides a powerful set of tools for data manipulation, allowing users to filter, add or modify columns, and group data for aggregation. Here’s a closer look at these techniques:

**1. Filtering Data**

Filtering is a crucial technique used to select subsets of data based on specific conditions.

* **Conditional Selection**: You can filter DataFrames using boolean indexing to retrieve rows that meet certain criteria.

import pandas as pd

# Example DataFrame

data = { 'Name': ['Alice', 'Bob', 'Charlie', 'David'], 'Age': [24, 30, 22, 35], 'Salary': [50000, 60000, 52000, 58000] }

df = pd.DataFrame(data)

# Filter rows where Age is greater than 25

filtered\_df = df[df['Age'] > 25]

print(filtered\_df)

* **Multiple Conditions**: You can also combine multiple conditions using logical operators (& for AND, | for OR).

# Filter rows where Age is greater than 25 and Salary is less than 60000

filtered\_df = df[(df['Age'] > 25) & (df['Salary'] < 60000)]

print(filtered\_df)

**2. Adding/Modifying Columns**

Pandas allows you to create new columns or modify existing ones easily.

* **Adding a New Column**: You can create a new column by assigning values based on existing columns.

# Add a new column 'Tax' which is 10% of Salary

df['Tax'] = df['Salary'] \* 0.10

print(df)

* **Modifying an Existing Column**: You can update values in an existing column by performing operations on it.

# Increase the Salary by 5%

df['Salary'] = df['Salary'] \* 1.05

print(df)

* **Using apply() to Modify Columns**: The apply() function allows you to apply a function to a DataFrame column.

# Define a function to categorize ages

def categorize\_age(age):

if age < 30:

return 'Young’

else:

return 'Old’

# Apply the function to create a new column 'Age Group’

df['Age Group'] = df['Age'].apply(categorize\_age)

print(df)

**3. Group By**

Grouping data is essential for aggregation and summary statistics.

* **Group By for Aggregation**: The groupby() method allows you to group data by one or more columns and then apply aggregation functions like sum(), mean(), count(), etc.

# Group by 'Age Group' and calculate the average Salary

group\_df = df.groupby('Age Group')['Salary'].mean().reset\_index()

print(group\_df)

* **Multiple Aggregations**: You can perform multiple aggregation operations using the agg() method.

# Group by 'Age Group' and get multiple statistics

agg\_df = df.groupby('Age Group').agg({ 'Salary': ['mean', 'max', 'min'], 'Age': 'count' }).reset\_index()

print(agg\_df)

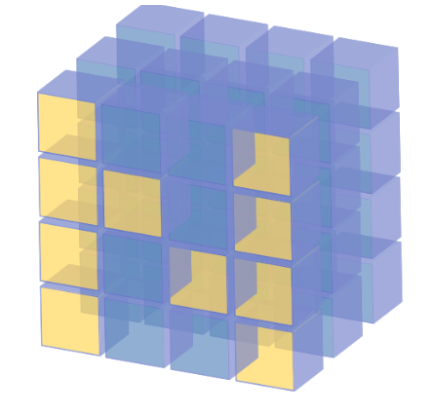
## 4.2.6 Applications of Pandas

Pandas is an essential library for data science and analytics, with applications that span several key areas. Here’s a breakdown of its primary uses:

* **Data Handling**:
  + **Importing Data**: Pandas can import data from various sources and formats, including CSV files, Excel spreadsheets, SQL databases, and even JSON files. This versatility simplifies working with data from different systems and makes it easy to integrate data sources for analysis.
  + **Handling Missing Data**: In real-world data, missing values are common. Pandas provides tools to identify, fill, replace, or drop missing data, making it easier to clean datasets and prepare them for analysis.
* **Data Wrangling**:
  + **Filtering and Grouping**: Pandas offers intuitive ways to filter data based on conditions, and group data by specific columns to obtain insights from different segments within a dataset.
  + **Merging and Joining**: You can merge or join multiple datasets based on shared keys or indexes, which is valuable when working with related data from multiple sources.
  + **Reshaping Data**: Pandas allows data reshaping through pivoting, stacking, and unstacking, which helps to reorganize data to fit analysis needs better.
* **Exploratory Data Analysis (EDA)**:
  + **Descriptive Statistics**: Pandas provides quick access to summary statistics like mean, median, min, max, and standard deviation, which helps understand data distributions and identify outliers.
  + **Data Visualization Support**: Though Pandas isn’t a dedicated visualization library, it integrates with Matplotlib and Seaborn for visualizing data, enabling quick, simple charts directly from DataFrames to visualize trends and patterns in the data.

