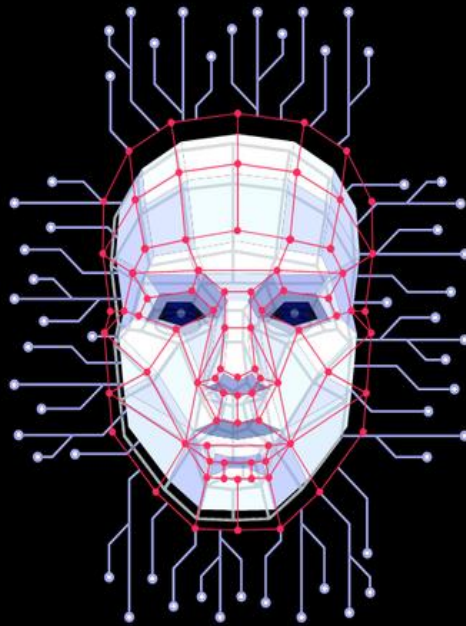# PYTHON MACHINE LEARNING

A Beginner's Guide to Scikit-Learn

RAJENDER KUMAR

PYTHON MACHINE LEARNING: A BEGINNER'S GUIDE TO SCIKIT-LEARN

CONTENTS

8 The Power of Combining: Ensemble Learning Methods

01

1 INTRODUCTION TO MACHINE LEARNING

———

Machine learning is a rapidly growing field that involves the use of algorithms and statistical models to analyze and make predictions or decisions based on data. This chapter will provide a background on the history and evolution of machine learning, as well as an overview of its different types and applications. Additionally, this chapter will introduce Python and scikit-learn, the popular machine learning library that will be used throughout the book. The goal of this chapter is to give readers a strong foundation in machine learning concepts and terminology, as well as the tools and techniques used in the field. By the end of this chapter, readers will have a clear understanding of the importance and potential of machine learning, and be ready to begin exploring the various algorithms and techniques used in the field.

1.1 BACKGROUND ON MACHINE LEARNING

⸺

Machine learning is a subfield of artificial intelligence (AI) and is a powerful tool for solving complex problems in a variety of industries, including finance, healthcare, transportation, and more.

The history of machine learning can be traced back to the 1940s and 1950s, when researchers first began exploring the use of computers for problem-solving and decision-making. In the early days of machine learning, algorithms were primarily used for simple tasks, such as classification and clustering. However, as technology advanced and data became more readily available, machine learning began to evolve and expand into more complex applications.

One of the key milestones in the history of machine learning was the development of the perceptron algorithm in the 1950s. The perceptron was the first algorithm capable of learning from data and was used for simple pattern recognition tasks. This was followed by the development of other algorithms, such as decision trees and artificial neural networks, which further expanded the capabilities of machine learning.

In the 1980s and 1990s, machine learning began to gain more widespread acceptance, with the development of more sophisticated algorithms and the increasing availability of data. The introduction of big data, powerful computing resources and more advanced algorithms such as Random Forest, Gradient Boosting Machine and Support Vector Machine (SVM) has led to the current state of machine learning where it is applied in a wide range of industries to solve complex problems.

~ 1940-1950s	Simple task: Classification, Clustering etc.
~ 1950s	Perception Algorithm
~ 1980-1990s	More Sophisticated algorithms: Decision Tree, Artificial Neural Networks
	Random Forest, Gradient Boosting Machine, Support Vector Machine
	Deep Learning & other complex algorithms
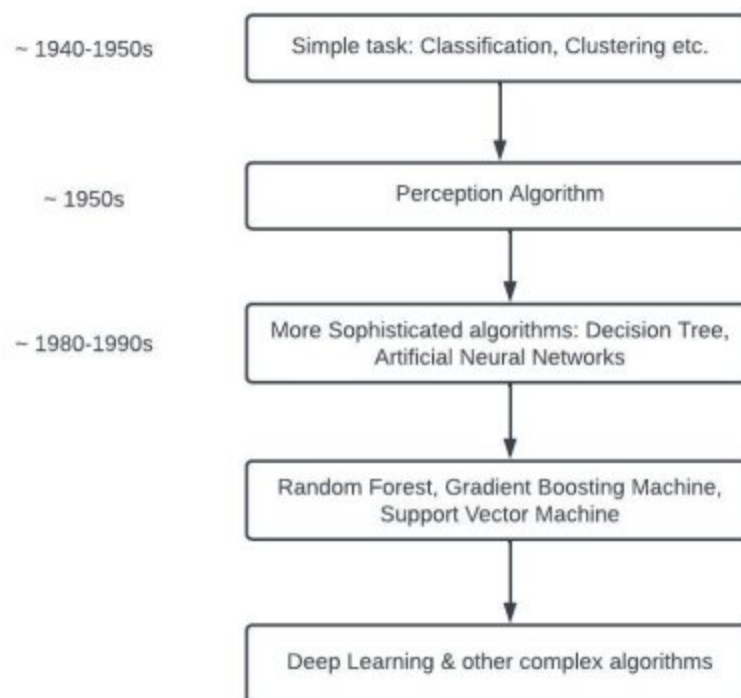
THERE ARE SEVERAL DIFFERENT types of machine learning, including supervised learning, unsupervised learning, and deep learning. Supervised learning involves using labeled

data to train a model, which can then be used to make predictions on new, unseen data. Unsupervised learning, on the other hand, involves using unlabeled data to identify patterns or structures in the data. Deep learning, a subset of machine learning, uses artificial neural networks to analyze large amounts of data and make predictions or decisions.

Machine learning is a powerful tool for solving complex problems, but it is not without its limitations. One of the main challenges of machine learning is dealing with large amounts of data, which can be difficult to process and analyze. Additionally, machine learning models can be prone to overfitting and underfitting, which can lead to inaccurate predictions or decisions. Despite these limitations, machine learning has the potential to revolutionize a wide range of industries and has already been used to solve problems that were once thought to be impossible.

In conclusion, Machine Learning is a field of computer science that uses statistical models and algorithms to analyze and make predictions or decisions based on data. Its history can be traced back to the 1940s and 1950s, and has evolved over time with the development of more sophisticated algorithms and the increasing availability of data. With the power of big data, powerful computing resources and more advanced algorithms, machine learning has become a powerful tool for solving complex problems in a wide range of industries.

1.2 WHY PYTHON FOR MACHINE LEARNING

Python is a high-level programming language that is widely used in the field of machine learning. It is an open-source language, which means it is free to use and can be modified by anyone. Python's popularity has grown rapidly in recent years, due to its ease of use, readability, and versatility.

Python is one of the most popular programming languages for machine learning, and for good reason. It offers a wide range of powerful libraries and frameworks that make it easy to implement machine learning algorithms and preprocess data. In this section, we will discuss some of the reasons why Python is the go-to language for machine learning.

Ease of Use

ONE OF THE MAIN REASONS why Python is popular for machine learning is its ease of use. The language has a simple, easy-to-read syntax that makes it easy to write and understand code. Additionally, Python has a large and active community, which means that there are many resources available to help with any problems or questions that may arise.

Powerful Libraries and Frameworks

PYTHON HAS A WIDE RANGE of powerful libraries and frameworks that make it easy to implement machine learning algorithms and preprocess data. Some of the most popular libraries and frameworks include:

- **Scikit-learn**: A popular library for machine learning that provides a wide range of algorithms, including linear regression, decision trees, and k-means clustering.
- **TensorFlow**: A powerful library for deep learning that makes it easy to build, train, and deploy neural networks.
- **Keras:** A high-level library for deep learning that can be used with TensorFlow and other backends.
- **Pandas**: A library for data manipulation and analysis that makes it easy to work with structured data.
- **NumPy**: A library for numerical computation that provides support for large, multi-dimensional arrays and matrices.

These libraries and frameworks make it easy to implement machine learning algorithms and preprocess data, which means that developers can focus on the problem they are trying to solve, rather than the details of the implementation.

Support for Machine Learning

PYTHON HAS A LARGE and active community that is dedicated to machine learning. This means that there are many resources available to help with any problems or

questions that may arise. Additionally, there are many tutorials, books, and online courses that can help developers learn how to use Python for machine learning.

Support for Big Data

PYTHON ALSO HAS A WIDE range of libraries and frameworks that make it easy to work with big data, such as PySpark, Dask, and Pandas. These libraries make it easy to work with large datasets, which is important for machine learning, as the more data that is available, the better the model can perform.

Python's versatility is also one of its key advantages. It can be used for a wide range of tasks, including web development, data analysis, and scientific computing. Additionally, Python has strong support for data visualization, which is important for understanding and interpreting machine learning models.

Despite its many advantages, Python is not without its limitations. One of the main disadvantages is that it can be slow for certain tasks, such as image processing and other computationally intensive tasks. Additionally, Python is a dynamically typed language, which can make it more difficult to debug and optimize code.

In conclusion, Python is a popular choice for machine learning because it is easy to use, has a wide range of powerful

libraries and frameworks, has a large and active community, and supports big data. Additionally, it has a simple and easy-to-read syntax, which makes it easy to write and understand code, which is important in machine learning because it makes it easier to experiment with different models and algorithms. It also has a wide range of resources available which makes it easy for developers to learn the language and its machine learning libraries.

1.3 OVERVIEW OF SCIKIT-LEARN

Scikit-learn is a popular machine learning library for Python. It is an open-source library, which means it is free to use and can be modified by anyone. Scikit-learn provides a wide range of tools and algorithms for building and training machine learning models, making it a powerful and versatile library. Here in the below section, we will provide some of the advantages and limitation of scikit-learn:

Advantage of scikit-learn

HERE ARE SOME OF THE advantages of using scikit-learn:

1. **Easy to use**: Scikit-learn is designed to be easy to use, even for beginners who are just starting out in machine learning. It provides a simple and consistent interface for building and evaluating models, which can help save time and reduce errors.

2. **Comprehensive**: Scikit-learn provides a comprehensive set of tools for data preprocessing, feature selection, model selection, and evaluation. It includes a wide range of machine learning algorithms, including linear regression, logistic regression, decision trees, random forests, support vector machines, and neural networks, among others.

3. **Open source**: Scikit-learn is an open-source library, which means that it is free to use and can be customized and modified to suit your specific needs. It is also constantly being updated and improved by a large community of developers and researchers.

4. **Fast and scalable**: Scikit-learn is designed to be fast and scalable, even for large datasets. It can run on multiple cores and supports distributed computing, which makes it suitable for use in production environments.

5. **Interoperable**: Scikit-learn is interoperable with other libraries and tools, including NumPy, Pandas, and Matplotlib, which makes it easy to integrate into your existing data analysis and machine learning workflows.

6. **Extensible**: Scikit-learn is highly extensible, which means that you can add your own custom models, algorithms, and metrics to the library. This allows you to tailor the library to your specific needs and use cases.

7. **Well documented**: Scikit-learn is well documented and provides a wide range of examples and tutorials, which can help you get up and running quickly. It also has an active community forum where you can ask questions and get help from other users.

Overall, scikit-learn is a powerful and versatile machine learning library that provides a range of tools for data analysis and predictive modeling. Its ease of use, comprehensive set of tools, and open-source nature make it a

popular choice for machine learning practitioners and researchers.

Limitation of scikit-learn

WHILE IT HAS MANY STRENGTHS and is widely used in the data science community, there are also some limitations to consider.

1. **Limited Deep Learning Capabilities**: Scikit-learn is not designed for deep learning, which is an area of machine learning that focuses on neural networks with many layers. While the library has some neural network functionality, it is not as extensive as other deep learning frameworks like TensorFlow or PyTorch.

2. **Lack of Support for Streaming Data**: Scikit-learn is primarily designed for batch learning, which means that it works well with datasets that can fit into memory. It does not provide support for streaming data, which is a scenario where data is continuously generated and must be processed in real-time.

3. **Limited Support for Unstructured Data**: Scikit-learn is designed for structured data, which means that it is not well-suited to dealing with unstructured data like text or images. While there are some tools for text analysis in the library, they are not as extensive as those available in specialized NLP (Natural Language Processing) libraries.

4. **Limited Parallelism**: While scikit-learn does support parallel processing, it is not optimized for distributed computing. This means that it can be slow to process very large datasets or to run complex models on large clusters.

5. **Limited AutoML Capabilities**: Scikit-learn does not have built-in tools for automating machine learning workflows, such as hyperparameter tuning or feature engineering. While some third-party libraries like TPOT provide this functionality, it is not as integrated as in other AutoML platforms.

Despite these limitations, scikit-learn is still a powerful tool for many machine learning tasks and is widely used in the data science community. It is important to understand its strengths and weaknesses when choosing a machine learning library for a given project.

1.4 SETTING UP THE DEVELOPMENT ENVIRONMENT

Adevelopment environment is a set of tools and software that allow you to write, test, and debug code. In this section, we will discuss how to set up a development environment for working with Python and machine learning.

Installing Python

THE FIRST STEP IN SETTING up a development environment is to install Python. Python can be downloaded from the official website (python.org) and can be installed on Windows, macOS, or Linux.

In this section, we will discuss the step-by-step process for installing Python on each operating system, along with screenshots.

For Windows:

1. Go to the official Python website (https://www.python.org/).
2. Click on the "Downloads" button.
3. Click on the "Windows" button.
4. Click on the "Latest Python 3 Release" button. For example, it is "Python 3.11.12" at this moment.

5. Click on the "Windows x86-64 executable installer" button to download the installer.
6. Once the download is complete, double-click on the installer file to begin the installation process.
7. On the first screen of the installer, click on the "Install Now" button.

Python 3.11.2 (64-bit) Setup

Install Python 3.11.2 (64-bit)

Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

→ Install Now
C:\Users\rajen\AppData\Local\Programs\Python\Python311

Includes IDLE, pip and documentation
Creates shortcuts and file associations

→ Customize installation
Choose location and features

☑ Use admin privileges when installing py.exe
☐ Add python.exe to PATH

Cancel

1. On the next screen, you will be asked to confirm the installation location. Leave the default location selected and click on the "Next" button.
2. On the next screen, you will be asked if you want to add Python to your system's PATH. It is recommended to select "Add Python 3.x to PATH" and click on the "Install" button.
3. Once the installation is complete, click on the "Close" button.
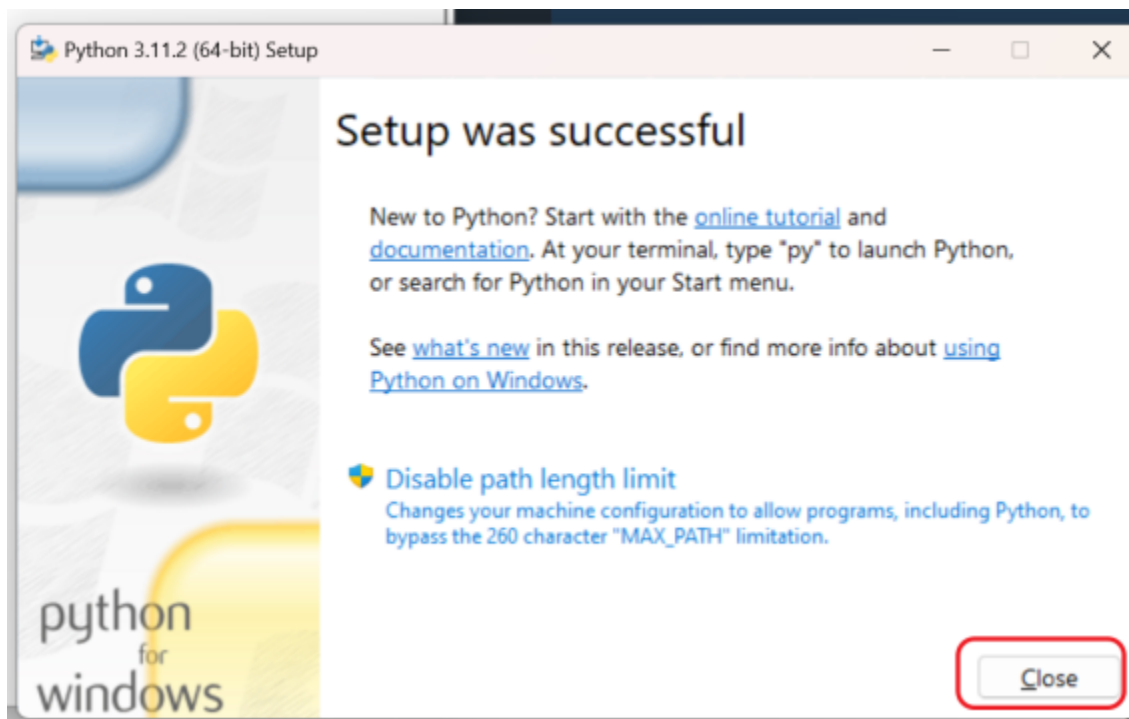
For MacOS:

1. Go to the official Python website (https://www.python.org/
2. Click on the "Downloads" button.
3. Click on the "MacOS" button.

4. Click on the "Latest Python 3 Release" button.
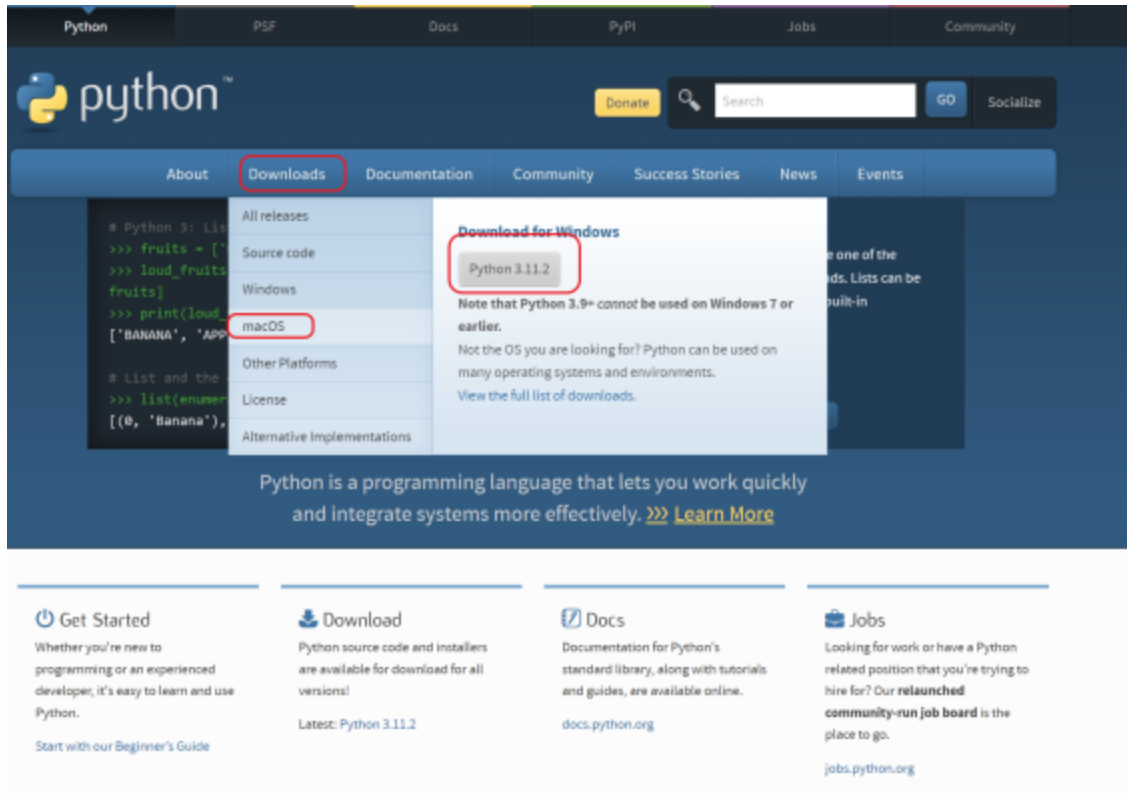
1. Click on the "macOS 64-bit installer" button to download the installer.
2. Once the download is complete, double-click on the installer file to begin the installation process.
3. On the first screen of the installer, click on the "Continue" button.
4. On the next screen, you will be asked to confirm the installation location. Leave the default location selected and click on the "Install" button.
5. Once the installation is complete, click on the "Close" button.

For Linux:

1. Open a terminal window.
2. Run the command "*sudo apt-get update*" to update your system's package list.
3. Run the command "*sudo apt-get install python3*" to install Python.
4. To check if Python is installed correctly, run the command "*python3 -V*". This should display the version of Python that is currently installed on your system.

It is important to note that the version of Python you have installed is important because some libraries and frameworks may only be compatible with specific versions of Python. It is recommended to install latest stable version of Python available.

Once Python is installed, it is recommended to install a package manager such as pip, which will make it easier to install and manage libraries and frameworks.

Installing Libraries & Frameworks

THE NEXT STEP IS TO install the libraries and frameworks that will be used in the development environment. The most popular libraries and frameworks for machine learning include NumPy, SciPy, and scikit-learn. These libraries can be installed using pip by running the command "*pip install numpy scipy scikit-learn*" on the command line.

In this section, we will discuss the step-by-step process for installing the most popular libraries and frameworks for machine learning, such as NumPy, SciPy, and scikit-learn.

The first step in installing libraries and frameworks is to make sure that Python is installed on your system. Once Python is installed, you can use the package manager pip to install libraries and frameworks.

The process of installing libraries and frameworks using pip is as follows:

1. Open a terminal or command prompt window
2. Run the command "*pip install numpy*" to install the NumPy library
3. Run the command "*pip install scipy*" to install the SciPy library
4. Run the command "*pip install scikit-learn*" to install the scikit-learn library

It is important to note that the above commands will install the latest version of the libraries and frameworks. If you need to install a specific version, you can use the command "*pip install numpy==x.x.x*" (where x.x.x is the version number)

Another way to install these libraries and frameworks is by using the Anaconda distribution which comes with a lot of useful libraries and frameworks pre-installed, and it also has

a built-in package manager called Conda. You can install these libraries by running the following commands in the Anaconda prompt:

1. Open the Anaconda prompt
2. Run the command "*conda install numpy*" to install the NumPy library
3. Run the command "*conda install scipy*" to install the SciPy library
4. Run the command "*conda install scikit-learn*" to install the scikit-learn library

Once the libraries and frameworks are installed, you can import them in your Python script and start using them for building and training machine learning models.

It's important to note that, even though the process of installing libraries and frameworks is simple, it's also important to keep them updated. This is because new versions of libraries and frameworks may have bug fixes, performance improvements, or new features. You can update the libraries and frameworks by running the following commands:

1. Open a terminal or command prompt window
2. Run the command "*pip install—upgrade numpy*" to update the NumPy library

3. Run the command "*pip install—upgrade scipy*" to update the SciPy library
4. Run the command "*pip install—upgrade scikit-learn*" to update the scikit-learn library

It is also recommended to install a code editor or integrated development environment (IDE) such as Jupyter Notebook, Spyder, or PyCharm. These tools provide a user-friendly interface for writing, testing, and debugging code. They also provide features such as code completion and debugging tools that can make the development process more efficient.

To further enhance the development process, it is also recommended to have a version control system (VCS) such as Git, which allows you to keep track of changes to your code and collaborate with other developers.

In addition to these tools, it is also a good idea to have access to a large dataset that can be used to train and test machine learning models. There are many publicly available datasets that can be used for machine learning, such as the UCI Machine Learning Repository, which provides a wide range of datasets for classification, regression, and clustering tasks. You can learn more about free data available online in Appendix D at the end of book.

Setting up a development environment is an important step in working with machine learning using Python. A

development environment includes tools such as Python, pip, libraries and frameworks, a code editor or IDE, version control system and datasets. By having all these tools in place, it will make the development process more efficient and streamlined.

1.5 UNDERSTANDING THE DATASET

Understanding the dataset is a crucial step in the machine learning process. It involves gaining a deep understanding of the data that will be used to train and test a model, including its structure, quality, and characteristics. This understanding is essential for selecting the appropriate model, developing a robust algorithm, and interpreting the results. In this section, we will discuss the importance of understanding the dataset, and the key considerations when working with a dataset.

The Importance of Understanding the Dataset

BEFORE A MODEL CAN be trained and tested, the dataset must be understood. This is because the dataset is the foundation upon which the model will be built, and a poor understanding of the dataset can lead to poor model performance. For example, if the dataset is not representative of the problem or is biased, the model will not perform well. Similarly, if the dataset is too small or has missing data, the model will be under-trained and will not generalize well to new data.