# 4.3 NumPy Library: Handling arrays and numerical data

NumPy, short for Numerical Python, is one of the core libraries in Python for numerical computing. It’s a foundational package in the data science ecosystem, widely used for performing efficient and fast computations on large datasets. NumPy is designed to handle multi-dimensional arrays and matrices efficiently, which makes it essential for data analysis, machine learning, and scientific computing.



Source: https://www.scaler.com/topics/architecture-of-data-warehouse/

## 4.3.1 Key Features of NumPy:

* **Efficient Handling of Numerical Data**:
  + NumPy is optimized for operations on large arrays of data, which are stored in contiguous memory blocks. This setup allows for quick data access and computation, making NumPy much faster for numerical tasks compared to standard Python lists.
* **Support for Multi-dimensional Arrays**:
  + The array is the central data structure in NumPy, and it can be one-dimensional (vectors), two-dimensional (matrices), or multi-dimensional (tensors). This versatility makes NumPy suitable for complex mathematical operations and multi-dimensional data manipulation.
* **Comprehensive Mathematical Functions**:
  + NumPy provides a broad range of mathematical functions for array operations, including linear algebra, statistical functions, random sampling, and more. These functions make it easy to perform a wide array of calculations, from basic arithmetic to advanced linear algebra operations.
* **Essential for Scientific Computing and Data Analysis**:
  + Due to its speed and efficiency, NumPy serves as the backbone of many other Python libraries for scientific computing and data analysis, such as Pandas, Matplotlib, and SciPy. Together, they form a powerful ecosystem for analyzing, visualizing, and working with large-scale data.

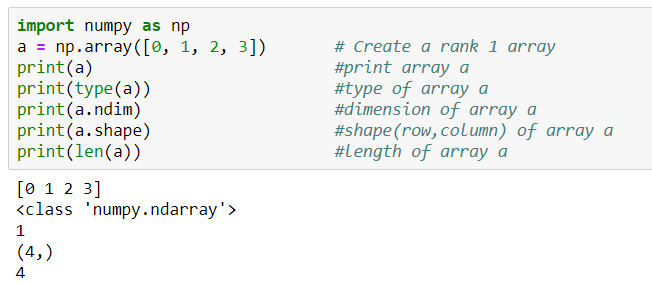
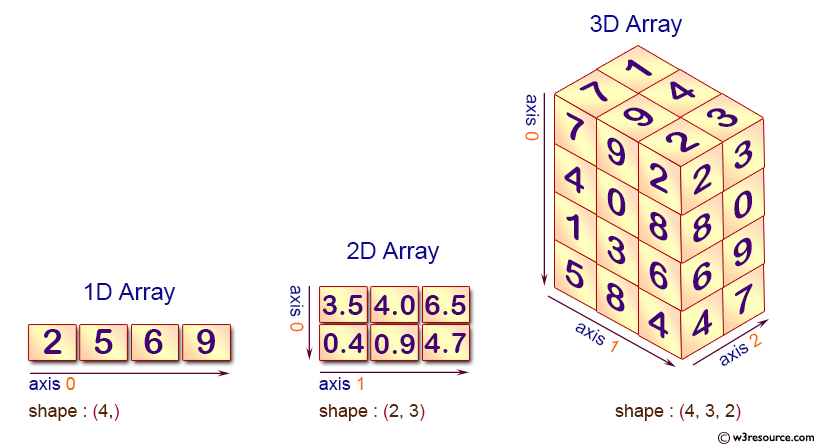
## 4.3.2 NumPy Array

* The NumPy array is a multi-dimensional data structure that serves as the foundation for scientific computing in Python. Arrays are the primary way of storing and manipulating numerical data in NumPy, and they are highly optimized for performance, making them essential in data analysis and machine learning.

### Key Characteristics of NumPy Arrays:

* **Homogenous Data**: All values within a NumPy array are homogenous, meaning they are of the same data type (e.g., all integers, all floats, etc.). This uniformity makes processing faster and more memory-efficient, as operations can be applied uniformly across all elements.
* **Multi-Dimensional Structure**: Unlike traditional lists in Python, NumPy arrays can be multi-dimensional. This allows you to create arrays of any shape (1D vectors, 2D matrices, or higher-dimensional tensors), making them versatile and powerful for representing complex data structures, such as images or time-series data.
* **Automatic Vectorization and Broadcasting**:
  + **Vectorization**: NumPy arrays support automatic vectorization, meaning you can perform mathematical operations on entire arrays without explicit loops. This leads to code that is not only faster but also more concise and readable.
  + **Broadcasting**: NumPy’s broadcasting feature enables operations on arrays of different shapes, which simplifies many mathematical operations. For example, you can add a scalar value to an array, or add two arrays of different shapes, without extra code.

**Explanation of the code:**



* This code begins by importing the NumPy library as np, which is a popular library in Python for numerical computations.
* It then creates a one-dimensional (rank 1) array a containing the integers 0 through 3.
* The code proceeds to print the array a itself, displays its type (which will confirm it's a NumPy array), and checks the number of dimensions using a.ndim, which should return 1 for a one-dimensional array.
* Additionally, it retrieves and prints the shape of the array with a.shape, which indicates the size of each dimension; since a is one-dimensional, this will return a tuple with a single value representing the length of the array.
* Finally, it prints the length of the array using the built-in len() function, which confirms the number of elements in a, returning 4 in this case.

## 4.3.3 Working With NumPy:

**Creating a 2-D Array from a List**:

* The code first imports the NumPy library as np. It then defines a list called mylist, which is a nested list containing three sublists: [1, 2, 3], [4, 5, 6], and [7, 8, 9].
* The np.array(mylist) function is used to convert this nested list into a 2-D NumPy array. This results in a 2-D array with three rows and three columns.

A screenshot of a computer program

Description automatically generated

**Creating Random Arrays**:

* **np.random.rand(5):** This function generates a 1-D array of 5 random values drawn from a uniform distribution between 0 and 1.
* **np.random.rand(4, 4):** This generates a 2-D array with 4 rows and 4 columns, filled with random values from a uniform distribution between 0 and 1.
* **np.random.randn(4):** This creates a 1-D array of 4 random values drawn from a standard normal distribution (mean 0 and standard deviation 1).

## 4.3.4 Commonly Used NumPy Methods

NumPy provides a variety of methods that are essential for creating and manipulating arrays. Here are some of the most commonly used functions:

* **np.array()**:
  + This function creates a **NumPy array** from a list or any other iterable. It’s the fundamental method to convert standard Python data structures into NumPy arrays, enabling all the advantages of array operations.
  + **Example**: np.array([1, 2, 3]) creates a one-dimensional array with three elements.
* **np.random.randint(0, 10, 1)**:
  + Generates a **random integer** between 0 and 10, with the specified size. This method is useful for simulations and random sampling.
  + **Example**: np.random.randint(0, 10, 1) might produce an output like array([4]).
* **np.unique()**:
  + Finds the **unique elements** in an array and returns them in sorted order. This is particularly useful for data analysis when you want to identify distinct values.
  + **Example**: np.unique([1, 2, 2, 3]) would return array([1, 2, 3]).
* **np.dot()**:
  + Computes the **dot product** of two arrays or matrices. This is fundamental in linear algebra for various applications, including machine learning algorithms.
  + **Example**: np.dot(np.array([1, 2]), np.array([3, 4])) yields 11, which is calculated as 1∗3+2∗41\*3 + 2\*41∗3+2∗4.
* **np.power(a, 2)**:
  + Raises each element of the array a to the power of 2. This is a convenient way to apply element-wise operations on arrays.
  + **Example**: If a = np.array([1, 2, 3]), then np.power(a, 2) results in array([1, 4, 9]).
* **np.sqrt(a)**:
  + Returns the **square root** of each element in the array a. This is useful for statistical analysis and data transformation.
  + **Example**: For a = np.array([1, 4, 9]), np.sqrt(a) would return array([1., 2., 3.]).
* **np.ones()**:
  + Creates a new array of the specified shape, filled with **ones**. This can be useful for initializing matrices or tensors in computations.
  + **Example**: np.ones((2, 3)) creates a 2x3 array filled with ones.
* **np.zeros()**:
  + Similar to np.ones(), this method creates a new array of the specified shape, but filled with **zeros**. It’s often used to initialize arrays before populating them with data.
  + **Example**: np.zeros((2, 3)) generates a 2x3 array filled with zeros.
* **np.max()**:
  + Returns the **maximum value** in the array. This is helpful for identifying the largest element in datasets.
  + **Example**: For a = np.array([1, 5, 3]), np.max(a) would return 5.
* **np.arange(10)**:
  + Creates an array with **evenly spaced values** from 0 to 9. It’s similar to Python’s built-in range() function but returns a NumPy array instead.
  + **Example**: np.arange(10) results in array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]).

## 4.3.5 Applications of NumPy

* NumPy is an essential library for numerical computing in Python, with a wide range of applications in data analysis, scientific computing, and machine learning. Here are some key areas where NumPy shines:
* **Array Manipulation**:
  + **Creation**: NumPy allows for the easy creation of arrays from lists, tuples, or even other arrays. This flexibility makes it straightforward to initialize data structures for various applications.
  + **Manipulation**: With NumPy, you can efficiently manipulate arrays by reshaping, merging, or splitting them, enabling complex data transformations.
  + **Slicing**: NumPy provides powerful slicing capabilities, allowing users to extract specific portions of arrays. This feature is crucial for data analysis tasks, where you often need to focus on particular subsets of your data.
* **Mathematical Operations**:
  + **Element-Wise Operations**: NumPy enables element-wise operations on arrays, allowing you to perform calculations on entire datasets without explicit loops. This leads to cleaner and more efficient code.
  + **Linear Algebra**: NumPy provides comprehensive support for linear algebra operations, such as matrix multiplication, determinants, and eigenvalues, which are fundamental in various scientific and engineering applications.
  + **Statistical Operations**: The library includes a wide array of statistical functions (mean, median, standard deviation, etc.) that help analyze data distributions and trends, making it essential for data analysis tasks.
* **Integration with Other Libraries**:
  + NumPy works seamlessly with other popular Python libraries, such as Pandas (for data manipulation), SciPy (for scientific computing), and Matplotlib (for data visualization). This integration enhances its functionality and allows users to leverage the strengths of each library in their data workflows.
* **Performance**:
  + NumPy is optimized for speed and performance, making it ideal for large-scale data analysis. It uses efficient memory management and vectorized operations, enabling computations to be executed much faster than with traditional Python lists. This efficiency is crucial for handling big data and complex numerical tasks in fields like machine learning, statistics, and scientific research.

# 4.4 Data Preprocessing

* Data preprocessing is a crucial process that transforms raw data into a clean, organized, and usable format that is suitable for analysis. It lays the foundation for effective data analysis by ensuring that the data is accurate, consistent, and ready for the next steps in any analytical workflow.
* Data preprocessing involves several essential tasks that enhance data quality and ensure consistency. Properly preprocessed data improves the reliability of analysis, leading to more accurate insights and better decision-making.

Data preprocessing typically involves four key steps:

* Discovery: Understanding the data, identifying its structure, format, and potential issues.
* Transformation: Converting data into a suitable format for analysis, including handling missing values, outliers, and inconsistencies.
* Validation: Ensuring data accuracy and consistency through checks and verification.
* Publishing: Preparing the processed data for analysis and consumption.

## 4.4.1 Discovery Phase

* The primary goal of the Discovery Phase is to understand the data in the context of the specific use case. This involves gaining insights into the data's characteristics, structure, and potential challenges that might affect analysis.