Understanding the dataset also helps to identify potential issues such as outliers, missing data, and duplicate records, which can affect the performance of the model. By identifying

these issues early, they can be addressed before the model is trained, resulting in a more robust model.

Key Considerations when Working with a Dataset

WHEN WORKING WITH A dataset, there are several key considerations to keep in mind:

- **Representativeness**: The dataset should be representative of the problem being solved, otherwise the model may not perform well on new data.
- **Size**: The size of the dataset will affect the performance of the model. A larger dataset can result in a more robust model, but it can also increase the computational resources required to train and test the model.
- **Quality**: The quality of the data can affect the performance of the model. Missing data, outliers, and duplicate records can all affect the performance of the model.
- **Features**: The features of the data should be carefully selected, as they will be used to train and test the model. The features should be relevant to the problem and should not include redundant information.
- **Preprocessing**: The data may need to be preprocessed before it can be used to train and test the model. This can include cleaning, normalizing, and transforming the data.

In conclusion, understanding the dataset is a crucial step in the machine learning process. It involves gaining a deep understanding of the data, including its structure, quality, and characteristics. This understanding is essential for selecting the appropriate model, developing a robust algorithm, and interpreting the results. By understanding the types of data, potential issues, and key considerations when working with a dataset, machine learning practitioners can ensure that their models are robust, accurate, and generalize well to new data.

1.6 TYPE OF DATA

When working with datasets in machine learning, it is important to understand the different types of data that can be encountered. Data can be broadly classified into two categories: structured and unstructured. Understanding the different types of data and their characteristics can help in selecting the appropriate machine learning algorithms and preprocessing techniques.

Structured Data

STRUCTURED DATA IS data that is organized and can be easily understood by a computer. It can be represented in tabular form and can be easily stored in a relational database. Examples of structured data include numerical data (such as age, income) and categorical data (such as gender, occupation). Structured data can be further classified into two types: numerical and categorical.

Numerical Data: Numerical data is data that consists of numbers. It can be further divided into two types: **discrete and continuous**. Discrete data is data that can only take on certain values, such as the number of children in a family. Continuous data, on the other hand, can take on any value within a range, such as the temperature.

Categorical Data: Categorical data is data that consists of categories. Examples include gender, occupation, and country of origin. Categorical data can be further divided into two types: **ordinal and nominal**. Ordinal data is data that can be ordered, such as education level (high school, undergraduate, graduate). Nominal data is data that cannot be ordered, such as gender (male, female).
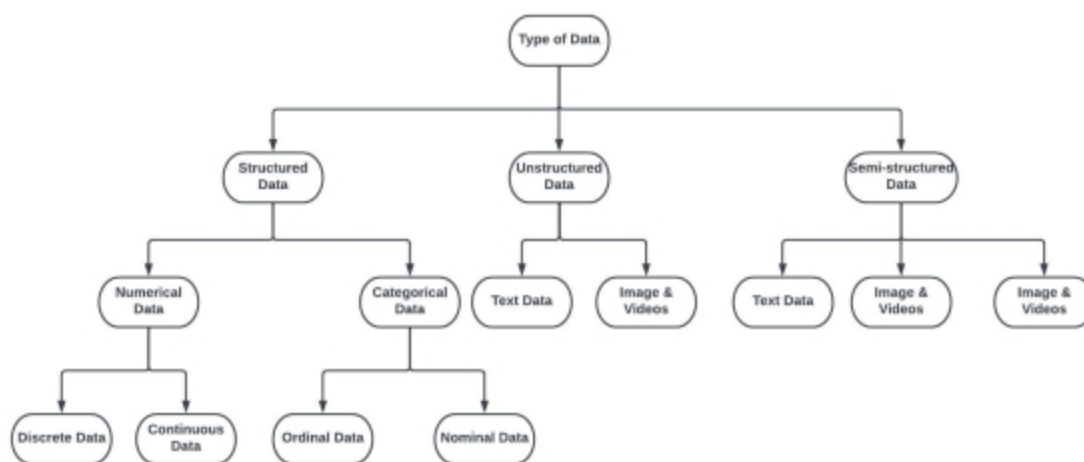
Unstructured Data

UNSTRUCTURED DATA IS data that is not organized and is difficult for a computer to understand. Examples include text, images, and audio. Unstructured data is often more challenging to work with, as it requires additional preprocessing steps to convert it into a format that can be understood by a computer. This can include text mining, image processing, and audio processing.

Text data is one of the most common types of unstructured data. It can be found in emails, social media posts, and customer reviews. Text data requires preprocessing techniques such as tokenization, stemming, and lemmatization to convert it into a format that can be understood by a machine learning model.

Images and videos are also common types of unstructured data. They require preprocessing techniques such as image processing, object detection, and feature extraction to convert them into a format that can be understood by a machine learning model.

Semi-structured Data

SEMI-STRUCTURED DATA is a type of data that does not conform to a specific data model or schema. It contains elements of both structured and unstructured data. Semi-structured data is often represented in a format that can be easily processed by computers, such as JSON, XML, or CSV files, but it may not have a fixed structure or defined data types.

One of the main advantages of semi-structured data is its flexibility. Because the data does not have to conform to a specific schema, it can be easily modified or extended to accommodate new data elements. This makes it well-suited

for applications that deal with rapidly changing data or data that has a high degree of variability.

Semi-structured data is commonly used in applications such as web scraping, social media analysis, and machine learning. It is also often used as an intermediary format for converting data between different systems or applications.

However, working with semi-structured data can also have some drawbacks. Because the data is not well-defined, it can be more difficult to perform certain types of analysis, such as querying or joining data across multiple sources. It may also require more preprocessing and cleaning before it can be used in a particular application.

An **example** of working with different types of data in machine learning is building a model to predict the price of a house based on various features. Let's assume that the dataset contains the following features:

- Sale Price (Target variable)
- Number of Bedrooms (Numerical Data)
- Number of Bathrooms (Numerical Data)
- Size of the House in square feet (Numerical Data)
- Type of the House (Categorical Data, Nominal)
- Year Built (Numerical Data)
- Zipcode (Categorical Data, Nominal)

In this example, the target variable is the Sale Price, which is a numerical data as it is represented by a number, and we want to predict this variable based on other variables.

The first feature is the number of bedrooms, which is numerical data as it is represented by a number. This feature can be used as is in the model, and it can be useful in predicting the Sale Price as the number of bedrooms can affect the overall size of the house and the price.

The second feature is the number of bathrooms, which is also numerical data as it is represented by a number. This feature can also be used as is in the model, and it can be useful in predicting the Sale Price as the number of bathrooms can affect the overall amenities of the house and the price.

The third feature is the size of the house in square feet, which is numerical data as it is represented by a number. This feature can also be used as is in the model, and it can be useful in predicting the Sale Price as the size of the house can affect the overall space of the house and the price.

The fourth feature is the type of the house, which is categorical data, nominal type, as it consists of categories such as "Single Family", "Townhouse", "Apartment" etc. This feature can not be used as is in the model, as the model will not be able to understand the categorical data. So, we will use one-hot encoding to convert this feature into numerical

data. One-hot encoding creates a new binary column for each category and assigns a value of 1 or 0 depending on whether the category is present in the original data or not.

The fifth feature is the year built, which is numerical data as it is represented by a number. This feature can also be used as is in the model, and it can be useful in predicting the Sale Price as the age of the house can affect the overall condition of the house and the price.

The sixth feature is the zipcode, which is also categorical data, nominal type, as it consists of categories such as "90210", "10001" etc. Similar to the type of the house, this feature also needs to be transformed using one-hot encoding.

Once the data is preprocessed, it can be used to train and test a machine learning model, such as a linear regression or a decision tree, to predict the Sale Price based on the other features. By understanding the types of data and their characteristics, we were able to preprocess the data and feed it into a model for further analysis.

In this example, we have covered three types of data: numerical, categorical (nominal), and unstructured. Understanding the types of data and preprocessing them accordingly is an important step in the machine learning process, as it ensures that the data is in a format that can be

understood by a machine learning model, and that it accurately represents the problem we are trying to solve.

Understanding the different types of data and their characteristics is an important step in the machine learning process. It can help in selecting the appropriate machine learning algorithms and preprocessing techniques, and in understanding the potential issues and challenges that can be encountered when working with a dataset. Structured data is easier to work with as it is organized and can be easily understood by a computer. Unstructured data is more challenging as it requires additional preprocessing steps to convert it into a format that can be understood by a machine learning model.

1.7 TYPES OF MACHINE LEARNING MODELS

Machine learning models are algorithms that are used to make predictions or take decisions based on data. There are several types of machine learning models, each with their own strengths and weaknesses. Understanding the different types of models is crucial for selecting the right model for a specific problem and for interpreting the results of a model. The main types of machine learning models are:

1. **Supervised learning**: Supervised learning models are used to make predictions based on labeled data. The model is trained on a labeled dataset, where the output variable is known. Once the model is trained, it can be used to make predictions on new, unseen data. Examples of supervised learning models include linear regression, logistic regression, and decision trees.

2. **Unsupervised learning**: Unsupervised learning models are used to find patterns or structure in unlabeled data. The model is not given any labeled data, and instead must find patterns and structure on its own. Examples of unsupervised learning models include k-means clustering, hierarchical clustering, and principal component analysis.

3. **Semi-supervised learning**: Semi-supervised learning models are a combination of supervised and unsupervised learning. The model is given some labeled data, but not enough to fully train the model. The model must use the labeled data and the structure of the unlabeled data to make predictions. Examples of semi-supervised learning models include self-training and co-training.

4. **Reinforcement learning**: Reinforcement learning models are used to make decisions in an environment where the model is given feedback in the form of rewards or penalties. The model learns to make decisions by maximizing the rewards over time. Examples of reinforcement learning models include Q-learning and SARSA.

5. **Deep Learning**: Deep learning models are a subfield of machine learning that is inspired by the structure and function of the human brain. These models are composed of multiple layers of interconnected nodes, called artificial neurons, which can learn and represent highly complex patterns in data. Examples of deep learning models include convolutional neural networks, recurrent neural networks, and deep belief networks.

In conclusion, understanding the different types of machine learning models is crucial for selecting the right model for a specific problem and for interpreting the results of a model.

Each model has its own strengths and weaknesses, so it is important to evaluate the suitability of each model for a specific task.

We will discuss different models in the next few chapters in detail.

1.8 SUMMARY

- Introduction to Machine Learning and why Python is a popular choice for ML
- Setting up the environment by installing Python and required libraries, including a tutorial on Jupyter Notebook
- Understanding Python data structures and the importance of understanding the dataset before starting the ML process
- Introduction to Scikit-learn, a popular machine learning library in Python
- Understanding the API of scikit-learn and how it can be used to train and test models.

1.9 TEST YOUR KNOWLEDGE

I. What is the main goal of machine learning?

a. To understand and analyze data

b. To make predictions or decisions

c. To automate tasks

d. All of the above

I. What is the main advantage of using Python for machine learning?

a. It has a simple, easy-to-read syntax

b. It has a wide range of powerful libraries and frameworks

c. It has a large and active community

d. All of the above

I. What is Jupyter Notebook?

a. A library for data manipulation and analysis

b. A web-based interactive development environment

c. A machine learning algorithm

d. A database management system

I. What is the difference between structured and unstructured data?

a. Structured data is organized and can be easily understood by a computer, while unstructured data is not organized and is difficult for a computer to understand
b. Structured data is numerical, while unstructured data is categorical
c. Structured data is easily stored in a relational database, while unstructured data is not
d. All of the above

I. **What is Exploratory Data Analysis (EDA)?**

a. A machine learning algorithm
b. A technique used to understand and analyze data
c. A preprocessing step for unstructured data
d. A data visualization library

I. **What is one-hot encoding?**

a. A technique to convert categorical data into numerical data
b. A technique to convert numerical data into categorical data
c. A technique to remove outliers from the data
d. A technique to normalize the data

I. **What is the main benefit of preprocessing the data?**

a. It makes the data easier to understand

b. It makes the data more representative of the problem

c. It improves the performance of the machine learning model

d. All of the above

I. What is the main disadvantage of using unstructured data?

a. It is not organized and is difficult for a computer to understand

b. It requires additional preprocessing steps to convert it into a format that can be understood by a machine learning model

c. It is not as easily stored in a relational database

d. All of the above

I. What is the main benefit of using scikit-learn?

a. It provides a wide range of machine learning algorithms

b. It is easy to use and understand

c. It has a large and active community

d. All of the above

I. What is the main importance of evaluating model performance?

a. It helps to determine the accuracy of the model

b. It helps to identify areas for improvement

c. It helps to compare the performance of different models

d. All of the above

1.10 ANSWERS

I. Answer:
   b)   To make predictions or decisions

I. Answer:
   d)   All of the above

I. Answer:
   b)   A web-based interactive development environment

I. Answer: Structured data is organized and can be easily understood by a
   a)   computer, while unstructured data is not organized and is difficult
         for a computer to understand

I. Answer:
   b)   A technique used to understand and analyze data

I. Answer:
   a)   A technique to convert categorical data into numerical data

I. Answer:
   d)   All of the above

I. Answer: All of the above
   d)

I. Answer:
   d)   All of the above

I. Answer:
   d)   All of the above

02

2 PYTHON: A BEGINNER'S OVERVIEW

Welcome to the chapter on Python: A Beginner's Overview. This chapter is designed to give you a solid foundation in the Python programming language. Python is a versatile and widely-used programming language that is perfect for beginners and experts alike. It is known for its simplicity, readability, and ease of use, making it an ideal choice for a wide range of applications. In this chapter, we will cover the basics of Python, including its uses, and basic syntax. We will also delve into more advanced topics such as control flow, functions, and working with data. By the end of this chapter, you will have a good understanding of the basics of Python programming and be well-equipped to continue learning and exploring the language.

2.1 PYTHON BASICS

Python is a powerful and versatile programming language that is widely used in a variety of industries, from web development to data science. It is known for its simple syntax, easy readability, and vast ecosystem of libraries and frameworks. If you're new to programming or are looking to learn Python, this section will cover the basics of the language, including its syntax, comments, and variables.

Syntax of Python

THE SYNTAX OF PYTHON is designed to be easy to read and understand, with a focus on readability and simplicity. Python uses **indentation** to indicate code blocks, rather than curly braces or keywords like "begin" and "end." This makes the code more organized and easy to read. Python also uses a simple, consistent syntax for basic operations like assignment and comparison. For example, the assignment operator is the single equal sign (=), and the comparison operator is the double equal sign (==).

Comments in Python

COMMENTS IN PYTHON are used to add notes and explanations to your code, making it easier to understand and maintain. They are ignored by the interpreter and do not affect the execution of the program.

In Python, comments start with a hash symbol (#) and continue until the end of the line. For example, the following line is a comment:

This is a comment

YOU CAN ALSO PLACE comments at the end of a line of code, after the statement:

x = 5 # This is a comment

ADDITIONALLY, PYTHON supports multi-line comments, which are denoted by triple quotes (either single or double). For example, the following code demonstrates a multi-line comment:

"""

This is a

multi-line comment

"""

Multi-line comments are often used to add documentation to a module, class, or function, and are also known as **docstrings**.

IT IS CONSIDERED A best practice to include comments in your code, especially for complex or non-obvious sections of code. Comments should be clear, concise, and informative, and should be used to explain the purpose and function of the code.

It's also important to keep comments up-to-date, and delete or update them when the code changes. Comments that are no longer relevant or accurate can be confusing and misleading.

Indentation in Python

IN PYTHON, INDENTATION is used to indicate code blocks. This means that the amount of whitespace at the beginning of a line is used to determine the level of nesting for a block of code. For example, in the following code, the statements in the if block are indented four spaces to the right of the if statement:

```python
if x > 0:

print("x is positive")

x = x - 1
```

THIS INDENTATION IS important because it helps to make the code more organized and readable. It is also used to indicate the scope of loops, functions, and classes. The amount of

indentation is not fixed and can be any multiple of spaces or tabs, as long as it is consistent within the same block of code.

It is also important to note that Python raises **IndentationError** when there is an inconsistent use of whitespaces. This means that if you use different amounts of whitespaces for different code blocks, you'll get an error. For example, if you use 4 spaces for one block and 2 spaces for another, you'll get an error.

To avoid this, it is recommended to use spaces or tabs consistently throughout your code, and use an editor that automatically converts tabs to spaces. Most popular Python IDEs like PyCharm, VSCode, and Jupyter notebook have this feature enabled by default.

In summary, indentation is an important aspect of Python's syntax, as it is used to indicate code blocks and helps to make the code more organized and readable. It is important to use consistent indentation throughout your code to avoid errors and improve readability.

Variable in Python

IN PYTHON, VARIABLES are used to store and manipulate data in a program. They are used to give a name to a value, so that the value can be referred to by its name rather than its value. Variables are declared by assigning a value to a

variable name. For example, the following code declares a variable named "x" and assigns the value 5 to it:

```
x = 5
```

PYTHON IS A DYNAMICALLY-typed language, which means that the type of a variable is determined at runtime. This means that you don't have to specify the type of a variable when you declare it. For example, the following code assigns a string to a variable:

```
name = "John"
```

It's also important to note that Python variable names must start with a letter or an underscore and can only contain letters, numbers, and underscores. Additionally, Python has a number of reserved words that cannot be used as variable names. These reserved words include keywords such as "if", "else", "for", "in", "and", "or", etc.

PYTHON ALSO SUPPORTS multiple assignment, which allows you to assign values to multiple variables in a single line. For example, the following code assigns values to three variables in a single line:

```
x, y, z = 1, 2, 3
```

ADDITIONALLY, PYTHON also supports variable swapping, where the values of two variables can be swapped in a single line of code, without the need of a temporary variable. For example, the following code swaps the values of x and y:

```
x, y = y, x
```

IN CONCLUSION, PYTHON is a powerful and versatile programming language that is easy to learn and use. Its simple syntax, easy readability, and vast ecosystem of libraries and frameworks make it a popular choice for a wide range of applications. This section has covered the basics of the language, including its syntax, comments, and variables. With a solid understanding of these concepts, you'll be well on your way to becoming a proficient Python programmer.

2.2 DATA TYPES IN PYTHON

In Python, data types are used to define the type of a variable or a value. Different data types have different properties and behaviors, and they are used to store different types of data. The most commonly used data types in Python are:

1. **Numbers**: Python has various types of numerical data types, such as int (integer), float (floating point number), and complex. Integers are whole numbers, positive or negative, and they do not have decimal points. For example:

x = 5 # int

y = -10 # int

Floating point numbers are numbers that have decimal points and they are used for representing real numbers. For example:

y = 3.14 # float

z = -0.5 # float

Complex numbers are numbers that consist of a real and an imaginary part. They are written in the form of a+bj, where a and b are real numbers and j is the imaginary unit. For example:

```
z = 3 + 4j  # complex
```

1. **Strings**: A string is a sequence of characters. They can be declared using single quotes ('') or double quotes (""). Strings are used to represent text and they are immutable, meaning they cannot be modified after they are created. For example:

```
name = "John"  # string
```

1. **Lists**: Lists are ordered sequences of items, which can be of any data type. They are enclosed in square brackets and the items are separated by commas. Lists are mutable, meaning they can be modified after they are created. For example:

```
fruits = ["apple", "banana", "orange"]  # list
```

1. **Tuples**: Tuples are similar to lists but they are immutable, meaning they cannot be modified after they are created. They are also enclosed in parentheses and the items are separated by commas. For example:

```
coordinates = (3, 4)  # tuple
```

1. **Dictionaries**: Dictionaries are used to store key-value pairs. They are enclosed in curly braces {} and the items are separated by commas. The keys must be unique and immutable. Dictionaries are also mutable, meaning they can be modified after they are created. For example:

```
person = {"name": "John", "age": 30}  # dictionary
```

1. **Booleans**: Booleans represent true or false values. They can be either True or False. They are mostly used in conditional statements and loops. For example:

```
is_valid = True # Boolean
```

It's important to note that in Python, the type of a variable or a value can be determined using the built-in function type(). For example:

```
x = 5
```

```
print(type(x))  # <class 'int'>
```

In conclusion, data types are an important aspect of Python programming as they define the type of a variable or a value, and they have different properties and behaviors. The most commonly used data types in Python are numbers (int, float, complex), strings, lists, tuples, dictionaries, and booleans. Choosing the appropriate data type for the task at hand is

crucial, as it affects how the data can be used and manipulated in your program. Additionally, knowing the mutability of data types helps to make efficient use of memory and prevent unexpected behavior in your code.

2.3 CONTROL FLOW IN PYTHON

Control flow refers to the order in which statements in a program are executed. In Python, control flow is achieved through the use of statements such as if-else, for loops, and while loops. These statements allow you to control the flow of execution in your program and make decisions based on certain conditions.

The if-else statement is used to make decisions in your code. It allows you to execute a block of code only if a certain condition is true. The syntax for an if-else statement is as follows:

if condition**:**

execute this code block if condition is true

else:

execute this code block if condition is false

FOR EXAMPLE, THE FOLLOWING code checks if a variable x is greater than 5, and if it is, it prints "x is greater than 5".

x = 7

if x > 5**:**

print("x is greater than 5"**)**

else:

print("x is not greater than 5"**)**
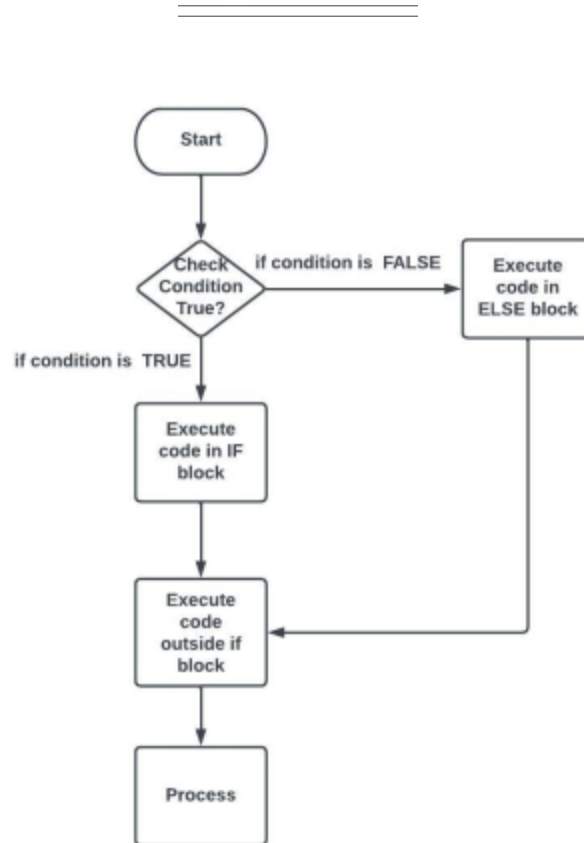
ANOTHER TYPE OF CONTROL flow structure is the **for loop**. The for loop is used to iterate through a sequence of items, such as a list, tuple, or string. The syntax for a for loop is as follows:

for variable **in** sequence**:**

execute this code block for each item in the sequence

FOR EXAMPLE, THE FOLLOWING code iterates through a list of numbers, and prints each number:

numbers **= [**1**,** 2**,** 3**,** 4**,** 5**]**

for number **in** numbers**:**

print(number**)**

For Loop
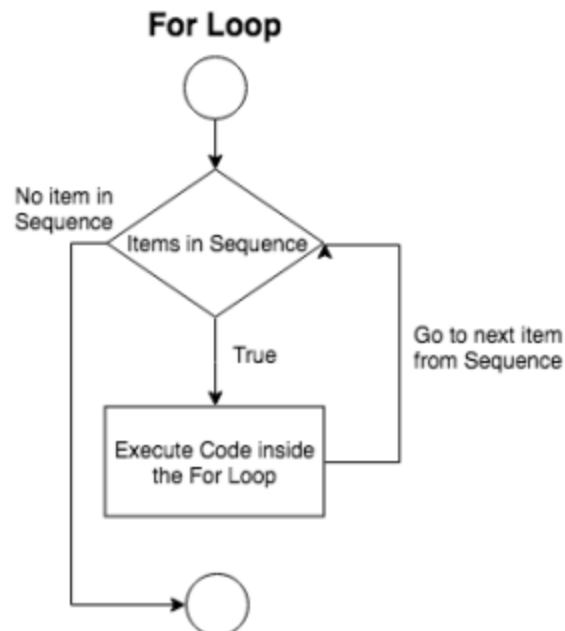
THE **while loop** is another type of control flow structure. It is used to execute a block of code repeatedly as long as a certain condition is true. The syntax for a while loop is as follows:

while condition**:**

execute this code block while the condition is true

FOR EXAMPLE, THE FOLLOWING code uses a while loop to print the numbers from 1 to 5:

```
i = 1

while i <= 5:

print(i)

i += 1
```

PYTHON ALSO PROVIDES the **break** and **continue** statements for more fine-grained control over the flow of execution within loops. The **break** statement allows you to exit a loop early, while the **continue** statement allows you to skip the current iteration and move on to the next one.