**Activities**:

* **Clean**: Identify and address initial data quality issues, such as missing values or outliers that could impact the analysis.
* **Structure**: Determine how the data is organized and whether it is in a suitable format for analysis. This may involve assessing whether the data is in tabular form, time-series format, etc.
* **Organize**: Arrange the data logically to facilitate further processing. This might include categorizing variables or defining the relationships between different datasets.
* **Map Data**: Create a data dictionary or schema that outlines the data types, variable definitions, and relationships among different datasets. This helps clarify how each piece of data contributes to the analysis.
* **Outcome**: The outcome of the Discovery Phase is to determine the best approach for data preprocessing. By thoroughly understanding the data, you can make informed decisions about the methods and techniques that will be most effective for transforming the data into a usable format for analysis.

## 4.4.2 Transformation Phase

The Transformation Phase focuses on preparing the data for analysis through various techniques.

**Key activities include:**

* **Structuring**:
  + Organizing data into a specific format suitable for analysis, such as converting data into tabular formats or creating structured datasets from unstructured sources.
  + **Example**: Transforming raw text data into a structured table with columns for different attributes.
* **Normalizing**:
  + Adjusting data values to a common scale to ensure that no particular feature disproportionately affects the results. This is essential in algorithms sensitive to the scale of input data.
  + **Example**: Scaling numerical features to a range between 0 and 1, or standardizing them to have a mean of 0 and a standard deviation of 1.
* **De-normalizing**:
  + Converting normalized data back into its original format or scale, if needed, to interpret results in a more understandable context.
  + **Example**: Converting scaled values back to their original units for reporting or presentation purposes.
* **Cleaning**:
  + Removing or correcting errors in the data, such as inconsistencies, duplicates, or incorrect values. This is crucial for ensuring the integrity and reliability of the dataset.
  + **Example**: Identifying and correcting typos in categorical variables or removing duplicate records.
* **Enriching**:
  + Adding additional information to the data to provide more context or enhance the analysis. This could involve merging datasets, adding new features, or integrating external data sources.
  + **Example**: Combining customer data with demographic information to enrich insights for a marketing analysis.

## 4.4.3 Validation Phase

* The Validation Phase is critical for evaluating the quality of data after transformation. This step ensures that the data meets specific standards of accuracy, reliability, and security, making it fit for analysis and decision-making.

**Validation Rules**:

* During this phase, a set of validation rules is applied. These are repetitive programming procedures designed to verify the integrity of the data. Key aspects include:
  + **Accuracy**: Ensuring that the data correctly represents the real-world scenarios it is supposed to depict. This may involve cross-checking against known values or validating calculations.
  + **Reliability**: Confirming that the data remains consistent and trustworthy across different analyses or datasets. This might include checks for data consistency over time.
  + **Security**: Ensuring that sensitive data is handled appropriately, adhering to privacy standards and regulations. This includes checking for unauthorized access and validating that data is stored securely.

**Common Validation Techniques**:

* **Automated Scripts**: Utilizing scripts to run checks on data quality metrics, such as completeness, validity, and uniqueness.
* **Statistical Checks**: Applying statistical methods to validate distributions and ranges of numerical data.
* **Peer Review**: Involving team members in reviewing the data and the validation processes to catch potential issues that automated methods may miss.

## 4.4.4 Publishing Phase

* The Publishing Phase involves the final delivery of the processed data for use in projects. This step is crucial for making the data accessible and usable for analysis and decision-making by stakeholders.

**Includes**:

* **Transformed and Validated Data**:
  + The core output of this phase is the **cleaned and processed dataset**, which has undergone the entire preprocessing pipeline, ensuring its quality and reliability.
  + This data is structured and ready for analysis, typically stored in formats such as CSV, JSON, or in databases.
* **Metadata**:
  + Along with the dataset, accompanying **metadata** is provided. Metadata includes important information about the data, such as:
    - **Data Source**: Where the data was originally collected or obtained.
    - **Transformation Steps**: A summary of the preprocessing steps taken, which is crucial for transparency and reproducibility.
    - **Data Dictionary**: Definitions of each variable, including types and units of measurement, which helps users understand the data better.
* The Publishing Phase ensures that the processed data is delivered in a way that is both usable and understandable, enabling effective utilization in analysis, reporting, and decision-making processes.

## 4.4.5 Data Preprocessing Techniques - Data Cleaning

* The primary goal of Data Cleaning is to handle incorrect, incomplete, inconsistent, or missing values in the dataset. Data cleaning is a critical step in the preprocessing phase because the quality of data directly impacts the accuracy and reliability of the analysis. Proper data cleaning helps ensure that the insights derived from the data are valid and actionable.

**Techniques for Data Cleaning**

* **Handling Missing Values**:
  + **Removing Rows/Columns**:
    - One straightforward approach is to remove rows or columns that contain NULL or missing values. This is useful when the amount of missing data is minimal and won’t significantly impact the overall dataset.
    - **Example**: If a column has more than 50% missing values, it may be more practical to remove the entire column rather than trying to fill in those gaps.
  + **Imputing Missing Values**:
    - When it’s not feasible to remove missing data, imputation techniques can be employed to fill in the gaps. Common methods include:
      * **Mean Imputation**: Replacing missing values with the mean of the available data for that variable. This method is straightforward but can distort the original data distribution.
      * **Mode Imputation**: Using the mode (most frequent value) for categorical variables to fill in missing entries.
      * **Regression Imputation**: Predicting missing values based on other variables in the dataset using regression techniques. This approach can be more accurate, especially when dealing with complex datasets.
      * **K-Nearest Neighbors (KNN)**: Utilizing the KNN algorithm to find the nearest observations and using their values to impute the missing data. This method considers the relationships between different data points.
* **De-noising**:
  + **Removing Noise**: Data often contains **noise**, which refers to random errors or fluctuations that can obscure the underlying trends in the data. Common sources of noise include entry errors, faulty data collection methods, or sensor inaccuracies.
  + **Techniques for De-noising**:
    - **Binning**: This technique involves grouping continuous data into discrete intervals or "bins." By aggregating values into bins, you can reduce the impact of noise and highlight trends. For example, instead of analyzing individual temperatures, you could analyze temperature ranges (e.g., 0-10°C, 11-20°C).
    - **Regression Smoothing**: This approach uses regression analysis to fit a smooth curve to the data, effectively filtering out noise while preserving the main trends. Techniques like polynomial regression can be useful here.
    - **Clustering**: Clustering algorithms can identify and group similar data points, allowing you to detect and potentially remove outliers or erroneous entries that do not fit within any cluster. For instance, K-means clustering can help identify natural groupings in the data, making it easier to spot anomalies.

## 4.4.6 Data Preprocessing Techniques - Data Reduction

* The primary purpose of Data Reduction is to reduce the volume of data while maintaining its integrity and meaning. This process enhances storage efficiency and improves the speed and performance of data analysis. By minimizing data complexity, analysts can focus on the most relevant information, leading to faster and more insightful analyses.

**Techniques for Data Reduction**

* **Dimensionality Reduction**:
  + **Goal**: To reduce the number of features (dimensions) in a dataset while retaining the most important information. High-dimensional data can be challenging to analyze due to the "curse of dimensionality," which can lead to overfitting and increased computational costs.
  + **Methods**:
    - **Feature Selection**: This involves identifying and retaining only the most relevant features of the dataset while removing those that contribute little or no information. Techniques such as correlation analysis, recursive feature elimination, or using machine learning algorithms (like decision trees) can help in selecting significant features.
    - **Principal Component Analysis (PCA)**: PCA is a mathematical technique that transforms the data into a new set of dimensions (principal components) that capture the maximum variance. This method reduces dimensionality by projecting the data onto fewer dimensions while preserving as much variance as possible. It’s widely used in exploratory data analysis and for visualizing high-dimensional datasets.
* **Numerosity Reduction**:
  + **Goal**: To create smaller, more manageable data representations while retaining essential characteristics of the original data.
  + **Methods**:
    - **Regression Models**: Utilizing regression techniques to summarize and represent relationships within the data can significantly reduce the data volume. Instead of working with raw data points, you can analyze the coefficients and predictions from regression models, effectively condensing the information.
    - **Data Aggregation**: This technique involves summarizing data by combining multiple records into single data points. For example, instead of storing individual sales transactions, you could store total sales by month or category.
* **Data Compression**:
  + **Goal**: To reduce the size of data files to save storage space and improve transmission speeds.
  + **Methods**:
    - **Lossless Compression**: This method compresses data without any loss of information, allowing the original data to be perfectly reconstructed. Common lossless compression algorithms include ZIP and PNG for images. This is crucial when data integrity is paramount.
    - **Lossy Compression**: In contrast, lossy compression reduces file size by removing less critical information, which can result in a loss of quality. This method is commonly used for media files, such as JPEG for images and MP3 for audio, where a slight loss of quality is acceptable in exchange for significantly reduced file sizes.