For example, the following code uses a for loop to iterate through a list of numbers, but it uses the **break** statement to exit the loop early if the number is equal to 3:

```
numbers = [1, 2, 3, 4, 5]

for number in numbers:

if number == 3:

break

print(number)
```

THE OUTPUT OF THIS code will be 1, 2, but not 3.

In conclusion, control flow is an important aspect of programming and Python provides a variety of tools to control the flow of execution in your program. The if-else statement, for loops, and while loops are used to make decisions and execute code repeatedly based on certain conditions. Additionally, the **break** and **continue** statements provide more fine-grained control over the flow of execution within loops. Understanding and using these control flow structures effectively will help you write more efficient and organized code.

2.4 FUNCTIO IN PYTHON

In Python, a function is a block of reusable code that performs a specific task. Functions are useful for organizing and structuring code, making it more readable and maintainable. They can also be reused across multiple parts of a program, reducing code duplication.

Functions are defined using the **def** keyword, followed by the function name, and a set of parentheses that may contain parameters. For example, the following code defines a simple function called **greet** that takes a single parameter **name**:

def greet**(**name**):**

print("Hello, " **+** name**)**

ONCE A FUNCTION IS defined, it can be called or invoked by using its name followed by parentheses. For example:

greet**(**"John"**)**

THIS WILL OUTPUT "HELLO, John".

Functions can also return a value using the **return** statement. For example, the following function takes two

parameters **a** and **b**, and returns their sum:

```python
def add(a, b):

return a + b

result = add(3, 4)

print(result)
```

THIS WILL OUTPUT 7.

It's also important to note that in Python, functions can have default values for their parameters. This means that if a value is not passed for a parameter when calling the function, the default value will be used instead. For example:

```python
def greet(name, message = "Hello"):

print(message + ", " + name)

greet("John")
```

IN THIS EXAMPLE, THE **message** parameter has a default value of "Hello", so if it's not passed when calling the function, the default value will be used.

Functions can also be used as arguments in other functions, a technique called Higher-Order functions. This makes the code

more flexible and allows the developer to write more generic and reusable functions.

In conclusion, functions are an important aspect of Python programming. They provide a way to organize and structure code, making it more readable and maintainable. Functions can also be reused across multiple parts of a program, reducing code duplication. They can also take parameters and return values.

2.5 ANONYMOUS (LAMBDA) FUNCTION

In Python, an anonymous function, also known as a lambda function, is a function without a name. They are defined using the **lambda** keyword, followed by a set of parameters, a colon, and a single expression. The expression is evaluated and returned when the function is called.

For example, the following code defines a lambda function that takes two parameters **a** and **b**, and returns their product:

```python
multiply = lambda a, b: a * b

result = multiply(3, 4)

print(result)
```

THIS WILL OUTPUT 12.

Lambda functions are useful when a small, simple function is needed, such as a callback function for a button click or a sorting key for a list. They can also be used as arguments in other functions, such as the **filter()** and **map()** functions.

For example, the following code uses the **filter()** function to filter a list of numbers, keeping only the even numbers:

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

even_numbers = filter(lambda x: x % 2 == 0, numbers)

print(list(even_numbers))
```

It's important to note that, unlike regular functions, lambda functions do not have a return statement. The value of the expression is returned by default. Also, lambda functions cannot contain statements, only expressions.

2.6 FUNCTION FOR LIST

Python provides a variety of built-in functions that can be used to manipulate lists. These functions make it easy to perform common operations on lists, such as sorting, filtering, and mapping.

The **len()** function is used to determine the length of a list, which is the number of elements it contains. For example:

fruits **= [**'apple'**,** 'banana'**,** 'orange'**]**

print(len(fruits**))**

THIS WILL OUTPUT 3.

The **sort()** function is used to sort the elements of a list in ascending order. It modifies the original list and does not return a new one. For example:

numbers **= [**3**,** 1**,** 4**,** 2**,** 5**]**

numbers**.**sort**()**

print(numbers**)**

THIS WILL OUTPUT [1, 2, 3, 4, 5].

The **sorted()** function is similar to the **sort()** function but it returns a new list rather than modifying the original one.

```
numbers = [3, 1, 4, 2, 5]

sorted_numbers = sorted(numbers)

print(sorted_numbers)
```

———————

THIS WILL OUTPUT [1, 2, 3, 4, 5].

The **filter()** function is used to filter the elements of a list based on a certain condition. It returns an iterator, which can be converted to a list. For example:

```
numbers = [1, 2, 3, 4, 5]

even_numbers = filter(lambda x: x % 2 == 0, numbers)

print(list(even_numbers))
```

———————

THIS WILL OUTPUT [2, 4].

The **map()** function is used to apply a certain operation to each element of a list. It returns an iterator, which can be converted to a list. For example:

```
numbers = [1, 2, 3, 4, 5]

squared_numbers = map(lambda x: x ** 2, numbers)

print(list(squared_numbers))
```

THIS WILL OUTPUT [1, 4, 9, 16, 25].

The **reduce()** function is used to reduce a list of elements to a single value by applying a certain operation. It is a part of the **functools** module. For example:

```
from functools import reduce

numbers = [1, 2, 3, 4, 5]

product = reduce(lambda x, y: x * y, numbers)

print(product)
```

THIS WILL OUTPUT 120.

In conclusion, Python provides a variety of built-in functions that make it easy to manipulate lists. The **len()**, **sort()**, **sorted()**, **filter()**, **map()** and **reduce()** functions are commonly used for tasks such as determining the length of a list, sorting elements, filtering elements based on a certain condition, applying an operation to each element of a list, and reducing a list of elements to a single value. These functions can make your code more readable, efficient, and organized. It's important to note that some of these functions such as **filter()** and **map()** returns an iterator, which can be converted to a list using the **list()** function. Additionally, **reduce()** is part of the **functools** module and needs to be

imported before using it. Understanding and using these built-in functions effectively can help you write more efficient and organized code.

2.7 FUNCTION FOR DICTIONARY

Python provides a variety of built-in functions that can be used to manipulate dictionaries. These functions make it easy to perform common operations on dictionaries, such as iterating through keys and values, adding and updating key-value pairs, and checking for the existence of a key or value.

The **len()** function is used to determine the number of key-value pairs in a dictionary. For example:

person **=** {"name"**:** "John"**,** "age"**:** 30}

print(len(person**))**

THIS WILL OUTPUT 2.

The **keys()** function is used to return a view of all the keys in a dictionary. For example:

person **=** {"name"**:** "John"**,** "age"**:** 30}

print(person**.**keys**())**

THIS WILL OUTPUT **dict_keys(['name', 'age'])**

The **values()** function is used to return a view of all the values in a dictionary. For example:

```python
person = {"name": "John", "age": 30}

print(person.values())
```

THIS WILL OUTPUT **dict_values(['John', 30])**

The **items()** function is used to return a view of all the key-value pairs in a dictionary as a list of tuple. For example:

```python
person = {"name": "John", "age": 30}

print(person.items())
```

THIS WILL OUTPUT **dict_items([('name', 'John'), ('age', 30)])**

The **get()** function is used to retrieve the value of a key in a dictionary. If the key is not found, it returns None or a default value that can be specified as an argument. For example:

```python
person = {"name": "John", "age": 30}

print(person.get("name")) # Output: John

print(person.get("address", "Unknown")) # Output: Unknown
```

THE **update()** function is used to update the key-value pairs of a dictionary. It can take another dictionary or key-value pairs as argument. For example:

```python
person = {"name": "John", "age": 30}

person.update({"name": "Jane", "gender": "female"})

print(person) # Output: {'name': 'Jane', 'age': 30, 'gender': 'female'}
```

THE **pop()** function is used to remove a key-value pair from a dictionary. It takes the key as an argument and returns the value of the key that was removed. If the key is not found, it raises a **KeyError** or return a default value that can be specified as an argument. For example:

```python
person = {"name": "John", "age": 30}

print(person.pop("name")) # Output: John

print(person) # Output: {'age': 30}
```

THE **in** keyword is used to check if a key or a value exists in a dictionary. For example:

```python
person = {"name": "John", "age": 30}

print("name" in person) # Output: True

print("address" in person) # Output: False
```

IN CONCLUSION, PYTHON provides a variety of built-in functions that make it easy to manipulate dictionaries. The **len()**, **keys()**, **values()**, **items()**, **get()**, **update()**, **pop()** and the **in** keyword are commonly used for tasks such as determining the number of key-value pairs, iterating through keys and values, adding and updating key-value pairs, checking for the existence of a key or value and removing key-value pairs. These functions can make your code more readable, efficient, and organized. Understanding and using these built-in functions effectively can help you write more efficient and organized code.

2.8 STRING MANIPULATION FUNCTION

Python provides a variety of built-in functions and methods for manipulating strings. These functions and methods make it easy to perform common operations on strings, such as concatenation, slicing, formatting, and searching.

The **len()** function is used to determine the length of a string, which is the number of characters it contains. For example:

name = "John"

print(len(name**))**

THIS WILL OUTPUT 4.

The **+** operator is used to concatenate two or more strings together. For example:

first_name = "John"

last_name = "Doe"

full_name = first_name **+** " " **+** last_name

print(full_name**)**

THIS WILL OUTPUT "JOHN Doe".

The * operator is used to repeat a string a certain number of times. For example:

name = "John"

print(name * 3**)**

THIS WILL OUTPUT "JOHNJOHNJOHN".

The **[:]** notation is used to slice a string. It allows you to extract a substring from a string by specifying the start and end index. For example:

name = "John Doe"

print(name**[0:4])**

THIS WILL OUTPUT "JOHN".

The **in** keyword is used to check if a substring exists in a string. For example:

name = "John Doe"

print("John" **in** name**)** # Output: True

print("Jane" **in** name**)** # Output: False

THE **replace()** method is used to replace a substring with another substring in a string. For example:

```
name = "John Doe"

new_name = name.replace("John", "Jane")

print(new_name)
```

THIS WILL OUTPUT "JANE Doe".

The **split()** method is used to split a string into a list of substrings using a specified delimiter. For example:

```
name = "John,Doe,30"

name_list = name.split(",")

print(name_list)
```

THIS WILL OUTPUT **['John', 'Doe', '30']**

The **format()** method is used to format strings by replacing placeholders with values. Placeholders are represented by curly braces **{}**. For example:

```
name = "John"

age = 30

print("My name is {} and I am {} years old.".format(name, age))
```

THIS WILL OUTPUT "MY name is John and I am 30 years old."

The **join()** method is used to join a list of strings into a single string using a specified delimiter. For example:

```python
names = ["John", "Doe", "Jane"]

string_of_names = ",".join(names)

print(string_of_names)
```

THIS WILL OUTPUT "JOHN,Doe,Jane"

In conclusion, Python provides a variety of built-in functions and methods for manipulating strings. The **len()**, **+** operator, ***** operator, **[:]** notation, **in** keyword, **replace()**, **split()**, **format()** and **join()** are commonly used for tasks such as determining the length of a string, concatenating multiple strings, repeating a string, slicing substrings, checking for the existence of a substring, replacing substrings, splitting strings into a list of substrings, formatting strings with placeholders and joining a list of strings into a single string. These functions and methods can make your code more readable, efficient, and organized. Understanding and using these built-in functions and methods effectively can help you write more efficient and organized code. It's important to note that some

methods like **replace()**, **split()**, **format()** and **join()** are specific to strings and can't be used on other data types.

2.9 EXCEPTION HANDLING

Exception handling is a mechanism in Python that allows you to handle errors and exceptional situations in your code. It allows you to write code that can continue to execute even when an error occurs. This is important because it allows your program to continue running and avoid crashing, which can lead to a better user experience.

The **try** and **except** keywords are used to handle exceptions in Python. The **try** block contains the code that may raise an exception. The **except** block contains the code that will be executed if an exception is raised. For example:

```
try:

num1 = 7

num2 = 0

print(num1 / num2)

except ZeroDivisionError:

print("Division by zero is not allowed.")
```

IN THIS EXAMPLE, THE code in the **try** block raises a **ZeroDivisionError** exception when it attempts to divide **num1** by **num2**, which has a value of 0. The code in the

except block is executed and the message "Division by zero is not allowed." is printed.

You can use multiple **except** blocks to handle different types of exceptions. For example:

try:

variable = "hello"

print(int(variable**))**

except ValueError:

print("ValueError: could not convert string to int."**)**

except TypeError:

print("TypeError: int() argument must be a string, a bytes-like object or a number, not 'list'"**)**

IN THIS EXAMPLE, THE code in the **try** block raises a **ValueError** exception when it attempts to convert the string "hello" to an integer, which is not possible. The first **except** block is executed and the message "ValueError: could not convert string to int." is printed.

You can also use the **finally** block to include code that will be executed regardless of whether an exception was raised or not. For example:

try:

```
num1 = 7

num2 = 0

print(num1 / num2)

except ZeroDivisionError:

print("Division by zero is not allowed.")

finally:

print("This code will be executed no matter what.")
```

IN THIS EXAMPLE, THE code in the **finally** block will be executed regardless of whether the code in the **try** block raises an exception or not.

In addition, you can use the **raise** keyword to raise an exception manually.

```
raise ValueError("Invalid Value")
```

IN CONCLUSION, EXCEPTION handling is an important feature in Python that allows you to handle errors and exceptional situations in your code. The **try** and **except** keywords are used to handle exceptions, while the **finally** block can be used to include code that will be executed regardless of whether an exception was raised or not. Using multiple **except** blocks allows you to handle different types of

exceptions separately. The **raise** keyword can be used to raise an exception manually. Exception handling allows your program to continue running and avoid crashing, which can lead to a better user experience. It is important to use exception handling in your code to anticipate and handle unexpected situations, and to make your code more robust and reliable.

2.10 FILE HANDLING IN PYTHON

File handling in Python allows you to read from and write to files on your computer's file system. Python provides a variety of built-in functions and methods for working with files, including opening, reading, writing, and closing files.

The **open()** function is used to open a file. It takes the file name and the mode in which the file should be opened as arguments. The mode can be 'r' for reading, 'w' for writing, and 'a' for appending. For example:

```
file = open('example.txt', 'r')
```

THIS OPENS THE FILE 'example.txt' in read mode.

The **read()** method is used to read the contents of a file. For example:

```
file = open('example.txt', 'r')

contents = file.read()

print(contents)

file.close()
```

THIS READS THE CONTENTS of the file 'example.txt' and stores it in the variable 'contents' and prints it. it's important to close the file after reading it by calling **file.close()**

The **write()** method is used to write to a file. It takes a string as an argument. For example:

file **= open(**'example.txt'**,** 'w'**)**

file**.**write**(**"Hello World!"**)**

file**.**close**()**

———————

THIS WRITES THE STRING "Hello World!" to the file 'example.txt'.

The **append()** method is used to add new data to the end of a file without overwriting the existing contents. It works similar to the write method. For example:

file **= open(**'example.txt'**,** 'a'**)**

file**.**write**(**"\nThis is an added text"**)**

file**.**close**()**

———————

THIS WILL ADD "THIS is an added text" to the end of the file 'example.txt' without overwriting the existing contents.

The **with** statement is used to open a file and automatically close it when you are done with it. This is considered as a best practice to avoid file not closed errors. For example:

```python
with open('example.txt', 'r') as file:

contents = file.read()

print(contents)
```

THIS WILL OPEN THE file 'example.txt' in read mode, read its contents and print it, and then automatically close the file when it's done.

The **readline()** method is used to read a single line from a file. For example:

```python
with open('example.txt', 'r') as file:

line = file.readline()

print(line)
```

THIS WILL READ THE first line of the file 'example.txt' and print it.

File handling in Python allows you to read from and write to files on your computer's file system. Python provides a variety of built-in functions and methods for working with

files, including opening, reading, writing, and closing files. It's important to close the file after reading or writing to it, and the **with** statement is considered as a best practice to avoid file not closed errors. These functions and methods can make your code more efficient and organized. Understanding and using these built-in functions and methods effectively can help you write more efficient and organized code.

2.11 MODLUES IN PYTHON

Modules in Python are pre-written code libraries that you can use to add extra functionality to your Python programs. They are a way to organize and reuse code, and they can help you write more efficient and organized code. Python has a wide variety of built-in modules, and you can also install third-party modules using package managers such as pip.

The **import** statement is used to import a module in Python. For example:

import math

THIS IMPORTS THE MATH module, which provides mathematical functions such as **sqrt()**, **sin()**, and **cos()**.

You can also use the **from** keyword to import specific functions or variables from a module. For example:

from math **import** sqrt

THIS IMPORTS ONLY THE **sqrt()** function from the math module.

You can also use the **as** keyword to give a module or function a different name when you import it. For example:

import math **as** m

THIS IMPORTS THE MATH module and gives it the name 'm', so you can use **m.sqrt()** instead of **math.sqrt()**.

You can also use the * wildcard character to import all functions and variables from a module. For example:

from math **import** *

This imports all functions and variables from the math module, so you can use **sqrt()** instead of **math.sqrt()**.

Python also provides a way to find out the list of functions or variable inside a module using **dir()** function. For example

import math

print(dir(math**))**

This will print a list of all the functions and variables inside the math module.

It's important to note that using the * wildcard character to import all functions and variables from a module can cause naming conflicts if there are functions or variables with the same name in different modules. It's recommended to use

the **import** statement to import the specific functions or variables that you need, or to use the **as** keyword to give them different names.

In conclusion, modules in Python are pre-written code libraries that you can use to add extra functionality to your Python programs. They are a way to organize and reuse code, and they can help you write more efficient and organized code. The **import** statement, **from** keyword, **as** keyword, and * wildcard character are used to import modules, functions, and variables. It's important to use the import statement to import the specific functions or variables that you need and to use the **dir()** function to find out the list of functions or variables inside a module. Understanding and using modules effectively can help you write more efficient and organized code, and it can save you time by not having to re-write code that already exists.

2.12 STYLE GUIDE FOR PYTHON CODE

To make the code readable, maintainable, and shareable, there are certain style conventions that need to be followed. These conventions are documented in the Python Style Guide, also known as PEP 8.

PEP 8 provides a set of guidelines for formatting Python code. These guidelines cover topics such as naming conventions, indentation, whitespace, line length, comments, and more. By following these guidelines, you can improve the readability and consistency of your code, which makes it easier to understand and maintain.

Here are some of the key style conventions that are recommended by PEP 8:

1. Naming Conventions

In Python, there are naming conventions for variables, functions, classes, and modules. These conventions make it easier to understand the purpose of each object. Here are the basic naming conventions:

- Variables and functions should be lowercase, with words separated by underscores.

- Classes should use CamelCase (words are separated by capital letters).
- Modules should be lowercase, with words separated by underscores.

1. Indentation

Python uses indentation to define blocks of code, rather than using braces or other delimiters. It's recommended to use 4 spaces for each level of indentation, rather than using tabs.

1. Whitespace

Whitespace can be used to improve the readability of your code. It's recommended to use a single space after commas and operators, and to avoid using spaces between function names and parentheses.

1. Line Length

Long lines of code can be difficult to read, especially on smaller screens or in a terminal window. PEP 8 recommends limiting lines to a maximum of 79 characters. If a line of code exceeds this limit, you can break it up into multiple lines using backslashes or parentheses.

1. Comments

Comments can be used to explain the purpose of your code and how it works. PEP 8 recommends using comments sparingly, and only when necessary. Comments should be written in complete sentences, and should start with a capital letter.

In addition to these conventions, PEP 8 also recommends a few other best practices. For example, it's recommended to use the built-in functions and modules whenever possible, rather than writing your own. It's also recommended to use the Python 3.x syntax whenever possible, rather than using deprecated features from Python 2.x.

By following these conventions, you can make your Python code more readable, maintainable, and shareable. If you're working on a large project with other developers, it's especially important to follow these guidelines, as it makes it easier for everyone to understand the code.

PEP 8 provides a set of guidelines for formatting Python code. By following these guidelines, you can improve the readability and consistency of your code, which makes it easier to understand and maintain.

2.13 DOCSTRING CONVENTIONS IN PYTHON

In Python, a docstring is a string literal that appears as the first statement of a module, function, class, or method definition. It is used to provide documentation about the object being defined.

There are several conventions for writing docstrings in Python, with the most common being the PEP 257 convention. This convention specifies that a docstring should be a multi-line string that includes a one-line summary of the object, followed by a blank line and a more detailed description of the object. The detailed description should be in the form of complete sentences, and should provide information on the parameters, return value, and any exceptions that the function may raise.

Here's an example of a function definition with a docstring that follows the PEP 257 convention:

```python
def add_numbers(x, y):

    """

    Add two numbers together and return the result.