## 4.4.7 Data Preprocessing Techniques - Data Transformation

* The primary goal of Data Transformation is to convert data into a format that is suitable for efficient machine learning (ML) modeling and insightful analysis. Transforming data helps enhance its quality and makes it more compatible with various algorithms, ultimately improving the performance of predictive models.

**Methods for Data Transformation**

* **Smoothing**:
  + **Purpose**: To **remove noise** from the data and identify underlying patterns. Noise can obscure the true signals in the data, making it difficult to derive meaningful insights or train accurate models.
  + **Techniques**:
    - **Moving Averages**: A technique that smooths data by averaging a fixed number of preceding values. This method is often used in time series analysis to identify trends over time.
    - **Exponential Smoothing**: A more sophisticated smoothing technique that applies decreasing weights to older observations, making recent data more influential in the analysis.
    - **Kernel Smoothing**: This technique uses a kernel function to weigh nearby data points differently, allowing for flexible smoothing that can adapt to the underlying data structure.
* **Aggregation**:
  + **Purpose**: To **summarize large volumes of data** into more manageable and insightful formats. Aggregation helps in reducing data complexity and facilitating analysis by focusing on key metrics.
  + **Techniques**:
    - **Group By Operations**: Summarizing data by grouping it based on one or more keys (e.g., summing sales data by product category or calculating averages by region).
    - **Statistical Aggregation**: Calculating metrics such as mean, median, mode, and standard deviation to summarize data distributions. This helps in understanding central tendencies and variability within datasets.
* **Discretization**:
  + **Purpose**: To convert **continuous variables** into discrete intervals or bins. This process can help in simplifying the analysis and making data more interpretable, especially for algorithms that perform better with categorical inputs.
  + **Techniques**:
    - **Equal Width Binning**: Dividing the range of continuous data into intervals of equal size. This method is straightforward but may lead to bins with varying numbers of observations.
    - **Equal Frequency Binning**: Creating bins so that each bin contains approximately the same number of data points. This method helps in maintaining a balanced representation across intervals.
    - **Clustering-based Discretization**: Using clustering algorithms to determine the best boundaries for bins based on the distribution of data points.
* **Normalization**:
  + **Purpose**: To adjust numeric variables to a specified range, typically between 0 and 1 or -1 and 1. Normalization is crucial when features have different scales, as many machine learning algorithms assume data is centered around zero or has comparable ranges.

# References:

<https://www.javatpoint.com/python-features>

<https://www.geeksforgeeks.org/python-language-advantages-applications/>

<https://docs.anaconda.com/anaconda/install/index.html>

<https://www.w3schools.com/python/python_datatypes.asp>

<https://realpython.com/python-conditional-statements/>

<https://www.w3schools.com/python/python_classes.asp>

<https://www.geeksforgeeks.org/inheritance-in-python/>

<https://pandas.pydata.org/docs/getting_started/intro_tutorials/index.html>

<https://numpy.org/doc/stable/user/quickstart.html>

<https://towardsdatascience.com/data-cleaning-with-pandas-bd8e8b6a7c9a>



Disclaimer: The content is curated from online/offline resources and used for educational purpose only