    Args:

    x (int): The first number to add.
```

y (int): The second number to add.

Returns:

int: The sum of x and y.

"""

return x + y

IN THIS EXAMPLE, THE docstring provides a summary of the function and a detailed description of the parameters and return value. The parameter types are specified using type hints, which are optional but recommended in Python 3.

Following a consistent docstring convention can help make your code more readable and easier to understand. It also makes it easier for automated tools to generate documentation from your code.

2.14 PYTHON LIBRARY FOR DATA SCIENCE

Python is a popular programming language that has gained immense popularity in the data science community. It offers a vast array of libraries and tools that have made data science tasks easier and more efficient. In this article, we will discuss some of the most popular Python libraries for data science.

1. **NumPy**: NumPy is a fundamental library for scientific computing in Python. It provides support for large, multi-dimensional arrays and matrices, along with a large library of mathematical functions. NumPy is designed to integrate with other libraries like SciPy and Pandas.
2. **Pandas**: Pandas is a library that provides data manipulation and analysis tools. It offers data structures like Series and DataFrame that allow for easy manipulation and transformation of data. Pandas provides support for working with tabular, structured, and time-series data.
3. **Matplotlib**: Matplotlib is a 2D plotting library that enables users to create various types of charts and plots. It supports a wide range of plot types, including line, bar, scatter, and histogram. Matplotlib is an excellent tool for visualizing data and generating insights.

4. **Scikit-learn**: Scikit-learn is a machine learning library for Python. It provides simple and efficient tools for data mining and data analysis. Scikit-learn offers support for various supervised and unsupervised learning algorithms.
5. **TensorFlow**: TensorFlow is an open-source machine learning library developed by Google. It is widely used for building deep learning models. TensorFlow provides an extensive set of tools for building and training neural networks, along with a high-level Keras API for building deep learning models.
6. **PyTorch**: PyTorch is another popular open-source machine learning library that is widely used for building deep learning models. It provides support for both CPU and GPU processing and is known for its simplicity and ease of use.
7. **Keras**: Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It is designed to enable fast experimentation with deep neural networks, and it provides a simple, consistent interface for building and training deep learning models.
8. **Seaborn**: Seaborn is a data visualization library based on Matplotlib. It provides a high-level interface for drawing informative and attractive statistical graphics. Seaborn provides support for a wide range of plot types, including heatmaps, violin plots, and categorical plots.

In conclusion, Python has become the go-to programming language for data science due to the wide range of libraries and tools available. The libraries listed above are some of the most popular Python libraries for data science and are used by data scientists and machine learning engineers worldwide.

2.15 SUMMARY

- Python is a high-level programming language that is widely used for web development, data science, and other fields.
- Python has a simple and easy-to-learn syntax, making it a popular choice for beginners.
- Python has a large and active community, which provides a wide range of resources and libraries for programmers.
- There are different versions of Python, including Python 2 and Python 3, and it is recommended to use the latest version (Python 3) as it has many improvements over the older version.
- Python supports various data types including numbers, strings, lists, and dictionaries.
- Python has built-in functions and modules that can be used for tasks such as mathematical operations, string manipulation, and file handling.
- Python also has a feature called "Control Flow" which allows the programmer to control the flow of the program, making use of the 'if' and 'else' statements, the 'for' and 'while' loops.
- Python also allows defining and using functions which can be defined using the 'def' keyword and called by their name.

- Python also has a feature called "Exception Handling" which allows the programmer to handle errors and exceptional situations in their code, making use of the 'try' and 'except' keywords.
- Python also has a wide variety of libraries for data science and machine learning, including NumPy, pandas, Matplotlib, Seaborn, scikit-learn, TensorFlow, Keras, PyTorch, SciPy, and statsmodels.

Python is a powerful and versatile programming language that can be used for a wide variety of tasks. It has a simple and easy-to-learn syntax, making it a popular choice for beginners. It also has a large and active community, which provides a wide range of resources and libraries for programmers. Understanding the basics of Python, including its data types, built-in functions and modules, control flow, and exception handling, is essential for any beginner looking to start programming with Python. Additionally, being familiar with the most popular libraries for data science and machine learning can help beginners to easily implement complex algorithms and perform advanced data analysis tasks.

2.16 TEST YOUR KNOWLEDGE

I. **What is the recommended version of Python to use?**

a. Python 2
b. Python 3
c. Python 4
d. Python 5

I. **What is the purpose of the try and except keywords in Python?**

a. To control the flow of the program
b. To handle errors and exceptional situations
c. To import modules
d. To define and call functions

I. **What is the purpose of the import statement in Python?**

a. To control the flow of the program
b. To handle errors and exceptional situations
c. To import modules
d. To define and call functions

I. **Which data type in Python is used to store multiple items in a single variable?**

a. String

b. Integer

c. List

d. Tuple

I. What is the purpose of the dir() function in Python?

a. To control the flow of the program

b. To handle errors and exceptional situations

c. To import modules

d. To find out the list of functions or variables inside a module

I. What is the purpose of the open() function in Python?

a. To open a file

b. To close a file

c. To read a file

d. To write to a file

I. Which library in Python is widely used for topic modeling and document similarity analysis?

a. NumPy

b. pandas

c. Gensim

d. Matplotlib

I. What is the purpose of the * wildcard character when importing modules in Python?

a. To import all functions and variables from a module

b. To import specific functions or variables from a module

c. To give a module or function a different name when importing

d. To open a file

I. What is the purpose of the with statement in Python when working with files?

a. To open a file and automatically close it when you are done with it

b. To read a file

c. To write to a file

d. To find out the list of functions or variables inside a module

I. Which library in Python is widely used for machine learning and data science competitions?

a. NumPy

b. pandas

c. XGBoost

d. Matplotlib

2.17 ANSWERS

I. Answer:
b) Python 3

I. Answer:
b) To handle errors and exceptional situations

I. Answer: c) To import modules

I. Answer: c) List

I. Answer:
d) To find out the list of functions or variables inside a module

I. Answer:
a) To open a file

I. Answer: c) Gensim

I. Answer:
a) To import all functions and variables from a module

I. Answer: To open a file and automatically close it when you are done with
a) it

I. Answer: c) XGBoost

03

3 DATA PREPARATION

In machine learning, the quality and characteristics of the data plays a crucial role in the performance of the model. The data preparation step is the process of cleaning, transforming, and organizing the data to make it suitable for the machine learning model. The goal of this chapter is to provide an understanding of the data preparation process and the various techniques used to preprocess the data. We will cover data cleaning, feature scaling, feature selection, and data transformation, as well as the importance of evaluating the quality of the data. By the end of this chapter, you will have a clear understanding of how to prepare your data for machine learning, and how to ensure that it is of the highest quality.

3.1 IMPORTING DATA

Importing and cleaning data is an essential step in the machine learning process. The quality and characteristics of the data plays a crucial role in the performance of the model, and it is important to ensure that the data is in a format that can be understood by the machine learning model. In this section, we will discuss the process of importing and cleaning data, and the various techniques used to preprocess the data.

The first step in the data preparation process is to import the data into the program. In Python, there are several libraries that can be used to import data, including Pandas, NumPy, and CSV.

Pandas is a library that provides easy-to-use data structures and data analysis tools. It can be used to import data from a variety of sources, including CSV, Excel, and SQL databases.

NumPy is a library for numerical computation that provides support for large, multi-dimensional arrays and matrices. It can be used to import data in the form of arrays.

CSV (Comma Separated Values) is a common file format for storing data in a tabular form. In Python, the CSV module can be used to import data from a CSV file.

A common example of importing data in Python is using the Pandas library to import a CSV file. The following is an example of how to import a CSV file called "example_data.csv" using Pandas. The file "example_data.csv" contain employee data with following columns:

EMPLOYEE_ID, FIRST_NAME, LAST_NAME, EMAIL, PHONE_NUMBER, HIRE_DATE, JOB_ID, YEAR_OF_EXP, EDUCATION, CERTIFICATION, COMMISSION_PCT, MANAGER_ID, DEPARTMENT_ID, AGE, SALARY

The file will be available on this book GitHub repository. Here is the code for importing the csv file:

```
import pandas as pd

# Import the data from the CSV file

data = pd.read_csv("example_data.csv")
```

IN THIS EXAMPLE, THE first line imports the Pandas library and assigns it the alias "pd". The second line uses the "read_csv" function provided by Pandas to import the data from the "example_data.csv" file and store it in a variable called "data". The "read_csv" function automatically detects the delimiter (comma) used in the CSV file and separates the data into columns and rows.

Once the data is imported, it can be easily manipulated and analyzed. For example, you can view the first five rows of the data using the following code:

```
print(data.head())
```

YOU CAN ALSO ACCESS specific columns and rows of the data using the following code:

```
# Access the column "AGE"

age = data["AGE"]

# Access the row with index 2

row_2 = data.loc[2]
```

IN THIS EXAMPLE, WE have shown how to import a CSV file into python using Pandas, and also how to access specific columns and rows of the data once imported. Pandas is a powerful library that makes it easy to import and manipulate structured data, it's easy to use and understand, it's widely adopted in data science and machine learning, and it is one of the most important libraries for data preparation.

List of function for importing data

THERE ARE SEVERAL PYTHON functions that can be used to import data into a python script, including:

1. **pandas.read_csv()** - This function can be used to import data from a CSV file into a pandas DataFrame.
2. **pandas.read_excel()** - This function can be used to import data from an Excel file into a pandas DataFrame.
3. **pandas.read_json()** - This function can be used to import data from a JSON file into a pandas DataFrame.
4. **pandas.read_sql()** - This function can be used to import data from a SQL database into a pandas DataFrame.
5. **pandas.read_html()** - This function can be used to import data from an HTML file into a pandas DataFrame.
6. **pandas.read_pickle()** - This function can be used to import data from a pickle file into a pandas DataFrame.
7. **pandas.read_fwf()** - This function can be used to import data from a fixed-width-format file into a pandas DataFrame.
8. **pandas.read_stata()** - This function can be used to import data from a STATA file into a pandas DataFrame.
9. **pandas.read_sas()** - This function can be used to import data from a SAS file into a pandas DataFrame.
10. **pandas.read_clipboard()** - This function can be used to import data from the clipboard into a pandas DataFrame.

These are some of the most common functions used for importing data in python using pandas library. Depending on the format and source of the data, different functions can be used to import it into python.

In addition, the Pandas library provides many useful functions for data manipulation, such as sorting, filtering, and aggregating the data. This makes it a powerful tool for data analysis and preparation.

Once the data is imported, it can be easily manipulated and analyzed.

> To explore the data importing, cleaning and transformation in more detail, you can read the book "Python for Data Analysis" by the same author. There we explain all the above process and their relevant library such as NumPy, pandas in more detail.

3.2 CLEANING DATA

After importing the data, the next step is to clean it. Data cleaning is the process of removing or correcting inaccurate, incomplete, or irrelevant data. This step is important because the quality of the data can have a significant impact on the performance of the machine learning model. The following are some common data cleaning techniques:

Removing duplicate data

REMOVING DUPLICATE data is an important step in data preprocessing as it can improve the accuracy and efficiency of machine learning models. Duplicate data can occur for various reasons, such as data entry errors, data merging, or data scraping.

There are several techniques for removing duplicate data, including:

1. **Removing duplicate rows:** This method involves identifying and removing duplicate rows based on one or more columns. This method can be useful when the duplicate data is limited to a small number of rows.
2. **Removing duplicate columns:** This method involves identifying and removing duplicate columns based on one

or more columns. This method can be useful when duplicate data is limited to a small number of columns.

3. **Removing duplicate records based on a subset of columns:** This method involves identifying and removing duplicate records based on a subset of columns. This method can be useful when duplicate data is limited to a specific subset of columns.

We can do this using the following code snippet in python using the pandas library:

```python
# Import the data

data = pd.read_csv("example_data.csv")

# Remove duplicate rows

data = data.drop_duplicates()

# Remove duplicate columns

data = data.loc[:,~data.columns.duplicated()]

#Remove duplicate records based on a subset of columns

data = data.drop_duplicates(subset = ['EMPLOYEE_ID', 'EMAIL',
'PHONE_NUMBER'])
```

IN THIS EXAMPLE, THE first line imports the data from the "example_data.csv" file and store it in a variable called "data".

The second line uses the drop_duplicates() method from pandas to remove the duplicate rows from the dataframe.

The third line uses the drop_duplicates() method from pandas to remove the duplicate columns from the dataframe by using the option of **.loc[:,~data.columns.duplicated()]**

The fourth line uses the drop_duplicates() method with the subset parameter, to remove the duplicate records based on a subset of columns, in this case columns **'EMPLOYEE_ID', 'EMAIL',** and **'PHONE_NUMBER'**.

It's important to keep in mind that when removing duplicate data, it's crucial to consider the size of the dataset, as removing duplicate data can cause a significant loss of information.

It's also important to check the distribution of the data after removing duplicate data to ensure that it makes sense for the data.

Sometimes it's not always necessary to remove duplicate data, for example, in case of time-series data, duplicate data can be useful for studying the trends over time. Therefore, it is important to consider the context of the data and the research question before removing duplicate data.

It's also important to keep in mind that when removing duplicate data, it's crucial to use the appropriate method based on the specific characteristics of the data and the research question. For example, if duplicate data is limited to a specific subset of columns, it's more appropriate to remove duplicate records based on that subset of columns rather than removing all duplicate data.

Handling missing data

HANDLING MISSING VALUES is an important step in data preprocessing as it can have a significant impact on the performance of machine learning models. Missing values can occur for various reasons, such as data entry errors, measurement errors, or non-response.

There are several techniques for handling missing values, including:

1. **Deleting the rows or columns with missing values**: This is the simplest method, but it can result in a loss of important information if a large number of observations are removed.
2. **Imputing the missing values**: This method involves replacing the missing values with a substitute value, such as the mean, median, or mode of the variable. This method can be useful when the number of missing values

is small, but it can introduce bias if the missing values are not missing at random.

3. **Using a predictive model to impute missing values**: This method involves training a model to predict the missing values based on the non-missing values. This method can be useful when the number of missing values is large, but it can be computationally expensive.

We can do this using the following code snippet in python using the scikit-learn library:

```python
from sklearn.impute import SimpleImputer

# Import the data

data = pd.read_csv("example_data.csv")

X = (data.drop(columns =["EMPLOYEE_ID", "FIRST_NAME", "LAST_NAME", "EMAIL",

"PHONE_NUMBER", "HIRE_DATE", "JOB_ID", "COMMISSION_PCT",

"MANAGER_ID", "DEPARTMENT_ID", "SALARY"], axis=1))

# Create an imputer object

imputer = SimpleImputer(strategy="mean")

# Fit the imputer object to the data

imputer.fit(X)

# Transform the data

X_imputed = imputer.transform(X)
```

IN THIS EXAMPLE, THE first line imports the SimpleImputer class from the scikit-learn library. The second line imports the data from the "example_data.csv" file and store it in a variable called "data". The third line separates the predictor variables (X) and all non-numeric columns (namely EMPLOYEE_ID, FIRST_NAME, LAST_NAME, EMAIL, PHONE_NUMBER, HIRE_DATE, JOB_ID, , COMMISSION_PCT, MANAGER_ID, DEPARTMENT_ID, SALARY) from the target variable. The fourth line creates an imputer object and sets the strategy parameter to "mean" which means that the missing values will be replaced with the mean value of the feature. The fifth line fits the imputer object to the data. The sixth line uses the transform method to replace the missing values with the mean value of the feature.

It's important to keep in mind that the chosen method for handling missing values should be based on the specific characteristics of the data and the research question. Additionally, it's also important to check the distribution of the data after handling missing values to ensure that it makes sense for the data.

Handling outliers

HANDLING OUTLIERS IS an important step in data preprocessing as it can have a significant impact on the performance of machine learning models. Outliers are observations that deviate significantly from the majority of

the data. They can occur for various reasons, such as measurement errors, data entry errors, or data from a different distribution.

There are several techniques for handling outliers, including:

1. **Removing outliers**: This method involves identifying and removing observations that deviate significantly from the majority of the data. This method can be useful when the number of outliers is small, but it can result in a loss of important information if a large number of observations are removed.
2. **Transforming the data**: This method involves transforming the data using techniques such as log transformation, square root transformation, or reciprocal transformation to reduce the impact of outliers.
3. **Imputing outliers**: This method involves replacing outliers with a substitute value, such as the mean, median, or mode of the variable. This method can be useful when the number of outliers is small, but it can introduce bias if the outliers are not missing at random.
4. **Using robust models**: This method involves using models that are less sensitive to outliers such as decision trees or linear discriminant analysis.

We can do this using the following code snippet in python using the scikit-learn library:

```python
from sklearn.covariance import EllipticEnvelope

# Import the data

data = pd.read_csv("example_data.csv")

X = (data.drop(columns =["EMPLOYEE_ID", "FIRST_NAME", "LAST_NAME",

"EMAIL",

"PHONE_NUMBER", "HIRE_DATE", "JOB_ID",

"COMMISSION_PCT",

"MANAGER_ID", "DEPARTMENT_ID", "SALARY"],

axis=1))

########## Handling missing values ##############

# Create an imputer object

imputer = SimpleImputer(strategy="mean")

# Fit the imputer object to the data

imputer.fit(X)

# Transform the data

X_imputed = imputer.transform(X)

X = X_imputed

########## Handling outliers ##################

# Create an EllipticEnvelope object

outlier_detector = EllipticEnvelope(contamination=0.05)

# Fit the outlier detector to the data
```

```
outlier_detector.fit(X)

# Predict the outliers

y_pred = outlier_detector.predict(X)

# Identify the outliers

outliers = X[y_pred == -1]
```

IN THIS EXAMPLE, THE first line imports the EllipticEnvelope class from the scikit-learn library. The second line imports the data from the "example_data.csv" file and store it in a variable called "data". The third line separates the predictor variables (X) and non-numeric variable from the target variable. After that, we handle missing values as described in previous section.

The next line after comment (########## *Handling outliers #################*) creates an outlier detector object and sets the contamination parameter to 0.05 which means that 5% of the data is considered as outliers. The next line fits the outlier detector to the data. The next line uses the predict method to identify the observations that deviate significantly from the majority of the data. The next line identifies the outliers by using the predictions from the outlier detector.

It's important to keep in mind that the chosen method for handling outliers should be based on the specific characteristics of the data and the research question. Additionally, it's also important to check the distribution of the data after handling outliers to ensure that it makes sense for the data.

Formatting data

FORMATTING DATA IS an important step in data preprocessing as it ensures that the data is in a consistent and usable format for machine learning models. Formatting data involves converting data into a format that can be easily consumed by machine learning algorithms. This can include tasks such as converting data types, encoding categorical variables, and standardizing variable names.

There are several techniques for formatting data, including:

1. Converting data types: This method involves converting data into the appropriate data type, such as converting text data into numerical data or converting date/time data into a timestamp format. This can be done using functions such as **to_numeric**, **to_datetime** in pandas library.
2. Encoding categorical variables: This method involves converting categorical variables, such as text data, into a numerical format that can be used by machine learning

models. This can be done using techniques such as one-hot encoding, ordinal encoding, or dummy encoding.

3. Standardizing variable names: This method involves converting variable names into a consistent format, such as converting variable names to lowercase or removing spaces. This can be done using functions such as **str.lower()**, **str.strip()** in pandas library.

We can do this using the following code snippet in python using the pandas library:

```python
# Import the data

data = pd.read_csv("example_data.csv")

#Convert data types

data['AGE'] = data[['AGE']].astype(int)

data['HIRE_DATE'] = pd.to_datetime(data['HIRE_DATE'])

#Encoding categorical variables

data = pd.get_dummies(data, columns=["EDUCATION_LEVEL"])

#Standardizing variable names

data.rename(columns={'EMAIL': 'EMAIL_ID'}, inplace=True)

data.columns = data.columns.str.strip().str.lower().str.replace(' ', '_')
```

_____

IN THIS EXAMPLE, THE first line imports the data from the "example_data.csv" file and store it in a variable called

"data". The second line converts the data types of column 'age' to integer using the **astype()** method and the column 'date' to datetime using the **pd.to_datetime()** method. The third line uses the **pd.get_dummies()** method to encode the categorical variable 'color' by one-hot encoding The fourth line uses the **rename()** method to standardize the variable name 'name' to 'full_name' and also use **str.strip()**, **str.lower()**, **str.replace()** to standardize all columns name to lowercase, remove spaces and replace spaces with underscore.

It's important to keep in mind that the chosen method for formatting data should be based on the specific characteristics of the data and the research question. Additionally, it's also important to check the distribution of the data after formatting data to ensure that it makes sense for the data.

In conclusion, Formatting data is an important step in data preprocessing as it ensures that the data is in a consistent and usable format for machine learning models. Formatting data involves converting data into a format that can be easily consumed by machine learning algorithms. This can include tasks such as converting data types, encoding categorical variables, and standardizing variable names. There are several techniques for formatting data, including converting data types, encoding categorical variables, and standardizing variable names. The chosen method should be based on the

specific characteristics of the data and the research question. Formatting data correctly can help to improve the performance of machine learning models and make the data easier to work with.

It's also important to check the data for any inconsistencies or errors, such as typos or mislabeled data, and to correct them as necessary.

It's important to note that data cleaning can be a time-consuming process, but it is critical to improve the performance of the machine learning model.

List of function to clean data

THERE ARE SEVERAL PYTHON functions that can be used to clean data in a python script, including:

1. **pandas.DataFrame.drop()** - This function can be used to drop specified labels from rows or columns. It can be used to drop rows or columns based on their index or column name, and can also be used to drop rows or columns based on certain conditions.
2. **pandas.DataFrame.fillna()** - This function can be used to fill missing values with a specific value or method. It can be used to fill missing values with a specific value, such as the mean or median of the data, or it can be used to forward-fill or backward-fill missing values.

3. **pandas.DataFrame.replace()** - This function can be used to replace values in a DataFrame. It can be used to replace a specific value or a set of values with another value or set of values.

4. **pandas.DataFrame.drop_duplicates()** - This function can be used to remove duplicate rows from a DataFrame. It can be used to drop duplicate rows based on one or more columns and can also be used to keep the first or last occurrence of duplicate rows.

5. **pandas.DataFrame.query()** - This function can be used to filter rows of a DataFrame based on a Boolean expression. It can be used to filter rows based on certain conditions, such as removing rows with missing values or removing rows with specific values.

6. **pandas.DataFrame.rename()** - This function can be used to rename columns or row indexes of a DataFrame. It can be used to rename one or multiple columns or indexes.

7. **pandas.DataFrame.sort_values()** - This function can be used to sort a DataFrame by one or more columns. It can be used to sort the data in ascending or descending order and can also be used to sort based on multiple columns.

8. **pandas.DataFrame.groupby()** - This function can be used to group rows of a DataFrame based on one or more columns. It can be used to group data by a specific

column and perform calculations such as mean, sum, or count on the grouped data.

For example,

```python
import pandas as pd
# Import the data
data = pd.read_csv("example_data.csv")
# Remove missing values
data = data.dropna()
# Replace specific values
data = data.replace({'JOB_ID': {'ST_CLERK': 'STATION_CLERK'}})
# Remove duplicate rows
data = data.drop_duplicates()
# Rename columns
data = data.rename(columns={'FIRST_NAME': 'GIVEN_NAME'})
```

IN THIS EXAMPLE, THE first line imports the data from the "example_data.csv" file and store it in a variable called "data". The second line uses the **dropna()** function to remove all the rows that contain missing values. The third line uses the **replace()** function to replace all occurrences of the value 'green' in the 'color' column with 'blue'. The fourth line uses the **drop_duplicates()** function to remove all

duplicate rows from the DataFrame. The fifth line uses the **rename()** function to rename the column 'name' to 'full_name'.

It's important to keep in mind that the chosen method for cleaning data should be based on the specific characteristics of the data and the research question. Additionally, it's also important to check the distribution of the data after cleaning to ensure that it makes sense for the data. It's also important to keep in mind that data cleaning is an iterative process and it's necessary to repeat the process until the data is in the desired format.

A common example of cleaning data in Python is using the Pandas library to handle missing values. The following is an example of how to handle missing values in a DataFrame:

```python
import pandas as pd

# Import the data from the CSV file

data = pd.read_csv("example_data.csv")

# Checking the number of missing values in each column

print(data.isnull().sum())

# Dropping rows with missing values

data = data.dropna()

# Filling missing values with mean of the column

data = data.fillna(data.mean())
```

IN THIS EXAMPLE, THE first line imports the Pandas library and assigns it the alias "pd". The second line uses the "read_csv" function provided by Pandas to import the data from the "example_data.csv" file and store it in a variable called "data".

The third line uses the "isnull()" function provided by Pandas to check for missing values in each column of the data, and the "sum()" function to count the number of missing values in each column.

The fourth line uses the "dropna()" function to drop all the rows that contain missing values. This method is useful when the missing values are in a small number and can be dropped without affecting the overall data.

The fifth line uses the "fillna()" function to fill missing values with the mean of the column. This method is useful when the missing values are in a large number and need to be replaced with a suitable value, in this case, the mean of the column.

It's important to note that there are many other ways to handle missing values such as using median, mode, or interpolation methods. The choice of method depends on the characteristics of the data and the specific requirements of the project.

In this example, we have shown how to handle missing values in a DataFrame using the Pandas library. We have used the "isnull()" and "sum()" functions to check for missing values, the "dropna()" function to drop rows with missing values, and the "fillna()" function to fill missing values with the mean of the column. These are just some of the ways to handle missing values, and different methods may work better depending on the characteristics of the data.

It's important to keep in mind that data cleaning is an iterative process and it may require multiple steps to clean the data properly. It's also important to check the data for any inconsistencies or errors, such as typos or mislabeled data, and to correct them as necessary.

In conclusion, cleaning data is an essential step in the machine learning process, it is important to ensure that the data is of high quality. This can be achieved by using various libraries and techniques such as Pandas, handling missing values, dropping rows, filling missing values with suitable values, checking for inconsistencies and errors, these steps are crucial to improve the performance of the machine learning model. The more time and effort you invest in data cleaning, the better the performance of your machine learning model will be.

3.3 EXPLORATORY DATA ANALYSIS

Exploratory Data Analysis (EDA) is a technique used to understand and analyze data. The goal of EDA is to gain insights and understand the underlying structure of the data, as well as identify any patterns, relationships, or anomalies that may be present. EDA is an iterative process, and it is typically the first step in the data analysis pipeline. In this section, we will discuss the process of EDA, and the various techniques used to explore and understand the data.

Univariate Analysis

THE FIRST STEP IN EDA is to perform univariate analysis, which involves **analyzing each variable individually**. This helps to understand the distribution of the data and identify any outliers or anomalies. Some common techniques used in univariate analysis include:

Descriptive statistics

DESCRIPTIVE STATISTICS are a crucial part of data analysis that allows us to summarize and describe the main characteristics of a dataset. In univariate analysis, descriptive statistics are used to summarize and describe the properties of a single variable.

Descriptive statistics in univariate analysis can be classified into two broad categories: measures of central tendency and measures of variability.

Measures of Central Tendency

MEASURES OF CENTRAL tendency are used to describe the typical or central value of a distribution. The most common measures of central tendency are:

1. **Mean**: It is the sum of all observations in the dataset divided by the total number of observations.
2. **Median**: It is the middle value in a dataset. When a dataset has an even number of observations, the median is the average of the two middle values.
3. **Mode**: It is the value that occurs most frequently in a dataset.

Measures of Variability

MEASURES OF VARIABILITY are used to describe the spread or dispersion of the data. The most common measures of variability are:

1. **Range**: It is the difference between the maximum and minimum values in a dataset.

2. **Variance**: It is the average of the squared differences of each value from the mean.
3. **Standard Deviation**: It is the square root of the variance and is used to describe the spread of data around the mean.

Other measures of variability include percentiles, which are used to describe the distribution of the data over the entire range.

Descriptive statistics can be presented using different types of graphical representations such as histograms, boxplots, and scatterplots. These visualizations help to identify patterns and outliers in the data.

A common example of univariate analysis is using the Pandas library to calculate descriptive statistics of a dataset. The following is an example of how to perform univariate analysis on a variable "Age" in a dataset "data":

```python
import pandas as pd

# Import the data

data = pd.read_csv("example_data.csv")

# Extract the variable "Age"

age = data["AGE"]

# Calculate descriptive statistics

print("Mean: ", age.mean())
```

```python
print("Median: ", age.median())

print("Mode: ", age.mode())

print("Standard Deviation: ", age.std())

print("Minimum Value: ", age.min())

print("Maximum Value: ", age.max())
```

IN THIS EXAMPLE, THE first line imports the Pandas library and assigns it the alias "pd". The second line uses the "read_csv" function provided by Pandas to import the data from the "example_data.csv" file and store it in a variable called "data".

The third line uses the bracket notation to extract the variable "AGE" from the dataframe and store it in a variable called "age".

The fourth line uses the mean() function to calculate the mean of the variable "Age", the median() function to calculate the median of the variable, the mode() function to calculate the mode of the variable, the std() function to calculate the standard deviation of the variable, the min() function to calculate the minimum value of the variable, and the max() function to calculate the maximum value of the variable.

The results of the descriptive statistics can be used to understand the distribution of the variable "Age", for example, if the mean is close to the median, it indicates that the variable is distributed normally, if the mean and median are far apart it indicates that the variable is distributed skew. The standard deviation can also be used to understand the variability of the variable, where a low standard deviation indicates that the variable is distributed closely around the mean, while a high standard deviation indicates that the variable is distributed widely around the mean.

In this example, we have shown how to perform univariate analysis using the Pandas library, specifically calculating descriptive statistics of a variable. This is just one of the many techniques that can be used to perform univariate analysis, and different methods may work better depending on the characteristics of the data.

In conclusion, descriptive statistics in univariate analysis are essential for summarizing and describing the properties of a single variable. These statistics provide valuable insights into the distribution of the data and help to identify any patterns or outliers. Data analysts and data scientists use these measures to make informed decisions about the data and to build models for data prediction and forecasting.

Histograms

HISTOGRAMS ARE A GRAPHICAL representation of the distribution of numerical data. They display the frequency or proportion of values that fall within specific ranges or bins. In univariate analysis, histograms are used to visualize the distribution of a single variable.

For example, let's say we have a dataset that contains the heights of a group of people. We can create a histogram to visualize the distribution of these heights. The histogram will show the frequency or proportion of people with heights in each bin.

To create a histogram in Python, we can use the Matplotlib library. Here's an example code snippet:

```python
import matplotlib.pyplot as plt

import numpy as np

# Generate some random data

data = np.random.normal(size=1000)

# Create histogram

plt.hist(data, bins=30, alpha=0.5, color='b')

# Add labels and title

plt.xlabel('Height')

plt.ylabel('Frequency')

plt.title('Distribution of Heights')
```

Show plot

plt.show()

THE OUTPUT LOOK LIKE this:
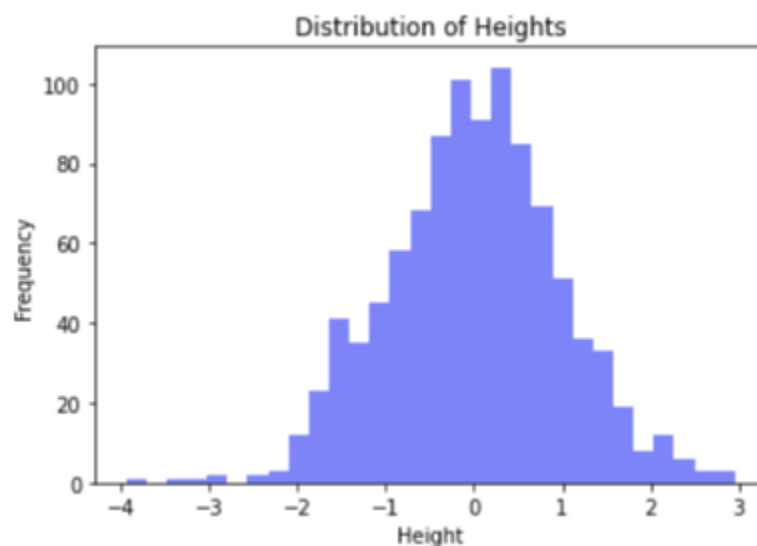
IN THIS EXAMPLE, WE first generate some random data using the NumPy library. We then create a histogram using the **plt.hist()** function, specifying the number of bins to use, the transparency and color of the bars, and other properties. We then add labels and a title to the plot and display it using **plt.show()**.

The resulting histogram will display the distribution of the heights in the data, with the x-axis showing the height range

and the y-axis showing the frequency or proportion of people with heights in that range.

Histograms are a useful tool in univariate analysis as they allow us to quickly visualize the distribution of a variable and identify patterns or outliers in the data.

Box plots

IN STATISTICS, UNIVARIATE analysis involves the examination of a single variable. One way to visually summarize the distribution of a single variable is through box plots. A box plot, also known as a box and whisker plot, displays a summary of the data's median, quartiles, and range. This graph is useful in identifying outliers and comparing the distribution of different datasets.

To construct a box plot, you need the minimum value, lower quartile (Q1), median, upper quartile (Q3), and maximum value. The interquartile range (IQR) is the distance between the upper and lower quartiles. The box represents the IQR, with the median shown as a line inside the box. The whiskers extend from the box to the minimum and maximum values, excluding outliers.
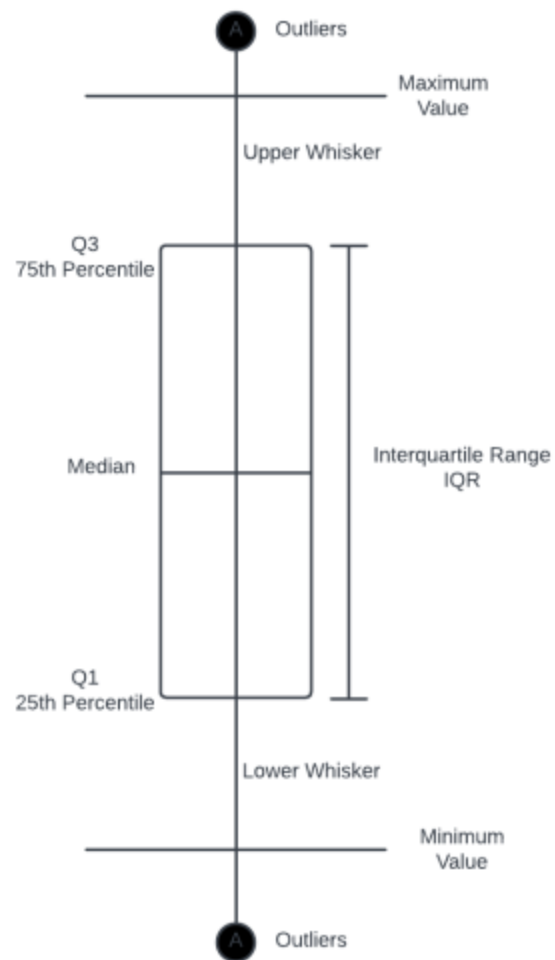
HERE IS AN EXAMPLE of a box plot for a dataset of exam scores:

Dataset: 40, 60, 70, 75, 80, 85, 90, 95, 100

Minimum value: 40

Lower quartile (Q1): 70

Median: 80

Upper quartile (Q3): 90

Maximum value: 100

IQR**:** Q3 **-** Q1 **=** 20

40 60 70 75 80 85 90 95 100

| |___| |___|

Q1 Median Q3

Whiskers**:** 40 **and** 100

Outliers**: None**

———————

IN THIS EXAMPLE, THE dataset has a minimum value of 40, a lower quartile of 70, a median of 80, an upper quartile of 90, and a maximum value of 100. The IQR is calculated as 20. The box plot shows that the majority of the data falls between 70 and 90, with two scores outside this range (75 and 100).

Box plots can also be used to compare multiple datasets. In the case of multiple datasets, each box plot is drawn side by side, making it easy to visually compare their median, IQR, and range. This comparison can help to identify differences in the distribution of the data.

here is an example of how to create a box plot in Python using the Matplotlib library:

import matplotlib**.**pyplot **as** plt

import numpy **as** np

```python
# generate some random data

data = np.random.normal(size=100)

# create a box plot of the data

fig, ax = plt.subplots()

ax.boxplot(data)

# set the title and labels

ax.set_title('Box Plot of Random Data')

ax.set_ylabel('Values')

# show the plot

plt.show()
```

THE OUTPUT WILL LOOK like this

IN THIS EXAMPLE, WE first import the necessary libraries: **matplotlib** and **numpy**. We then generate some random data using the **numpy.random.normal()** function. This generates an array of 100 values sampled from a normal distribution with a mean of 0 and standard deviation of 1.

Next, we create a figure and axis object using the **subplots()** function. We then call the **boxplot()** function on the axis object, passing in the data as a parameter. This creates a box plot of the data.

We then set the title and y-axis label using the **set_title()** and **set_ylabel()** methods on the axis object, respectively.

Finally, we call the **show()** function to display the plot.

The resulting plot should show a box with a line in the middle (the median), a box on either side of the median representing the interquartile range (IQR), and whiskers extending from the boxes to show the minimum and maximum values that are within 1.5 times the IQR from the box. Any values beyond the whiskers are considered outliers and are plotted as individual points.

In summary, box plots are a useful tool for visualizing the distribution of a single variable or comparing the distributions of multiple variables. They provide a concise summary of the

data's median, quartiles, and range, as well as identifying potential outliers.

Bivariate Analysis

ONCE THE UNIVARIATE analysis is complete, the next step is to perform bivariate analysis, which involves **analyzing the relationship between two variables**. This helps to understand how the variables are related and identify any patterns or correlations in the data. Some common techniques used in bivariate analysis include:

Scatter plots

IN STATISTICAL ANALYSIS, bivariate analysis refers to the analysis of two variables. One common method of visualizing bivariate data is through a scatter plot. A scatter plot is a graph in which the values of two variables are plotted along two axes, with each individual data point represented by a dot on the graph. The position of the dot on the graph indicates the value of the two variables for that particular data point.

For example, suppose we want to analyze the relationship between the height and weight of a group of individuals. We can create a scatter plot with height on the x-axis and weight on the y-axis, with each individual represented by a dot on the graph.

In the resulting scatter plot, each dot represents a single individual, and the position of the dot on the graph indicates both the height and weight of that individual. By looking at the scatter plot, we can see the overall pattern of the relationship between the two variables.

If the dots on the graph form a roughly linear pattern, we can say that there is a positive correlation between the two variables, which means that as one variable increases, so does the other. On the other hand, if the dots on the graph are spread out randomly with no discernible pattern, we can say that there is no correlation between the two variables.

In addition to examining the overall pattern of the relationship between the two variables, scatter plots can also be used to identify outliers, which are individual data points that fall far outside the range of the other data points.

Here's an example code for creating a scatter plot using the **matplotlib** library in Python:

```python
import matplotlib.pyplot as plt

# Sample data

x = [1, 2, 3, 4, 5, 6, 7, 8, 9, 9, 8, 7, 6, 5, 4, 3, 3]

y = [3, 5, 2, 7, 4, 2, 4, 5, 5, 6, 2, 3, 4, 5, 7, 2, 1]

# Create scatter plot

plt.scatter(x, y)
```

```
# Add axis labels and title

plt.xlabel('X-axis')

plt.ylabel('Y-axis')

plt.title('Scatter Plot Example')

# Show the plot

plt.show()
```

IN THIS EXAMPLE, WE first import the **matplotlib.pyplot** module and create two lists of data, **x** and **y**. We then call the **scatter()** function from **matplotlib.pyplot** to create the scatter plot using the data.

Next, we add axis labels and a title to the plot using the **xlabel()**, **ylabel()**, and **title()** functions.

Finally, we call the **show()** function to display the plot.

Scatter Plot Example
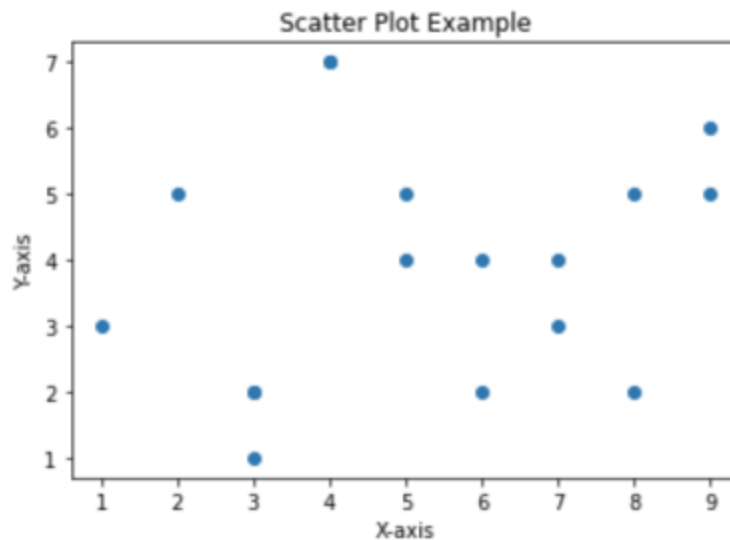
THE RESULTING SCATTER plot will have the **x** values on the horizontal axis and the **y** values on the vertical axis, with a point representing each pair of values. The axis labels and title provide additional information about the data being plotted.

Overall, scatter plots provide a useful way to explore the relationship between two variables and identify any patterns or outliers in the data.

Correlation

IN STATISTICS, CORRELATION is a measure of the strength and direction of the relationship between two variables. Correlation analysis is used in bivariate analysis to identify the extent to which two variables are related to each other.

For example, suppose we are interested in examining the relationship between the age of a car and its price. We can use correlation analysis to determine if there is a relationship between these two variables, and if so, whether it is a positive or negative relationship.

Correlation is usually measured using a correlation coefficient, which ranges from -1 to +1. A coefficient of +1 indicates a perfect positive correlation, a coefficient of -1 indicates a perfect negative correlation, and a coefficient of 0 indicates no correlation.

There are two main types of correlation: Pearson correlation and Spearman correlation. Pearson correlation is used when both variables are normally distributed, while Spearman correlation is used when one or both variables are not normally distributed.

Here is an example of using Pearson correlation to examine the relationship between the age of a car and its price in Python:

```python
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

# create a sample dataset

age = np.array([1, 3, 5, 7, 9])
```

```python
price = np.array([10000, 9000, 7000, 5000, 3000])

# calculate the correlation coefficient

corr_coef = np.corrcoef(age, price)[0, 1]

# plot the data and the regression line

plt.scatter(age, price)

plt.plot(age, np.poly1d(np.polyfit(age, price, 1))(age))

plt.xlabel('Age (years)')

plt.ylabel('Price ($)')

plt.title(f'Correlation Coefficient: {corr_coef:.2f}')

plt.show()
```

IN THIS EXAMPLE, WE first create a sample dataset with the age of the car in years and its price in dollars. We then calculate the Pearson correlation coefficient using the **np.corrcoef** function, and plot the data using **plt.scatter**. Finally, we plot the regression line using **np.polyfit** and **plt.plot**, and display the correlation coefficient in the title using string formatting.

Correlation Coefficient: -0.99
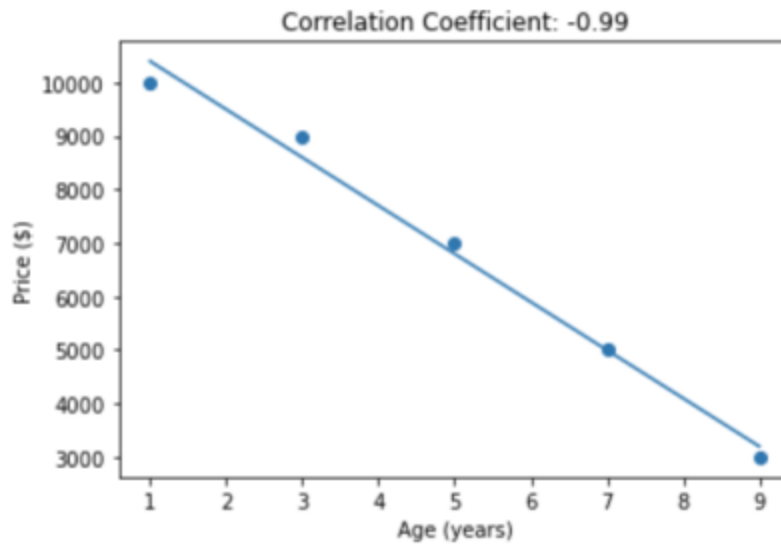
THE RESULTING PLOT shows that there is a strong negative correlation between the age of the car and its price, with a correlation coefficient of -0.99. This indicates that as the age of the car increases, its price decreases.

This is just one of the many techniques that can be used to perform bivariate analysis, and different methods such as heatmap, line plot, bar plot, etc. may work better depending on the characteristics of the data. It's important to note that bivariate analysis is a powerful tool that can be used to identify patterns, relationships, and correlations between variables, which can be useful for understanding the underlying structure of the data and guiding further analysis.

Multivariate Analysis

AFTER PERFORMING UNIVARIATE and bivariate analysis, the next step is to perform multivariate analysis, which involves analyzing the relationship among three or more variables. This helps to understand how the variables are related and identify any patterns or correlations in the data. Some common techniques used in multivariate analysis include:

Heatmaps

A HEATMAP IS A GRAPHICAL representation of data that uses a color-coding system to represent different values. It is a useful tool for visualizing the relationships between multiple variables in a dataset. In multivariate analysis, a heatmap is used to display the correlation between multiple variables, making it easier to identify patterns and trends in large datasets.

To create a heatmap in Python, we can use the **seaborn** library. First, we need to generate some random data to work with. We can use the **numpy** library to generate random data and set a seed to ensure that the data is the same each time the code is run.

Here is an example of how a heatmap can be used in multivariate analysis:

Suppose we want to understand the relationship between student performance and various factors such as study time,

family income, and parental education level. We can create a heatmap to visualize the correlation between these variables.

To generate the data, we can use the NumPy library and set a seed to ensure the data is consistent each time. Here is an example code:

```python
import numpy as np

import matplotlib.pyplot as plt

np.random.seed(42)

# generate random data

study_time = np.random.normal(6, 2, 100)

family_income = np.random.normal(50000, 15000, 100)

parent_education = np.random.normal(12, 3, 100)

test_score = study_time * 10 + family_income * 0.001 + parent_education * 3 + np.random.normal(0, 5, 100)

# plot heatmap

plt.figure(figsize=(8, 6))

correlation_matrix = np.corrcoef([study_time, family_income, parent_education, test_score])

heatmap = plt.imshow(correlation_matrix, cmap='coolwarm', interpolation='nearest')

plt.colorbar(heatmap)

plt.xticks([0, 1, 2, 3], ['Study Time', 'Family Income', 'Parent Education', 'Test Score'])
```

```
plt.yticks([0, 1, 2, 3], ['Study Time', 'Family Income', 'Parent Education', 'Test
Score'])

plt.title('Correlation Matrix Heatmap')

plt.show()
```

IN THIS EXAMPLE, WE generate 100 random values for each of the variables using the **np.random.normal()** function, which generates values from a normal distribution. We then calculate the test score based on a linear combination of these variables plus some random noise. Finally, we use the **np.corrcoef()** function to calculate the correlation coefficients between the variables, which we plot using the **plt.imshow()** function.

Correlation Matrix Heatmap
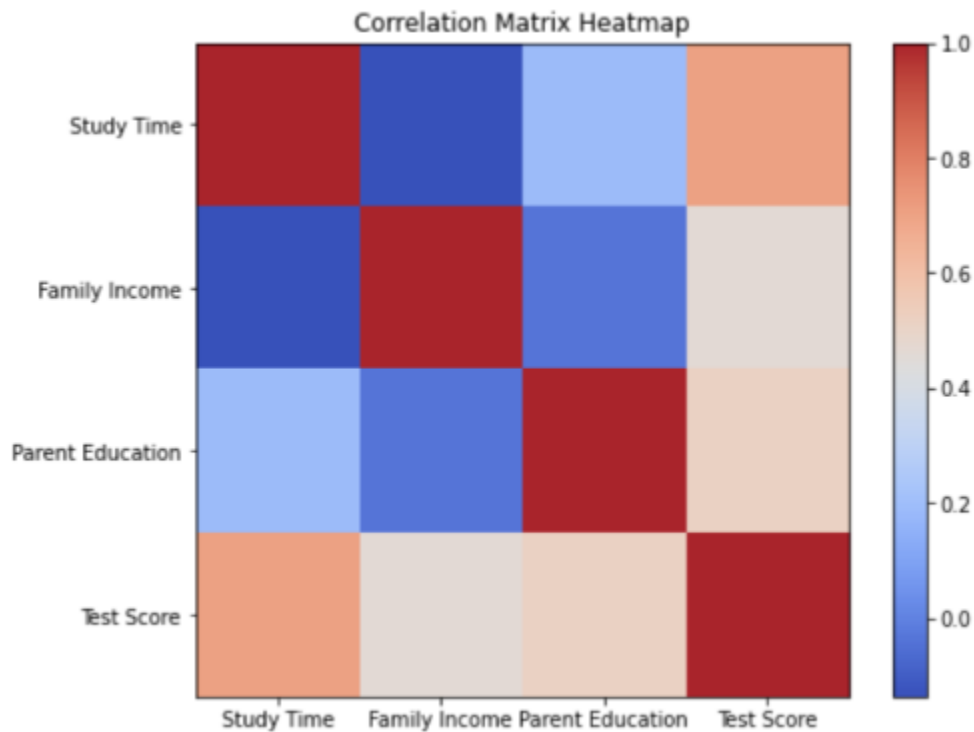
THE RESULTING HEATMAP will show the correlation between each pair of variables. The darker the color, the higher the correlation. For example, we can see that there is a strong positive correlation between study time and test score, and a weaker positive correlation between family income and test score. There is also a weak negative correlation between parent education and test score. Overall, the heatmap gives us a quick and easy way to visualize the relationship between multiple variables.

Parallel coordinates

Parallel coordinates is a powerful technique for visualizing high-dimensional data. In this technique, each variable is represented by a vertical axis, and a line is drawn connecting the values of each variable for each data point. This allows us to identify patterns and relationships between variables that may not be apparent in other types of visualizations.

Here's an example scenario where parallel coordinates can be used in multivariate analysis:

Suppose we have a dataset of customer reviews for a restaurant, with features such as food quality, service, ambiance, price, and overall rating. We want to explore the relationship between these features and the overall rating given by the customers.

We can create a parallel coordinates plot to visualize this relationship. Here's an example code using Python's matplotlib library:

```python
import numpy as np

import matplotlib.pyplot as plt

np.random.seed(42)

# Generate random data for 5 features and 100 samples

food_quality = np.random.randint(1, 6, size=100)

service = np.random.randint(1, 6, size=100)

ambiance = np.random.randint(1, 6, size=100)
```

```python
price = np.random.randint(1, 6, size=100)

overall_rating = np.random.randint(1, 6, size=100)

# Create parallel coordinates plot

fig, ax = plt.subplots()

plt.title('Customer Reviews for a Restaurant')

plt.xlabel('Features')

plt.ylabel('Rating')

plt.xticks(range(5), ['Food Quality', 'Service', 'Ambiance', 'Price', 'Overall
Rating'])

plt.ylim(0, 5)

plt.plot([0, 1, 2, 3, 4], [food_quality, service, ambiance, price, overall_rating],
color='blue', alpha=0.1)

plt.plot([0, 1, 2, 3, 4], [np.mean(food_quality), np.mean(service),
np.mean(ambiance), np.mean(price), np.mean(overall_rating)], color='red',
alpha=0.9)

plt.show()
```

IN THIS EXAMPLE, WE first generated random data for the five features and overall rating for 100 customer reviews, using the **numpy.random.randint()** function with a seed of 42 to ensure the same data is generated each time.

We then created a parallel coordinates plot using matplotlib's **plot()** function, where each feature is plotted as a separate

line and the overall rating is shown on the y-axis. The blue lines represent individual customer reviews, while the red line represents the mean rating for each feature.

Customer Reviews for a Restaurant

FROM THIS PLOT, WE can observe the following:

- Customers tend to rate food quality and service higher than ambiance and price.
- The overall rating is positively correlated with food quality and service, but less so with ambiance and price.
- There is a wide range of ratings for each feature, indicating that different customers have different preferences and priorities.

This type of visualization can be useful for quickly identifying patterns and relationships in multivariate data, and can help

guide further analysis and decision-making.

Cluster Analysis

CLUSTER ANALYSIS, ALSO known as clustering, is a technique used in multivariate analysis to group a set of objects in a way that objects in the same group (called a cluster) are more similar to each other than to those in other groups. The objective of cluster analysis is to find a structure or pattern in the data that can be useful in discovering relationships or identifying groups within the data.

Cluster analysis can be used for a wide range of applications in different fields, such as market segmentation, image segmentation, anomaly detection, customer profiling, biological classification, and many others.

There are different methods of cluster analysis, but the most common ones are hierarchical clustering and k-means clustering.

Hierarchical Clustering

HIERARCHICAL CLUSTERING is a method of cluster analysis that builds a hierarchy of clusters by recursively dividing the data set into smaller and smaller groups. There are two types of hierarchical clustering: agglomerative and divisive. Agglomerative clustering starts with each data point as its own cluster and then successively merges the closest pairs of

clusters until all the data points belong to a single cluster. Divisive clustering starts with all the data points in a single cluster and then successively divides the cluster into smaller clusters until each data point belongs to its own cluster.

Here's an example of hierarchical clustering using Python's **scipy** library:

```python
import numpy as np

from scipy.cluster import hierarchy

import matplotlib.pyplot as plt

# Generate some random data

np.random.seed(123)

x = np.random.rand(10, 2)

# Perform hierarchical clustering

dendrogram = hierarchy.dendrogram(hierarchy.linkage(x, method='ward'))

# Visualize the dendrogram

plt.title('Hierarchical Clustering Dendrogram')

plt.xlabel('Data point')

plt.ylabel('Distance')

plt.show()
```

IN THIS EXAMPLE, WE first generate a random dataset with 10 data points and 2 features. We then use the **hierarchy.linkage** function from **scipy** to perform hierarchical clustering using the "ward" method. Finally, we use the **hierarchy.dendrogram** function to generate a visualization of the dendrogram.

Hierarchical Clustering Dendrogram
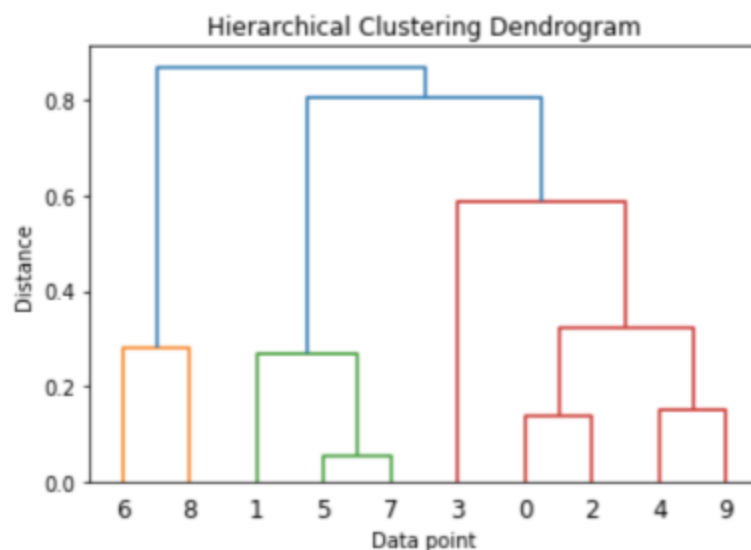
THE RESULTING PLOT shows the hierarchical clustering dendrogram, where the y-axis represents the distance between the clusters being merged and the x-axis represents the data points being clustered. The height of each horizontal line in the dendrogram indicates the distance between the two clusters being merged. We can use this plot to determine the optimal number of clusters to use in our analysis, by

selecting a horizontal line that cuts through the longest vertical line without intersecting any other lines.

K-means Clustering

K-MEANS CLUSTERING is a method of cluster analysis that partitions the data set into k clusters by minimizing the sum of squared distances between each data point and the centroid of its cluster. The algorithm starts by randomly assigning each data point to one of the k clusters, and then iteratively updates the centroids and re-assigns the data points to the closest cluster until convergence.

Here's an example of how to perform cluster analysis using K-Means algorithm in Python:

```python
# Import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
# Generate some random data
np.random.seed(123)
x = np.random.normal(0, 1, 100)
y = np.random.normal(0, 1, 100)
z = np.random.normal(0, 1, 100)
```

```python
data = pd.DataFrame({'X': x, 'Y': y, 'Z': z})

# Perform cluster analysis

kmeans = KMeans(n_clusters=3, random_state=0).fit(data)

labels = kmeans.labels_

centroids = kmeans.cluster_centers_

# Plot the clusters

colors = ['r', 'g', 'b']

fig = plt.figure(figsize=(10, 10))

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

for i in range(len(data)):

ax.scatter(data['X'][i], data['Y'][i], data['Z'][i], c=colors[labels[i]], alpha=0.8)

ax.scatter(centroids[:, 0], centroids[:, 1], centroids[:, 2], marker='*', s=300,
c='#050505')

ax.set_xlabel('X')

ax.set_ylabel('Y')

ax.set_zlabel('Z')

plt.show()
```

———————

IN THIS EXAMPLE, WE first import the necessary libraries including **numpy**, **pandas**, **matplotlib**, and **sklearn.cluster** which contains the KMeans algorithm.

We then generate random data using **numpy.random.normal** function and create a Pandas dataframe. We then perform cluster analysis on this data using the KMeans algorithm with **n_clusters=3** (i.e. we want to divide the data into 3 clusters) and **random_state=0** (to make the results reproducible).

Finally, we plot the clusters using **matplotlib** where each cluster is shown in a different color and the centroids of the clusters are shown as stars. The plot is in 3D to show the clusters in a multi-dimensional space.
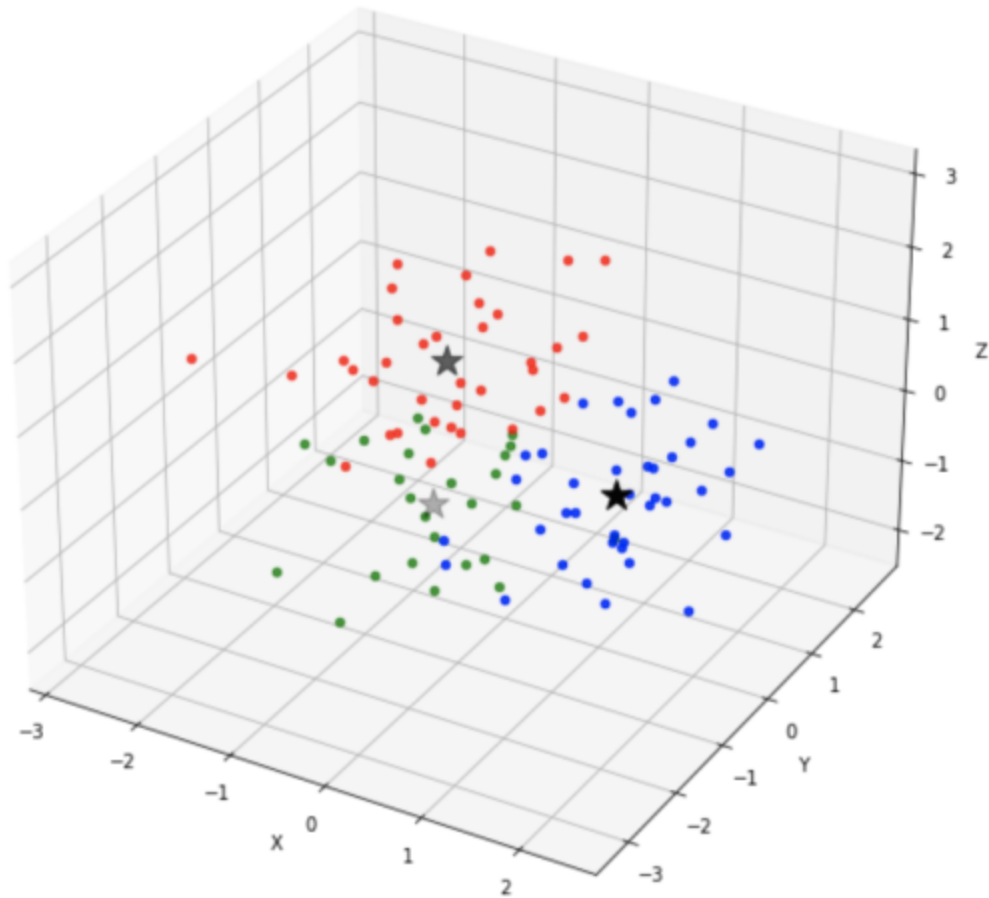
THIS IS JUST A SIMPLE example, but cluster analysis can be used in various fields like customer segmentation, fraud detection, image segmentation, and more.

Both hierarchical clustering and k-means clustering have their strengths and weaknesses, and the choice of method depends on the specific problem and data set.

For example, let's say we have a data set of customers who purchased different products from an online store. We want to

segment the customers into different groups based on their purchase behavior. We can use cluster analysis to identify groups of customers who are similar in their purchase behavior, and then use this information to tailor marketing campaigns and product offerings to each group.

In this case, we can use k-means clustering to partition the customers into k clusters based on their purchase behavior, and then use the cluster centroids to describe the purchase behavior of each group. We can also use hierarchical clustering to build a hierarchy of clusters, which can be useful for exploring the structure of the data and identifying natural breaks in the data set.

Principal Component Analysis (PCA)

THIS IS A DIMENSIONALITY reduction technique that can be used to identify the underlying structure of the data and reduce the number of variables. We will discuss this in more detail in future chapter under "Unsupervised Learning".

It's important to note that multivariate analysis can help to understand the relationship between multiple variables and identify patterns and correlations that may not be apparent in univariate or bivariate analysis.

Data Visualization

DATA VISUALIZATION is the process of creating graphical representations of data to make it more interpretable and understandable. It is an important aspect of Exploratory Data Analysis (EDA), as it allows us to identify patterns and relationships in the data that might be difficult to detect using numerical methods alone. There are many different types of data visualizations that can be used in EDA, such as line plots, bar plots, scatter plots, and heat maps. In this section, we will discuss the importance of data visualization, the different types of data visualizations, and best practices for creating effective data visualizations.

Importance of Data Visualization

Data visualization is an important aspect of data analysis because it allows us to quickly and easily identify patterns, relationships, and anomalies in the data. It can also help to communicate complex data to a non-technical audience, such as stakeholders or management. Additionally, data visualization can help to identify outliers or anomalies in the data, which can be important for identifying errors or inconsistencies in the data.

Types of Data Visualizations

There are many different types of data visualizations that can be used in EDA, each with their own strengths and

weaknesses. Some of the most common types of data visualizations include:

- Line plots: Line plots are used to visualize the relationship between two variables over time. They are useful for identifying trends and patterns in the data.
- Bar plots: Bar plots are used to visualize the distribution of a categorical variable. They are useful for comparing the distribution of the variable across different groups.
- Scatter plots: Scatter plots are used to visualize the relationship between two quantitative variables. They are useful for identifying patterns and correlations in the data.
- Heat maps: Heat maps are used to visualize the relationship between multiple variables. They are useful for identifying patterns and correlations in the data.
- Histograms: Histograms are used to visualize the distribution of a quantitative variable. They are useful for identifying the underlying distribution of the variable.
- Box plots: Box plots are used to visualize the distribution of a quantitative variable. They are useful for identifying outliers and the underlying distribution of the variable.

We already some of above visualization techniques such as scatter plot, heat map, box plot, histograms, correlation coefficients in previous sections. The

remaining visualization like line graph, bar graph will be explained in future sections.

Best Practices for Creating Effective Data Visualizations

When creating data visualizations, it is important to keep in mind certain best practices to ensure that the visualizations are effective and interpretable. Some of these best practices include:

- Use appropriate scales and axes: It is important to use appropriate scales and axes for the data being visualized, as this can affect the interpretability of the visualization. For example, if the data has a wide range of values, it may be necessary to use a logarithmic scale.
- Use meaningful labels and annotations: It is important to use meaningful labels and annotations for the visualization, as this can help to interpret the data. This includes labeling the x-axis, y-axis, and any other relevant information such as units of measurement.
- Use appropriate color schemes: It is important to use appropriate color schemes for the visualization, as this can affect the interpretability of the data. For example, using a colorblind-friendly color scheme can make the visualization more accessible to those with color vision deficiencies.

- Keep it simple: It is important to keep the visualization simple and avoid using unnecessary elements. This can help to improve the interpretability of the visualization.
- Use appropriate chart types: It is important to use the appropriate chart type for the data being visualized. For example, line plots are more appropriate for time series data, while bar plots are more appropriate for categorical data.

In conclusion, data visualization is an important aspect of data analysis that allows us to quickly and easily identify patterns, relationships, and anomalies in the data. There are many different types of data visualizations that can be used in EDA, such as line plots, bar plots, scatter plots, and heat maps. To create effective data visualizations, it is important to keep in mind best practices such as using appropriate scales and axes, meaningful labels and annotations, appropriate color schemes, keeping it simple, and using appropriate chart types. By following these best practices, we can create data visualizations that are interpretable, easy to understand, and effectively communicate the insights and information contained within the data. Data visualization is a powerful tool that can be used to gain a deeper understanding of the data, guide further analysis and inform decision making.

3.4 FEATURE ENGINEERING

⸻

Feature engineering is the process of transforming raw data into features that can be used in a machine learning model. It is a crucial step in the machine learning pipeline as the quality and nature of the features can greatly affect the performance of the model. The goal of feature engineering is to extract meaningful information from the raw data and create new features that can improve the predictive power of the model.

There are many different techniques that can be used in feature engineering, such as:

Feature extraction

FEATURE EXTRACTION is a technique in feature engineering that involves creating new features from the raw data by applying mathematical functions or algorithms. The goal is to extract meaningful information from the raw data and create new features that can improve the predictive power of the model. There are several ways to perform feature extraction, some of which include:

- **Mathematical functions**: Applying mathematical functions, such as square root, logarithm, or trigonometric functions, to a feature can extract new

information from the data. For example, taking the square root of a feature that represents the area of a house can create a new feature that represents the side length of the house.

- **Aggregations**: Grouping the data by a certain variable and calculating aggregate statistics, such as mean, median, or standard deviation, can create new features. For example, taking the mean of a feature that represents the temperature of a city over a period of time can create a new feature that represents the average temperature of the city.

- **Derived features**: Creating new features by combining or manipulating existing features. For example, creating a new feature that represents the ratio of two existing features, such as the ratio of income to expenditure.

- **Signal processing**: Applying signal processing techniques, such as Fourier transform or wavelet transform, to time series data can extract new features from the data. For example, applying a Fourier transform to a time series of stock prices can create new features that represent the frequencies of the price movements.

- **NLP techniques**: Applying natural language processing techniques to text data can extract new features. For example, counting the number of words in a sentence, or creating bag-of-words representations of a document.

An example of feature extraction is as follows: Consider a dataset that contains information about houses, including the number of bedrooms, number of bathrooms, square footage, and price. We would like to use this data to train a model to predict the price of a house based on the other features. One of the features "Square footage" may be very large and may not be useful in predicting the price of a house directly. Instead, we can extract a new feature from the square footage by applying a mathematical function such as square root. This can create a new feature that represents the side length of the house, which may be more meaningful and useful in predicting the price.

We can do this using the following code snippet in python using the Pandas library:

```python
import numpy as np

import pandas as pd

# Set random seed for reproducibility

np.random.seed(123)

# Generate random data for house pricing

num_houses = 1000

sqft = np.random.normal(1500, 250, num_houses)

bedrooms = np.random.randint(1, 6, num_houses)

bathrooms = np.random.randint(1, 4, num_houses)
```

```python
year_built = np.random.randint(1920, 2022, num_houses)

price = (sqft * 200) + (bedrooms * 10000) + (bathrooms * 7500) - (house_age *
1000) + np.random.normal(0, 50000, num_houses)

# Create pandas dataframe from random data

houses = pd.DataFrame({

'sqft': sqft,

'bedrooms': bedrooms,

'bathrooms': bathrooms,

'year_built': year_built,

'price': price

})

# # Extract the new feature

houses["house_age"] = 2022 - houses["year_built"]

print("New feature (house_age) after using feature extraction by mathematic
function is here: ")

print(houses["house_age"])

# Extract the new feature

houses["side_length"] = houses["sqft"].apply(np.sqrt)

print("New feature (side_length) after using feature extraction by mathematic
function is here: ")

print(houses["side_length"])
```

———————————————

IN THIS EXAMPLE, THE first line imports the Pandas library. After that, we create data using random function from numPy and create a data frame from that data. In the next few lines, we extracted two new variables namely house_age and side_length using mathematical functions.

This is just one example of how feature extraction can be used in practice. It's important to note that feature extraction should be done carefully and with domain knowledge, as it can greatly affect the performance of the model. Additionally, it's a good practice to test multiple different features and techniques to identify the optimal set of features for a given problem.

Feature selection

FEATURE SELECTION IS a technique in feature engineering that involves selecting a subset of the original features based on their relevance or importance for the prediction task. The goal is to select a subset of features that contain the most informative and useful information, while reducing the dimensionality of the data and avoiding the inclusion of irrelevant or redundant features. This can lead to improved model performance, faster training times, and more interpretable models.

There are several ways to perform feature selection, some of which include:

- **Filter methods**: Filter methods evaluate each feature independently and select a subset based on a certain criteria, such as correlation or mutual information. These methods are simple and fast, but they do not take into account the relationship between the features and the target variable.
- **Wrapper methods**: Wrapper methods evaluate the feature subset in combination with a specific model, and select a subset based on the performance of the model. These methods are more computationally expensive, but they take into account the relationship between the features and the target variable.
- **Embedded methods**: Embedded methods select features as part of the model training process. These methods are typically used in ensemble methods and tree-based models, such as random forests and gradient boosting.
- **Lasso regularization**: Lasso regularization is a technique that can be used to select features by adding a penalty term to the cost function that encourages the model to select a smaller number of features.

An example of feature selection using filter method is as follows: Consider a dataset that contains information about houses, including the number of bedrooms, number of bathrooms, square footage, and price. We would like to use

this data to train a model to predict the price of a house based on the other features. One way to perform feature selection is to use a filter method called correlation-based feature selection (CFS). This method selects a subset of features based on the correlation between the features and the target variable.

We can do this using the following code snippet in python using the scikit-learn library:

```python
import pandas as pd

from sklearn.feature_selection import SelectKBest

from sklearn.feature_selection import chi2, f_classif

# Import the data

data = pd.read_csv("house-prices.csv")

X = data.drop(columns = ["Price", "Neighborhood", "Brick"], axis=1)

y = data["Price"]

# Perform feature selection using CFS

selector = SelectKBest(score_func=f_classif, k=2)

X_new = selector.fit_transform(X, y)
```

IN THIS EXAMPLE, THE first two lines import the pandas and the feature selection module of scikit-learn. The third line uses the "read_csv" function provided by pandas to import

the data from the "house-prices.csv" file and store it in a variable called "data". The fourth line separates the predictor variables (X) from the target variable(y) and also drop the non-numeric columns. The fifth line instantiates the feature selection method by specifying the scoring function "f_classif" and the number of features to select "k=2" The sixth line applies the feature selection method to the data and returns a new dataset that contains only the selected features.

In the above example, the feature selection method used is correlation-based feature selection (CFS) which is a filter method. CFS calculates the correlation between each feature and the target variable and selects the k features with the highest correlation. It is a simple and fast method but it does not take into account the relationship between the features and the target variable, which may be important for certain types of data and problems.

Another example of feature selection method is Wrapper Method, which uses a specific model to evaluate the feature subset and select a subset based on the performance of the model. This method is more computationally expensive, but it takes into account the relationship between the features and the target variable. An example of wrapper method is Recursive Feature Elimination (RFE) which uses a specific model (such as SVM) and recursively removes the feature with the lowest coefficient until the desired number of features are selected.

It's important to note that feature selection should be an iterative process, and it's not uncommon to test multiple different feature selection techniques and subsets of features to identify the optimal set of features for a given problem. Additionally, feature selection should be done carefully and with domain knowledge, as it can greatly affect the performance of the model. It's also important to keep in mind that feature selection should be done after cleaning and preprocessing the data, so that the features selected are based on the actual characteristics of the data and not the noise.

Another important aspect of feature selection is evaluating the performance of the model after feature selection. It is important to use appropriate evaluation metric and compare the performance of the model with the selected features and the model with the original features. This can help to identify the optimal set of features that provide the best performance.

Keep in mind the interpretability of the model when performing feature selection. A model with fewer features is often more interpretable and easier to understand than a model with many features. This can be important for certain types of problems, such as medical diagnosis or financial forecasting, where the

interpretability of the model can be critical for making informed decisions.

There are also some advanced techniques for feature selection such as **genetic algorithm**, which is a optimization-based method inspired by the process of natural selection. It's an iterative method that uses the genetic algorithm to find the optimal subset of features.

In conclusion, feature selection is a technique in feature engineering that involves selecting a subset of the original features based on their relevance or importance for the prediction task. It can lead to improved model performance, faster training times, and more interpretable models. There are several ways to perform feature selection, such as filter methods, wrapper methods, embedded methods, and Lasso regularization. It's important to keep in mind that feature selection should be an iterative process, done carefully and with domain knowledge. It's also important to evaluate the performance of the model after feature selection and consider the interpretability of the model.

Feature scaling

FEATURE SCALING IS a technique in feature engineering that involves transforming the scale of a feature to a common range, such as between 0 and 1. This can be important for some machine learning algorithms that are sensitive to the

scale of the input features. Feature scaling is typically applied to the data before training a model, and it can have a significant impact on the performance of the model.

There are several ways to perform feature scaling, some of which include:

Min-Max scaling:

Min-Max scaling scales the data to a specific range, such as [0,1]. It is calculated by subtracting the minimum value of the feature from each value and then dividing by the range (max-min). This method is sensitive to outliers, so it's important to make sure that the data is cleaned before applying min-max scaling.

$$X_{new} = \frac{X_i - \min(X)}{\max(x) - \min(X)}$$

AN EXAMPLE OF FEATURE scaling using min-max scaling is as follows: Consider a dataset that contains information about houses, including the number of bedrooms, number of bathrooms, square footage, and price. We would like to use this data to train a model to predict the price of a house based on the other features. One of the features "Square footage" may have a much larger scale than the other features and may not be useful in predicting the price of a

house directly. Instead, we can scale the "Square footage" feature to a common range using min-max scaling.

We can do this using the following code snippet in python using the scikit-learn library:

```python
import pandas as pd

from sklearn.preprocessing import MinMaxScaler

# Import the data

data = pd.read_csv("house-prices.csv")

X = data.drop(columns = ["Price", "Neighborhood", "Brick"], axis=1)
```

———————

```python
# INITIALIZE THE SCALER

scaler = MinMaxScaler()
# Fit and transform the data

X_scaled = scaler.fit_transform(X[["SqFt"]])

print(X_scaled)
```

———————

IN THIS EXAMPLE, THE first two lines import the pandas and the MinMaxScaler module of scikit-learn. The third line uses the "read_csv" function provided by pandas to import the data from the "house-prices.csv" file and store it in a variable called "data". The fourth line separates the predictor variables (X) from the target variable. The fifth line

instantiates the MinMaxScaler object. The sixth line applies the min-max scaling method to the "SqFt" feature of the data and returns a new dataset that contains the scaled feature.

Min-Max scaling scales the data to a specific range, such as [0,1]. It is calculated by subtracting the minimum value of the feature from each value and then dividing by the range (max-min). This method is sensitive to outliers, so it's important to make sure that the data is cleaned before applying min-max scaling.

In this example, we first import the necessary libraries, and then we import our dataset " house-prices.csv" and separate the predictor variables (X) from the target variable. After that, we initialize the MinMaxScaler and then fit and transform the data. Toy can check the result by printing the final output using print(X_scaled).

Standardization:

Standardization scales the data to have a mean of 0 and a standard deviation of 1. It is calculated by subtracting the mean of the feature from each value and then dividing by the standard deviation. This method is not sensitive to outliers, but it assumes a normal distribution of the data, which may not be the case.

An example of feature scaling using standardization is as follows: Consider a dataset that contains information about houses, including the number of bedrooms, number of bathrooms, square footage, and price. We would like to use this data to train a model to predict the price of a house based on the other features. One of the features "Square footage" may have a much larger scale than the other features and may not be useful in predicting the price of a house directly. Instead, we can scale the "Square footage" feature to a common scale using standardization.

We can do this using the following code snippet in python using the scikit-learn library:

```python
import pandas as pd

from sklearn.preprocessing import StandardScaler

# Import the data

data = pd.read_csv("house-prices.csv")

X = data.drop(columns = ["Price", "Neighborhood", "Brick"], axis=1)

# Initialize the Scaler

scaler = StandardScaler()

# Fit and transform the data

X_scaled = scaler.fit_transform(X[["SqFt"]])

print(X['SqFt'], X_scaled)
```

IN THIS EXAMPLE, THE first two lines import the pandas and the StandardScaler module of scikit-learn. The third line uses the "read_csv" function provided by pandas to import the data from the "house-prices.csv" file and store it in a variable called "data". The fourth line separates the predictor variables (X) from the target variable and also drop the non-numeric columns. The fifth line instantiates the StandardScaler object. The sixth line applies the standardization method to the "SqFt" feature of the data and returns a new dataset that contains the scaled feature.

It is important to note that while standardization is not sensitive to outliers, it assumes a normal distribution of the data. Therefore, it's important to check the distribution of the data before applying standardization. If the data is not normally distributed, other methods such as min-max scaling or normalization may be more appropriate.

Normalization:

Normalization scales the data to have a minimum value of 0 (zero) and a maximum value of 1. It is calculated by subtracting the minimum value of the feature from each value and then dividing by the range (max-min). This method is sensitive to outliers, so it's important to make sure that the data is cleaned before applying normalization.

Normalization Formula

$$X_{normalized} = \frac{(x - x_{minimum})}{(x_{minimum} - x_{minimum})}$$

AN EXAMPLE OF FEATURE scaling using normalization is as follows: Consider a dataset that contains information about houses, including the number of bedrooms, number of bathrooms, square footage, and price. We would like to use this data to train a model to predict the price of a house based on the other features. One of the features "SqFt" may have a much larger scale than the other features and may not be useful in predicting the price of a house directly. Instead, we can scale the "SqFt" feature to a common scale using normalization.

We can do this using the following code snippet in python using the scikit-learn library:

```python
import pandas as pd

from sklearn.preprocessing import Normalizer

# Import the data

data = pd.read_csv("house-prices.csv")

X = data.drop(columns = ["Price", "Neighborhood", "Brick"], axis=1)

# Initialize the Scaler

scaler = Normalizer()
```

```
# Fit and transform the data

X_scaled = scaler.fit_transform(X[["SqFt"]])

print(X['SqFt'], X_scaled)
```

IN THIS EXAMPLE, THE first two lines import the pandas and the Normalizer module of scikit-learn. The third line uses the "read_csv" function provided by pandas to import the data from the "housing-prices.csv" file and store it in a variable called "data". The fourth line separates the predictor variables (X) from the target variable. The fifth line instantiates the Normalizer object. The sixth line applies the normalization method to the "SqFt" feature of the data and returns a new dataset that contains the scaled feature.
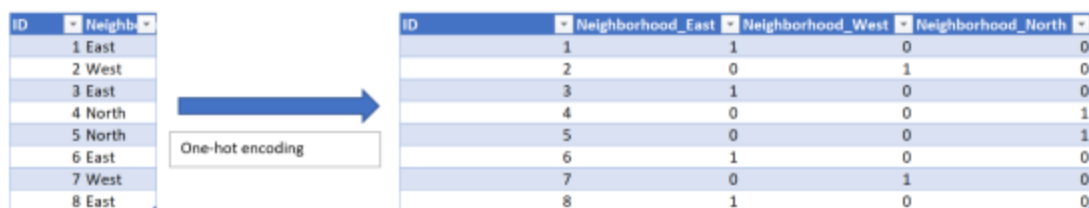
It's important to note that while normalization is not sensitive to outliers, it assumes that the data is already cleaned, and it can be affected by the presence of any outliers, it's important to check the data for outliers and remove them before applying normalization.

It's also important to keep in mind that normalization is typically used when the data is not normally distributed, and when the data has a similar scale but different ranges. This method is particularly useful in cases where the data has a large range of values, and we want to bring the values to a similar scale.

In summary, normalization is a technique that can be used to scale the features of a dataset to a common range. It is particularly useful when the data is not normally distributed, and when the data has a similar scale but different ranges. It's important to note that normalization is sensitive to outliers, so it's important to check the data for outliers and remove them before applying normalization, to avoid having any negative impact on the performance of the model.

One-hot encoding

ONE-HOT ENCODING IS a technique used in feature engineering to represent categorical variables as numerical values. It is particularly useful when working with categorical variables that have a small number of possible values, also called nominal variables.

In one-hot encoding, a new binary feature is created for each possible value of the categorical variable. The value of the new feature is 1 if the original variable has that value and 0 (zero) otherwise. For example, if we have a categorical variable called "Neighborhood" with three possible values "East", "West", and "North", three new binary features will be

created: "Neighborhood_East", "Neighborhood_West", and "Neighborhood_North". The value of "Neighborhood_East" will be 1 if the original Neighborhood variable is "East" and 0 otherwise. The same applies to "Neighborhood_West" and "Neighborhood_North".

We can do this using the following code snippet in python using the pandas library:

```python
import pandas as pd
# Import the data
data = pd.read_csv("house-prices.csv")
# Perform one-hot encoding on the "Neighborhood" variable
data = pd.get_dummies(data, columns=["Neighborhood"])
data.head()
```

IN THIS EXAMPLE, THE first line imports the pandas library, the second line imports the data from "house-prices.csv" and store it in a variable called "data". The third line uses the "get_dummies" function provided by pandas to perform one-hot encoding on the "Neighborhood" variable. The function creates new binary features for each possible value of the "Neighborhood" variable and adds them to the original dataset.

|	Home	Price	SqFt	Bedrooms	Bathrooms	Offers	Brick	Neighborhood_East	Neighborhood_North	Neighborhood_West
0	1	114300	1790	2	2	2	No	1	0	0
1	2	114200	2030	4	2	3	No	1	0	0
2	3	114800	1740	3	2	1	No	1	0	0
3	4	94700	1980	3	2	3	No	1	0	0
4	5	119800	2130	3	3	3	No	1	0	0

It's important to keep in mind that one-hot encoding can result in a large number of new features if the categorical variable has a large number of possible values. This can lead to a phenomenon called the "**Curse of Dimensionality**". In such cases, other encoding techniques such as **ordinal encoding** or **binary encoding** may be more appropriate.

In conclusion, One-hot encoding is a technique used in feature engineering to represent categorical variables as numerical values. It is particularly useful when working with categorical variables that have a small number of possible values, also called nominal variables. The technique creates new binary features for each possible value of the categorical variable, and it's a useful method for categorical feature representation in machine learning.

Binning

BINNING IS A TECHNIQUE used in feature engineering to group continuous variables into discrete bins or intervals. It is particularly useful when working with continuous variables

that have a large range of values, or when the distribution of the variable is not clear.

2 X 2 Binning
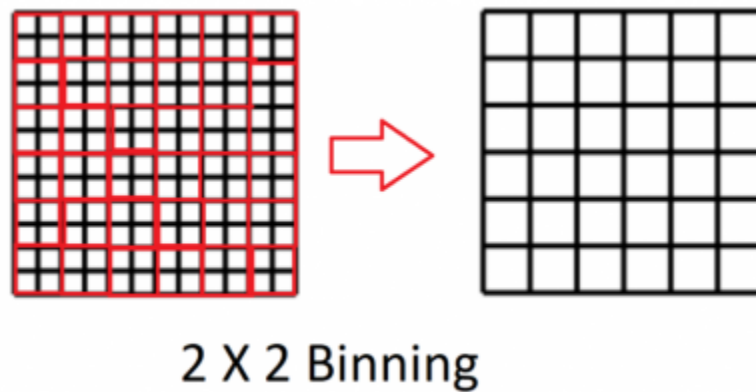
THE PROCESS OF BINNING involves dividing the range of a continuous variable into a specific number of bins or intervals, and then assigning each value of the variable to one of the bins. For example, if we have a continuous variable called "age" that ranges from 0 to 100, we can divide the range into 5 bins: (0-20), (20-40), (40-60), (60-80) and (80-100). Each value of the "age" variable will be assigned to one of the bins.

We can do this using the following code snippet in python using the pandas library:

```python
import pandas as pd

# Import the data
data = pd.read_csv("example_data.csv")
```

```
# Define the bins

bins = [0, 20, 40, 60, 80, 100]

# Create the binned variable

data["age_binned"] = pd.cut(data["AGE"], bins)
```
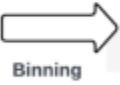
IN THIS EXAMPLE, THE first line imports the pandas library, the second line imports the data from "example_data.csv" and store it in a variable called "data". The third line defines the bins, in this case, the bins are defined as [0, 20, 40, 60, 80, 100]. The fourth line uses the "cut" function provided by pandas to create a new variable called "age_binned" that contains the binned values of the "age" variable. The function takes the "age" variable and the defined bins as inputs and assigns each value of the "age" variable to one of the bins.

Binning

AGE		age_binned
30		(20, 40]
35		(20, 40]
40		(20, 40]
45		(40, 60]
50	Binning	(40, 60]
55		(40, 60]
60		(40, 60]
65		(60, 80]
70		(60, 80]
75		(60, 80]

IT'S IMPORTANT TO KEEP in mind that the choice of the number of bins and the bin boundaries can have a significant impact on the performance of the model. A good rule of thumb is to use a larger number of bins for variables with a large range of values, and a smaller number of bins for variables with a small range of values. It's also important to check the distribution of the variable before binning to ensure that the binning makes sense for the data.

In conclusion, Binning is a technique used in feature engineering to group continuous variables into discrete bins or intervals. It is particularly useful when working with continuous variables that have a large range of values, or when the distribution of the variable is not clear. The process

of binning involves dividing the range of a continuous variable into a specific number of bins or intervals, and then assigning each value of the variable to one of the bins. It's important to keep in mind that the choice of the number of bins and the bin boundaries can have a significant impact on the performance of the model.

Binning can be useful in many situations, for example, it can be used to group age data in different age groups to analyze the data by age group, or it can be used to group salary data into different salary ranges to analyze the data by salary range.

Another advantage of binning is that it can reduce the effect of outliers by putting them in the same bin with similar values and this can improve the performance of the model.

It's important to note that feature engineering should be an iterative process, and it's not uncommon to test multiple different features and techniques to identify the optimal set of features for a given problem. Additionally, feature engineering should be done carefully and with domain knowledge, as it can greatly affect the performance of the model.

3.5 SPLITTING THE DATA INTO TRAINING AND TESTING SETS

─────

Splitting the data into training and testing sets is an important step in the machine learning process. It is used to evaluate the performance of the model on unseen data, which helps to prevent overfitting and to assess the generalization ability of the model.

The process of splitting the data involves randomly dividing the data into two subsets: a training set and a testing set. The training set is used to train the machine learning model, while the testing set is used to evaluate the performance of the model. The standard split ratio is typically 80% for the training set and 20% for the testing set, although this ratio can be adjusted depending on the specific needs of the project.

We can do this using the following code snippet in python using the scikit-learn library:

```
from sklearn.model_selection import train_test_split

# Import the data

data = pd.read_csv("example_data.csv")

X = data.drop("SALARY", axis=1)

y = data["SALARY"]
```

```
# Split the data

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

IN THIS EXAMPLE, THE first line imports the train_test_split function from the scikit-learn library. The second line imports the data from the "example_data.csv" file and store it in a variable called "data". The third line separates the predictor variables (X) from the target variable (y). The fourth line uses the train_test_split function to split the data into training and testing sets. The function takes the predictor variables, target variable, test size, and random_state as inputs. The test_size parameter is set to 0.2, which means that 20% of the data will be used for testing, and the rest for training. The random_state parameter is set to 42, which ensures that the data is split in the same way each time the code is executed.

It's important to keep in mind that the data should be randomly split to ensure that the training and testing sets are representative of the entire dataset, and that the model is tested on unseen data. The random_state parameter can be used to ensure that the data is split in the same way each time the code is executed. Additionally, it's also important to use stratified sampling techniques such as stratifiedKFold or StratifiedShuffleSplit when dealing with imbalanced datasets to ensure that the training and testing sets are representative

of the class distribution in the entire dataset (will discuss in detail in later chapter).

Furthermore, it's also important to keep in mind that the results of the model should be evaluated based on the performance on the test set, as this represents the performance of the model on unseen data. This will give a better idea of how well the model will perform when deployed in the real world.

3.6 SUMMARY

- Data preparation is an important step in the machine learning process as it ensures that the data is in a consistent and usable format for models.
- Techniques used in data preparation include: importing and cleaning data, exploratory data analysis, feature engineering, data visualization and splitting the data into training and testing sets.
- Importing data can be done using functions such as **pandas.read_csv()**, **pandas.read_excel()**, **pandas.read_json()** and others.
- Cleaning data can include tasks such as handling missing values, removing duplicate data, handling outliers, formatting data and normalizing and transforming data.
- Exploratory data analysis is the process of analyzing and summarizing the main characteristics of the data through visualizations and statistics.
- Feature engineering is the process of creating new features or transforming existing features to improve the performance of the model.
- Data visualization is the process of creating graphical representations of the data to better understand the underlying patterns and relationships.

3.7 TEST YOUR KNOWLEDGE

I. **What is the main purpose of data preparation in the machine learning process?**

a. To improve the performance of the model

b. To ensure the data is in a consistent format

c. To create new features

d. To analyze and summarize the main characteristics of the data

I. **What is one of the most common functions used for importing data into python?**
   a. **pandas.read_csv()**
   b. **pandas.read_html()**
   c. **pandas.read_sas()**
   d. **pandas.read_excel()**

II. **What is one of the ways to handle missing values in a DataFrame?**
   a. **Forward-fill**
   b. **Backward-fill**
   c. **Drop the rows**
   d. **All of the above**

III. **What is one of the ways to handle duplicate data in a DataFrame?**
   a. **Keep the first occurrence of duplicate rows**

b. Keep the last occurrence of duplicate rows

c. Drop the duplicate rows

d. All of the above

IV. What is one of the techniques used in Exploratory Data Analysis (EDA)?

    a. Univariate Analysis

    b. Bivariate Analysis

    c. Multivariate Analysis

    d. All of the above

V. What is feature engineering?

    a. Creating new features or transforming existing features

    b. Importing data

    c. Cleaning data

    d. Splitting data into training and testing sets

VI. What is one of the methods for data visualization?

    a. Bar chart

    b. Line chart

    c. Scatter plot

    d. All of the above

VII. What is the purpose of splitting the data into training and testing sets?

    a. To improve the performance of the model

    b. To evaluate the model's performance

    c. To ensure the data is in a consistent format

    d. To create new features

VIII. **What is the process of normalizing and transforming data?**
   a. **Changing the format of the data**
   b. **Scaling the data**
   c. **Changing the distribution of the data**
   d. **All of the above**

IX. **What is the importance of data preparation in the machine learning process?**
   a. **It ensures that the data is in a consistent and usable format for models**
   b. **It can improve the performance of the model**
   c. **It can create new features**
   d. **All of the above**

3.8 ANSWERS

I. Answer: b) To ensure the data is in a consistent format

I. Answer: a) pandas.read_csv()

I. Answer: d) All of the above

I. Answer: d) All of the above

I. Answer: d) All of the above

I. Answer: a) Creating new features or transforming existing features

I. Answer: d) All of the above

I. Answer: b) To evaluate the model's performance

I. Answer: d) All of the above

I. Answer: d) All of the above

04

4 SUPERVISED LEARNING

Supervised learning is a type of machine learning where the model is trained on labeled data to make predictions on new, unseen data. In supervised learning, the goal is to learn mapping from input features to output labels. The input features are often represented as a set of numerical or categorical values, while the output labels can be either continuous or discrete values. In this chapter, we will delve into the different types of supervised learning algorithms, and explore how to use scikit-learn to train, evaluate and improve models for different types of problems. We will cover topics such as linear and logistic regression, decision trees, and support vector machines, among others, as well as techniques for handling overfitting, underfitting, and improving model performance.

4.1 LINEAR REGRESSION

Linear regression is a supervised learning algorithm used for modeling the linear relationship between a dependent variable and one or more independent variables. The goal of linear regression is to find the best-fitting straight line through the data points. Linear regression can be used for both simple and multiple regression analysis.

Simple Linear Regression

IN SIMPLE LINEAR REGRESSION, there is only one independent variable and the line of best fit is represented by the equation:

$y = mx + b$
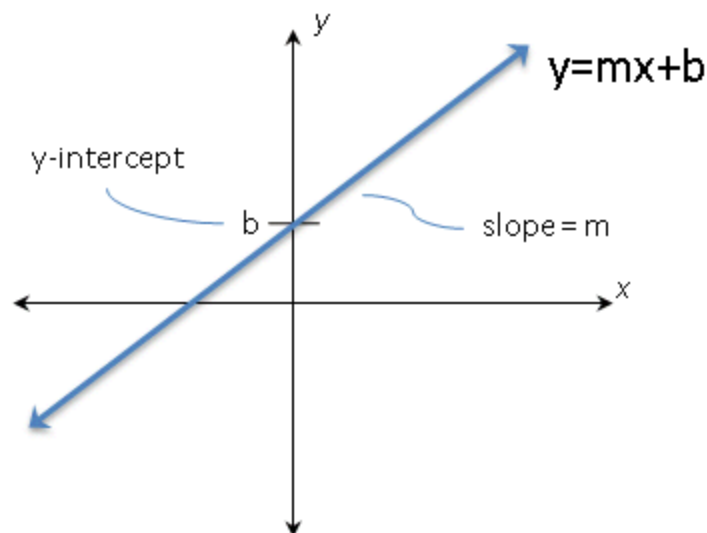
WHERE Y IS THE DEPENDENT variable, x is the independent variable, m is the slope of the line, and b is the y-intercept. The slope of the line represents the relationship between x and y, while the y-intercept represents the point at which the line crosses the y-axis.

Multiple Linear Regression

IN MULTIPLE LINEAR regression, there are two or more independent variables, and the line of best fit is represented by the equation:

$$y = b0 + b1x1 + b2x2 + ... + bnxn$$

Where y is the dependent variable, x1, x2, ..., xn are the independent variables, and b0, b1, b2, ..., bn are the coefficients of the line. Each coefficient represents the change in the dependent variable for a one-unit change in the corresponding independent variable, holding all other independent variables constant.

Linear regression can be used for both continuous and categorical dependent variables, but the independent variables must be continuous. The method of least squares is

used to find the coefficients of the line of best fit, which minimize the sum of the squared differences between the predicted and actual values.

Scikit-learn library in Python provides an easy implementation of linear regression via the LinearRegression class. The class provides several methods for model fitting and prediction such as **fit(), predict(), score()** etc. Linear regression can be used for various real-world problems such as predicting housing prices, stock prices, and many more.

Here's a coding example to illustrate linear regression using scikit-learn library and a randomly generated dataset:

```python
# Import required libraries

import numpy as np

from sklearn.linear_model import LinearRegression

from sklearn.model_selection import train_test_split

# Generate random dataset

np.random.seed(123)

num_samples = 100

x1 = np.random.normal(10, 2, num_samples) # Independent variable 1 (age)

x2 = np.random.normal(5, 1, num_samples) # Independent variable 2 (income)

y = 2*x1 + 3*x2 + np.random.normal(0, 1, num_samples) # Dependent variable (savings)

# Reshape independent variables
```

```python
X = np.column_stack((x1, x2))

# Split dataset into train and test sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=123)

# Create a Linear Regression model

model = LinearRegression()

# Fit the model on training data

model.fit(X_train, y_train)

# Predict the savings using the test set

y_pred = model.predict(X_test)

# Print model coefficients and intercept

print('Coefficients:', model.coef_)

print('Intercept:', model.intercept_)

# Print model performance metrics

from sklearn.metrics import mean_squared_error, r2_score

print('Mean squared error: %.2f' % mean_squared_error(y_test, y_pred))

print('R2 score: %.2f' % r2_score(y_test, y_pred))
```

THE OUTPUT WILL LOOK like this:

Coefficients: [2.00633802 3.0088422]

Intercept: -0.19265156903176717

Mean squared error: 0.85

R2 score: 0.95

In this example, we are generating a random dataset with 2 independent variables and 1 dependent variable. The independent variables are age and income, and the dependent variable is savings. We are assuming that savings is a linear combination of age and income, with some added noise.

We are using the **numpy** library to generate the random dataset, and the **LinearRegression** class from the **sklearn.linear_model** module to create a linear regression model.

We are then splitting the dataset into a training set and a test set using the **train_test_split** function from the **sklearn.model_selection** module.

We are fitting the linear regression model on the training data using the **fit** method, and using the **predict** method to predict the savings for the test set.

We are then printing the model coefficients and intercept, and evaluating the model performance using the mean squared error and R2 score metrics from the **sklearn.metrics** module.

Note that we have used meaningful and realistic

variable names to make the example easier to understand. We have also added a test-train split to evaluate the model's performance on unseen data.

After going through above example, you may have question that What is Mean Squared Error or What is R2 Score? Let's discuss these two concept first below:

What is Mean Squared Error?

MEAN SQUARED ERROR (MSE) is a popular metric used to measure the average squared difference between the actual and predicted values of a regression problem. It is a common evaluation metric used in regression analysis and is the average of the squared differences between predicted and actual values. The MSE provides a relative measure of how well a regression model fits the data. The lower the MSE, the better the model is at predicting the target variable.

Observed Values
Predicted Values
Regression Line
Y Scale Difference Between Observed and Predicted Values
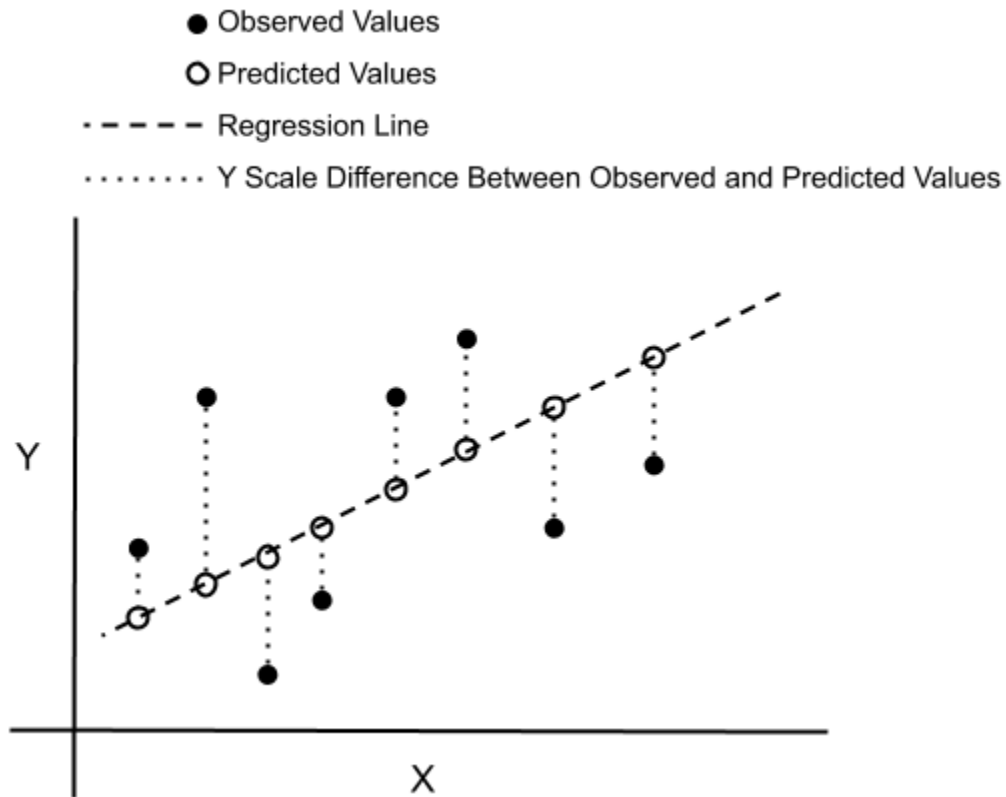
THE FORMULA FOR MSE is:

MSE = (1/n) * ∑(yi - ŷi)^2

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} \left(Y_i - \hat{Y}_i\right)^2$$

Mean **Error** **Squared**

WHERE:

- n: the number of observations in the dataset
- yi: the actual value of the target variable for the ith observation
- ŷi: the predicted value of the target variable for the ith observation

In essence, MSE measures the average squared distance between the predicted and actual values. By squaring the differences between the predicted and actual values, the metric is able to give higher weights to larger errors, providing a more accurate reflection of the overall performance of the model.

What is R2 Score?

R2 SCORE, ALSO KNOWN as the coefficient of determination, is a statistical measure that represents the proportion of variance in the dependent variable that can be explained by the independent variable(s).

R2 score is a value between 0 and 1, where a value of 1 indicates that the model perfectly fits the data, while a value of 0 indicates that the model does not explain any of the variability in the data.

The formula for R^2 score is:

$$R^2 = 1 - \frac{unexplained\ variation}{total\ variation} = 1 - \frac{\sum_i \left(y_i - \hat{y}_i\right)^2}{\sum_i \left(y_i - \bar{y}\right)^2}$$

Here's an example of calculating the R2 score using Python:

```
from sklearn.linear_model import LinearRegression

from sklearn.metrics import r2_score

import numpy as np

# Generate some random data

x = np.array([1, 2, 3, 4, 5, 6])

y = np.array([2, 4, 5, 6, 7, 8])

# Fit a linear regression model

model = LinearRegression().fit(x.reshape(-1,1), y)

# Predict the y values using the model

y_pred = model.predict(x.reshape(-1,1))

# Calculate the R2 score

r2 = r2_score(y, y_pred)

print(f"R2 score: {r2}")
```

IN THIS EXAMPLE, WE generated some random data and fit a linear regression model to it using scikit-learn's **LinearRegression** class. We then used the **r2_score**

function from scikit-learn's **metrics** module to calculate the R2 score of the model. The resulting R2 score tells us how well the model fits the data.

If you want to create a linear regression model on your own data, you can use the below code.

```python
from sklearn.linear_model import LinearRegression

from sklearn.model_selection import train_test_split

import pandas as pd

# read data

data = pd.read_csv("data.csv")

# split data into dependent and independent variables

X = data[['feature1', 'feature2', 'feature3']]

y = data['target']

# split data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

# create linear regression object

reg = LinearRegression()

# fit the model to the training data

reg.fit(X_train, y_train)

# predict the target variable for the test data

y_pred = reg.predict(X_test)
```

```
# check the accuracy of the model

score = reg.score(X_test, y_test)

print("Accuracy:", score)
```

IN THIS EXAMPLE, WE first import the necessary libraries: LinearRegression and train_test_split from scikit-learn and pandas for data manipulation.

We then read the data from a csv file using the pandas read_csv() method and split it into dependent and independent variables. The independent variables are stored in the X variable and the dependent variable is stored in the y variable.

Next, we split the data into training and testing sets using the train_test_split() method. The test_size parameter determines the proportion of the data that will be used for testing. In this case, we set it to 0.2, meaning 20% of the data will be used for testing.

We then create a LinearRegression object and fit the model to the training data using the fit() method.

We then use the predict() method to predict the target variable for the test data and store it in the y_pred variable.

Finally, we use the score() method to check the accuracy of the model on the test data. The score ranges from 0 to 1, with a higher score indicating a better fit. In the above example, we are assuming that the data is in a form of csv file, but it can be in different formats as well.

It's important to note that while this code gives an idea on how to implement linear regression using scikit-learn but it is not a complete implementation and may require additional preprocessing and feature selection steps depending on the nature of the data. Also, the accuracy of the model may be affected by the assumptions that are made in Linear Regression like linearity and normality of data.

Linear regression assumes linearity between independent and dependent variables and can't capture complex non-linear relationships. Also, it assumes that the data is free of outliers and follows a normal distribution. In case of violation of these assumptions, the results may be unreliable.

One real-life application of linear regression is in the field of finance, where it can be used to predict stock prices. By analyzing historical stock data such as price, volume, and company performance, a linear regression model can be trained to predict future stock prices. This can be useful for investors looking to make informed buying and selling decisions. Another example is in the field of real estate,

where linear regression can be used to predict property prices based on factors such as location, square footage, and number of bedrooms. This can help buyers and sellers make more informed decisions about buying or selling a property. In general, linear regression can be applied in any domain where there is a need to predict a continuous variable based on one or more independent variables.

4.2 LOGISTIC REGRESSION

Logistic regression is a type of supervised machine learning algorithm used for classification tasks. It is used to predict the probability of an outcome belonging to a particular class. The goal of logistic regression is to find the best fitting model to describe the relationship between the independent variables and the dependent variable, which is binary in nature (0/1, true/false, yes/no).

The logistic regression model is an extension of the linear regression model, with the main difference being that the outcome variable is dichotomous, and the linear equation is transformed using the logistic function, also known as the sigmoid function. The logistic function produces an S-shaped curve, which allows the model to predict the probability of the outcome belonging to one class or the other. The difference between linear and logistic regression can be seen visually below:

Linear Regression / Logistic Regression
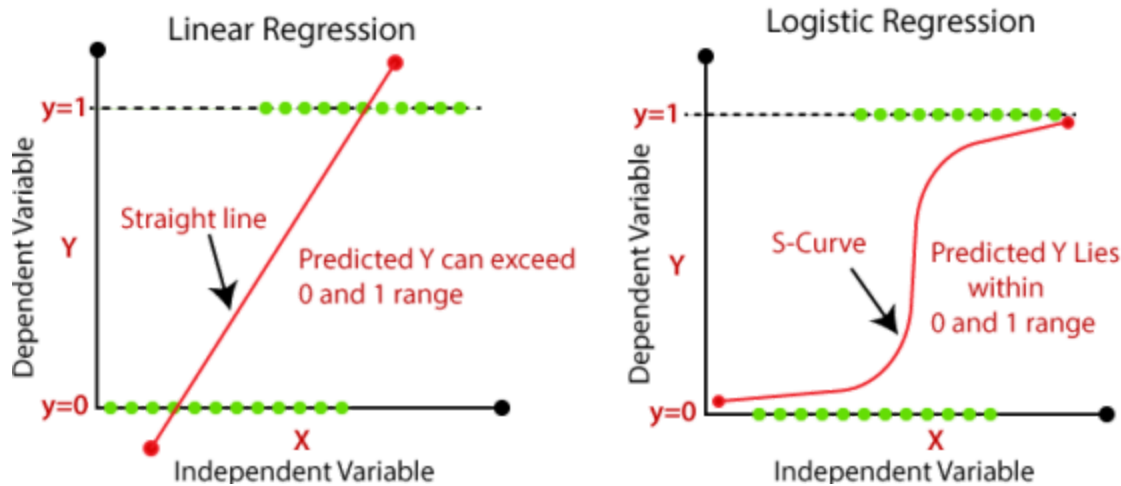
The logistic regression model estimates the probability that a given input belongs to a particular class, using a probability threshold. If the predicted probability is greater than the threshold, the input is classified as belonging to the class. Otherwise, it is classified as not belonging to the class.

One of the main advantages of logistic regression is that it is easy to implement and interpret. It also does not require a large sample size, and it is robust to noise and outliers. However, it can be prone to overfitting if the number of independent variables is large compared to the number of observations. Logistic regression is used in various fields such as healthcare, marketing, and social sciences to predict the likelihood of a certain event happening.

Coding example:

Here is an example of how to implement logistic regression using Python's Scikit-learn library:

```python
import numpy as np

from sklearn.linear_model import LogisticRegression

from sklearn.model_selection import train_test_split

# Generate random dataset

np.random.seed(123)

age = np.random.normal(40, 10, 100)

income = np.random.normal(50000, 10000, 100)

education = np.random.choice([0, 1], size=100)

# Define dependent and independent variables

X = np.column_stack((age, income, education))

y = np.random.choice([0, 1], size=100)

# Split data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=123)

# Create logistic regression model

model = LogisticRegression()

# Train the model using the training data

model.fit(X_train, y_train)

# Make predictions on the test data

y_pred = model.predict(X_test)

# Print the accuracy of the model

print("Accuracy:", model.score(X_test, y_test))
```

IN THIS EXAMPLE, WE first generate a random dataset with three independent variables: age, income, and education. We also generate a binary outcome variable, y. We then split the data into training and testing sets using the **train_test_split** function from Scikit-learn.

Next, we create a logistic regression model using the **LogisticRegression** function from Scikit-learn. We then train the model using the training data by calling the **fit** function and passing in the independent variables and outcome variable.

Once the model is trained, we can use it to make predictions on the test data by calling the **predict** function and passing in the independent variables. Finally, we print the accuracy of the model on the test data using the **score** function.

The output of the code will be the accuracy of the logistic regression model on the test data.

One of the key features of logistic regression is that it estimates the probability of an instance belonging to a particular class. The class with the highest probability is the one that the instance is assigned to. This makes logistic regression a popular choice for binary classification problems, such as spam detection and medical diagnosis.

The logistic regression algorithm is trained by maximizing the likelihood of the observed data given the model parameters. The maximum likelihood estimates of the model parameters are then used to make predictions on new data.

One of the main advantages of logistic regression is that it is a simple and interpretable model. It is easy to understand and explain the relationship between the input features and the output class. It also has a low variance, making it less prone to overfitting compared to other models.

However, logistic regression does have some limitations. It assumes that the relationship between the input features and the output class is linear, which may not be the case for all problems. Additionally, it is sensitive to outliers and may not perform well when the data is highly imbalanced.

4.3 DECISION TREES

Decision trees are a popular and widely used supervised learning algorithm for both classification and regression problems. The algorithm creates a tree-like model of decisions and their possible consequences, with the goal of correctly classifying or predicting the outcome of new instances.

At the top of the tree is a root node that represents the entire dataset. The root node is then split into two or more child nodes, each representing a subset of the data that has certain characteristics. This process continues recursively until the leaf nodes are reached, which represent the final decision or prediction.
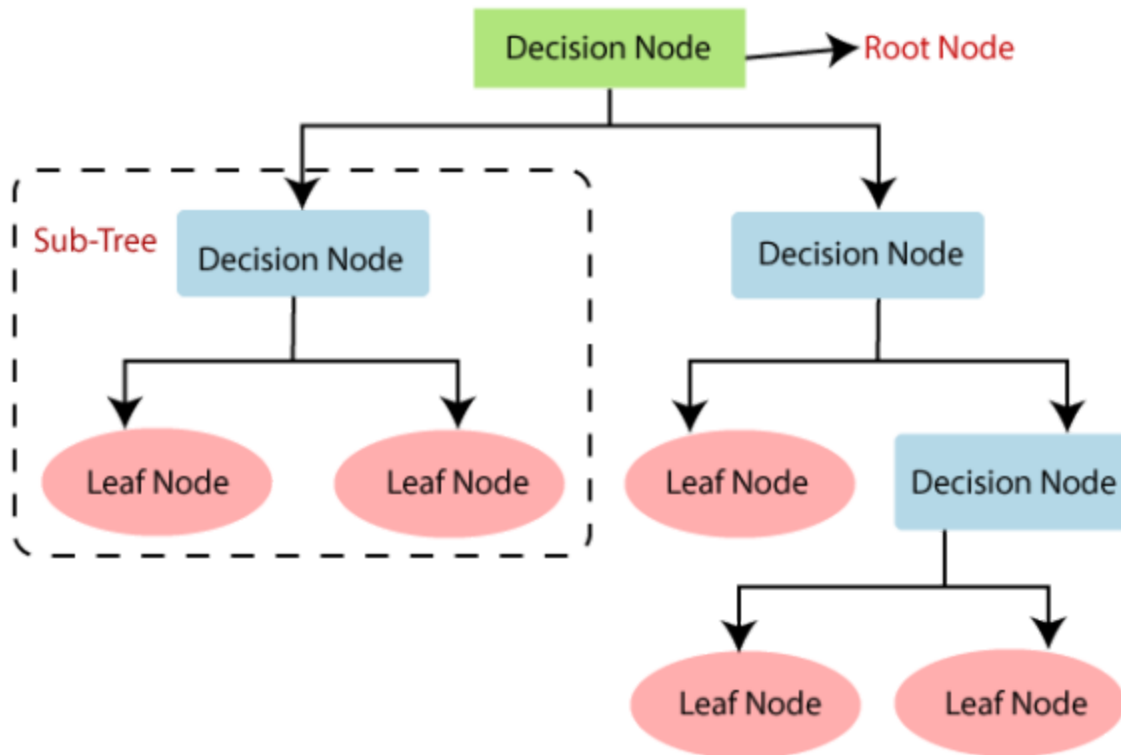
One of the main advantages of decision trees is their interpretability. The tree structure makes it easy to understand the logic behind the predictions and the decision-making process. Additionally, decision trees can handle both categorical and numerical data and can handle missing values without the need for imputation.

However, decision trees also have some limitations. They can easily overfit the training data, especially if the tree is allowed to grow deep. To prevent overfitting, techniques such as pruning, limiting the maximum depth of the tree, or using ensembles of trees like random forests can be used.

Decision trees are also sensitive to small changes in the data. A small change in the training data can lead to a completely different tree being generated. This can be mitigated by using ensembles of trees like random forests, which average the predictions of multiple trees to reduce the variance.

Here is an example of how to train and use a decision tree classifier using the scikit-learn library in Python:

```python
import numpy as np

from sklearn.tree import DecisionTreeClassifier, plot_tree

import matplotlib.pyplot as plt

# Generate a random dataset with 4 independent variables and 1 dependent variable

np.random.seed(42)

age = np.random.randint(18, 65, size=100)

income = np.random.randint(20000, 150000, size=100)

education = np.random.randint(0, 4, size=100)

occupation = np.random.randint(0, 5, size=100)

loan_approved = np.random.randint(0, 2, size=100)

# Combine the independent variables into a feature matrix X

X = np.column_stack((age, income, education, occupation))

# Assign the dependent variable to y

y = loan_approved
```

```python
# Split the dataset into training and testing sets

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Fit the decision tree classifier to the training set

clf = DecisionTreeClassifier(max_depth=3, random_state=42)

clf.fit(X_train, y_train)

# Make predictions on the testing data

y_pred = clf.predict(X_test)

# Print the accuracy score

print("Accuracy:", np.mean(y_pred == y_test))

# Visualize the decision tree

fig, ax = plt.subplots(figsize=(12, 8))

plot_tree(clf, filled=True, feature_names=['age', 'income', 'education',
'occupation'], class_names=['rejected', 'approved'], ax=ax)

plt.show()
```
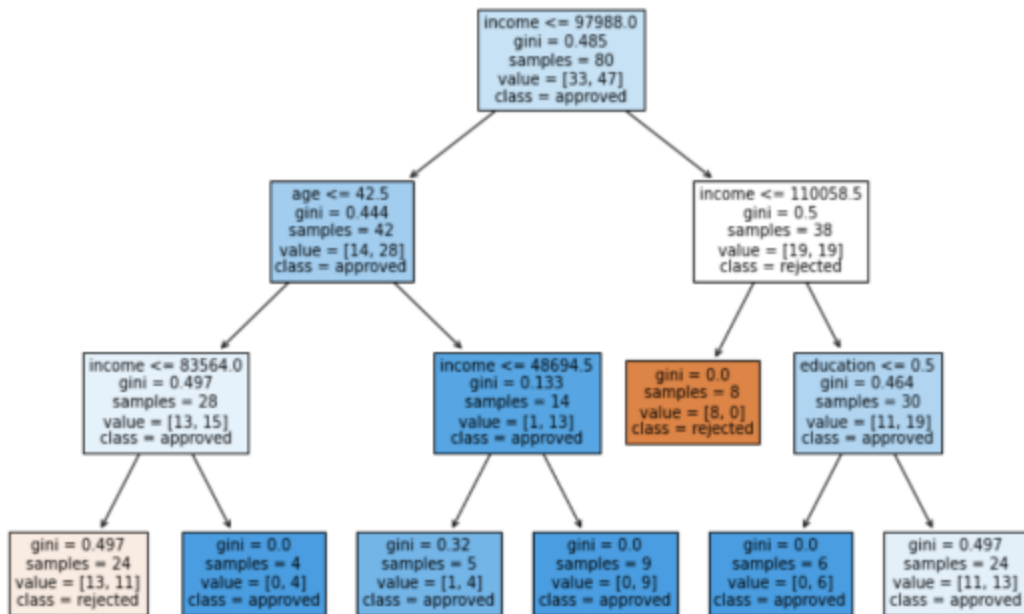

THE OUTPUT WILL LOOK like this:

Accuracy: 0.4

Explanation:

- The code first imports necessary modules and libraries: numpy for generating random data, sklearn.tree.DecisionTreeClassifier for building the decision tree classifier model, matplotlib.pyplot for visualizing the decision tree, and sklearn.model_selection.train_test_split for splitting the dataset into training and testing sets.
- A random dataset is generated using numpy's random functions. The dataset consists of 4 independent variables (age, income, education, and occupation) and 1 dependent variable (loan_approved).

- The independent variables are combined into a feature matrix X, and the dependent variable is assigned to y.
- The dataset is split into 80% training and 20% testing sets using train_test_split.
- A decision tree classifier is created with a maximum depth of 3 and a random seed of 42, and then fit to the training set using the fit method.
- Finally, the resulting decision tree is visualized using matplotlib's plot_tree function, with the feature names and class names specified in the corresponding parameters.

The resulting decision tree plot shows the decision rules learned by the model, based on the relationships between the independent and dependent variables in the dataset. The root node represents the feature that best splits the dataset, while the leaf nodes represent the resulting decision outcomes. The color coding indicates the class distribution of the samples that fall into each node.

Decision tree are prone to overfitting and it's a best practice to limit the maximum depth of the tree or use ensembles like random forest to reduce the variance.

A real-life application of Decision tree is in the field of medicine. For example, a decision tree can be used to predict the likelihood of a patient developing a certain disease based on their medical history, symptoms, and lab test results. By
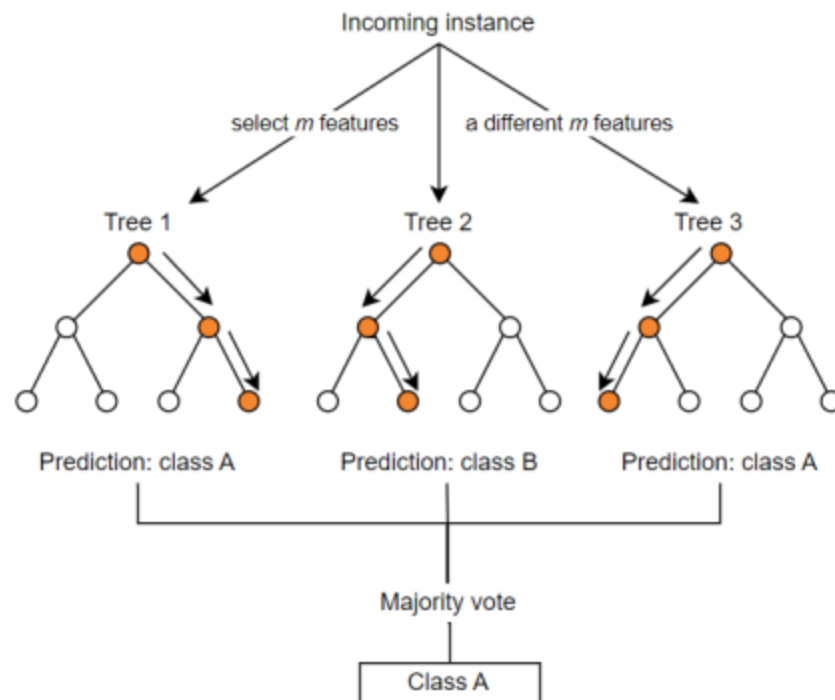
training a decision tree model on a large dataset of patient records, the model can learn to identify patterns and relationships that are indicative of the disease, and make accurate predictions for new patients.

In conclusion, decision trees are a powerful and interpretable algorithm for both classification and regression problems. They can handle both categorical and numerical data and can handle missing values. However, they can easily overfit the training data and be sensitive to small changes in the data. Techniques such as pruning, limiting the maximum depth of the tree, or using ensembles of trees can be used to mitigate these limitations.

4.4 RANDOM FORESTS

───

Random forests is an ensemble learning method for classification and regression problems in machine learning. It is a type of decision tree algorithm that combines multiple decision trees to create a more robust and accurate model. The basic idea behind random forests is to randomly sample the data, build a decision tree on each sample, and then combine the results of all the trees to make a final prediction.

To create a random forest, the algorithm first randomly selects a subset of data from the original dataset, called a **bootstrap sample**. It then builds a decision tree on this sample and repeats this process for a specified number of times. Each decision tree is built on a different bootstrap sample, so each tree will have a slightly different structure. The final predictions are made by taking the majority vote of all the trees in the forest.

Incoming instance
select *m* features | a different *m* features
Tree 1 | Tree 2 | Tree 3
Prediction: class A | Prediction: class B | Prediction: class A
Majority vote
Class A

THE RANDOMNESS IN THE random forest comes from two sources: the random selection of data for each tree, and the random selection of features for each split in the tree. This randomness helps to reduce overfitting, which is a common problem in decision tree algorithms. Random forests are also less sensitive to outliers and noise in the data, making them more robust than a single decision tree.

Random forests can be used for both classification and regression problems. In classification problems, it can handle categorical and numerical features, and it can handle missing data as well. In regression problems, it can also handle

categorical and numerical features, and it can handle missing data as well.

A real-life application of random forests is in the field of finance, where it is used for risk management. Random forests can be used to identify important factors that contribute to risk and to develop a model that predicts the risk level of a portfolio. It can also be used in medicine to predict the likelihood of a patient developing a disease based on their medical history and other factors.

In Python, the scikit-learn library provides the RandomForestClassifier and RandomForestRegressor classes for building and using random forest models. These classes provide a simple and consistent interface for building, training and evaluating random forest models, and they are compatible with other scikit-learn tools such as cross-validation, grid search and feature importance analysis.

One of the key advantages of random forests is that it reduces overfitting, which is a common problem in decision tree algorithms. This is because a random forest is made up of multiple decision trees, each of which is built on a different subset of the data. As a result, the final predictions are less sensitive to the specific data points in the training set.

Another advantage of random forests is that it is able to handle missing data and categorical variables. It can also

handle high dimensional data and is less affected by outliers.

A real life example of random forests is in the field of finance. Random forests can be used to predict whether a customer will default on a loan. The algorithm can take into account factors such as the customer's credit score, income, and employment history to make the prediction.

Here's an example of how to implement Random Forests algorithm using scikit-learn library in Python:

```python
import numpy as np

import pandas as pd

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score, confusion_matrix

from sklearn.model_selection import train_test_split

import matplotlib.pyplot as plt

# Set seed for reproducibility

np.random.seed(42)

# Generate random dataset

height = np.random.normal(loc=170, scale=5, size=1000)

weight = np.random.normal(loc=70, scale=10, size=1000)

age = np.random.normal(loc=30, scale=5, size=1000)

gender = np.random.randint(low=0, high=2, size=1000)

# Create a DataFrame to hold the dataset
```

```python
df = pd.DataFrame({

'height': height,

'weight': weight,

'age': age,

'gender': gender

})

# Define the dependent and independent variables

X = df[['height', 'weight', 'age']]

y = df['gender']

# Split the dataset into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Instantiate a Random Forest classifier with 100 trees

rfc = RandomForestClassifier(n_estimators=100)

# Fit the model to the training data

rfc.fit(X_train, y_train)

# Use the model to make predictions on the testing data

y_pred = rfc.predict(X_test)

# Evaluate the model's accuracy

accuracy = accuracy_score(y_test, y_pred)

print("Accuracy:", accuracy)

# Create a confusion matrix to visualize the performance of the model
```

```python
cm = confusion_matrix(y_test, y_pred)

# Print the confusion matrix

print(cm)

plt.imshow(cm, cmap=plt.cm.Blues)

plt.colorbar()

plt.xticks([0, 1])

plt.yticks([0, 1])

plt.xlabel('Predicted label')

plt.ylabel('True label')

plt.title('Confusion matrix')

plt.show()
```

HERE IS HOW THE OUTPUT will look like: