RQF LEVEL 5

SWDML501

SOFTWARE DEVELOPMENT

Machine Learning Fundamentals

TRAINEE'S MANUAL

October, 2024

AUTHOR'S NOTE PAGE (COPYRIGHT)

MACHINE LEARNING FUNDAMENTALS

2024

ACKNOWLEDGEMENTS

The publisher would like to thank the following for their assistance in the elaboration of this training manual:

Rwanda TVET Board (RTB) extends its appreciation to all parties who contributed to the development of the trainer’s and trainee’s manuals for the TVET Certificate V in Software Development, specifically for the module **"SWDML501: Machine Learning Fundamentals".**

We extend our gratitude to KOICA Rwanda for its contribution to the development of these training manuals and for its ongoing support of the TVET system in Rwanda.

We extend our gratitude to the TQUM Project for its financial and technical support in the development of these training manuals.

We would also like to acknowledge the valuable contributions of all TVET trainers and industry practitioners in the development of this training manual.

The management of Rwanda TVET Board extends its appreciation to both its staff and the staff of the TQUM Project for their efforts in coordinating these activities.

This training manual was developed:

Under Rwanda TVET Board (RTB) guiding policies and directives

Under Financial and Technical support of

COORDINATION TEAM

RWAMASIRABO Aimable

MARIA Bernadette M. Ramos

MUTIJIMA Asher Emmanuel

Production Team

Authoring and Review

MUHIRWA David

NYIRANSHUTI Elizabeth

Validation

SEKABANZA Jean de la Paix

NYANDWI Rongin

NIYONSHUTI Yves

Conception, Adaptation and Editorial works

HATEGEKIMANA Olivier

GANZA Jean Francois Regis

HARELIMANA Wilson

NZABIRINDA Aimable

DUKUZIMANA Therese

NIYONKURU Sylvestre

Formatting, Graphics, Illustrations, and infographics

YEONWOO Choe

SUA Lim

SAEM Lee

SOYEON Kim

WONYEONG Jeong

NIYOMUGABO Silas

Financial and Technical support

KOICA through TQUM Project

TABLE OF CONTENT

ACRONYMS

AAA: Authentication, Authorization and Accountability

AI: Artificial intelligent

API: Application Programming Interfaces

AWS: Amazon Web services

CBT/A: Competency Based Training/ Assessment

CI/CD: Continuous Integration/Continuous Deployment

CMD: Command Line Interface

CPU: Central Processing Unit

DBMS: Database Management System

DL: Deep learning

DVC: Data Version Control

GPU: Graphics Processing Unit

HIPAA: Health Insurance Portability and Accountability Act

HTTPS: Hypertext Transfer Protocol Secure

IDE: Integrated Development Environment

IDLE: Integrated Development and Learning Environment

JSON: JavaScript Object Notation

JWT: JSON Web Token

KNN: k-Nearest Neighbours

KOICA: Korea International Cooperation Agency

KPI: key performance indicators

MaaS: Model-as-a-Service

ML: Machine Learning

NPM: Node package manager

ONNX: Open Neural Network Exchange

OS: Operating system

PC1: First Principal Component

PC2: Second Principal Component

PCA: Principal Component Analysis

PIP: Package installer for Python

RAM: Random Access Memory

RESTful: Representational state transfer

RL: Reinforcement Learning

ROM: Read Only Memory

RTB: Rwanda TVET Board

STEM: Science, Technology, Engineering, and Mathematics

TPU: Tensor Processing Unit

TQUM Project: TVET Quality Management Project

t-SNE: t-Distributed Stochastic Neighbor Embedding.

TVET: Technical Vocational and Education Training

INTRODUCTION

This trainee's manual includes all the knowledge and skills required in Software Development specifically for the module of **"Machine Learning Fundamentals".** Trainees enrolled in this module will engage in practical activities designed to develop and enhance their competencies. The development of this training manual followed the Competency-Based Training and Assessment (CBT/A) approach, offering ample practical opportunities that mirror real-life situations.

The trainee's manual is organized into Learning Outcomes, which is broken down into indicative content that includes both theoretical and practical activities. It provides detailed information on the key competencies required for each learning outcome, along with the objectives to be achieved.

As a trainee, you will start by addressing questions related to the activities, which are designed to foster critical thinking and guide you towards practical applications in the labour market. The manual also provides essential information, including learning hours, required materials, and key tasks to complete throughout the learning process.

All activities included in this training manual are designed to facilitate both individual and group work. After completing the activities, you will conduct a formative assessment, referred to as the end learning outcome assessment. Ensure that you thoroughly review the key readings and the 'Points to Remember' section.

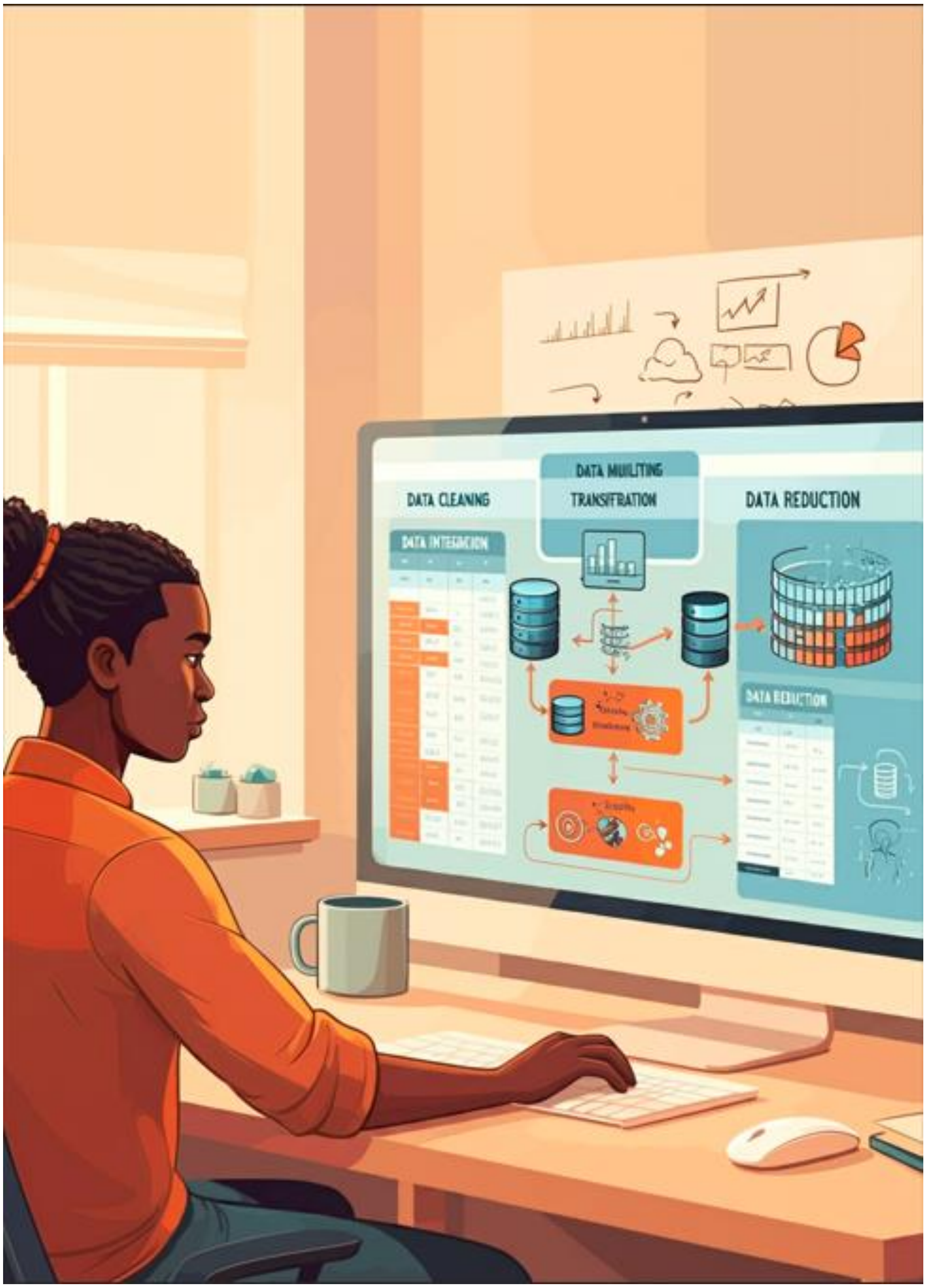
MODULE CODE AND TITLE: NITML501 MACHINE LEARNING FUNDAMENTALS

Learning Outcome 1: Apply Data Pre-processing.

Learning Outcome 2: Develop Machine Learning Model.

Learning Outcome 3: Perform Model Deployment.

Learning Outcome 1: Apply Data Pre-processing.



Indicative contents

1.1 Description of Machine learning concepts

1.2 Preparing Machine Learning environment

1.3 Data Collection and Acquisition

1.4 Interpret Data Visualization

1.5 Perform Data cleaning

Key Competencies for Learning Outcome 1: Apply Data Pre-processing

Knowledge	Skills	Attitudes
● Description of machine learning concepts ● Description of machine learning tools ● Description of Data Collection and Acquisition ● Description of data Visualization ● Description of data cleaning	● Preparing Machine Learning environment ● Gathering Machine Learning dataset ● Using Types of Data Visualization ● Applying data Visualization Best Practices ● Interpreting visualizations results ● Performing Data cleaning	● Having Teamwork spirit in preparing environment ● Being critical thinker in gathering the dataset ● Being Innovative interpreting data ● Being creative in cleaning data ● Having Problem-Solving Mindset in visualizing the dataset ● Being Attentive analysing dataset

Duration: 30 hrs

Learning outcome 1 objectives:

By the end of the learning outcome, the trainees will be able to:

1. Describe correctly basic concepts as used in machine learning
2. Describe correctly machine learning tools as used in model development
3. Prepare properly machine learning environment as used in developing model
4. Describe clearly data collection and acquisition as used in machine learning
5. Gather correctly machine learning dataset based on project
6. Interpret properly data Visualization based on dataset
7. Use properly the types of data visualization as used in machine learning
8. Describe correctly data cleaning based on dataset
9. Perform correctly data cleaning based on required dataset.

Resources

Equipment	**Tools**	**Materials**
● Computer ● Projector ● Storage Devices	● Python Distribution (CPython, Anaconda) ● Python IDEs (Jupyter Notebook, PyCharm, Visual Studio Code, or Spyder) ● Cloud jupyter notebook platform	● Internet ● Datasets

Indicative content 1.1: Description of Machine Learning Concepts

Duration: 6 hrs

Theoretical Activity 1.1.1: Description of machine learning concepts.

Tasks:

1: Answer the following questions:

- **i.** Define machine learning.
- **ii.** Explain machine learning life cycle.
- **iii.** State and explain the applications of Machine Learning.
- **iv.** Explain advantages and disadvantages of machine learning
- **v.** Differentiate between machine learning, artificial intelligence and deep learning

2: Present the findings to the trainer and whole class.

3: Ask questions where necessary.

3: For more clarification, read the key readings 1.1.1.

Key readings 1.1.1: Description of machine learning concepts.

- **achine learning overview**

In the real world, we are surrounded by humans who can learn everything from their experiences with their learning capability, and we have computers or machines which work on our instructions. But can a machine also learn from experiences or past data like a human does? So here comes the role of **Machine Learning**.

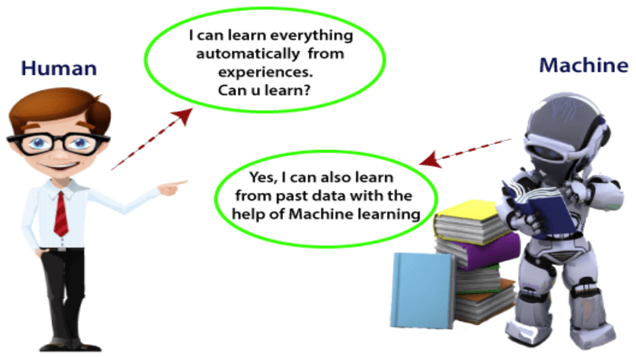

Human
I can learn everything automatically from experiences. Can u learn?
Machine
Yes, I can also learn from past data with the help of Machine learning

Figure 1: Human vs machine learning

✓

efinition of machine learning

Machine learning is subset of artificial intelligence that enables systems to learn from data and improve without explicit programming.

✓

achine learning Life cycle

Machine learning life cycle is a process that guides the development and deployment of machine learning models in a structured way.

it is a cyclic process to build an efficient machine learning project. The main purpose of the life cycle is to find a solution to the problem or project.

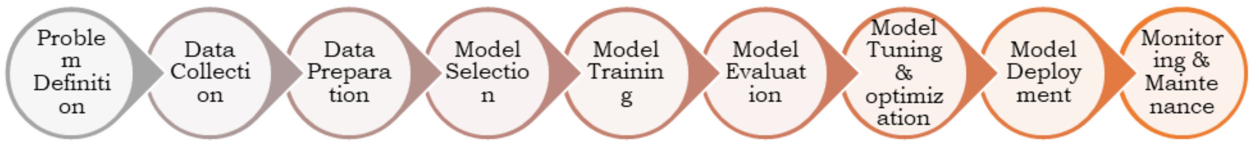


Figure 2: Machine Learning Life cycle

roblem definition:

The first step involves clearly defining the problem you want to solve and determining the objectives.

- Identifying the specific business or research question.
- Defining the desired outcome or goal (e.g., prediction, classification, clustering).
- Specifying the key performance indicators (KPIs) that will be used to evaluate the model's success.

ata Collection:

This involves gathering the data that will be used to train and evaluate the machine learning model. This might include:

Data Gathering: The next step is collecting data that is relevant to the problem you're trying to solve. Data may come from multiple sources like databases, APIs, or scraping web data, public datasets.

Data Labeling: In supervised learning, data should be labeled with the correct output, which can involve manual labeling or using pre-labeled datasets.

- Collecting the data using appropriate methods (e.g., web scraping, data acquisition APIs).
- Ensuring data quality and addressing potential biases.

ata Preparation:

This stage focuses on transforming the raw data into a suitable format for the machine learning model. Analysis and preprocessing steps can be done at this stage. Key activities include:

- Data cleaning: Handling missing values, removing duplicates, and correcting errors.
- Data transformation: Converting data into appropriate formats (e.g., feature scaling, one-hot encoding).

odel Selection:

Based on the problem definition and the characteristics of the data, you choose the appropriate machine learning algorithm. This might involve:

- Considering the type of problem (e.g., classification, regression, clustering, dimensionality reduction…).
- Evaluating the strengths and weaknesses of different algorithms.
- Experimenting with different models to find the best fit.

odel training:

This is where the chosen algorithm learns from the prepared data. Divide the dataset into training, validation, and test sets to avoid overfitting and ensure generalization.

The model iteratively adjusts its parameters to minimize the difference between its predictions and the actual labels in the training data (i.e. error).

odel Evaluation:

Once trained, the model's performance is assessed using Performance metrics (like accuracy, precision, recall, F1-score, or mean squared error, MSE, RMSE for regression).

This helps determine the model's effectiveness and identify areas for improvement.

odel Tuning & optimization

This step involves fine-tuning the model's hyperparameters (e.g., learning rate) to improve its performance.

Model Deployment

This step intends to Integrate the trained and optimized model into a real-world application or production environment by ensuring the model can handle real-world data efficiently.

- Deploy the model as a service (e.g., APIs, cloud platforms).
- Set up infrastructure for scalability and improved performance.

Monitoring & Maintenance:

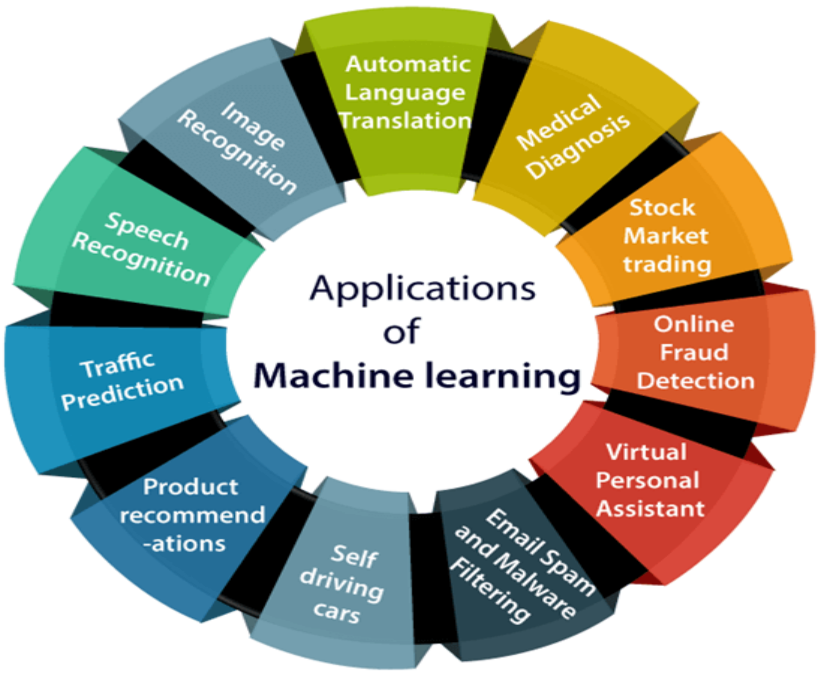
Continuously monitoring the deployed model's performance and addressing any issues that arise.

- Track metrics like accuracy, latency, or drift in predictions.
- Retrain the model if data distribution changes (concept drift).
- Maintain and upgrade infrastructure for long-term usability.

✓

pplications of Machine learning

Machine learning is a buzzword for today's technology, and it is growing very rapidly day by day. machine learning is used in our daily life even without knowing it such as Google Maps, Google assistant, Alexa, etc. Below are some most trending real-world applications of Machine Learning:



Applications of Machine learning
Automatic Language Translation
Medical Diagnosis
Stock Market trading
Online Fraud Detection
Virtual Personal Assistant
Email Spam and Malware Filtering
Self driving cars
Product recommend -ations
Traffic Prediction
Speech Recognition
Image Recognition

Figure 3: examples of Machine learning Applications

mage Recognition:

Image recognition is one of the most common applications of machine learning. It is used to identify objects, persons, places, digital images, etc.

peech Recognition

While using Google, click on option of "Search by voice," it comes under speech recognition, and it's a popular application of machine learning.

Speech recognition is a process of converting voice instructions into text, and it is also known as "Speech to text", or "Computer speech recognition." At present, machine learning algorithms are widely used by various applications of speech recognition. Google assistant, Siri, Cortana, and Alexa are using speech recognition technology to follow the voice instructions.

raffic prediction:

while visiting a new place, take help of Google Maps, which shows the correct path with the shortest route and predicts the traffic conditions.

It predicts the traffic conditions such as whether traffic is cleared, slow-moving, or heavily congested with the help of two ways:

- Real Time location of the vehicle form Google Map app and sensors
- Average time has taken on past days at the same time.

Everyone who is using Google Map is helping this app to make it better. It takes information from the user and sends back to its database to improve the performance.

roduct recommendations:

Machine learning is widely used by various e-commerce and entertainment companies such as Amazon, Netflix, etc., for product recommendation to the user. When searching for some product on Amazon, then it started proposing an advertisement for the same product while internet surfing on the same browser and this is because of machine learning.

Google understands the user interest using various machine learning algorithms and suggests the product as per customer interest.

As similar, when using Netflix, some recommendations for entertainment series, movies, etc. are shown, and this is also done with the help of machine learning.

elf-driving cars:

One of the most exciting applications of machine learning is self-driving cars. Machine learning plays a significant role in self-driving cars. Tesla, the most popular car manufacturing company is working on self-driving car. It is using unsupervised learning method to train the car models to detect people and objects while driving.

mail Spam and Malware Filtering:

When receiving a new email, it is filtered automatically as important, normal, and spam. an important mail in the inbox with the important symbol and spam emails in the spam box, and the technology behind this is Machine learning. Below are some spam filters used by Gmail:

- Content Filter
- Header filter
- General blacklists filter
- Rules-based filters
- Permission filters

Some machine learning algorithms such as Multi-Layer Perceptron, Decision tree, and Naïve Bayes classifier are used for email spam filtering and malware detection.

irtual Personal Assistant:

There are various virtual personal assistants such as Google assistant, Alexa, Cortana, Siri. As the name suggests, they help in finding the information using the voice instruction. These assistants can help in various ways just by the voice instructions such as Play music, call someone, Open an email, Scheduling an appointment, etc.

These virtual assistants use machine learning algorithms as an important part.

These assistants record the voice instructions, send it over the server on a cloud, and decode it using ML algorithms and act accordingly.

nline Fraud Detection:

Machine learning is making the online transaction safe and secure by detecting fraud transaction. When performing some online transaction, there may be various ways that a fraudulent transaction can take place such as fake accounts, fake ids, and steal money in the middle of a transaction. So to detect this, Feed Forward Neural network helps by checking whether it is a genuine transaction or a fraud transaction.

For each genuine transaction, the output is converted into some hash values, and these values become the input for the next round. For each genuine transaction, there is a specific pattern which gets change for the fraud transaction hence, it detects it and makes the online transactions more secure.

tock Market trading:

Machine learning is widely used in stock market trading. In the stock market, there is always a risk of up and downs in shares, so for this machine learning's long short-term memory neural network is used for the prediction of stock market trends.

edical Diagnosis:

In medical science, machine learning is used for diseases diagnoses. With this, medical technology is growing very fast and able to build 3D models that can predict the exact position of lesions in the brain.

It helps in finding brain tumors and other brain-related diseases easily.

utomatic Language Translation:

Nowadays, while visiting a new place and no specific language skills then it is not a problem at all, as for this also machine learning helps by converting the text into the known languages. Google's GNMT (Google Neural Machine Translation) provide this feature, which is a Neural Machine Learning that translates the text into a familiar language, and it called as automatic translation.

The technology behind the automatic translation is a sequence to sequence learning algorithm, which is used with image recognition and translates the text from one language to another language.

- **Advantages and disadvantages of machine learning**

Table 1: table describing the advantages and disadvantages of machine learning

ML Advantages	ML Disadvantages
• Automation of a process	• Data Dependency (machine cannot learn if there is no data available)
• Learning Capability	• Chances of Error
• Improved decision making	• Complexity and Interpretability
• Personalization	• High Computational Costs
• Handling High-Dimensional Data	• Dependence on Skilled Professionals
• Wide Range of Applicability	• Reduce employment opportunities (for humans)
• Continuous Improvement	• Time and resources consuming (in case of big dataset)
• Enhanced Accuracy & Precision	
• Ability to Handle Unstructured Data	
• Risk Management by predicting	

- **ifference Between Machine Learning, Artificial Intelligence, and Deep Learning**

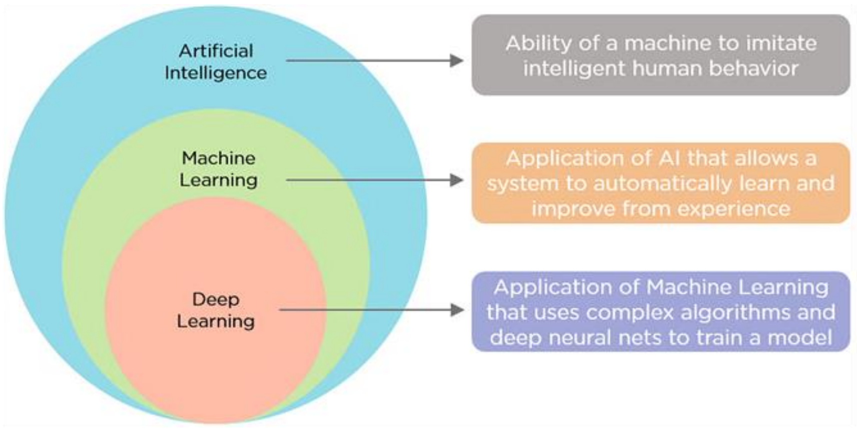

Artificial Intelligence
Ability of a machine to imitate intelligent human behavior
Machine Learning
Application of AI that allows a system to automatically learn and improve from experience
Deep Learning
Application of Machine Learning that uses complex algorithms and deep neural nets to train a model

Figure 4: comparison between ML-AI-DL

Table 2: table describing the differences among ML, AI, and DL

Aspect	Artificial Intelligence (AI)	Machine Learning (ML)	Deep Learning
Definition	A broad field that involves creating machines capable of performing tasks that typically require human intelligence.	A subset of AI that focuses on creating systems that learn from data and improve without explicit programming.	A subset of ML that uses neural networks with many layers to process vast amounts of data.
Scope	Encompasses everything from reasoning and problem-solving to learning and perception.	Focuses on training algorithms to learn from data to make predictions or decisions.	Specializes in complex tasks like image recognition, natural language processing, game AI…
Techniques	Includes machine learning, rule-based systems, expert systems, and evolutionary algorithms.	Primarily uses algorithms like decision trees, regression, and support vector machines.	Utilizes deep neural networks with multiple layers to mimic the human brain's functioning.
Data Requirement	Does not always require data, as rule-based systems can be designed without learning from data.	Requires large datasets for training and improving models.	Requires massive datasets and high computational power to process and learn from data.
Examples	Robotics, speech recognition, problem-solving, and decision-making systems like Siri or self-driving cars.	Spam detection, recommendation systems, and predictive analytics.	Image recognition (e.g., facial recognition), language translation, and voice assistants (e.g., Alexa).

Theoretical Activity 1.1.2: Description of machine learning types and tools

Tasks:

1: Answer the following questions:

i. Explain the following types of machine learning

a. Supervised

b. Unsupervised

c. Semi-supervised

d. Reinforcement

ii. Explain Machine Learning tools

2: Provide the answer for the asked questions and write them on papers.

3: Present the findings/answers to the whole class and trainer

4: For more clarification, read the key readings 1.1.2. In addition, ask questions where necessary.

Key readings 1.1.2: Description of machine learning types and tools

- **Types of Machine Learning**

✓ **Supervised Learning**

- In supervised learning, the model is trained on labeled data, meaning that the input comes with corresponding correct output labels. The objective is for the model to learn the mapping between inputs and outputs and predict the output for new, unseen data.
- Examples: Classification (e.g., spam detection), Regression (e.g., house price prediction).
- Algorithms: Linear Regression, Decision Trees, Random Forests, Support Vector Machines (SVM), Neural Networks.

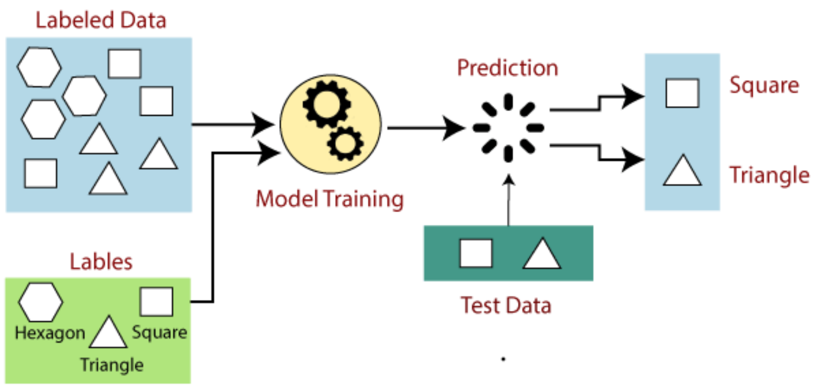


Figure 5: Supervised Learning illustration

✓ **Unsupervised Learning**

- In unsupervised learning, the model is given data without explicit labels. The goal is to find patterns, groupings, or structure in the data by analyzing the underlying relationships.
- Examples: Clustering (e.g., customer segmentation), Dimensionality Reduction (e.g., PCA, t-SNE).
- Algorithms: K-Means Clustering, Hierarchical Clustering, Principal Component Analysis (PCA), Autoencoders.

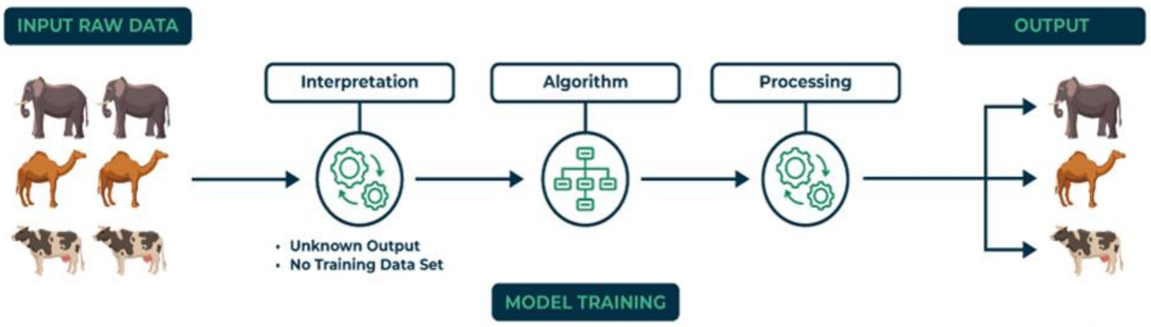


Figure 6: Unsupervised Learning illustration

✓ **Semi-Supervised Learning**

- Semi-supervised learning is a hybrid approach where the model is trained on a small amount of labeled data and a large amount of unlabeled data. This is useful when labeling data is expensive or time-consuming.
- Examples: Image classification with few labeled examples.
- Algorithms: Self-training, Co-training, Label Propagation.

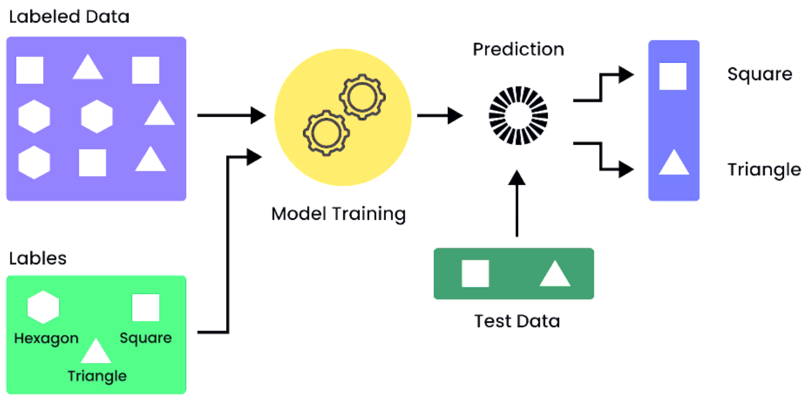


Figure 7: Semi-supervised Learning illustration

✓ **Reinforcement Learning**

- In reinforcement learning, an agent learns by interacting with an environment. It receives rewards or penalties based on its actions and learns to maximize cumulative rewards over time. This approach is used in scenarios where actions influence future states.
- Examples: Game playing (e.g., AlphaGo), Robotics, Autonomous driving.

- Algorithms: Q-Learning, Deep Q-Networks (DQN), Proximal Policy Optimization (PPO).

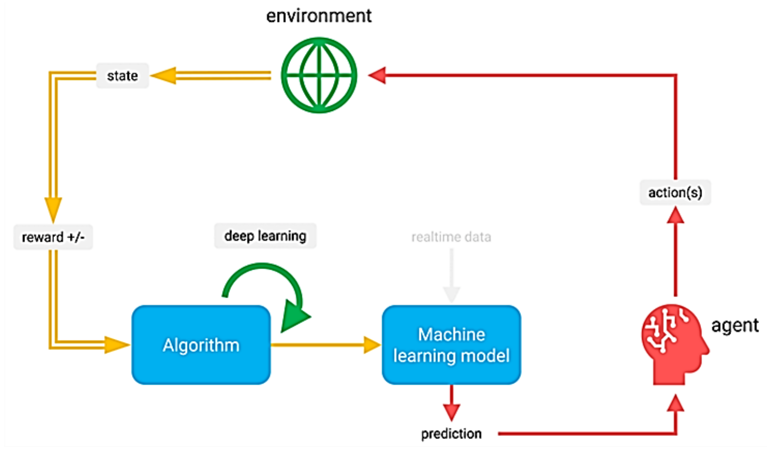


Figure 8: Reinforcement Learning illustration

Summarized types of machine learning

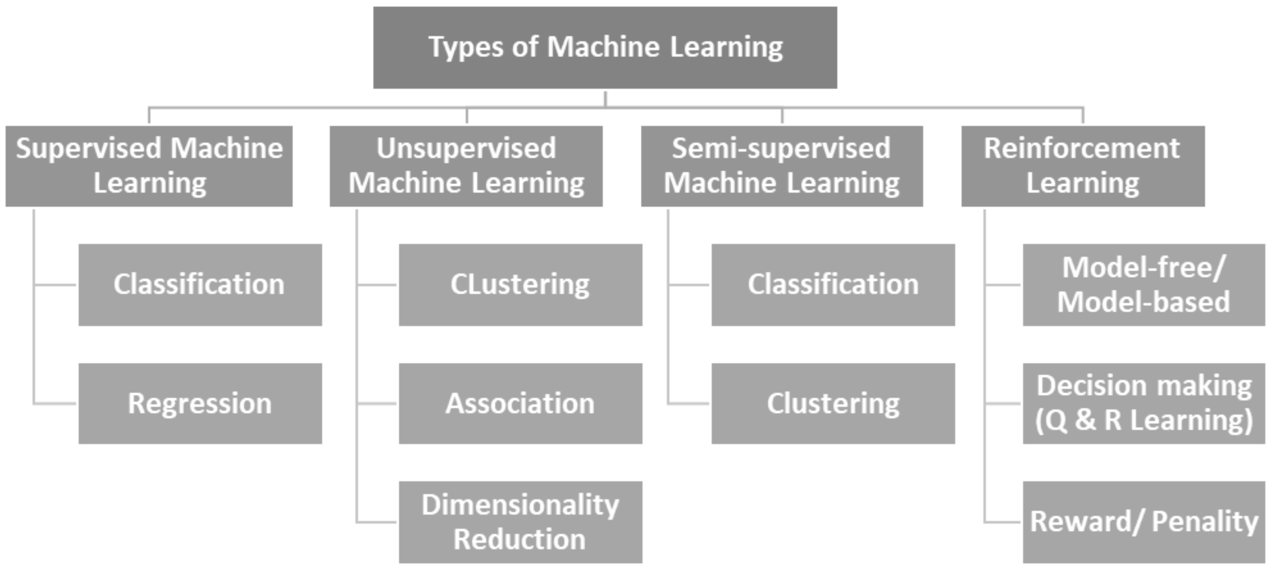


Figure 9: Diagram illustrating the types of machine learning

Supervised Learning Vs Semi-Supervised Learning Vs Unsupervised Learning

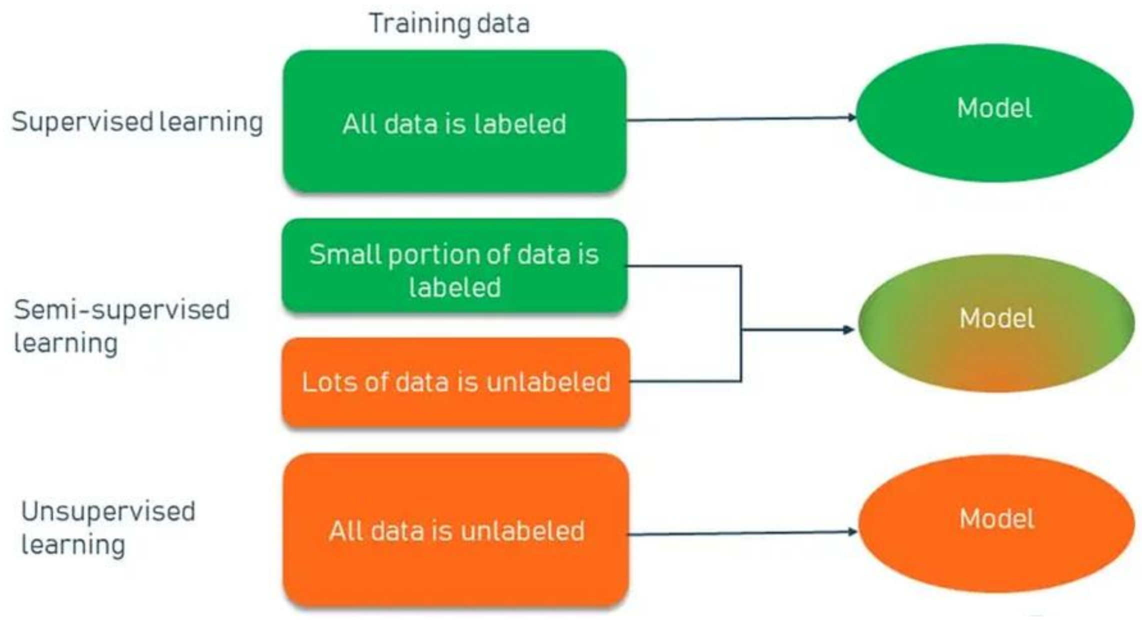


Figure 10: summary comparison of machine learning types

- **Machine Learning Tools**
  Machine learning involves various tools, frameworks, and libraries that help in building, training, and deploying models. These tools assist with different stages of the machine learning process, from data preprocessing to model development, evaluation, and deployment.

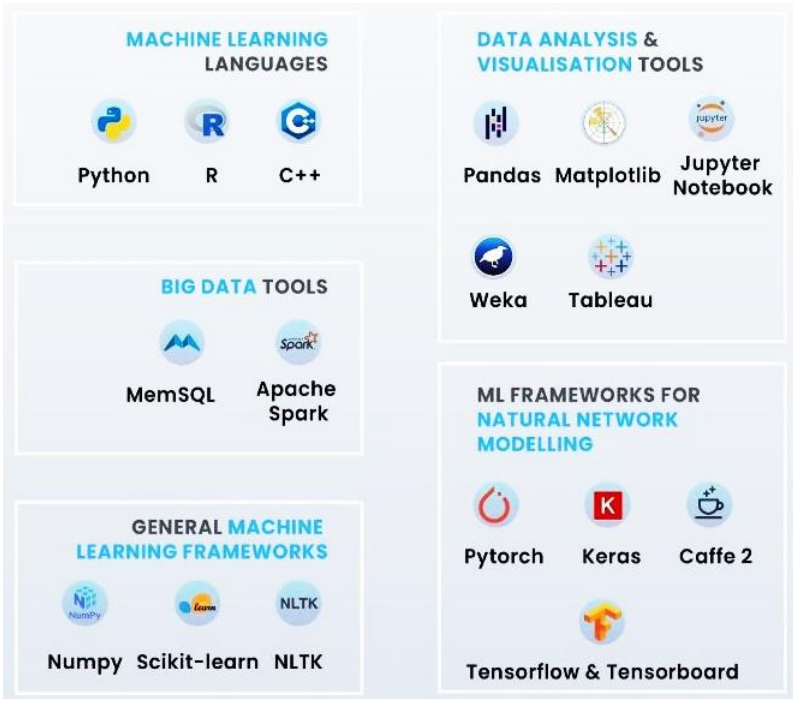


Figure 11: Examples of machine learning tools

✓ **Programming Languages**

- **Python**: The most popular language for machine learning due to its rich ecosystem of libraries like TensorFlow, PyTorch, and Scikit-learn.

- **R**: A statistical programming language used for data analysis and visualization, often employed in academic and research settings.

✓ **Libraries and Frameworks**

- **TensorFlow**: An open-source library developed by Google, widely used for both deep learning and machine learning applications.
- **PyTorch**: A deep learning framework developed by Facebook. It's popular for research and development because of its dynamic computational graph.
- **Scikit-learn**: A simple and efficient tool for data mining and machine learning in Python. It provides algorithms for classification, regression, clustering, and more.
- **Keras**: A high-level neural network API, running on top of TensorFlow. It's user-friendly and allows for fast experimentation.
- **XGBoost**: A powerful library for gradient boosting algorithms, widely used in machine learning competitions (e.g., Kaggle).

✓ **Data Preprocessing Tools**

- **Pandas**: A Python library for data manipulation and analysis. It’s widely used for preprocessing tasks like cleaning, transforming, and analyzing data.
- **NumPy**: A library used for numerical operations in Python, especially useful for dealing with large datasets in arrays.
- **Dask**: A library for parallel computing in Python that scales NumPy and Pandas for larger datasets.

✓ **Model Evaluation and Hyperparameter Tuning Tools**

- **GridSearchCV / RandomizedSearchCV**: Techniques available in Scikit-learn to perform hyperparameter tuning and cross-validation.
- **Hyperopt**: A Python library for automatic hyperparameter optimization.
- **Optuna**: A framework for hyperparameter optimization that helps in finding the best model parameters.

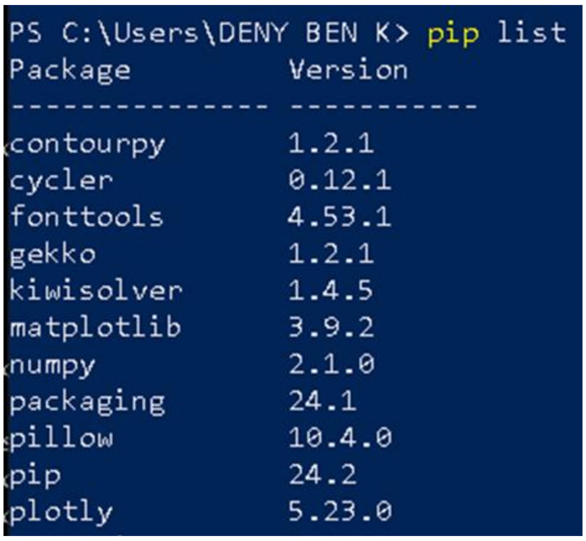
- ✓ **Data Visualization Tools**
  - **Matplotlib**: A plotting library for Python, useful for generating static, animated, or interactive visualizations.
  - **Seaborn**: Built on top of Matplotlib, it provides a high-level interface for drawing attractive and informative statistical graphics.
  - **Plotly**: A library for interactive, web-based data visualizations. It's used for dashboards and real-time model visualization.
- ✓ **Integrated Development Environments (IDEs)**
  - **Jupyter Notebooks**: An interactive web-based tool that allows you to write and run code, visualize outputs, and share notebooks.
  - **Spyder**: An IDE tailored for data science, often used in combination with Anaconda for Python-based machine learning projects.
  - **Google Colab**: A cloud-based Jupyter notebook environment that allows you to run Python code in the browser with free access to GPUs and TPUs.
- **Model Deployment and MLOps Tools**
  - **TensorFlow Serving**: A flexible, high-performance serving system for machine learning models designed for production environments.
  - **Docker**: Containerization tool to bundle machine learning models and their dependencies into containers for easy deployment across environments.
  - **Kubernetes**: An orchestration tool for managing machine learning models and applications in containers.
  - **MLflow**: An open-source platform for managing the machine learning lifecycle, including experimentation, reproducibility, and deployment.
  - **Apache Airflow**: A platform to programmatically author, schedule, and monitor workflows, often used to automate machine learning pipelines.
- **Cloud-based Machine Learning Platforms**
  - **Google Cloud AI Platform**: Offers tools for building, training, and deploying

machine learning models on Google's cloud infrastructure.

- **AWS SageMaker**: A fully managed service that provides developers and data scientists with tools to build, train, and deploy machine learning models quickly.
- **Microsoft Azure Machine Learning**: Provides cloud-based machine learning model development and deployment tools.

- **Data Labeling Tools**
  - **Labelbox**: A tool for data annotation and labeling, often used for training machine learning models, especially in supervised learning tasks.
  - **V7**: An AI-powered platform for creating and managing labeled datasets with collaborative annotation tools.
  - **SuperAnnotate**: A tool to label image, video, and text data, especially useful for computer vision tasks.

- **Version Control and Collaboration Tools**
  - **Git**: A version control system to track code changes and collaborate with other developers or data scientists.
  - **DVC (Data Version Control)**: An open-source tool to track machine learning projects' data, models, and pipelines. Helps version datasets similar to Git.
  - **Weights & Biases**: A tool for tracking machine learning experiments and managing model training and hyper-parameter tuning.



```
PS C:\Users\DENY BEN K> pip list
Package          Version
---------------- -----------
contourpy        1.2.1
cycler           0.12.1
fonttools        4.53.1
gekko            1.2.1
kiwisolver       1.4.5
matplotlib       3.9.2
numpy            2.1.0
packaging        24.1
pillow           10.4.0
pip              24.2
plotly           5.23.0
```

Figure 12: Picture showing installed machine learning tools

Points to Remember

- Ensure careful handling of machine learning tasks to achieve accurate outcomes.
- Follow the Machine Learning Life Cycle steps: Data Collection, Data Preparation, and Model Development.
- Choosing best machine learning types should be based on features and problem to be solved
- Preparation of machine learning environment should be based on machine learning tools.

Indicative content 1.2: Preparing Machine Learning Environment

Duration: 6 hrs

Practical Activity 1.2.1: Installing Python

Task:

1: You are requested to go to the computer lab to install python as used in machine learning.

2: Apply safety precautions

3: Read key reading 1.2.1 and ask clarification where necessary

4: Present out the steps to install python.

5: Referring to the steps provided on task 4, install python.

6: Present your work to the trainer and whole class.

Key readings 1.2.1: Installing Python

- **nstallation of Python**

Python is a popular high-level, general-use programming language. Python is a programming language that enables rapid development as well as more effective system integration. Python develops new versions with changes periodically and releases them according to version numbers.

- **ystem requirement for python**

  To use Python effectively for both hardware and software development in STEM fields, there are specific requirements and libraries you should install. Below is an overview of the tools needed for each category.

✓ **ardware and software requirements for Python programming**

  The minimum requirements needed to install Enthought Python and associated applications:

-Operating System: Windows OS (latest version),
Mac OS (latest version)
Linux (latest version)

-Memory: 4 GB RAM

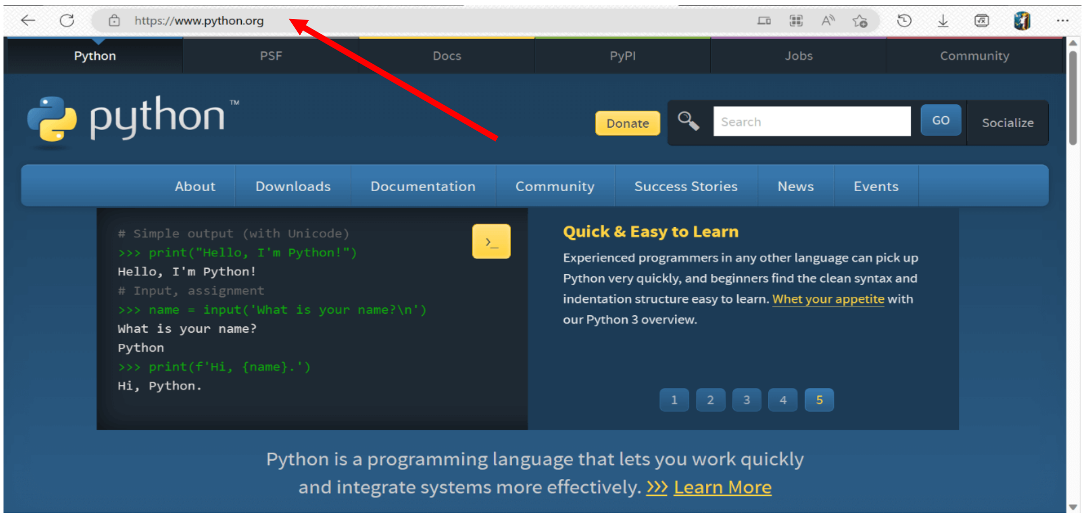
-Storage capacity: at least 5 GB free disk space

•

teps to Install Python

1. Installation on Windows

Visit the link ***https://www.python.org*** (if you have not the setup) to download the latest release of Python. In this process, we will install Python **3.11**



```
# Simple output (with Unicode)
>>> print("Hello, I'm Python!")
Hello, I'm Python!
# Input, assignment
>>> name = input('What is your name?\n')
What is your name?
Python
>>> print(f'Hi, {name}.')
Hi, Python.
```

Quick & Easy to Learn

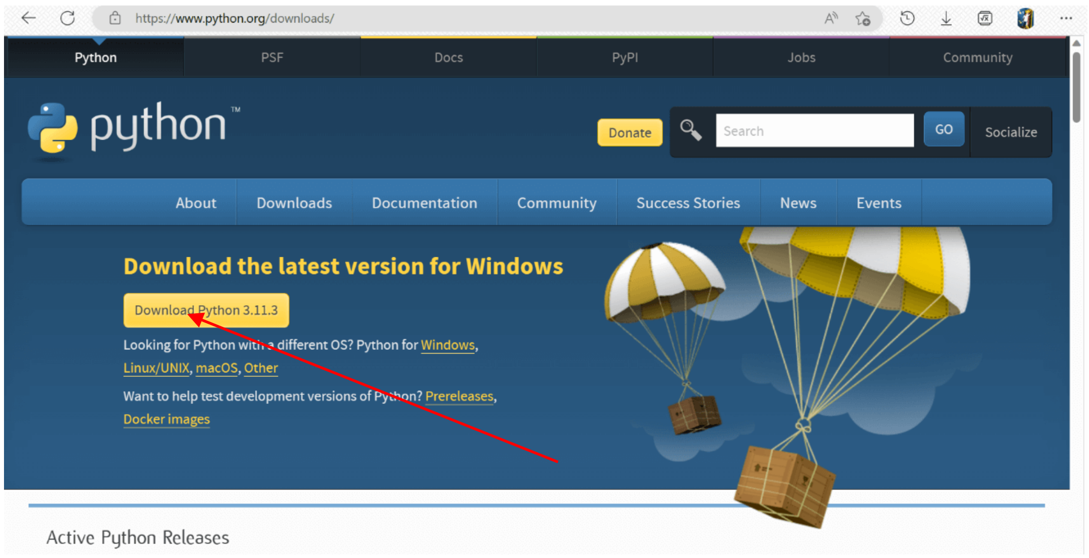
Experienced programmers in any other language can pick up Python very quickly, and beginners find the clean syntax and indentation structure easy to learn. Whet your appetite with our Python 3 overview.

Python is a programming language that lets you work quickly and integrate systems more effectively. >>> Learn More

Figure 13: Python Download page

Step - 1: Select the Python's version to download.

Click on the download button to download the exe file of Python.



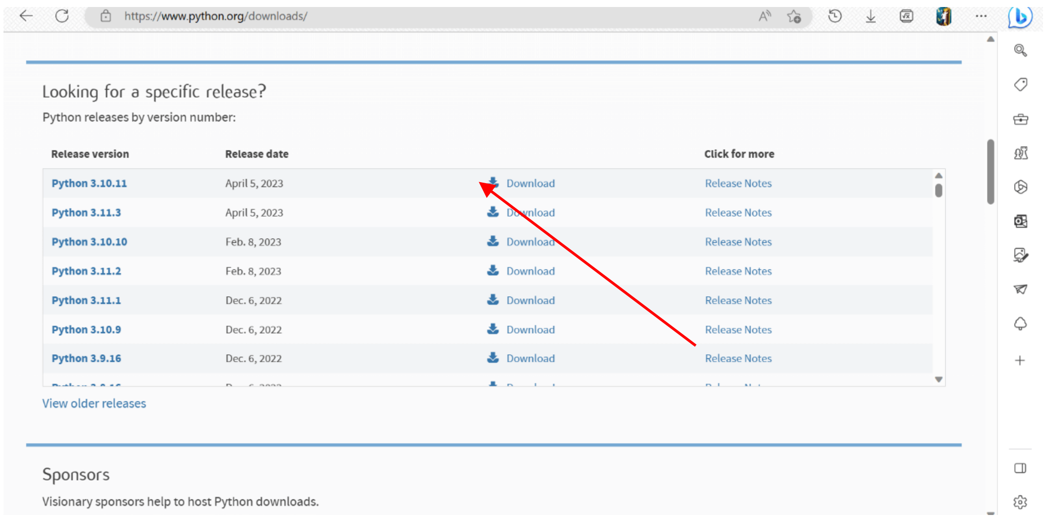
Download the latest version for Windows

Looking for Python with a different OS? Python for Windows, Linux/UNIX, macOS, Other

Want to help test development versions of Python? Prereleases, Docker images

Figure 14: Choose the python version to download

If in case, you want to download the specific version of Python. Then, you can scroll down further below to see different versions from 2 and 3 respectively. Click on download button right next to the version number you want to download.



Looking for a specific release?

Python releases by version number:

Release version	Release date		Click for more
Python 3.10.11	April 5, 2023	Download	Release Notes
Python 3.11.3	April 5, 2023	Download	Release Notes
Python 3.10.10	Feb. 8, 2023	Download	Release Notes
Python 3.11.2	Feb. 8, 2023	Download	Release Notes
Python 3.11.1	Dec. 6, 2022	Download	Release Notes
Python 3.10.9	Dec. 6, 2022	Download	Release Notes
Python 3.9.16	Dec. 6, 2022	Download	Release Notes

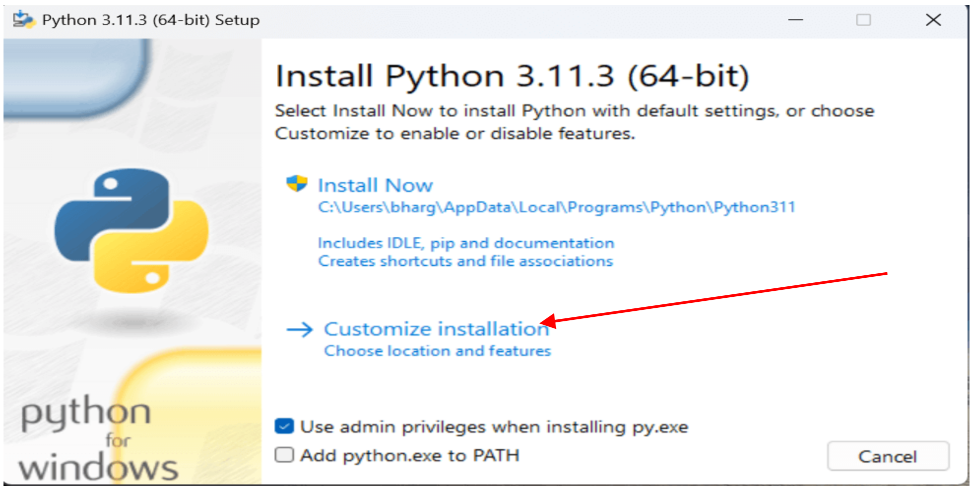
View older releases

Sponsors

Visionary sponsors help to host Python downloads.

Step - 2: Click on the Install Now

Double-click the executable file, which is downloaded.


Python 3.11.3 (64-bit) Setup

Install Python 3.11.3 (64-bit)

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

Install Now
C:\Users\bharg\AppData\Local\Programs\Python\Python311

Includes IDLE, pip and documentation
Creates shortcuts and file associations

→ Customize installation
Choose location and features

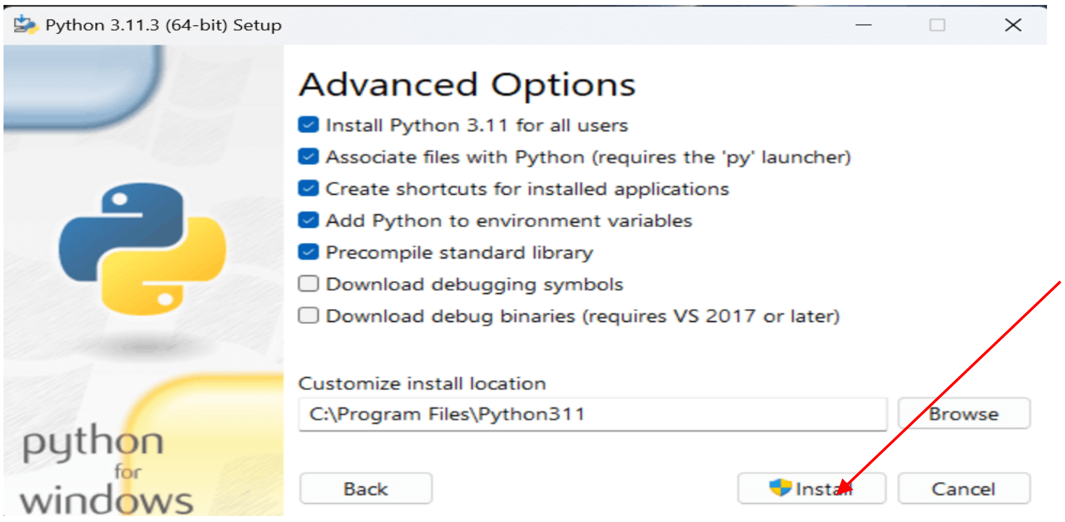
☑ Use admin privileges when installing py.exe
☐ Add python.exe to PATH

Figure 15: customize installation of python

The following window will open. Click on the Add Path check box, it will set the Python path automatically.

Now, Select Customize installation and proceed. also click on the customize installation to choose desired location and features. Other important thing is installing launcher for all user must be checked.

Here, under the advanced options, click on the checkboxes of " Install Python 3.11 for all users ", which is previously not checked in. This will check the other option " Precompile standard library " automatically. And the location of the installation will also be changed. Optionally, change it later, so leave the install location default. Then, click on the install button to finally install.


Python 3.11.3 (64-bit) Setup

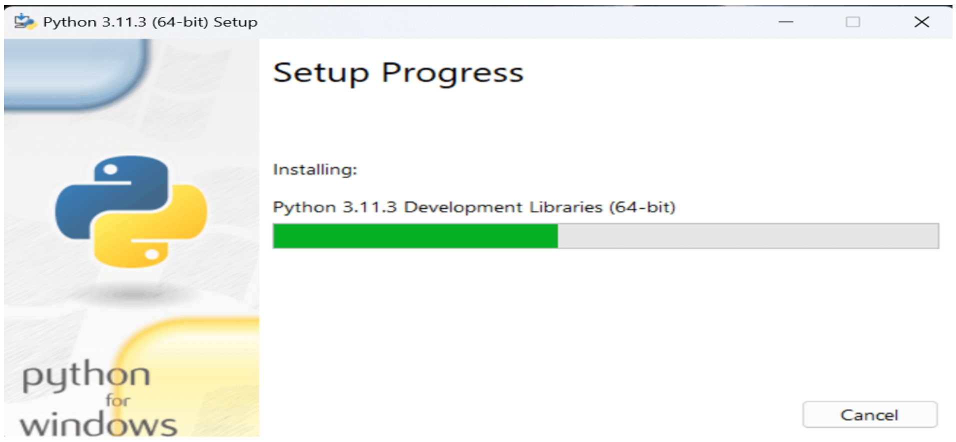
Advanced Options

☑ Install Python 3.11 for all users
☑ Associate files with Python (requires the 'py' launcher)
☑ Create shortcuts for installed applications
☑ Add Python to environment variables
☑ Precompile standard library
☐ Download debugging symbols
☐ Download debug binaries (requires VS 2017 or later)

Customize install location
C:\Program Files\Python311

Figure 16: python installation after customization

Step – 3: Installation in Process



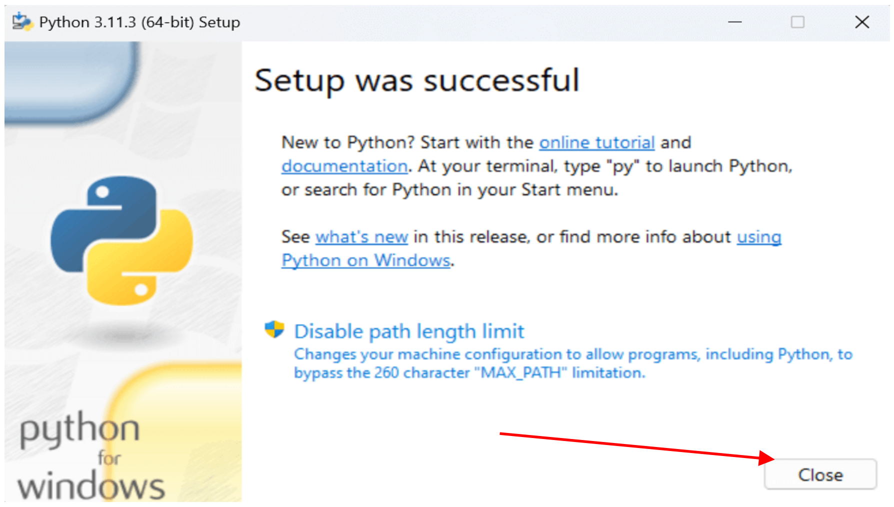
Setup Progress

Installing:

Python 3.11.3 Development Libraries (64-bit)

Figure 17: python is installed progressively

The setup is in progress. All the python libraries, packages, and other python default files will be installed in the system. Once the installation is successful, the following page will appear saying “Setup was successful ".



Setup was successful

New to Python? Start with the online tutorial and documentation. At your terminal, type "py" to launch Python, or search for Python in your Start menu.

See what's new in this release, or find more info about using Python on Windows.

Disable path length limit
Changes your machine configuration to allow programs, including Python, to bypass the 260 character "MAX_PATH" limitation.

Figure 18: close after Successful installation status is shown

Step - 4: Verifying the Python Installation

To verify whether the python is installed or not in the system, do the following.

- Go to "Start" button, and search " cmd ".
- Then type, " **python - - version** ".
- If python is successfully installed, then the version of the python installed appears.
- If not installed, then it will print the error as " **'python' is not recognized as an internal or external command, operable program or batch file.** ".

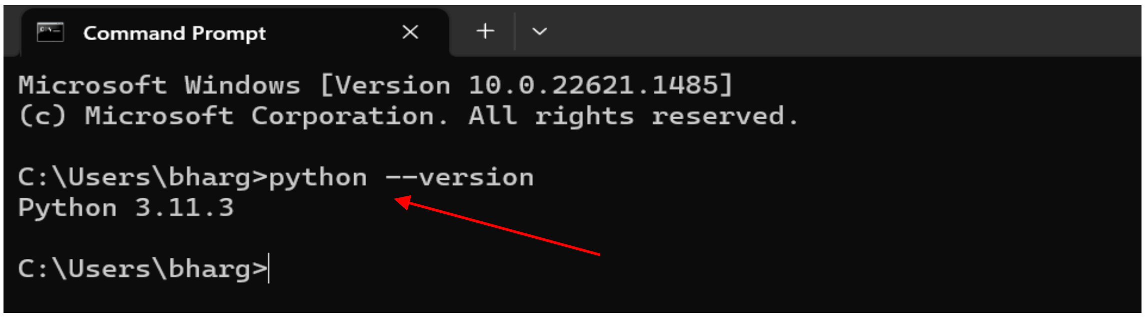
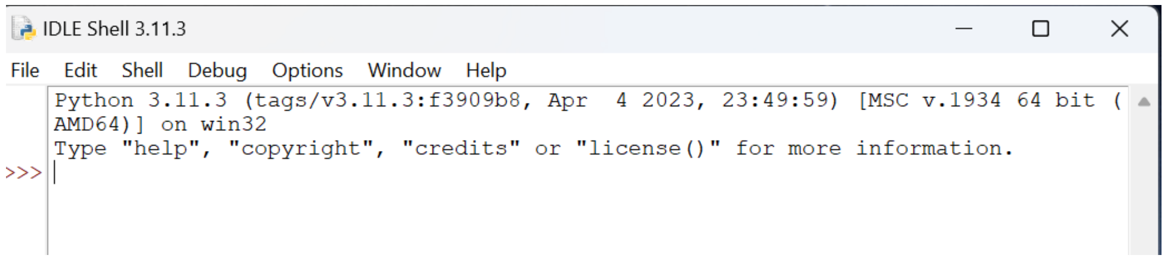

```
Microsoft Windows [Version 10.0.22621.1485]
(c) Microsoft Corporation. All rights reserved.

C:\Users\bharg>python --version
Python 3.11.3

C:\Users\bharg>
```

Figure 19: After installation verification, screenshot

Get ready to work with the Python.

Step - 5: Opening IDLE Shell

Now, to work on the first python program, move to the interactive interpreter prompt(idle). To open this, go to "Start" and type idle. Then, click on open to start working on idle.



```
Python 3.11.3 (tags/v3.11.3:f3909b8, Apr  4 2023, 23:49:59) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> 
```

Figure 20: Python IDLE screenshot

Practical Activity 1.2.2: Install Tools and Test Environment

Task:

1: You are requested to go to the computer lab to Install Tools and Test Environment as used in machine learning.

2: Apply safety precautions

3: Read key reading 1.2.2 and ask clarification where necessary

4: Present out the steps to Install Tools and Test Environment.

5: Referring to the steps provided on task 4, Install Tools and Test Environment.

6: Present your work to the trainer and whole class.

Key readings 1.2.2: Install Tools and Test Environment

- **nstallation of Tools**

Once Python is installed, the next step is to install essential tools, including libraries, IDEs, and frameworks, which are used to build machine learning models.

✓ **ractical Installation Steps:**

Step 1: Install Package Manager (pip)

- pip is used to install Python libraries and packages. It usually comes pre-installed with Python, but you can update it by running:

```
python -m pip install --upgrade pip
```

Step 2: Install Machine Learning Libraries

Essential libraries for ML include:

✓ **umPy**: For numerical computations.

✓ **andas**: For data manipulation and analysis.

✓ **cikit-learn**: A toolkit for building machine learning models.

✓ **atplotlib/Seaborn**: For data visualization.

✓ **ensorFlow/Keras/PyTorch**: For deep learning models.

You can install these libraries using pip:

```
pip install numpy pandas scikit-learn matplotlib seaborn tensorflow keras torch
```

Step 3: Install Integrated Development Environment (IDE)

An IDE simplifies coding by providing features like syntax highlighting, debugging, and code completion. Popular IDEs for Python include:

✓

upyter Notebook: An interactive environment to write and execute Python code, often used for data analysis and visualization.

```
pip install notebook
```

You can launch Jupyter Notebook by running jupyter notebook:

✓ **Visual Studio Code (VS Code)**: A lightweight code editor with powerful extensions for Python and ML development.

✓ **PyCharm**: An advanced IDE specifically designed for Python development, offering features like error detection and code refactoring.

Step 4: Install Anaconda (Optional)

- Anaconda is a popular distribution that simplifies package management and deployment. It comes with many pre-installed libraries and tools like Jupyter Notebook.
- Download and install Anaconda from the official website.
- After installation, you can manage environments and packages using conda:

```
conda create -n ml_env python=3.8

conda activate ml_env
```

- **Why Use Tools and Libraries?**

Machine learning requires various libraries to handle different tasks like data processing, model creation, and performance evaluation. Tools like IDEs and Jupyter notebooks help streamline the coding workflow, while libraries like TensorFlow and Scikit-learn abstract complex algorithms, enabling faster experimentation and implementation.

- **Choosing Between TensorFlow, PyTorch, and Keras**
  These deep learning frameworks offer different features:

✓ **TensorFlow**: Developed by Google, TensorFlow is highly scalable and widely used for both research and production. It allows distributed training.

✓ **PyTorch**: Preferred by researchers for its ease of use and dynamic computation graph, making debugging easier.

✓ **Keras**: A high-level API that runs on top of TensorFlow, making it easier for beginners to prototype neural networks.

✓ **Environment Testing**

Once the tools and libraries are installed, the environment must be tested to ensure that everything is working correctly.

✓ **Practical Steps for Testing:**

Step 1: Test Python Installation

- Open a terminal or command prompt and type:

```
python --version
```

This should return the installed Python version.

Step 2: Test IDE/Editor

- If using Jupyter Notebook, you can test it by launching it:

In the browser, create a new Python file and try running basic Python code like:

```
print ("Hello, Machine Learning!")
```

Step 3: Test ML Libraries

- Test the essential libraries by importing them in Python. For example:

```
import numpy as np
import pandas as pd
import tensorflow as tf
import sklearn
print("Libraries loaded successfully")
```

- If no errors occur, the libraries are installed correctly.

Points to Remember

- Set up a machine learning environment properly to ensure smooth functionality.
- Avoid skipping essential steps like verifying Python installation or neglecting to install key libraries, as this can lead to issues in running machine learning models later.

Application of learning 1.2.

As trainee in machine learning, you are requested to Install Python setup, Tools and Testing Python Environment in your computer lab for Preparing Machine Learning environment.

Indicative content 1.3: Data Collection and Acquisition

Duration: 6 hrs

Theoretical Activity 1.3.1: Data Collection and Acquisition

Tasks:

1: Answer to the following questions:

i. Describe the following terms used in data acquisition

a) Data

b) Information

c) Dataset

d) Data warehouse

e) Big data

ii. Explain the following source of data as used in machine learning?

a) IoT Sensors

b) Camera

c) Computer

d) Smartphone

e) Social data

f) Transactional data

iii. Describe 6 V's of Big Data

iv. Explain the types of data

2: Provide the answer for the asked questions and write them on papers.

3: Present the findings/answers to the whole class and trainer

4: For more clarification, read the key readings 1.3.1. In addition, ask questions where necessary.

Key readings 1.3.1.: Data Collection and Acquisition

- **Description of key terms**

- **Data:** Data refers to raw, unorganized facts and figures that are collected from various sources. It could be anything from numbers, text, or even sensor outputs (like sound or temperature). On its own, data has no specific meaning or context.

- **Information:** Information is data that has been processed, structured, and organized to provide context, relevance, and meaning. For example, a list of temperatures over a week becomes useful information when it is analysed to track weather patterns.
- **Dataset:** A **dataset** in machine learning is a collection of data that is used for training and testing models. It consists of structured data (such as tables with rows and columns) or unstructured data (like images, text, or audio) and plays a critical role in the model-building process.

  Common Formats for Datasets in Python

  –

  SV (Comma-Separated Values): A text file format where each line corresponds to a data record, and each record consists of fields separated by commas.

  –

  xcel Files: Spreadsheet files that can contain multiple sheets of data.

  –

  QL Databases: Datasets stored in relational databases, accessible through SQL queries.

  –

  SON (JavaScript Object Notation): A lightweight data-interchange format, often used for web APIs.

  –

  DF5: A binary data format commonly used for storing large amounts of scientific data.

- **Data warehouse:** A **Data Warehouse stores** a huge amount of data, which is typically collected from multiple heterogeneous sources like files, DBMS, etc.

  Benefits of Data Warehouse

  –

  etter business analytics

  –

  aster Queries

  –

  mproved data Quality:

  –

  istorical Insight

- **Big data: Big Data** refers to large, complex, and diverse datasets that are difficult to manage, process, or analyse using traditional data processing tools and methods.

- **Identification of Source of data**

When identifying sources of data, particularly in the context of big data and analytics, it's essential to recognize the different devices and systems that generate or capture the data.

- **IoT Sensors:** Internet of Things (IoT) sensors are devices that collect and transmit data about the environment they are monitoring. These sensors can measure various parameters like temperature, humidity, pressure, motion, light, and more.
- **Camera:** Cameras capture visual data in the form of images and videos. They can be used in surveillance, monitoring, and recognition systems.
- **Computer:** Computers generate a wide variety of data through different applications, software, and user interactions. This can include system logs, user activity, application data, and more.
- **Smartphone:** Smartphones are rich sources of data, generating information from various sensors (GPS, accelerometer, gyroscope), apps, and user interactions.
- **Social Data:** Social data comes from social media platforms like Twitter, Facebook, Instagram, LinkedIn, and others. It includes posts, comments, likes, shares, and other user-generated content.
- **Transactional Data:** Transactional data is generated from business transactions or operations, typically captured in databases. This data is vital for business operations, tracking sales, purchases, returns, payments, and more.

- **Description of 6 V's of Big Data**

The **6 V's of Big Data** represent key characteristics that define big data and help in understanding the complexities involved in managing and analyzing such vast amounts of information. Here's a description of each:

- **Volume:** Volume refers to the sheer amount of data generated every second across the globe. It is the most obvious characteristic of big data, as it deals with vast quantities of data.

  Example: Social media platforms generate petabytes of data daily from user interactions, posts, comments, and likes.
- **Variety:** describes the different types of data that are generated from multiple sources. Data can be structured, semi-structured, or unstructured.

  Example: Data types include text, images, videos, sensor data, log files, emails, social media posts, and more.
- **Veracity:** refers to the trustworthiness, accuracy, and reliability of the data. This dimension addresses data quality and the uncertainties present within the data.

  Example: Inconsistent data, missing values, and inaccuracies in data sources can lead to incorrect conclusions if not managed properly.
- **Value:** pertains to the usefulness of the data and the ability to turn big data into actionable insights that drive business decisions and add value to the organization.

Example: Analyzing customer behavior data to improve marketing strategies or using operational data to optimize supply chain processes.

- **Variability**: refers to the inconsistency of the data flow, which can be affected by various factors like seasonality, trends, or the context in which the data is generated.

Example: Retail sales data may vary significantly during holiday seasons compared to regular days, leading to fluctuating data volumes and types.

- **Velocity:** refers to the speed at which new data is generated, processed, and analyzed. In the era of real-time data processing, velocity is crucial for timely decision-making.

Example: Stock market transactions, social media updates, and IoT sensor data are generated and need to be processed in real-time or near-real-time.

- **Description of types of Data**
  - **Structured Data:** refers to data that is organized in a highly predictable and predefined manner, typically within a database or spreadsheet. This type of data is easily searchable and can be stored in tabular forms with rows and columns. Each field (column) contains a specific type of data (e.g., text, numbers), and each record (row) is a unique entry.
    - **Tabular Data**: Data organized in rows and columns, like spreadsheets or SQL databases. Eg.: customer information, financial records, and sensor readings.
    - **Time-Series Data**: Data collected at successive time points, such as stock prices or weather measurements. Eg.: stock prices, weather data, and website traffic logs.

Examples: Examples: Names, dates, phone numbers, currency or prices, heights or weights, word count or file size of a document, credit card numbers.

  - **Semi-structured Data** is a mix of structured and unstructured data. It does not reside in a traditional database format (tables and rows), but it does have some organizational properties, such as tags or markers, that make it easier to process than unstructured data. Semi-structured data is more flexible than structured data and can be stored in formats that allow for a variety of data types.
    - **XML Data**: Data marked up with tags, often used in web services.
    - **JSON Data**: Data in JavaScript Object Notation format, commonly used in web applications.

Examples: Images (both human or and machine-generated), video files, audio files, social media posts, product reviews, mobile SMS

  - **Unstructured Data** refers to data that does not follow a predefined structure or format. Unstructured data is more difficult to analyze and manage.

- **Text Data**: Data in the form of text, such as articles, emails, or social media posts.
- **Image Data**: Data consisting of visual information, used in tasks like image classification or object detection.
- **Audio Data**: Data captured in audio formats, like speech & music, used in speech recognition or audio classification.
- **Video Data**: Data in the form of video sequences, which is used in video classification or activity recognition.

Example: Emails, social media posts, multimedia content (images, videos, audio), and PDF documents.

- **Gathering Machine Learning Dataset**

Gathering a machine learning dataset is a critical step in the machine learning pipeline. The quality, relevance, and size of the dataset significantly influence the performance of the machine learning model. A well-curated dataset is essential for building accurate and robust models.

Below are key considerations for gathering datasets:

Define the Objective

- **Identify the Problem**: Clearly define the task (e.g., classification, regression, clustering, etc.) and the target variable (output).
- **Understand Data Needs**: Specify what type of data is required (e.g., images, text, numerical, or a combination).
- **Data Quantity**: Estimate the amount of data needed to train the model effectively. More complex models often require larger datasets.

Identify Data Sources

- **Public Datasets**: Look for datasets available on platforms like Kaggle, UCI Machine Learning Repository, Google Dataset Search, or government databases.
- **Sensor Data**: Collect data from sensors like temperature, humidity, or accelerometer readings…
- **Web Scraping**: Use tools like Beautiful Soup or Scrapy to collect data from websites.
- **APIs**: Access data through APIs from services like Twitter API, Google Books API, or any relevant database.
- **Synthetic Data**: Generate data using libraries like scikit-learn or faker if real data is scarce.
-

nternal Sources: Gather data from internal systems, such as CRM (customer relationship management) databases, Interviews, questionnaires, experiments, operational logs, or financial records.

Data Collection Techniques

- **APIs**: Access pre-existing data through APIs (e.g., Twitter API, OpenWeatherMap API).
- **Manual Data Entry**: For small datasets or domain-specific tasks.
- **Crowdsourcing**: Platforms like Amazon Mechanical Turk for labeling tasks.
- **Sensors and IoT Devices**: For real-world data collection, e.g., environmental data or wearables.

Dataset Storage:

- **Local Storage**: For smaller datasets, storing them locally on a machine or server may suffice.
- **Cloud Storage**: For large datasets, use cloud storage solutions like AWS S3, Google Cloud Storage, or Azure Blob Storage, which provide scalable, reliable, and secure data storage.

Practical Activity 1.3.2: Gathering Machine Learning dataset

Task:

1:You are requested to go to the computer lab to gather the dataset (1) iris dataset, (2) Predicting house prices, (2) classifying emails as spam or not, (4) identifying handwritten digits.
2: Apply safety precautions

3: Present out the steps to gather the dataset.

4: Referring to the steps provided, gather the dataset.

5: Present the work to the trainer and whole class.

6: Read key reading 1.3.2 and ask clarification where necessary

Key readings 1.3.2: Gathering Machine Learning dataset

Gathering the dataset can be done either manually or using Python libraries.

Below are various methods to obtain the (1) iris dataset, (2) Diabetes dataset, (3) Wine dataset, (4) Breast Cancer dataset, (5) Digits dataset., by downloading it directly from a URL or using **Pandas** and **Scikit-learn** libraries.

For sampling, let’s use the iris dataset.

1. Downloading the Dataset Manually

To download the dataset manually and load it from a local file, follow these steps:

→ **Download the Dataset**: Go to the UCI Machine Learning Repository's Iris Dataset page and download the file named iris.data.

→ **Load the Dataset from a Local File**:

```
import pandas as pd
# Load the Iris dataset from a local file
file_path = 'path_to_your_file/iris.data'  # Replace with your local file path
columns = ["SepalLength", "SepalWidth", "PetalLength", "PetalWidth", "Species"]
iris_data = pd.read_csv(file_path, header=None, names=columns)
# Display the first few rows
print(iris_data.head())
```

2. Loading the Iris Dataset Using Pandas

Load the Iris dataset directly from a URL using the Pandas library, as follows:

```
import pandas as pd
# Load the Iris dataset from the UCI Machine Learning Repository
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
columns = ["SepalLength", "SepalWidth", "PetalLength", "PetalWidth", "Species"]
iris_data = pd.read_csv(url, header=None, names=columns)
# Display the first few rows
print(iris_data.head())
```

2. Loading the Iris Dataset Using Scikit-learn

Scikit-learn provides a built-in method to load the Iris dataset easily. Here's how:

```
from sklearn.datasets import load_iris
import pandas as pd
# Load the Iris dataset
```

```
iris = load_iris()
# Create a DataFrame from the dataset
iris_data = pd.DataFrame(data=iris.data, columns=iris.feature_names)
iris_data['Species'] = iris.target_names[iris.target]
# Display the first few rows
print(iris_data.head())
```

Summary of the Dataset Structure

After loading the dataset, this is a DataFrame (iris_data) with the following structure:

SepalLength	SepalWidth	PetalLength	PetalWidth	Species
5.1	3.5	1.4	0.2	Iris Setosa
4.9	3.0	1.4	0.2	Iris Setosa
4.7	3.2	1.3	0.2	Iris Setosa
6.0	2.2	5.0	1.5	Iris Virginica
6.3	2.5	5.0	1.9	Iris Virginica

Points to Remember

- Data collection is key and involves sources like IoT sensors, cameras, computers, and social media.
- You can find machine learning datasets on platforms like UCI Machine Learning Repository, Kaggle, Scikit-learn, OpenML, and others such as Google Dataset Search, Data.gov, and GitHub.
- The gathered dataset can be stored on local storage or cloud storage.

Application of learning 1.3.

As a trainee in machine learning, you are requested to gather the dataset (1) Manually, (2) using **pandas and Scikit-learn** libraries

1. Visit any platform and download the following dataset
   (i) Diabetes dataset
   (ii) Wine dataset
   (iii) Breast Cancer dataset
   (iv) Car dataset

(v) COVID-19 Dataset

(vi) Energy Efficiency Dataset

(vii) penguins dataset

(viii) iris dataset

2. Load the above datasets using python libraries

Indicative content 1.4: Interpret Data Visualization.

Duration: 6 hrs

Theoretical Activity 1.4.1: Describing data Visualization tools

Tasks:

1: Answer the following questions related to the Description of data Visualization tools

i. Describe the following popular tools used for data visualization

a) Matplotlib

b) Seaborn

c) Plotly

d) Tableau

e) Power BI

2: Provide the answer for the asked questions and write them on papers.

3: Present the findings/answers to the whole class and trainer

4: For more clarification, read the key readings 1.4.1.

Key readings 1.4.1.: Describing data Visualization tools

- **Description of Data Visualization Tools**

Data visualization is the graphical representation of data using charts, graphs, and other visual tools. The primary purpose of data visualization is to make complex data more understandable by presenting it in a clear and concise visual format. It allows users to identify patterns, trends, and insights that might not be easily apparent from raw data alone. Visualizations make it easier to communicate findings, support decision-making, and uncover hidden relationships within the data.

Several tools are available to create compelling visual representations of data. Below are some popular tools used for data visualization:

✓ **Matplotlib**

Matplotlib is a Python-based plotting library that provides a wide range of static, animated, and interactive visualizations. It is highly customizable and allows users to create publication-quality graphs in a variety of formats.

Use Cases: It's commonly used for 2D plotting tasks such as line graphs, bar charts, histograms, scatter plots, and more. It’s suitable for basic, static visualizations and is often used in academic and scientific settings.

Advantages:

○ Highly customizable.

○ Simple to use for basic plots.

○ Extensive documentation and a large user community.

Disadvantages:

○ Requires more lines of code for complex visualizations compared to some other libraries.

○ Less aesthetically pleasing by default (requires customization for better looks).

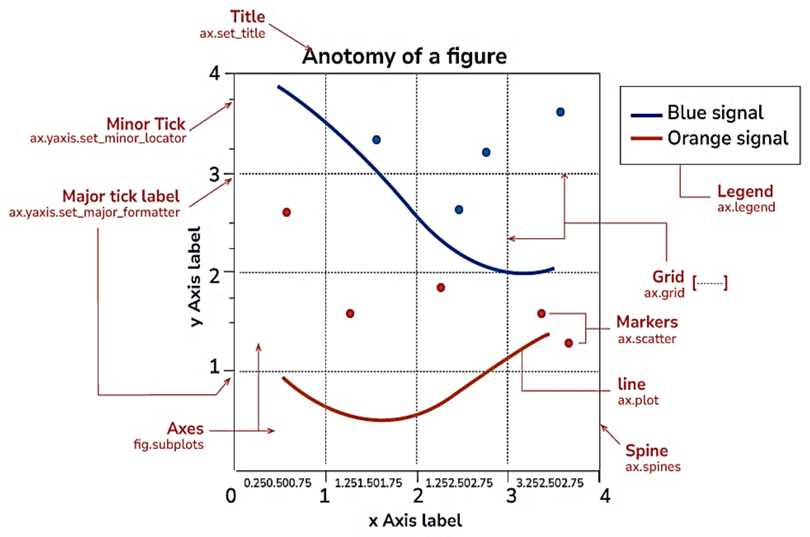


Figure 21: anatomy of matplotlib figure

✓ **Seaborn**

Seaborn is built on top of Matplotlib and provides a high-level interface for drawing attractive and informative statistical graphics. It is particularly good at creating complex visualizations in fewer lines of code compared to Matplotlib, with a focus on aesthetics and ease of use.

Use Cases: Seaborn is ideal for statistical plots, including heatmaps, box plots, violin plots, and categorical plots. It’s widely used in data analysis for making data distributions and relationships between variables easy to understand.

Advantages:

○ Simple, clean, and aesthetically pleasing visualizations.

○ Less code is required to generate complex plots.

○ Built-in support for working with data from pandas DataFrames.

Disadvantages:

○ Less flexible than Matplotlib for detailed customization.

○ Limited interactivity compared to other tools like Plotly.

Figure 22: seaborn illustration diagram

✓ **Plotly**

Plotly is an open-source graphing library that supports both static and interactive visualizations. It integrates well with Python and other programming languages, providing the ability to create rich, web-based plots that can be easily shared or embedded in web applications.

- **Use Cases**: Plotly is used for creating interactive visualizations such as line charts, scatter plots, 3D plots, and even maps. It is well-suited for dashboards and reports that require interactive elements, such as zooming, hovering, and panning.
- **Advantages**:

○ Offers highly interactive and visually appealing plots.
○ Supports 3D visualizations and geographical data.
○ Easy integration with web applications (e.g., Dash framework).

- **Disadvantages**:

○ More complex to set up compared to Matplotlib and Seaborn.
○ Some advanced features are available only in the paid version (Plotly Enterprise).

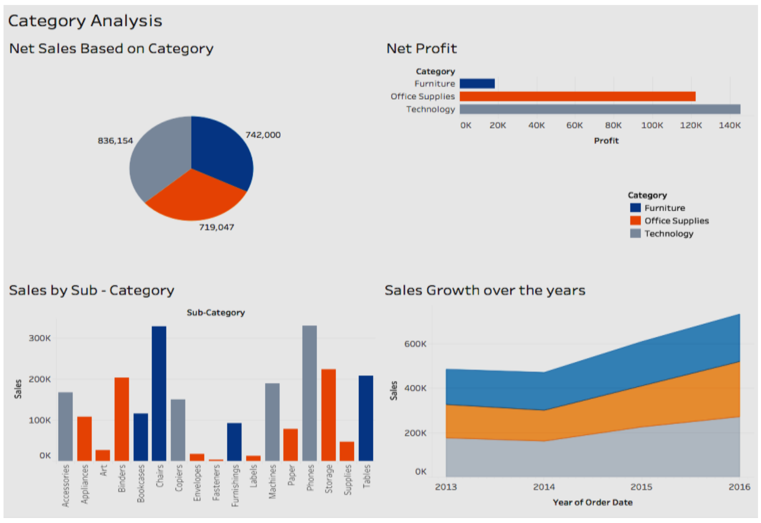


Figure 23: Plotly illustration diagram

✓ **Tableau**

Tableau is a powerful, user-friendly data visualization tool that allows users to create interactive and shareable dashboards. It provides drag-and-drop functionality, enabling users to quickly connect to data sources, create dashboards, and publish reports without any coding knowledge.

Use Cases: Tableau is commonly used for business intelligence and dashboard reporting. It's ideal for non-technical users who need to visualize data, conduct exploratory analysis, and create interactive reports.

Advantages:

- No programming required; easy to use.
- Highly interactive and visually appealing dashboards.
- Integrates with multiple data sources.

Disadvantages:

- Expensive for commercial use.
- Limited customization options compared to coding-based tools.

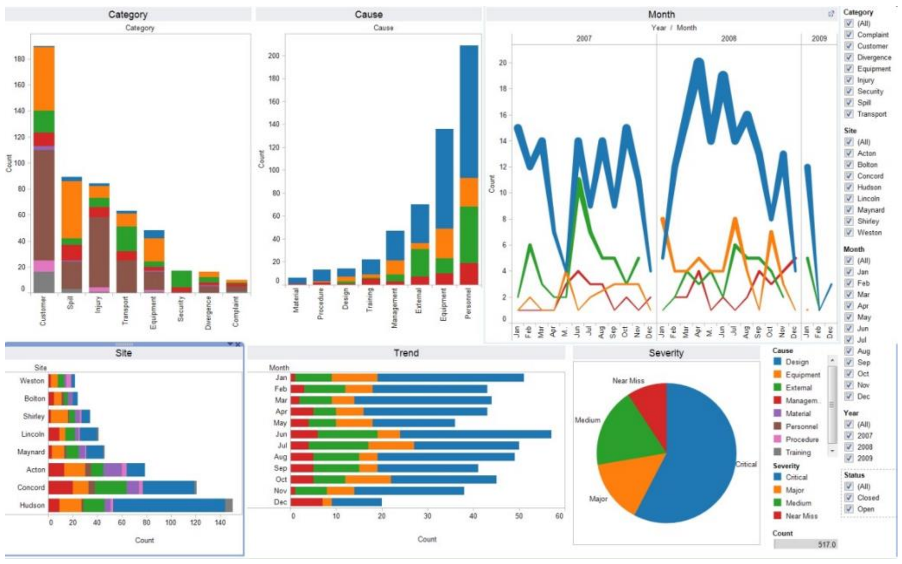


Figure 24: Illustration of Tableau tool

✓ **Power BI**

Power BI is a business analytics tool by Microsoft that enables users to create interactive reports and dashboards. It offers integration with various Microsoft services, making it ideal for enterprises using the Microsoft ecosystem. It supports both desktop and cloud-based solutions.

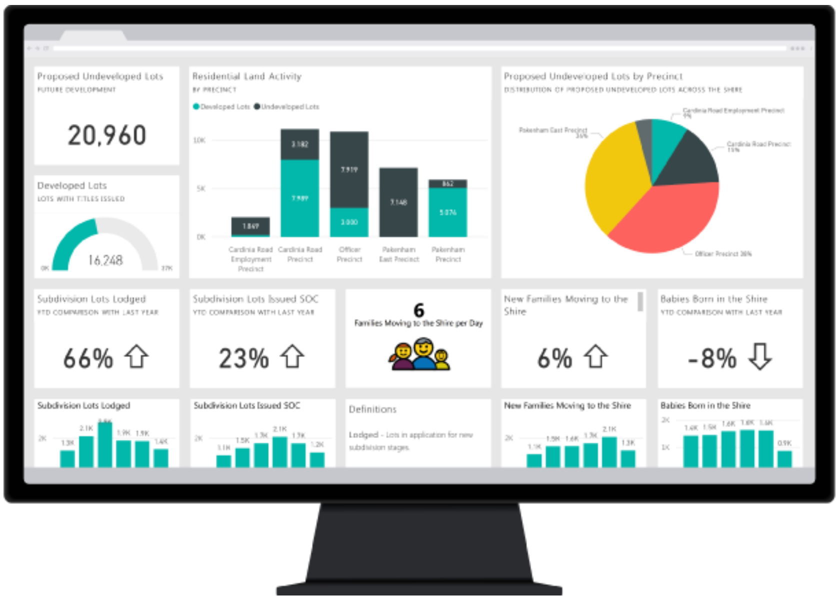
Use Cases: Power BI is used for business intelligence, reporting, and dashboard creation. It's particularly useful for companies that need to analyze large datasets, visualize KPIs, and share insights with stakeholders.

Advantages:

- Strong integration with Microsoft services like Excel, Azure, and SQL Server.
- User-friendly interface with drag-and-drop capabilities.
- Built-in AI features for advanced analytics.

Disadvantages:

- Limited flexibility in terms of customization.
- Performance can be an issue with extremely large datasets.



Proposed Undeveloped Lots
20,960
Developed Lots
16,248
Residential Land Activity
Proposed Undeveloped Lots by Precinct
Subdivision Lots Lodged
66%
Subdivision Lots Issued SOC
23%
6
Families Moving to the Shire per Day
New Families Moving to the Shire
6%
Babies Born in the Shire
-8%
Definitions

Figure 25: Illustration of Power BI tool

Each of these tools has its strengths depending on the level of interaction, complexity, and user experience required. Selecting the right tool depends on factors such as the data type, desired output, and the technical expertise of the user.

Practical Activity 1.4.2: Use Types of Data Visualization

Task:

1: You are requested to go to the computer lab to Use Types of Data Visualization as used in machine learning.

2: Apply safety precautions

3: Present out the steps to Use Types of Data Visualization.

4: Referring to the steps provided, use types of data visualization.

5: Present your work to the trainer and whole class

6: Read key reading 1.4.2 and ask clarification where necessary

Key readings 1.4.2 Use Types of Data Visualization

- **Types of Data Visualization**

Data visualization helps to understand relationships, trends, and patterns in data.

Let’s explore how some common types of visualizations are practically used, along with Python implementations using libraries like Matplotlib and Seaborn.

✓ **Scatter Plots**

Scatter plots visualize the relationship between two continuous variables, showing how one variable correlates with another. Points represent data values plotted along two axes.

- **Use Case**: Visualizing the correlation between height and weight of individuals, or the relationship between a company’s advertising spend and its sales revenue.

- **Practical Example** (Python with Matplotlib):

Plotting on scatter plot the height and weight data using matplotlib.

```
import matplotlib.pyplot as plt
# Example data
height = [160, 165, 170, 175, 180]
weight = [60, 65, 70, 75, 80]
# Scatter Plot
plt.scatter(height, weight)
plt.xlabel('Height (cm)')
plt.ylabel('Weight (kg)')
plt.title('Scatter Plot: Height vs Weight')
plt.show()
```

Answer:

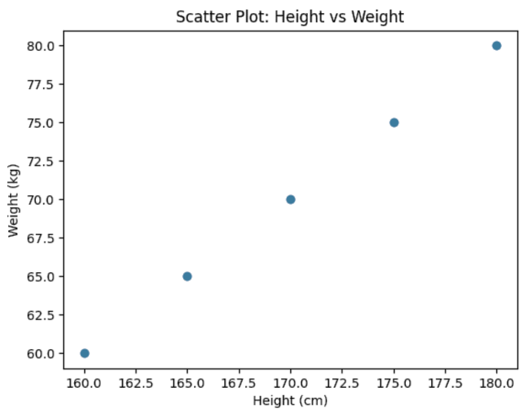


Figure 26: Simple scatter plot

<u>Using an existing sample dataset.</u>

Using scatter function with seaborn for a sample dataset (tips):

```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
tips = sns.load_dataset('tips')
sns.scatterplot(data=tips, x='total_bill', y='tip', hue='day')
plt.title('Total Bill vs Tip')
plt.show()
```

Answer:

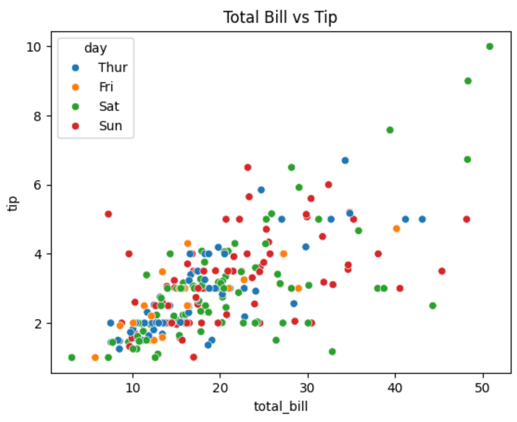


Figure 27: scatter plot of dataset using seaborn

✓ **Line Plots**

Line plots are used to display trends over time. They are particularly useful for showing changes in data points, such as stock prices, over regular intervals like days or months.

- **Use Case**: Tracking monthly sales or showing the trend of a company's revenue growth over the past years.

- **Practical Example**:

Plotting on a line chart the months and sales data using matplotlib.

```
import matplotlib.pyplot as plt
# Example data
months = ['Jan', 'Feb', 'Mar', 'Apr', 'May']
sales = [500, 600, 550, 650, 700]
# Line Plot
plt.plot(months, sales)
plt.xlabel('Month')
plt.ylabel('Sales')
plt.title('Monthly Sales Trend')
plt.show()
```

Answer:

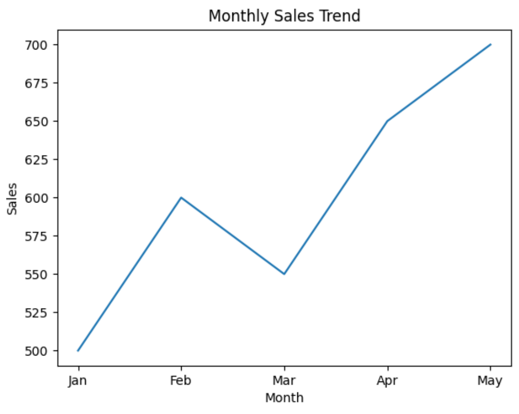


Figure 28: simple line plot using matplotlib

Draw a line in a diagram from position (10,20),(20,25), (30,35), (40,55):

```
import matplotlib.pyplot as plt
x = [10, 20, 30, 40]
y = [20, 25, 35, 55]
plt.plot(x, y, color='green', linewidth=3, marker='o', markersize=15, linestyle='--')
plt.title("Line Chart")
plt.ylabel('Y-Axis')
plt.xlabel('X-Axis')
plt.show()
```

Answer:

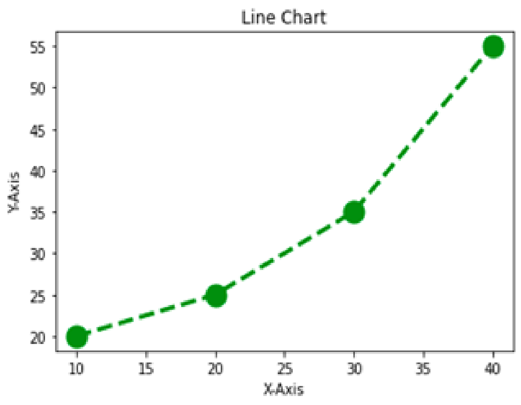


Figure 29: Line plot with aesthetics

Using combined functions with seaborn:

```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
tips = sns.load_dataset('tips')
plt.figure(figsize=(12, 6))
# Scatter plot
sns.scatterplot(data=tips, x='total_bill', y='tip', hue='time', alpha=0.7)
# Regression line
sns.regplot(data=tips, x='total_bill', y='tip', scatter=False, color='red')
plt.title('Total Bill vs Tip with Regression Line')
plt.show()
```

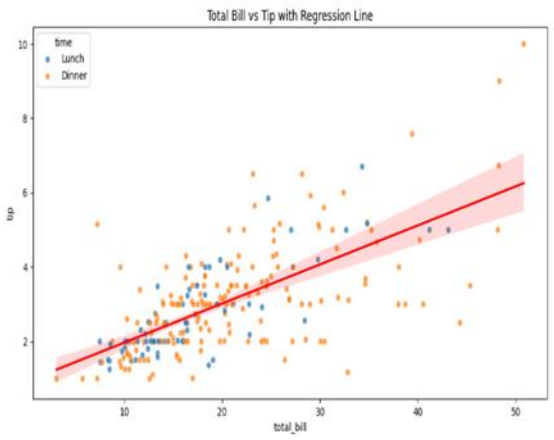


Figure 30: More functions combined in on plot using Seaborn

✓ **Bar Charts**

Bar charts are used to compare different categories or groups. They represent data using rectangular bars where the length of each bar is proportional to the data value.

- **Use Case**: Comparing the sales of different products in a store, or visualizing the population of different cities.

- **Practical Example**:

Plotting on bar chart the products and sales data using matplotlib.

```
import matplotlib.pyplot as plt
# Example data
products = ['Product A', 'Product B', 'Product C']
sales = [300, 450, 500]
# Bar Chart
plt.bar(products, sales)
plt.xlabel('Product')
plt.ylabel('Sales')
plt.title('Sales of Different Products')
plt.show()
```

Answer:

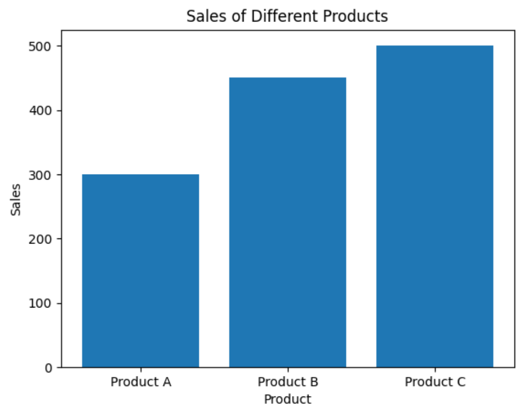


Figure 31: Vertical bar chart

Draw 4 horizontal bars:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])
plt.barh(x, y, color='orange')
plt.xlabel('Letter')
plt.ylabel('Quantity')
plt.title('Quantity of Leters')
plt.show()
```

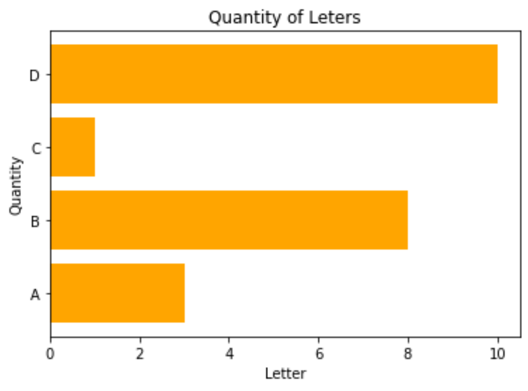


Figure 32: Horizontal boar chart

✓ **Histograms**

Histograms display the distribution of a dataset. They group data into bins and show how frequently data points fall within each bin.

- **Use Case**: Understanding the distribution of exam scores in a class or visualizing the distribution of salaries in a company.

- **Practical Example**:

Plotting on histogram chart the scores data using matplotlib.

```
import matplotlib.pyplot as plt
# Example data (randomly generated exam scores)
scores = [55, 65, 75, 85, 90, 95, 60, 70, 80, 85, 95, 100]
# Histogram
plt.hist(scores, bins=5)
plt.xlabel('Score')
plt.ylabel('Frequency')
plt.title('Distribution of Exam Scores')
plt.show()
```

Answer:

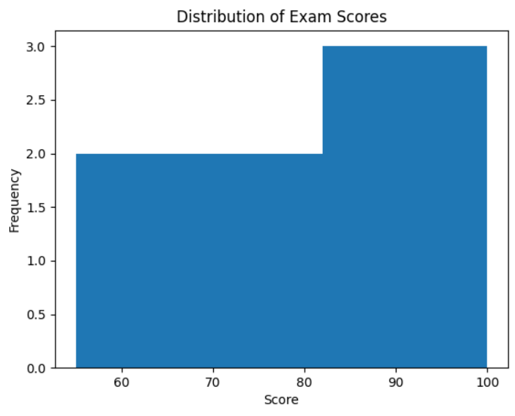


Figure 33: Histogram chart using matplotlib

✓ **Box Plots**

Box plots, or box-and-whisker plots, are used to show the distribution of a dataset through its quartiles. They help in identifying outliers and understanding the spread of the data.

- **Use Case**: Comparing the distribution of salaries in different departments of a company, or visualizing the spread of test scores among different groups of students.

- **Practical Example**:

Plotting on box plot the scores data generated randomly for two groups using matplotlib and seaborn.

```
import seaborn as sns
import matplotlib.pyplot as plt
# Example data (randomly generated
data for two groups)
data = {'Group': ['A', 'A', 'A', 'B', 'B', 'B'],
'Scores': [70, 85, 90, 60, 75, 80]}
# Box Plot
sns.boxplot(x='Group',          y='Scores',
data=data)
plt.title('Box Plot of Scores by Group')
plt.show()
```

Answer:

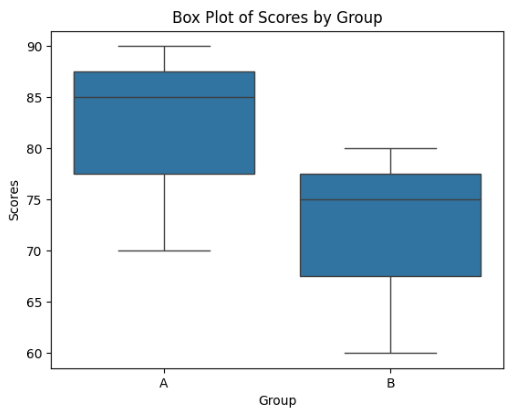


Figure 34: Box plot using seaborn

✓ **Heat Maps**

Heat maps are used to represent matrix-like data where the intensity of color represents the magnitude of values. It is often used to show correlations between variables or to display the density of data points.

- **Use Case**: Visualizing the correlation matrix of different features in a dataset or showing the intensity of activity on a website by hour and day.

- **Practical Example**:

Plotting on Heatmap plot the data using matplotlib, NumPy and seaborn.

```
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
# Example data (correlation matrix)
data = np.random.rand(5, 5)  # 5x5 random
matrix
# Heatmap
sns.heatmap(data,annot=True,cmap='coolwar
```

Answer:

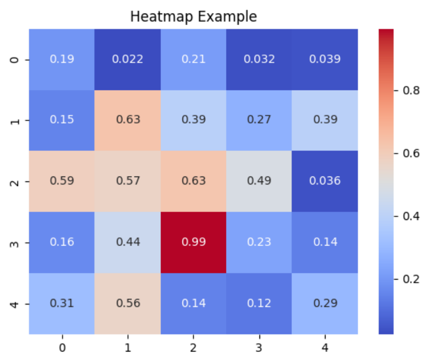


Figure 35: Heatmap plot using seaborn

```
m')
plt.title('Heatmap Example')
plt.show()
```

These visualization types are essential for exploratory data analysis and for conveying findings effectively in data-driven projects.

Practical Activity 1.4.3: Applying data Visualization Best Practices and Interpreting visualizations results.

Task:

1: You are requested to go to the computer lab to apply best practices data Visualization and Interpreting Visualized results

2: Apply safety precautions

3: Present out the steps to best practices data Visualization and Interpreting Visualized results.

4: Referring to the steps provided, apply best practices data Visualization and Interpreting Visualized results.

5: Present your work to the trainer and whole class

6: Read key reading 1.4.3 and ask clarification where necessary

Key readings 1.4.3: Applying data Visualization Best Practices and Interpreting visualizations results.

- **Applying Data Visualization Best Practices**

Creating effective data visualizations requires following certain best practices to ensure the visuals are clear, engaging, and easy to interpret. Here's how to apply these best practices:

✓ **Know Your Audience**

Tailor the visualization style to the knowledge level and interests of the audience. For example, technical audiences may prefer detailed graphs, while non-technical audiences need simple, easy-to-read visuals.

✓ **Simplify Visuals**

Avoid overloading your visualization with excessive details. Simplicity ensures that the audience can quickly grasp the key message without distractions.

✓ **Choose the Right Chart Type**

Selecting the appropriate chart type is crucial for conveying information effectively. For example, bar charts are better for comparisons, while line charts are suitable for trends over time.

✓ **Use Appropriate Scales**

Ensure axes, labels, and legends are scaled appropriately so that the data is not misleading.

✓ **Label Clearly**

All elements of the visualization, including axes, legends, and titles, should be clearly labeled for ease of understanding.

✓ **Choose Colors Wisely**

Colors are powerful for highlighting important data points, but they can also distract or mislead if used improperly.

✓ **Avoid Misleading Visualizations**

Misleading visuals can distort the message and lead to incorrect conclusions.

• **Interpreting Visualization Results**

Once visualizations are created, interpreting them correctly is essential to derive actionable insights. Interpretation involves recognizing patterns, identifying trends, and understanding relationships in the data.

✓ **Patterns and Trends**

Identifying recurring patterns and trends in data visualizations helps make predictions and understand the data over time or across categories.

- **Example**: A line plot showing the trend of rising sales over months can indicate growing market demand.

- **Interpretation**:

○ **Upward Trend**: Growth in the variable (e.g., increasing sales or population).

○ **Downward Trend**: Decline in the variable (e.g., decreasing website traffic).

○ **Seasonal Pattern**: Regular fluctuations based on specific times or conditions (e.g., retail sales peaking during holidays).

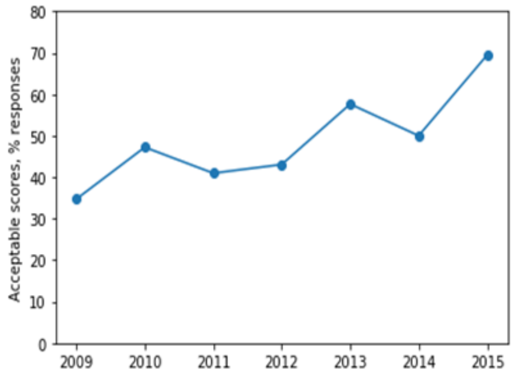


Figure 36: Line chart showing upward trends.

Example: Based on the codes, describe the trends of the given data.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
# Create some sample data with patterns and trends
data = np.linspace(0, 10, 100)
data = data + np.sin(data)
# Add a sine wave pattern
# Create a pandas DataFrame
df = pd.DataFrame({'data': data})
# Plot the data to visualize patterns and trends
plt.plot(df['data'])
plt.xlabel('Index')
plt.ylabel('Value')
plt.title('Sample Data with Patterns and Trends')
plt.grid(True)
plt.show()
```

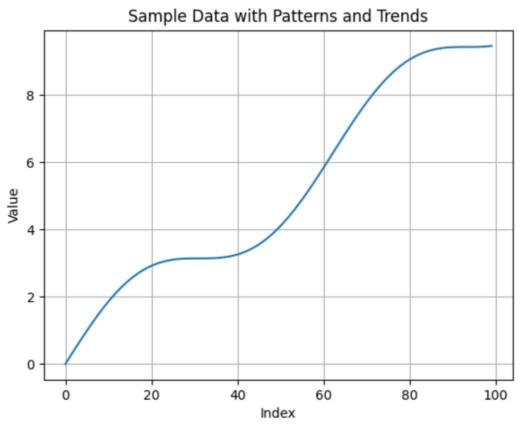


Figure 37: sample figure showing upward trends of data

✓ **Context and Background**

Always consider the context of the data being visualized. Without the right context, visualizations can be misinterpreted, leading to incorrect conclusions.

- **Example**: A sudden spike in sales data may need the context of a marketing campaign, holiday season, or product launch to fully understand the reason for the increase.

- **Interpretation**:

○**Contextual Analysis**: Take into account external factors like economic events, business changes, or global trends to explain the data.

○**Background Information**: Always include background information for the audience to understand the data source, time range, and relevance.

✓ **Correlations and Relationships**

Visualizations such as scatter plots and heat maps can reveal correlations between variables. Correlations help in understanding how one variable might affect or be associated with another.

- **Example**: A scatter plot showing the relationship between temperature and ice cream sales can reveal a positive correlation, where higher temperatures lead to increased sales.

- **Interpretation**:

○**Positive Correlation**: As one variable increases, the other increases (e.g., higher study hours leading to higher exam scores).

○**Negative Correlation**: As one variable increases, the other decreases (e.g., more rainfall leading to lower water consumption).

○**No Correlation**: No clear relationship between variables (e.g., shoe size and intelligence).

○**Causation vs. Correlation**: Be careful to distinguish between correlation (variables move together) and causation (one variable causes changes in another).

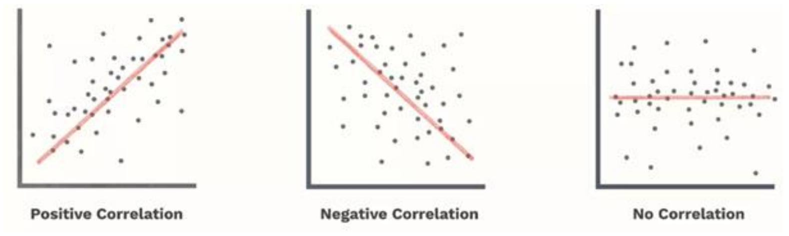


Figure 38: Correlation coefficient types

Points to Remember

- Apply best practices: Simplify visuals, choose appropriate charts, label axes, and avoid misleading representations.
- When interpreting visualizations, identify patterns, provide context, and understand relationships between data variables.

Application of learning 1.4.

As a trainee in machine learning, you are requested to develop a robust machine learning by choosing best Data Visualization tools and interpret visualization result based on iris dataset.

Indicative content 1.5: Perform Data Cleaning.

Duration: 6 hrs

Theoretical Activity 1.5.1: Description of data cleaning.

Tasks:

1: Answer the following questions related to the description of data cleaning

i. Define data cleaning and explain its purpose.

ii. Explain steps involve in data cleaning

iii. Describe characteristics of quality Data

iv. Explain the importance of data cleaning for inconsistencies rectification

v. Explain data cleaning techniques for inconsistencies rectification

vi. Discuss on importance of data normalization

vii. Describe data normalization techniques

viii. Explain the importance of data transformation

ix. Describe data transformation Techniques

x. Explain types of Data transformation

2: Provide the answer for the asked questions and write them on papers.

3: Present the findings/answers to the whole class and trainer

4: For more clarification, read the key readings 1.5.1. In addition, ask questions where necessary

Key readings 1.5.1: Description of data cleaning

- **Data Cleaning Overview**

Definition: Data Cleaning refers to the process of detecting and correcting (or removing) corrupt or inaccurate records from a dataset. It involves dealing with missing values, inconsistencies, duplicates, and noisy data to improve the dataset's quality.

Purpose: The **primary goal** of data cleaning is to ensure that the dataset is accurate, complete, and ready for analysis. Clean data improves the reliability of statistical results and the performance of machine learning models.

- **Steps of data cleaning:**

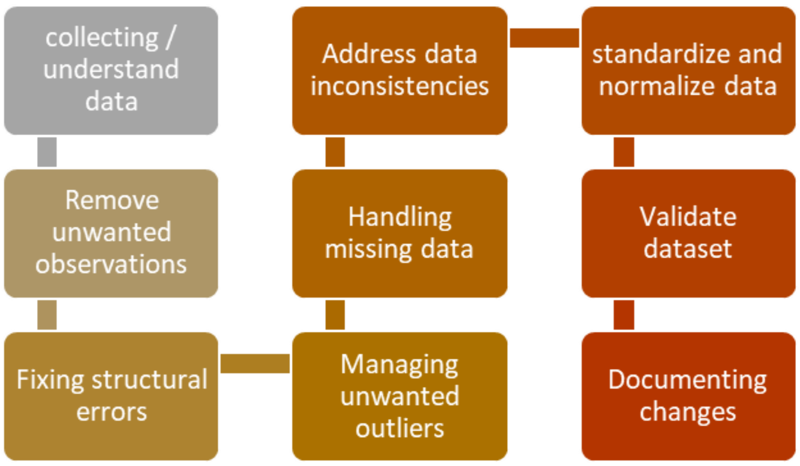

collecting / understand data
Remove unwanted observations
Fixing structural errors
Managing unwanted outliers
Handling missing data
Address data inconsistencies
standardize and normalize data
Validate dataset
Documenting changes

Figure 39: Data cleaning steps

1. Understand the data

- **Review Data Sources**: Understand where the data comes from (e.g., databases, APIs, or surveys).
- **Explore the Data**: Examine the structure, types of variables, and basic statistics (e.g., mean, median, mode, outliers).
- **Define Objectives**: Know what you want to achieve with the dataset.

2. Handling Missing Values

Identify Missing Data: Check for NaN, blank cells, or placeholders like "NA" or "?". Use methods like .isnull() in pandas to find missing values.

Imputation: Fill missing values using:

- Mean/median/mode for numerical data.
- Mode for categorical data.
- Forward fill or backward fill for time series data.

Removal: Drop rows or columns with excessive missing data.

3. Removing Duplicates

- **Identify Duplicate Rows**: Use tools like pandas' .duplicated() to check for duplicates.
- **Remove** duplicate records using methods like .drop_duplicates() in pandas. This will help to remove (duplicate or irrelevant) unwanted observations, to avoid redundancy and inflated results.

4. Fixing Structural Errors, Ensuring Consistent Data Types

- **Standardize Column Names**: Ensure uniform naming conventions.
- **Correct Data Types**: Convert columns to appropriate types (e.g., string, integer, date).
- **Resolve Inconsistencies**: Fix inconsistent formatting (e.g., "NYC" vs. "New York City").

5. Standardizing Data Formats

- Ensure consistent formats for dates, phone numbers, etc.
- Convert all text to lower/upper case to maintain uniformity.

6. Outlier Detection and Treatment

Detect and deal with outliers that could skew the analysis, either by removing or capping them.

- **Visualization**: Use box plots or scatter plots to identify outliers.
- **Statistical Methods**: Z-score or IQR methods to detect outliers.
- **Handling**: Remove or transform outliers based on the context.

7. Addressing Data Inconsistencies:

- Standardize the format of data entries (e.g., date formats, units of measurement, or text capitalization).
- Resolve issues like duplicate entries, mismatched units, or contradictory records.

8. Normalizing Data

- **Normalization**: Scale or transform variables to a consistent range, especially for numerical data, to ensure uniformity. Scale data to a range (e.g., 0 to 1) for algorithms sensitive to magnitude.

9. Validating Dataset

- Ensure that data adheres to predefined rules or constraints (e.g., age should be non-negative).
- **Cross-check Data**: Compare with external or reliable sources. Addressing Data Inconsistencies
- **Run Tests**: Validate data integrity and correctness with unit tests or rules.

10. Documenting Changes

Keep a log of the cleaning process to track changes made to the dataset.

Tools and Libraries

- **Pandas**: For data manipulation and cleaning.
- **NumPy**: For numerical operations.
- **Scikit-learn**: For preprocessing tasks.
- **Matplotlib/Seaborn**: For data visualization.

Example Workflow in Python

```
import pandas as pd

# Load data

data = pd.read_csv('data.csv')

# Identify and handle missing values

data.fillna(data.mean(), inplace=True)

# Remove duplicates

data.drop_duplicates(inplace=True)

# Standardize text

data['category'] = data['category'].str.lower()

# Encode categorical variables

data = pd.get_dummies(data, columns=['category'], drop_first=True)

# Normalize numerical features

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

data[['numerical_feature']] = scaler.fit_transform(data[['numerical_feature']])

# Display cleaned data

print(data.head())
```

1.5.2. Characteristics of quality Data in machine learning

High-quality data is critical for accurate analysis and decision-making. Several characteristics define data quality:

- **Accuracy:**

Accuracy ensures that data values are correct and reflect real-world observations. Errors in accuracy can lead to incorrect conclusions.

Example: If a dataset contains a column for "age," an entry showing "178" would be inaccurate, as it’s impossible for a human to live that long.

- **Completeness:**

Data is considered complete if there are minimal or no missing or incomplete entries that could hinder analysis. Each feature should have sufficient data points to ensure robust analysis

Example: A sales dataset missing the “date of purchase” for several records would be incomplete and potentially unreliable.

- **Consistency:**

Consistent data means that there are no conflicting values in the dataset, such as different formats or contradicting entries.

Example: - A dataset with dates in both MM/DD/YYYY and DD/MM/YYYY formats can lead to confusion and misinterpretation.

- If some entries for "country" are listed as "USA" and others as "United States," it creates inconsistency. A consistent dataset has similar entries.

- **Relevance:**

Relevance ensures that the data is pertinent and useful for the task or analysis at hand. Irrelevant data adds noise and may decrease model performance.

Example: Collecting historical weather data when analyzing customer purchasing behavior might not be relevant.

- If you’re predicting house prices, features like "square footage" and "number of bedrooms" are relevant, while "favorite color" is not.

- **Timeliness**

Data should be up-to-date and reflect the current state of the environment or phenomenon it represents, especially in dynamic fields.

Example: A dataset predicting stock prices should use the most recent trading data. Using old data can lead to inaccurate predictions.

- **Uniformity**

Data should follow a standard format (e.g., date formats, numerical precision). This includes consistent units of measurement.

Example: If dates are recorded in different formats (e.g., MM/DD/YYYY and DD/MM/YYYY), it can lead to confusion. A uniform dataset would use one consistent date format.

- **Granularity**

The data should have an appropriate level of detail. Too much granularity can lead to noise, while too little can obscure important patterns.

Example: For analyzing customer behavior, data collected daily (daily sales) might be too granular. Weekly or monthly summaries might provide a clearer overview without overwhelming detail.

- **Diversity**

A diverse dataset captures a wide range of scenarios and variations, which helps the model generalize better to new, unseen data.

Example: If a model predicting loan approvals is trained on data from only one demographic group, it may not perform well for others. A diverse dataset would include various age groups, income levels, and backgrounds.

- **Non-redundancy**

High-quality data should avoid redundancy. Duplicate records can skew results and lead to overfitting.

Example: If a customer's record appears multiple times in a dataset, it can skew results. A non-redundant dataset would have each customer listed once.

- **Scalability**

The dataset should be manageable in size, allowing for efficient processing and analysis without compromising performance.

Example: A dataset with millions of records may slow down machine learning algorithms. A scalable dataset is one that can be efficiently processed, possibly by sampling or aggregating.

- **Structured**

Data should be organized in a clear and logical manner. Structured data (like tables) is easier to analyze compared to unstructured data (like text).

Example: A structured dataset might be formatted as a table with rows and columns, such as a CSV file with clear headers like "Name," "Age," and "Occupation."

- **Validity:**

Data validity ensures that collected data conforms to the defined rules or constraints, such as data type, range, or specific formats. This means that the data should not only be accurate and consistent but also relevant to the specific needs of the analysis.

Example: A dataset with an age field containing negative values or characters (e.g., “-5” or “abc”) would be invalid.

Types of Validity:

Content Validity: Does the data cover the relevant aspects of the concept being measured.

Construct Validity: Does the data accurately represent the theoretical construct it is meant to measure.

Criterion Validity: Does the data correlate well with other measures of the same construct.

Example 1

Consider a dataset intended to predict employee performance based on their skills and experience. If the dataset includes a column for "years of experience," all entries should represent relevant work experience.

Invalid Scenario:

If you have an entry that lists "years of experience" as "10 years" and another as "5 years in high school," the second entry is invalid because high school experience is not relevant to job performance in a professional context.

Valid Scenario:

A valid dataset would only include relevant work experience, such as:

"10 years in software development"

"5 years in project management"

Example 2

A dataset with an age field containing negative values or characters (e.g., “-5” or “abc”) would be invalid.

1.5.3. Data cleaning for inconsistencies rectification

Inconsistencies in data refer to mismatches or conflicts in the dataset where data entries do not align with expected formats or values. Rectifying these inconsistencies is a crucial part of data cleaning, ensuring the data is reliable and ready for analysis or modeling.

Examples of Inconsistencies:

- **Date Format Differences:** Some entries might be in MM/DD/YYYY while others are in DD/MM/YYYY.
- **Currency or Units Mismatch:** Some records might use different currencies or measurement units.
- **Typos in Text Data:** Errors in categorical data such as New York being written as New York or new york.
- **Missing Data:** Different levels of missingness in different fields can cause inconsistency in the dataset.

Importance of Data Cleaning

- **Improves Data Quality:**
  High-quality data enhances the accuracy and reliability of insights derived from analyses or models. Clean data reduces the likelihood of errors that can skew results.
- **Enhances Model Performance:**
  Machine learning models trained on clean data are more likely to generalize well to unseen data. Poor quality data can lead to overfitting or underfitting.
- **Reduces Costs:**
  Cleaning data early in the process can save time and resources in the long run. It avoids the potential costs associated with making decisions based on flawed data.

- F**acilitates Better Decision-Making:**
  Accurate and clean data supports informed decision-making. Stakeholders can rely on data-driven insights when the data is trustworthy.
- **Increases Efficiency:**
  Clean data simplifies the data processing pipeline. It reduces the complexity involved in data manipulation and analysis.
- **Ensures Compliance:**
  In many industries, maintaining data quality is essential for compliance with regulations (e.g. General Data Protection Regulation (GDPR). Data cleaning helps ensure that data handling practices meet legal standards.

Data Cleaning Techniques

- **Handling Missing Values:**
  Imputation: Fill in missing values using statistical measures (mean, median, mode) or more advanced techniques (KNN imputation).
  Removal: Delete rows or columns with excessive missing values if imputation isn't feasible.
  Flagging: Create a separate column to indicate whether a value was missing.
- **Removing Duplicates:**
  Use functions (e.g., Pandas' .drop_duplicates()) to identify and remove duplicate records that can skew analysis and model training.
- **Standardizing Data Formats:**
  Ensure consistent formats for dates, currency, and other categorical variables. For example, convert all date entries to a standard format (YYYY-MM-DD).
- **Outlier Detection and Treatment:**
  Use statistical methods (Z-score, IQR) or visualization (box plots) to identify outliers. Treat outliers by removing them or transforming them based on the context of the analysis.
- **Encoding Categorical Variables:**
  Convert categorical variables into numerical format using:
  One-hot encoding: Creates binary columns for each category.
  Label encoding: Assigns numerical values to categories based on their order.
- **Feature Scaling:**
  Normalize or standardize numerical features to ensure they are on the same scale, improving model training efficiency.
  Techniques include Min-Max scaling and Z-score normalization.
- **Text Data Cleaning:**
  For text data, remove punctuation, special characters, and stop words. Apply stemming or lemmatization to reduce words to their base form.
- **Ensuring Consistent Data Types:**
  Check and convert data types as needed (e.g., converting strings to integers) to ensure uniformity across the dataset.

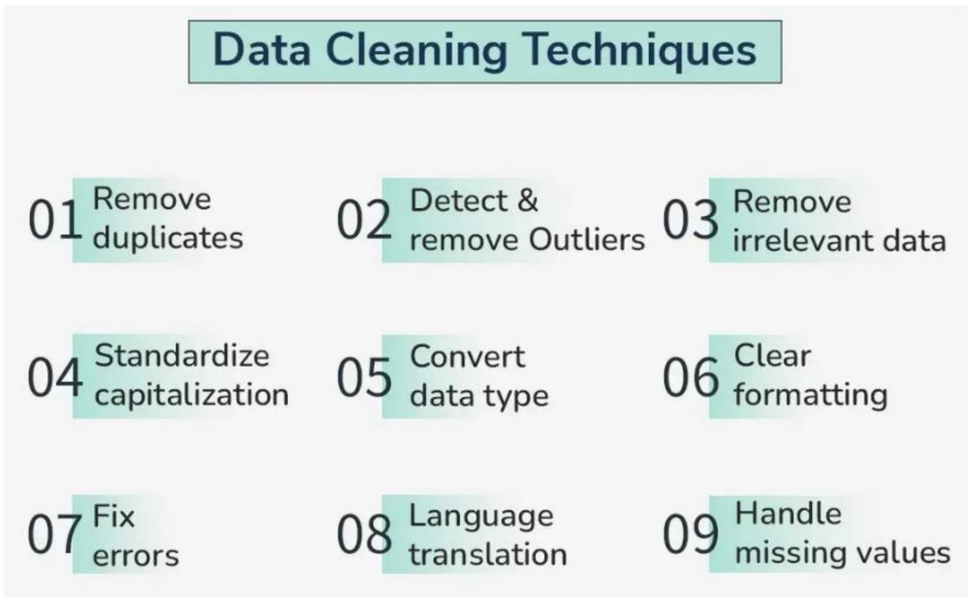


Figure 40: Data cleaning techniques.

1.5.4. Description of data normalization

1. Definition

Data Normalization is the process of adjusting the values of numeric data to a common scale, without distorting differences in the ranges of values. The main goal of normalization is to ensure that each feature contributes equally to the distance calculations in machine learning algorithms, particularly those sensitive to the scale of data, such as k-nearest neighbors (KNN) and gradient descent-based algorithms.

2. Importance of Data Normalization

- **Improves Model Performance:** Many machine learning algorithms, especially those based on distance metrics (like KNN and SVM), perform better when the data is on a similar scale. Normalization helps in achieving this uniformity.
- **Accelerates Convergence: In** algorithms that use gradient descent for optimization, such as linear regression and neural networks, normalization can speed up convergence. It helps the algorithm to reach the optimal solution faster.
- **Reduces Sensitivity to Input Scale:** Normalized data helps in reducing the sensitivity of models to the scale of input features, making models more robust.
- **Enhances Interpretability:** Normalized data can make it easier to interpret the results of the model, as the features are on a common scale.
- **Facilitates Feature Comparisons:** When features are normalized, it becomes easier to compare their effects on the output, aiding in feature selection and engineering.

3. Data Normalization Techniques

Several methods are used for normalizing data, depending on the type of model and the specific characteristics of the dataset.

1. Min-Max Normalization (Rescaling):

This technique scales the data to a fixed range, typically between 0 and 1.

- Formula:

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

  - Where X_{norm} is the normalized value, X is the original value, and X_{min} and X_{max} are the minimum and maximum values of the feature, respectively.

- **Use Case**: Ideal for datasets where the values need to be bounded, and the feature distribution is not Gaussian (normal distribution).
- **Example**:
  - Original data: [5, 10, 15, 20]
  - Normalized (0-1): [0, 0.33, 0.67, 1]

2. Z-Score Normalization (Standardization):

This method centers the data around the mean with a standard deviation of 1, making it suitable for features with a Gaussian distribution.

- Formula:

$$Z = \frac{X - \mu}{\sigma}$$

  - Where Z is the standardized value, X is the original value, μ is the mean of the feature, and σ is the standard deviation.

- **Use Case**: Useful for data with varying scales but a Gaussian distribution. It's often used in algorithms like logistic regression and support vector machines (SVM).
- **Example**:
  - Original data: [10, 20, 30, 40]
  - Mean: 25, Standard deviation: 11.18
  - Standardized: [-1.34, -0.45, 0.45, 1.34]

3. Decimal Scaling:

This technique involves moving the decimal point of the values to normalize them. The scaling factor is determined by the maximum absolute value in the data.

- Formula:

$$X_{norm} = \frac{X}{10^j}$$

  - Where j is the number of digits in the largest absolute value.

- **Use Case**: Used when the values vary by orders of magnitude, and you want to scale the data down uniformly.

4. MaxAbs Scaling:

Similar to Min-Max normalization, but here the data is scaled by dividing by the maximum absolute value of each feature. It maintains the sign of the original data.

- Formula:

$$X_{norm} = \frac{X}{X_{max}}$$

  - Where X_{max} is the maximum absolute value of the feature.

- **Use Case**: Useful when dealing with sparse data (e.g., in text processing or image classification), and you want to maintain the distribution's original sign while scaling.

5. Logarithmic Scaling:

This technique takes the logarithm of the values to reduce the impact of large outliers and bring skewed distributions closer to normality.

- Formula:

$$X_{norm} = \log(X + c)$$

  - Where c is a small constant added to avoid taking the log of zero.

- **Use Case**: Often applied in scenarios where data has exponential growth patterns (e.g., population growth, economic data).

6. Robust Scaler (Median and IQR Normalization):

This method normalizes data by removing the median and scaling the data according to the interquartile range (IQR). It's robust to outliers since it relies on percentiles rather than mean and variance.

- Formula:

$$X_{norm} = \frac{X - \text{median}}{\text{IQR}}$$

- **Use Case**: Best suited for datasets with many outliers where the standard scaling techniques would be affected by extreme values.

1.5.5. Data transformation

1. Definition

Data Transformation is the process of converting data from one format or structure into another. In the context of machine learning, it involves modifying the data to improve its quality and make it suitable for analysis and modeling. This can include scaling, encoding, aggregating, or applying mathematical functions to the data.

2. Importance of Data Transformation

- **Enhances Model Performance:** Properly transformed data can lead to better model accuracy and performance. Certain algorithms require data to be in specific formats or scales.
- **Improves Interpretability:** Transforming data can make it easier to understand and interpret, especially when dealing with complex datasets.
- **Facilitates Data Integration:** When combining data from different sources, transformation ensures that the formats and structures are compatible.
- **Addresses Data Quality Issues:** Transformation can help rectify inconsistencies, handle missing values, and reduce noise in the data.
- **Prepares Data for Analysis:** Many machine learning algorithms assume that the data follows certain distributions. Transformation can help meet these assumptions.

3. Data Transformation Techniques

The choice of data transformation technique depends on the characteristics of the data and the machine learning algorithm that is intended to be used on the data. Here are the major techniques among others:

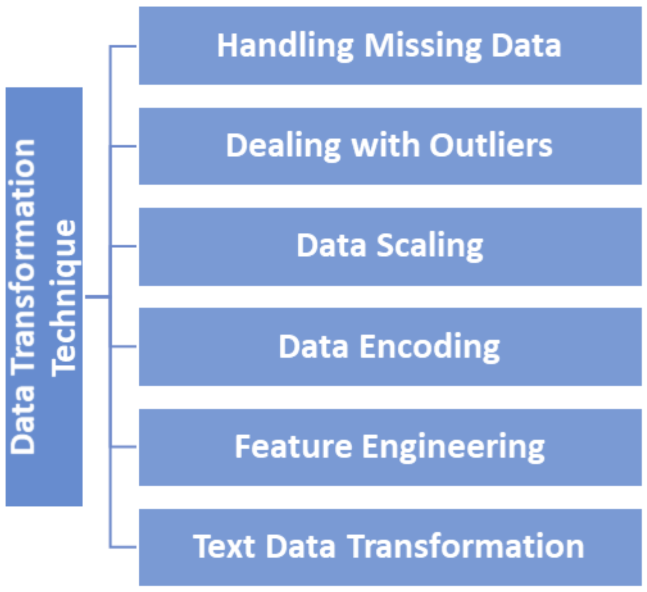


Figure 41: Data transformation techniques

Handling Missing Data

Most of the times the data received from different sources miss some of the values in it, by training the model on this data, the model might behave differently or even produce error while training. There are different techniques to handle the missing data:

☞**Removing the Missing Data**: For a small number of data is missing, delete the rows or columns which are having missing data. For a large value of missing data points in the dataset, consider some other technique.

☞**Imputation**: Remove or Replacing the missing values with estimated values (using mean, median, mode, or more advanced techniques like regression or machine learning).

Different missing values can be imputed with the help of KNNImputer(K-Nearest Neighbors) which is a part of sklearn.impute.

☞**Forward Fill or Backward Fill**: This method fills the missing value with the previous non missing value whereas backward fill method fills the missing value with the next non missing value to the missing value.

☞**Interpolation**: It involves predicting the missing values based on observed values in the dataset.

↳ ***Dealing with Outliers***

☞**Identification of Outlier**: Choosing the correct outlier detection method depends on the characteristics of data and the goals of analysis. 1st method: use visual inspection (e.g.: boxplot or scatterplot). 2nd method: use statistical methods including z-score and IQR (z-score)

☞**Removing Outliers**:

- **Winsorizing:** Replacing outliers with the nearest non-outlier value.
- Removing outliers if they are clearly errors.
- Capping outliers at a certain value.

☞**Truncation**: Is the method of setting a threshold and then adjusting all the points that are outside the range of the threshold value.

↳ ***Data Scaling: Normalization and Standardization***

☞**Normalization**: Rescales features to a specific/standard range of values which is usually 0-1. It helps in equalizing the range of the features and makes sure that the features contribute equally.

☞**Standardization**: Known **as z-score normalization,** its objective is to transform the feature such that the value of mean becomes 0 and the value of standard deviation becomes 1. It rescales features to have zero mean and unit variance.

↳ ***Encoding Categorical Variables***

The categorical features could be encoded into numerical valued features in different ways.

☞**One-Hot Encoding**:

It converts each category in a categorical feature into a different binary feature (i.e. 0 or 1).

Example: 'car', 'bike', 'bicycle', one-hot encoding will create three separate columns as 'is_car', 'is_bike', 'is_bicycle' and then label them as 0 if absent or 1 if present.

☞**Label Encoding**: assigns a unique numeric value to different categories in the same feature.

Example: 'small', 'medium', 'large', then the label encoding could label each value as 0, 1, 2 respectively.

☞**Ordinal Encoding**: It is similar to label encoding except the fact that in ordinal encoding the categorical feature is encoded according to some sort of hierarchy in the system.

Example: "High-School", "Bachelor's", and "Master's" the ordinal encoding will label this as 0, 1, 2 based on the educational hierarchy.

➯ ***Feature Engineering***

☞**Creating new features:** Derive new features from existing ones (e.g., combining features, calculating ratios).

☞**Feature Selection**: Choose the most relevant features for the model (e.g., correlation analysis, feature importance).

➯ ***Text Data Transformation***

☞**Text Cleaning:** textual data from any source (raw) data might contain HTML tags, punctuations, special characters, and symbols which is not usually useful in the analysis process, therefore, stripping them from the data might be a better option. Also, converting the characters in the data into a lowercase is often considered a good practice.

☞**Tokenization:** This process breaks down text into individual words or tokens. Example - "This is a statement" could be tokenized as "This", "is", "a", "statement".

☞**Stop word Removal:** We must consider removing the words that do not contribute to the overall meaning of the text or the words which are not essential for the analysis process. For example, words like - "and", "or", "the", etc.

☞**Stemming and Lemmatization:** Stemming is the reduction of words to their base forms like "sleeping" becomes associated with "sleep". lemmatization uses the core meaning of the word to get it to their base form. For example, "worse" could be associated with "bad".

☞**Word embeddings** helps in capturing the relationships between words.

Practical Activity 1.5.2: Performing data cleaning for inconsistencies

Task:

1. Read the task bellow:
2. As full-stack, you are asked to go to the computer lab to performing data cleaning for inconsistencies rectification of provided car dataset and save the cleaned data as **car_cleaned_dataset**.
3. Refers to provided key reading 1.5.2, perform the task described above.
4. Present your work to the trainer and whole class.

Key readings 1.5.2: Performing data cleaning for inconsistencies

Data cleaning is a crucial step in the data pre-processing phase of machine learning. It involves identifying and rectifying inconsistencies in the dataset to improve the quality of data, which in turn enhances model performance.

Below is a sample dataset in CSV format that you can use to apply the data cleaning steps. This dataset contains various inconsistencies, including missing values, duplicates, incorrect data types, and outliers.

Step 1: Load the Dataset

```
import pandas as pd
# Load the dataset
car_data = pd.read_csv('car_dataset.csv')
```

Step 2: Handling Missing Values

```
# Check for missing values
missing_values = car_data.isnull().sum()
# Fill or drop missing values based on your analysis
car_data['mileage'].fillna(car_data['mileage'].median(), inplace=True)
car_data.dropna(subset=['name', 'year'], inplace=True)  # Dropping rows with critical missing values
```

Step 3: Remove Duplicates

```
# Remove duplicate rows
car_data.drop_duplicates(inplace=True)
```

Step 4: Correct Data Types

```
# Convert data types if necessary
car_data['year'] = car_data['year'].astype(int)
car_data['selling_price'] = car_data['selling_price'].str.replace('₹', '').str.replace(',', '').astype(float)
```

Step 5: Handle Outliers

```
# Identify outliers using IQR method or z-score
Q1 = car_data['selling_price'].quantile(0.25)
Q3 = car_data['selling_price'].quantile(0.75)
IQR = Q3 - Q1
# Remove outliers
car_data = car_data[(car_data['selling_price'] >= (Q1 - 1.5 * IQR)) & (car_data['selling_price'] <= (Q3 + 1.5 * IQR))]
```

Step 6: Encode Categorical Variables

```
# One-hot encoding for categorical variables
car_data = pd.get_dummies(car_data, columns=['fuel', 'seller_type', 'transmission'], drop_first=True)
```

Step 7: Feature Engineering

```
# Example: Create new features from existing ones
car_data['age'] = 2023 - car_data['year']  # Assuming the current year is 2023
```

Step 8: Text Data Processing

```
# Clean text data if necessary (e.g., for the 'name' column)
car_data['name'] = car_data['name'].str.lower().str.strip()
```

Step 9: Final Data Check

```
# Final check of the cleaned data
print(car_data.info())
print(car_data.describe())
```

Step 10: Save the Cleaned Dataset

```
# Save the cleaned dataset
car_data.to_csv('car_cleaned_dataset.csv', index=False)
```

Explanation:

to_csv: This method is used to write the DataFrame to a CSV file.
'car_cleaned_dataset.csv': This specifies the name of the file where you want to save the cleaned dataset.
index=False: This argument prevents pandas from writing row indices to the CSV file, which is usually the desired behavior unless you need the indices.

Additional Notes

Adjust the handling of missing values according to the dataset's context.
Use visualizations (e.g., box plots) to help identify outliers.
Ensure the encoding method aligns with your model requirements.
Always back up your original dataset before making changes.

Conclusion

Now, our dataset is clean and ready for machine learning!

We have:

- Handled missing values.
- Removed duplicates.
- Fixed incorrect data types.
- Handled outliers.
- Normalized numerical values.
- Encoded categorical features.
- Engineered useful features.
- Transformed text data.

Example data cleaning on cardataset.csv

we are going to import pandas library that help us to manipulate our dataset

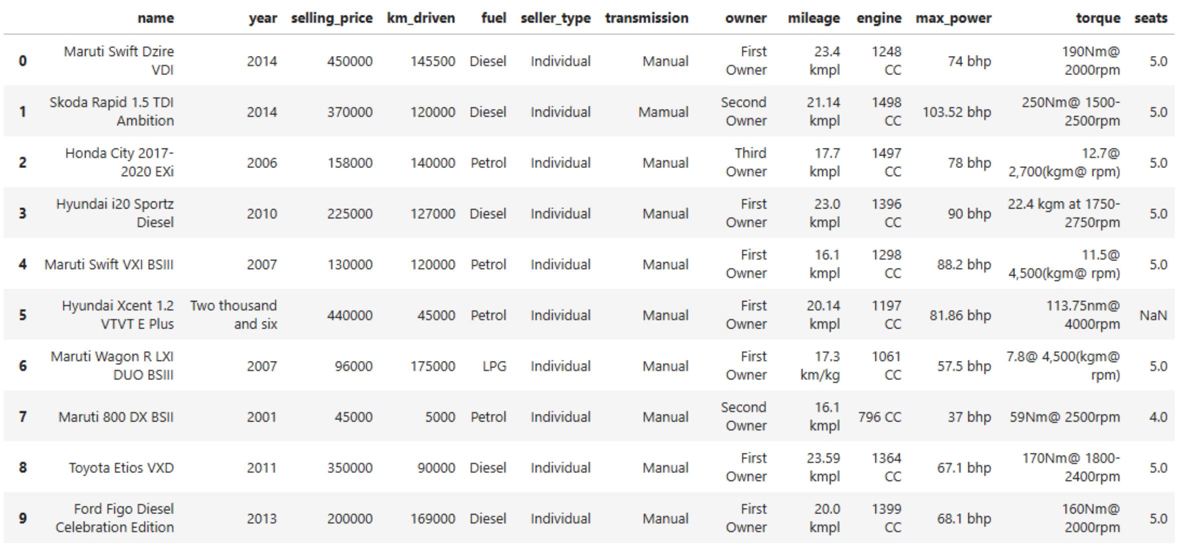
import pandas as pd

We are going to read our dataset, remember to upload your dataset from your local machine to a directory where you save your file in order to keep everything in the same place.

df = pd.read_csv("carDataset.csv")

we are going to display the dataset stored in df variable, there are two buildin function to take care about: head() which is by default display the first 5 records of a dataset but you can specify the number of the records you want as argument and tail() that display the last 5 records but also you can specify the number of records you want as arguments.

df.head(14)



	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage	engine	max_power	torque	seats
0	Maruti Swift Dzire VDI	2014	450000	145500	Diesel	Individual	Manual	First Owner	23.4 kmpl	1248 CC	74 bhp	190Nm@ 2000rpm	5.0
1	Skoda Rapid 1.5 TDI Ambition	2014	370000	120000	Diesel	Individual	Manual	Second Owner	21.14 kmpl	1498 CC	103.52 bhp	250Nm@ 1500-2500rpm	5.0
2	Honda City 2017-2020 EXi	2006	158000	140000	Petrol	Individual	Manual	Third Owner	17.7 kmpl	1497 CC	78 bhp	12.7@ 2,700(kgm@ rpm)	5.0
3	Hyundai i20 Sportz Diesel	2010	225000	127000	Diesel	Individual	Manual	First Owner	23.0 kmpl	1396 CC	90 bhp	22.4 kgm at 1750-2750rpm	5.0
4	Maruti Swift VXI BSIII	2007	130000	120000	Petrol	Individual	Manual	First Owner	16.1 kmpl	1298 CC	88.2 bhp	11.5@ 4,500(kgm@ rpm)	5.0
5	Hyundai Xcent 1.2 VTVT E Plus	Two thousand and six	440000	45000	Petrol	Individual	Manual	First Owner	20.14 kmpl	1197 CC	81.86 bhp	113.75nm@ 4000rpm	NaN
6	Maruti Wagon R LXI DUO BSIII	2007	96000	175000	LPG	Individual	Manual	First Owner	17.3 km/kg	1061 CC	57.5 bhp	7.8@ 4,500(kgm@ rpm)	5.0
7	Maruti 800 DX BSII	2001	45000	5000	Petrol	Individual	Manual	Second Owner	16.1 kmpl	796 CC	37 bhp	59Nm@ 2500rpm	4.0
8	Toyota Etios VXD	2011	350000	90000	Diesel	Individual	Manual	First Owner	23.59 kmpl	1364 CC	67.1 bhp	170Nm@ 1800-2400rpm	5.0
9	Ford Figo Diesel Celebration Edition	2013	200000	169000	Diesel	Individual	Manual	First Owner	20.0 kmpl	1399 CC	68.1 bhp	160Nm@ 2000rpm	5.0

The df.info() method is a useful tool in pandas that provides a concise summary of your DataFrame, including:

- The number of rows and columns.
- The column names and their data types.
- The number of non-null values in each column.
- The memory usage of the DataFrame.

df.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8134 entries, 0 to 8133
Data columns (total 13 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   name           8134 non-null   object 
 1   year           8134 non-null   object 
 2   selling_price  8134 non-null   int64  
 3   km_driven      8134 non-null   int64  
 4   fuel           8134 non-null   object 
 5   seller_type    8134 non-null   object 
 6   transmission   8134 non-null   object 
 7   owner          8134 non-null   object 
 8   mileage        7913 non-null   object 
 9   engine         7913 non-null   object 
 10  max_power      7919 non-null   object 
 11  torque         7912 non-null   object 
 12  seats          7911 non-null   float64
dtypes: float64(1), int64(2), object(10)
memory usage: 826.2+ KB
```

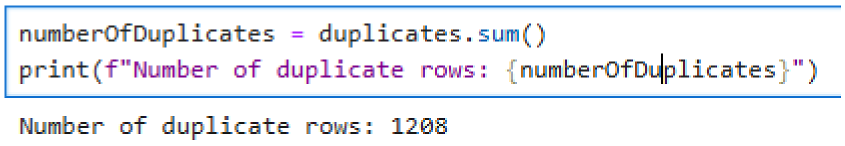
1. REMOVE DUPLICATES ROWS

The df.duplicated() method in pandas is used to identify duplicate rows in a DataFrame. It returns a Series of boolean values where True indicates that a row is a duplicate and False indicates it is unique.

```
duplicates = df.duplicated()
duplicates
```

```
0       False
1       False
2       False
3       False
4       False
        ...  
8129     True
8130     True
8131     True
8132     True
8133     True
Length: 8134, dtype: bool
```

The duplicates.sum() operation in pandas is used to count the number of True values in the Series returned by df.duplicated(). Since each True represents a duplicate row, duplicates.sum() gives the total number of duplicate rows in the DataFrame.



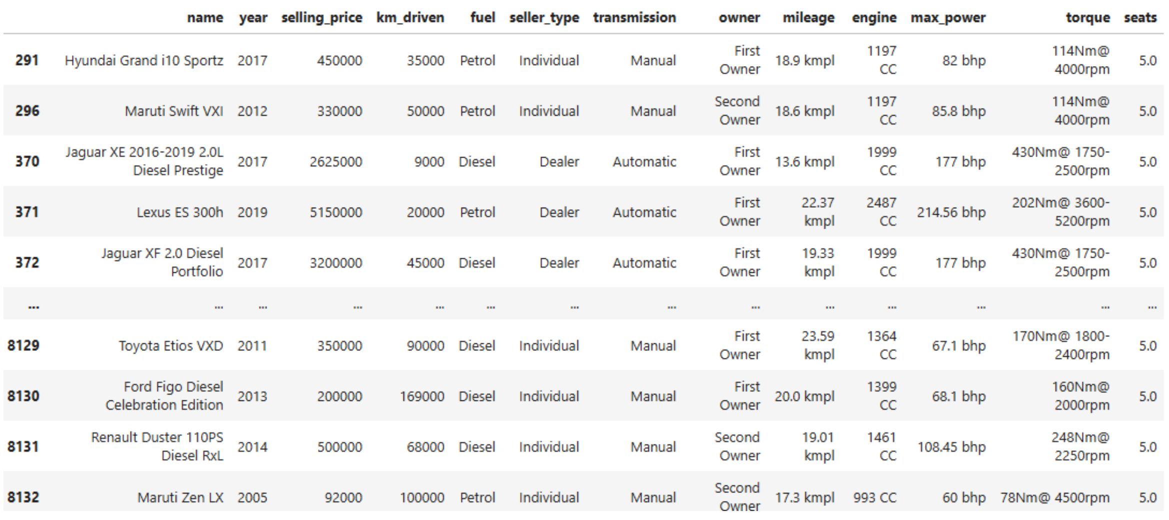
```
numberOfDuplicates = duplicates.sum()
print(f"Number of duplicate rows: {numberOfDuplicates}")
```

```
Number of duplicate rows: 1208
```

The line duplicate_rows = df[df.duplicated()] is used to filter out and create a new DataFrame containing only the duplicate rows from the original DataFrame df.

duplicate_rows = df[df.duplicated()]

duplicate_rows



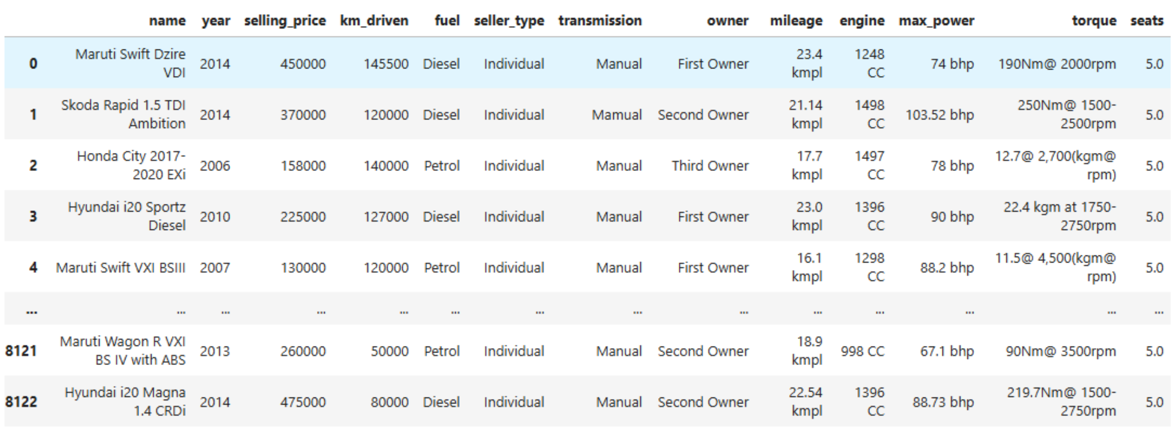
	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage	engine	max_power	torque	seats
291	Hyundai Grand i10 Sportz	2017	450000	35000	Petrol	Individual	Manual	First Owner	18.9 kmpl	1197 CC	82 bhp	114Nm@ 4000rpm	5.0
296	Maruti Swift VXI	2012	330000	50000	Petrol	Individual	Manual	Second Owner	18.6 kmpl	1197 CC	85.8 bhp	114Nm@ 4000rpm	5.0
370	Jaguar XE 2016-2019 2.0L Diesel Prestige	2017	2625000	9000	Diesel	Dealer	Automatic	First Owner	13.6 kmpl	1999 CC	177 bhp	430Nm@ 1750-2500rpm	5.0
371	Lexus ES 300h	2019	5150000	20000	Petrol	Dealer	Automatic	First Owner	22.37 kmpl	2487 CC	214.56 bhp	202Nm@ 3600-5200rpm	5.0
372	Jaguar XF 2.0 Diesel Portfolio	2017	3200000	45000	Diesel	Dealer	Automatic	First Owner	19.33 kmpl	1999 CC	177 bhp	430Nm@ 1750-2500rpm	5.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
8129	Toyota Etios VXD	2011	350000	90000	Diesel	Individual	Manual	First Owner	23.59 kmpl	1364 CC	67.1 bhp	170Nm@ 1800-2400rpm	5.0
8130	Ford Figo Diesel Celebration Edition	2013	200000	169000	Diesel	Individual	Manual	First Owner	20.0 kmpl	1399 CC	68.1 bhp	160Nm@ 2000rpm	5.0
8131	Renault Duster 110PS Diesel RxL	2014	500000	68000	Diesel	Individual	Manual	Second Owner	19.01 kmpl	1461 CC	108.45 bhp	248Nm@ 2250rpm	5.0
8132	Maruti Zen LX	2005	92000	100000	Petrol	Individual	Manual	Second Owner	17.3 kmpl	993 CC	60 bhp	78Nm@ 4500rpm	5.0

1208 rows × 13 columns

df.drop_duplicates(): This function removes duplicate rows from the DataFrame. By default, it keeps the first occurrence of each duplicate row and drops subsequent duplicates.

df2 = df.drop_duplicates()

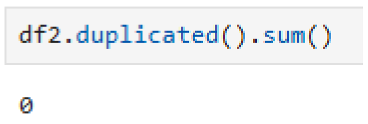
df2



	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage	engine	max_power	torque	seats
0	Maruti Swift Dzire VDI	2014	450000	145500	Diesel	Individual	Manual	First Owner	23.4 kmpl	1248 CC	74 bhp	190Nm@ 2000rpm	5.0
1	Skoda Rapid 1.5 TDI Ambition	2014	370000	120000	Diesel	Individual	Manual	Second Owner	21.14 kmpl	1498 CC	103.52 bhp	250Nm@ 1500-2500rpm	5.0
2	Honda City 2017-2020 EXi	2006	158000	140000	Petrol	Individual	Manual	Third Owner	17.7 kmpl	1497 CC	78 bhp	12.7@ 2,700(kgm@ rpm)	5.0
3	Hyundai i20 Sportz Diesel	2010	225000	127000	Diesel	Individual	Manual	First Owner	23.0 kmpl	1396 CC	90 bhp	22.4 kgm at 1750-2750rpm	5.0
4	Maruti Swift VXI BSIII	2007	130000	120000	Petrol	Individual	Manual	First Owner	16.1 kmpl	1298 CC	88.2 bhp	11.5@ 4,500(kgm@ rpm)	5.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
8121	Maruti Wagon R VXI BS IV with ABS	2013	260000	50000	Petrol	Individual	Manual	Second Owner	18.9 kmpl	998 CC	67.1 bhp	90Nm@ 3500rpm	5.0
8122	Hyundai i20 Magna 1.4 CRDi	2014	475000	80000	Diesel	Individual	Manual	Second Owner	22.54 kmpl	1396 CC	88.73 bhp	219.7Nm@ 1500-2750rpm	5.0

6926 rows × 13 columns

Here we confirm that the duplicate cleaning is over without any other duplicates in our dataset.



```
df2.duplicated().sum()
```

```
0
```

2. standardize the formats for data consistency

Splitting the name column into separate brand and model columns can be very useful for analysis, especially if different brands or models have distinct characteristics that may influence your predictions. Here's how you can do this in Python using pandas:

Steps to Split the name Column:

Inspect the name Column:

Identify a consistent pattern in the column. Typically, the brand is the first word, and the rest is the model description.

df.copy(): Creates a copy of the original DataFrame to ensure that the subsequent modifications do not result in the warning.

Assigning to df_copy[['brand', 'model']]: This is a direct assignment to the new DataFrame copy, avoiding the issue of modifying a slice of the original DataFrame.

By using df.copy(), you avoid potential pitfalls related to modifying a view versus a copy, ensuring that the changes are safely applied to a new DataFrame without unintended side effects.

```
df_copy = df2.copy()
df_copy[['brand', 'model']] = df_copy['name'].str.split(' ', n=1, expand=True)
```

```
df_copy
```

	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage	engine	max_power	torque	seats	brand	model
0	Maruti Swift Dzire VDI	2014	450000	145500	Diesel	Individual	Manual	First Owner	23.4 kmpl	1248 CC	74 bhp	190Nm@ 2000rpm	5.0	Maruti	Swift Dzire VDI
1	Skoda Rapid 1.5 TDI Ambition	2014	370000	120000	Diesel	Individual	Manual	Second Owner	21.14 kmpl	1498 CC	103.52 bhp	250Nm@ 1500-2500rpm	5.0	Skoda	Rapid 1.5 TDI Ambition
2	Honda City 2017-2020 EXi	2006	158000	140000	Petrol	Individual	Manual	Third Owner	17.7 kmpl	1497 CC	78 bhp	12.7@ 2,700(kgm@ rpm)	5.0	Honda	City 2017-2020 EXi
3	Hyundai i20 Sportz Diesel	2010	225000	127000	Diesel	Individual	Manual	First Owner	23.0 kmpl	1396 CC	90 bhp	22.4 kgm at 1750-2750rpm	5.0	Hyundai	i20 Sportz Diesel
4	Maruti Swift VXI BSIII	2007	130000	120000	Petrol	Individual	Manual	First Owner	16.1 kmpl	1298 CC	88.2 bhp	11.5@ 4,500(kgm@ rpm)	5.0	Maruti	Swift VXI BSIII

6926 rows × 15 columns

you can remove the original name column after splitting it into brand and model if it's no longer needed. This can be done using the drop() method.

```
df_copy.drop(columns=['name'], inplace=True)
```

```
df_copy
```

	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage	engine	max_power	torque	seats	brand	model
0	2014	450000	145500	Diesel	Individual	Manual	First Owner	23.4 kmpl	1248 CC	74 bhp	190Nm@ 2000rpm	5.0	Maruti	Swift Dzire VDI
1	2014	370000	120000	Diesel	Individual	Manual	Second Owner	21.14 kmpl	1498 CC	103.52 bhp	250Nm@ 1500-2500rpm	5.0	Skoda	Rapid 1.5 TDI Ambition
2	2006	158000	140000	Petrol	Individual	Manual	Third Owner	17.7 kmpl	1497 CC	78 bhp	12.7@ 2,700(kgm@ rpm)	5.0	Honda	City 2017-2020 EXi
3	2010	225000	127000	Diesel	Individual	Manual	First Owner	23.0 kmpl	1396 CC	90 bhp	22.4 kgm at 1750-2750rpm	5.0	Hyundai	i20 Sportz Diesel
4	2007	130000	120000	Petrol	Individual	Manual	First Owner	16.1 kmpl	1298 CC	88.2 bhp	11.5@ 4,500(kgm@ rpm)	5.0	Maruti	Swift VXI BSIII
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
8121	2013	260000	50000	Petrol	Individual	Manual	Second Owner	18.9 kmpl	998 CC	67.1 bhp	90Nm@ 3500rpm	5.0	Maruti	Wagon R VXI BS IV with ABS
8122	2014	475000	80000	Diesel	Individual	Manual	Second Owner	22.54 kmpl	1396 CC	88.73 bhp	219.7Nm@ 1500-2750rpm	5.0	Hyundai	i20 Magna 1.4 CRDi

Consistency

Uniform format: When data is standardized, it ensures that all entries follow the same format (e.g., title case, consistent spelling, and capitalization). This helps avoid confusion when analyzing or processing the data.

Avoid duplicates: Different variations of the same brand or model (e.g., "Apple" vs. "apple" or "Samsung " vs. " samsung") can result in duplicate entries in datasets, leading to errors in analysis or aggregation. Standardization ensures that similar entries are treated as identical.

To standardize the format of the "brand" , "model" ,"fuel", "seller type", "transmission" and "owner", columns in your dataset during cleaning, you can follow these steps:

Remove Extra Spaces: Strip any leading or trailing spaces and replace multiple spaces between words with a single space.

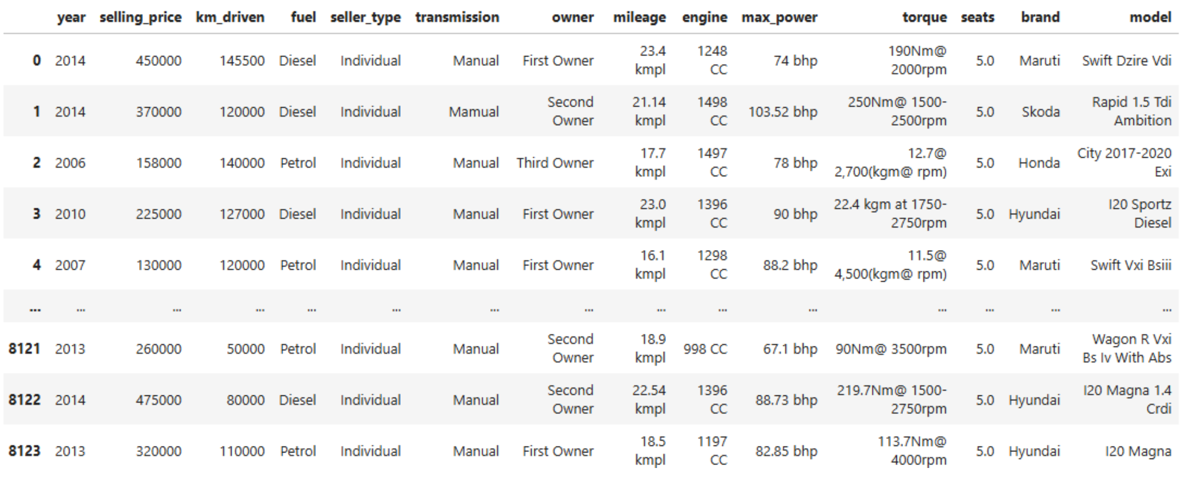
Capitalize Consistently: You can use title case (where each word starts with a capital letter) or uppercase/lowercase, depending on the required style. Typically, for brands and models, title case works well.

# Specify the columns to clean

columns_to_clean = ['brand', 'model', 'fuel', 'seller_type', 'transmission','owner'] # List the columns you want to clean

# Apply standardization to specified columns

df_copy[columns_to_clean] = df_copy[columns_to_clean].apply(lambda x: x.str.strip().str.title())



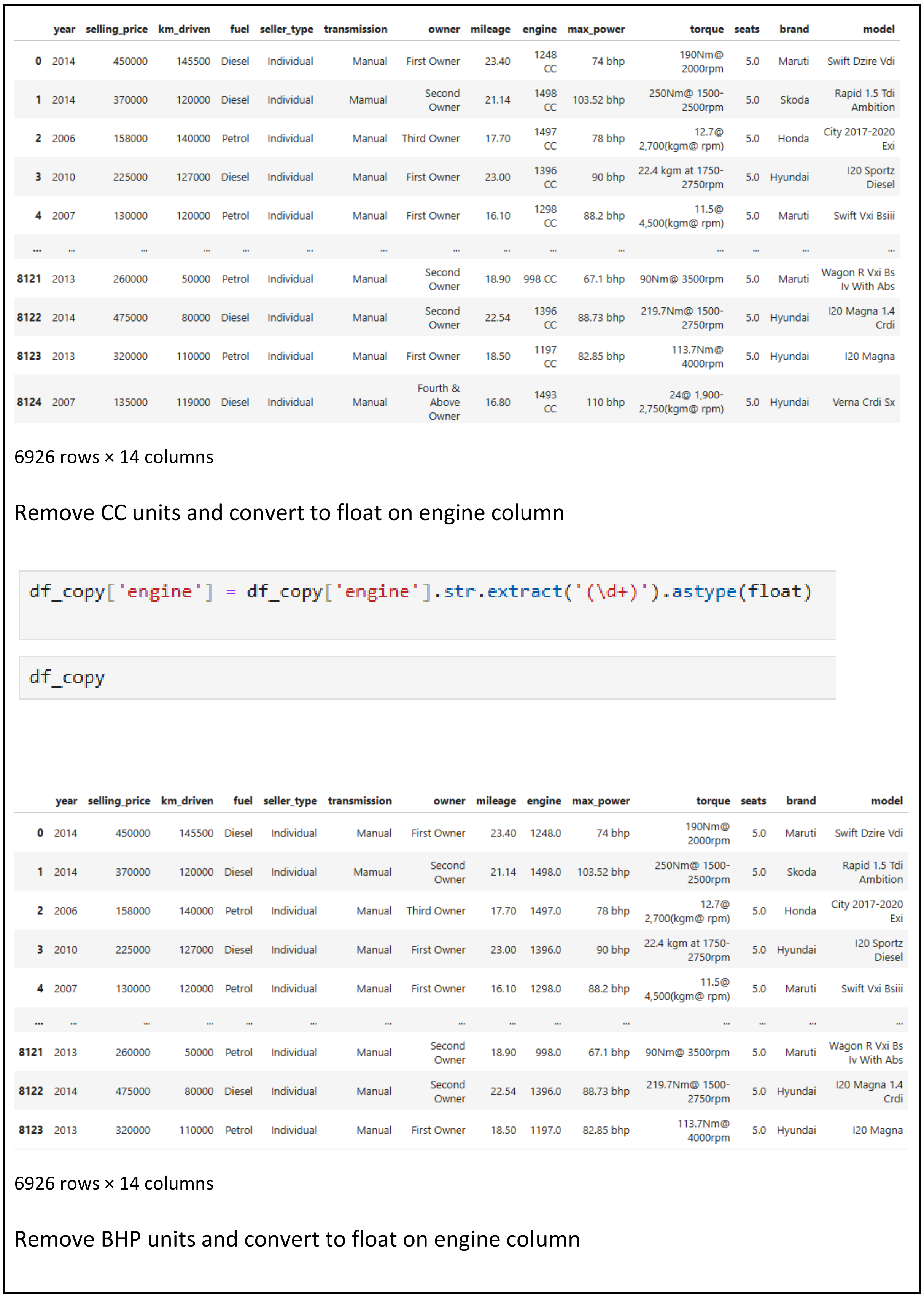
	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage	engine	max_power	torque	seats	brand	model
0	2014	450000	145500	Diesel	Individual	Manual	First Owner	23.4 kmpl	1248 CC	74 bhp	190Nm@ 2000rpm	5.0	Maruti	Swift Dzire Vdi
1	2014	370000	120000	Diesel	Individual	Manual	Second Owner	21.14 kmpl	1498 CC	103.52 bhp	250Nm@ 1500-2500rpm	5.0	Skoda	Rapid 1.5 Tdi Ambition
2	2006	158000	140000	Petrol	Individual	Manual	Third Owner	17.7 kmpl	1497 CC	78 bhp	12.7@ 2,700(kgm@ rpm)	5.0	Honda	City 2017-2020 Exi
3	2010	225000	127000	Diesel	Individual	Manual	First Owner	23.0 kmpl	1396 CC	90 bhp	22.4 kgm at 1750-2750rpm	5.0	Hyundai	I20 Sportz Diesel
4	2007	130000	120000	Petrol	Individual	Manual	First Owner	16.1 kmpl	1298 CC	88.2 bhp	11.5@ 4,500(kgm@ rpm)	5.0	Maruti	Swift Vxi Bsiii
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
8121	2013	260000	50000	Petrol	Individual	Manual	Second Owner	18.9 kmpl	998 CC	67.1 bhp	90Nm@ 3500rpm	5.0	Maruti	Wagon R Vxi Bs Iv With Abs
8122	2014	475000	80000	Diesel	Individual	Manual	Second Owner	22.54 kmpl	1396 CC	88.73 bhp	219.7Nm@ 1500-2750rpm	5.0	Hyundai	I20 Magna 1.4 Crdi
8123	2013	320000	110000	Petrol	Individual	Manual	First Owner	18.5 kmpl	1197 CC	82.85 bhp	113.7Nm@ 4000rpm	5.0	Hyundai	I20 Magna

Removing the unit from the data doesn't mean we discard important information; rather, it's a necessary step to make the data consistent, numerical, and ready for further analysis, comparisons, and conversions. After cleaning the numeric data, you can always reintroduce the unit in a standardized way, if needed, for presentation or reporting.

Remove units and convert to float on mileage column luckly here kmpl(kilometer per litter is equivalent to km/kg) cz 1kg = 1 L, If it is not the case we may first convert the unity to have consistent values.

df_copy['mileage'] = df_copy['mileage'].str.extract('(\d+\.\d+|\d+)').astype(float)

df_copy



	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage	engine	max_power	torque	seats	brand	model
0	2014	450000	145500	Diesel	Individual	Manual	First Owner	23.40	1248 CC	74 bhp	190Nm@ 2000rpm	5.0	Maruti	Swift Dzire Vdi
1	2014	370000	120000	Diesel	Individual	Manual	Second Owner	21.14	1498 CC	103.52 bhp	250Nm@ 1500-2500rpm	5.0	Skoda	Rapid 1.5 Tdi Ambition
2	2006	158000	140000	Petrol	Individual	Manual	Third Owner	17.70	1497 CC	78 bhp	12.7@ 2,700(kgm@ rpm)	5.0	Honda	City 2017-2020 Exi
3	2010	225000	127000	Diesel	Individual	Manual	First Owner	23.00	1396 CC	90 bhp	22.4 kgm at 1750-2750rpm	5.0	Hyundai	I20 Sportz Diesel
4	2007	130000	120000	Petrol	Individual	Manual	First Owner	16.10	1298 CC	88.2 bhp	11.5@ 4,500(kgm@ rpm)	5.0	Maruti	Swift Vxi Bsiii
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
8121	2013	260000	50000	Petrol	Individual	Manual	Second Owner	18.90	998 CC	67.1 bhp	90Nm@ 3500rpm	5.0	Maruti	Wagon R Vxi Bs Iv With Abs
8122	2014	475000	80000	Diesel	Individual	Manual	Second Owner	22.54	1396 CC	88.73 bhp	219.7Nm@ 1500-2750rpm	5.0	Hyundai	I20 Magna 1.4 Crdi
8123	2013	320000	110000	Petrol	Individual	Manual	First Owner	18.50	1197 CC	82.85 bhp	113.7Nm@ 4000rpm	5.0	Hyundai	I20 Magna
8124	2007	135000	119000	Diesel	Individual	Manual	Fourth & Above Owner	16.80	1493 CC	110 bhp	24@ 1,900-2,750(kgm@ rpm)	5.0	Hyundai	Verna Crdi Sx

6926 rows × 14 columns

Remove CC units and convert to float on engine column

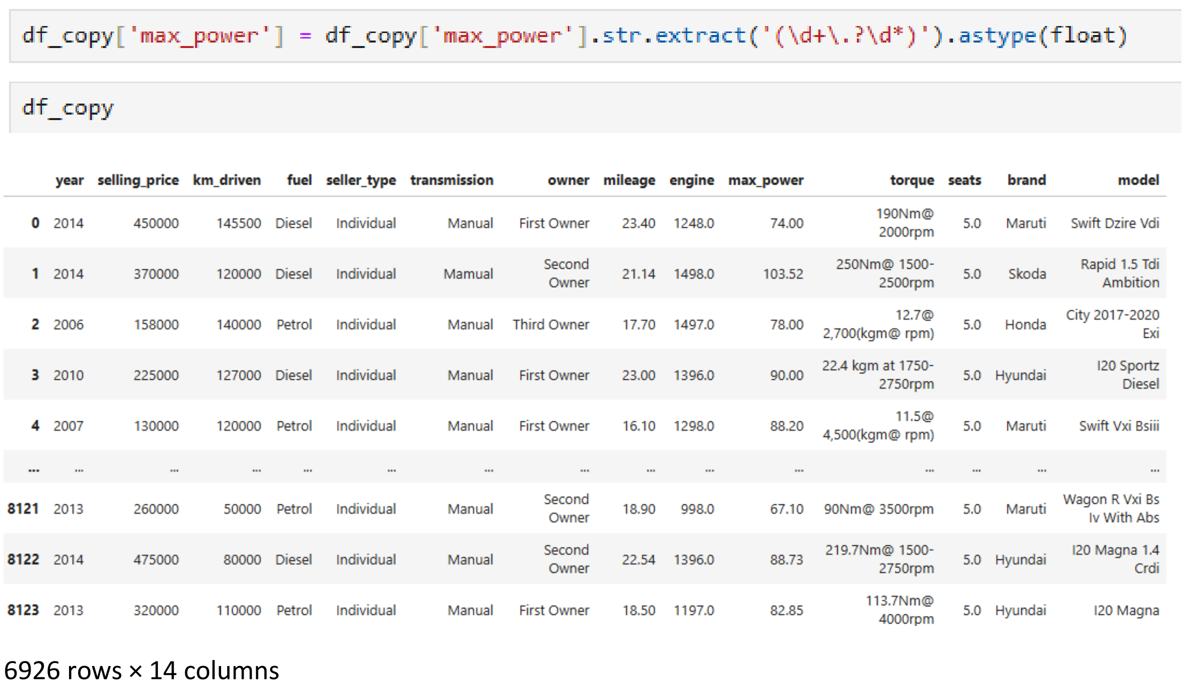
```
df_copy['engine'] = df_copy['engine'].str.extract('(\d+)').astype(float)
```

```
df_copy
```

	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage	engine	max_power	torque	seats	brand	model
0	2014	450000	145500	Diesel	Individual	Manual	First Owner	23.40	1248.0	74 bhp	190Nm@ 2000rpm	5.0	Maruti	Swift Dzire Vdi
1	2014	370000	120000	Diesel	Individual	Manual	Second Owner	21.14	1498.0	103.52 bhp	250Nm@ 1500-2500rpm	5.0	Skoda	Rapid 1.5 Tdi Ambition
2	2006	158000	140000	Petrol	Individual	Manual	Third Owner	17.70	1497.0	78 bhp	12.7@ 2,700(kgm@ rpm)	5.0	Honda	City 2017-2020 Exi
3	2010	225000	127000	Diesel	Individual	Manual	First Owner	23.00	1396.0	90 bhp	22.4 kgm at 1750-2750rpm	5.0	Hyundai	I20 Sportz Diesel
4	2007	130000	120000	Petrol	Individual	Manual	First Owner	16.10	1298.0	88.2 bhp	11.5@ 4,500(kgm@ rpm)	5.0	Maruti	Swift Vxi Bsiii
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
8121	2013	260000	50000	Petrol	Individual	Manual	Second Owner	18.90	998.0	67.1 bhp	90Nm@ 3500rpm	5.0	Maruti	Wagon R Vxi Bs Iv With Abs
8122	2014	475000	80000	Diesel	Individual	Manual	Second Owner	22.54	1396.0	88.73 bhp	219.7Nm@ 1500-2750rpm	5.0	Hyundai	I20 Magna 1.4 Crdi
8123	2013	320000	110000	Petrol	Individual	Manual	First Owner	18.50	1197.0	82.85 bhp	113.7Nm@ 4000rpm	5.0	Hyundai	I20 Magna

6926 rows × 14 columns

Remove BHP units and convert to float on engine column



```python
df_copy['max_power'] = df_copy['max_power'].str.extract('(\d+\.?\d*)').astype(float)
```

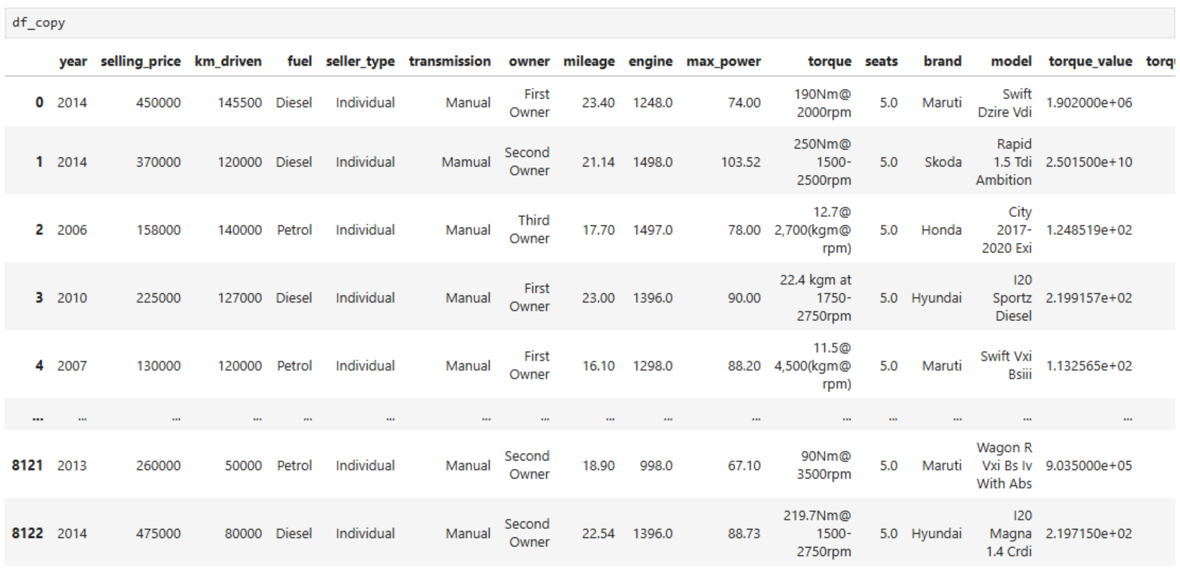
```python
df_copy
```

	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage	engine	max_power	torque	seats	brand	model
0	2014	450000	145500	Diesel	Individual	Manual	First Owner	23.40	1248.0	74.00	190Nm@ 2000rpm	5.0	Maruti	Swift Dzire Vdi
1	2014	370000	120000	Diesel	Individual	Manual	Second Owner	21.14	1498.0	103.52	250Nm@ 1500-2500rpm	5.0	Skoda	Rapid 1.5 Tdi Ambition
2	2006	158000	140000	Petrol	Individual	Manual	Third Owner	17.70	1497.0	78.00	12.7@ 2,700(kgm@ rpm)	5.0	Honda	City 2017-2020 Exi
3	2010	225000	127000	Diesel	Individual	Manual	First Owner	23.00	1396.0	90.00	22.4 kgm at 1750-2750rpm	5.0	Hyundai	I20 Sportz Diesel
4	2007	130000	120000	Petrol	Individual	Manual	First Owner	16.10	1298.0	88.20	11.5@ 4,500(kgm@ rpm)	5.0	Maruti	Swift Vxi Bsiii
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
8121	2013	260000	50000	Petrol	Individual	Manual	Second Owner	18.90	998.0	67.10	90Nm@ 3500rpm	5.0	Maruti	Wagon R Vxi Bs Iv With Abs
8122	2014	475000	80000	Diesel	Individual	Manual	Second Owner	22.54	1396.0	88.73	219.7Nm@ 1500-2750rpm	5.0	Hyundai	I20 Magna 1.4 Crdi
8123	2013	320000	110000	Petrol	Individual	Manual	First Owner	18.50	1197.0	82.85	113.7Nm@ 4000rpm	5.0	Hyundai	I20 Magna

6926 rows × 14 columns

```python
# Function to clean and standardize torque values
def clean_torque(value):
    if isinstance(value, str):  # Ensure value is a string before processing
        # Convert 'kgm' to Nm
        if 'kgm' in value:
            # Extract the numeric part before 'kgm'
            value = re.sub(r'[^0-9\.]', '', value)  # Remove any non-numeric characters
            value = float(value) * 9.81  # Convert to Nm (1 kgm = 9.81 Nm)
        else:
            # Extract numeric part for Nm values
            value = re.sub(r'[^0-9\.]', '', value)  # Remove any non-numeric characters
            value = float(value)  # Convert to float
        return value
    return None  # Return None for non-string values (including numeric or NaN)


# Function to extract RPM value
```

```
def extract_rpm(value):
    if isinstance(value, str):  # Ensure value is a string before processing
        # Extract the RPM part from the string
        rpm_match = re.search(r'(\d+[-\d]*)(rpm)', value)
        if rpm_match:
            rpm_str = rpm_match.group(1)
            if '-' in rpm_str:
                # If there is a range like 1500-2500, take the average
                rpm_range = rpm_str.split('-')
                rpm_value = (int(rpm_range[0]) + int(rpm_range[1])) / 2
            else:
                rpm_value = int(rpm_str.replace(",", ""))
            return rpm_value
    return None  # Return None for non-string values


# Apply the cleaning function to the torque column
df_copy['torque_value'] = df_copy['torque'].apply(clean_torque)


# Apply the RPM extraction function
df_copy['torque_rpm_value'] = df_copy['torque'].apply(extract_rpm)
```

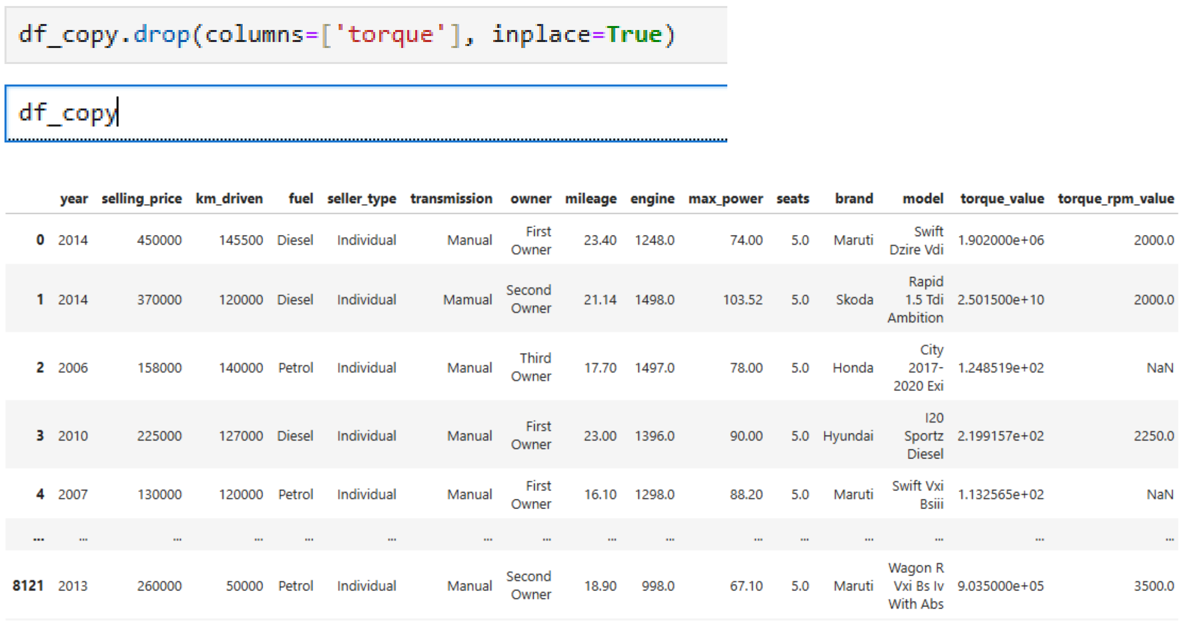


```
df_copy
```

	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage	engine	max_power	torque	seats	brand	model	torque_value	torq
0	2014	450000	145500	Diesel	Individual	Manual	First Owner	23.40	1248.0	74.00	190Nm@ 2000rpm	5.0	Maruti	Swift Dzire Vdi	1.902000e+06	
1	2014	370000	120000	Diesel	Individual	Manual	Second Owner	21.14	1498.0	103.52	250Nm@ 1500-2500rpm	5.0	Skoda	Rapid 1.5 Tdi Ambition	2.501500e+10	
2	2006	158000	140000	Petrol	Individual	Manual	Third Owner	17.70	1497.0	78.00	12.7@ 2,700(kgm@ rpm)	5.0	Honda	City 2017-2020 Exi	1.248519e+02	
3	2010	225000	127000	Diesel	Individual	Manual	First Owner	23.00	1396.0	90.00	22.4 kgm at 1750-2750rpm	5.0	Hyundai	I20 Sportz Diesel	2.199157e+02	
4	2007	130000	120000	Petrol	Individual	Manual	First Owner	16.10	1298.0	88.20	11.5@ 4,500(kgm@ rpm)	5.0	Maruti	Swift Vxi Bsiii	1.132565e+02	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
8121	2013	260000	50000	Petrol	Individual	Manual	Second Owner	18.90	998.0	67.10	90Nm@ 3500rpm	5.0	Maruti	Wagon R Vxi Bs Iv With Abs	9.035000e+05	
8122	2014	475000	80000	Diesel	Individual	Manual	Second Owner	22.54	1396.0	88.73	219.7Nm@ 1500-2750rpm	5.0	Hyundai	I20 Magna 1.4 Crdi	2.197150e+02	

6926 rows × 16 columns

you can remove the original Torque column after splitting it into torque_value and torque_rpm_value if it's no longer needed. This can be done using the drop() method.



```
df_copy.drop(columns=['torque'], inplace=True)
```

```
df_copy
```

	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage	engine	max_power	seats	brand	model	torque_value	torque_rpm_value
0	2014	450000	145500	Diesel	Individual	Manual	First Owner	23.40	1248.0	74.00	5.0	Maruti	Swift Dzire Vdi	1.902000e+06	2000.0
1	2014	370000	120000	Diesel	Individual	Manual	Second Owner	21.14	1498.0	103.52	5.0	Skoda	Rapid 1.5 Tdi Ambition	2.501500e+10	2000.0
2	2006	158000	140000	Petrol	Individual	Manual	Third Owner	17.70	1497.0	78.00	5.0	Honda	City 2017-2020 Exi	1.248519e+02	NaN
3	2010	225000	127000	Diesel	Individual	Manual	First Owner	23.00	1396.0	90.00	5.0	Hyundai	I20 Sportz Diesel	2.199157e+02	2250.0
4	2007	130000	120000	Petrol	Individual	Manual	First Owner	16.10	1298.0	88.20	5.0	Maruti	Swift Vxi Bsiii	1.132565e+02	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
8121	2013	260000	50000	Petrol	Individual	Manual	Second Owner	18.90	998.0	67.10	5.0	Maruti	Wagon R Vxi Bs Iv With Abs	9.035000e+05	3500.0

6926 rows × 15 columns

3. encoding categorical data

check for fuel

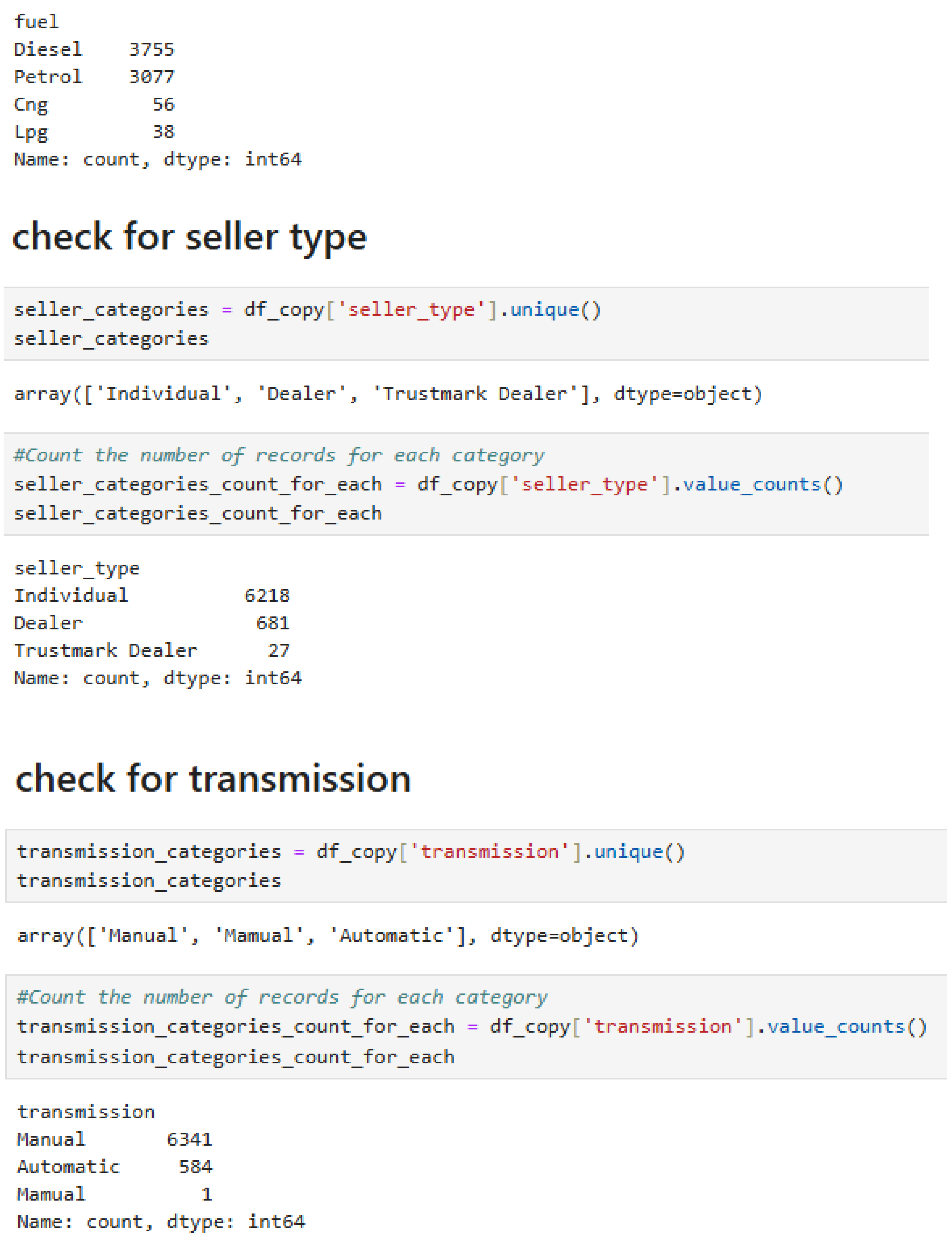
fuel_categories = df_copy['fuel'].unique()

fuel_categories

#Count the number of records for each category

fuel_categories_count_for_each = df_copy['fuel'].value_counts()

fuel_categories_count_for_each



```
fuel
Diesel    3755
Petrol    3077
Cng         56
Lpg         38
Name: count, dtype: int64
```

check for seller type

```
seller_categories = df_copy['seller_type'].unique()
seller_categories
```

```
array(['Individual', 'Dealer', 'Trustmark Dealer'], dtype=object)
```

```
#Count the number of records for each category
seller_categories_count_for_each = df_copy['seller_type'].value_counts()
seller_categories_count_for_each
```

```
seller_type
Individual          6218
Dealer               681
Trustmark Dealer      27
Name: count, dtype: int64
```

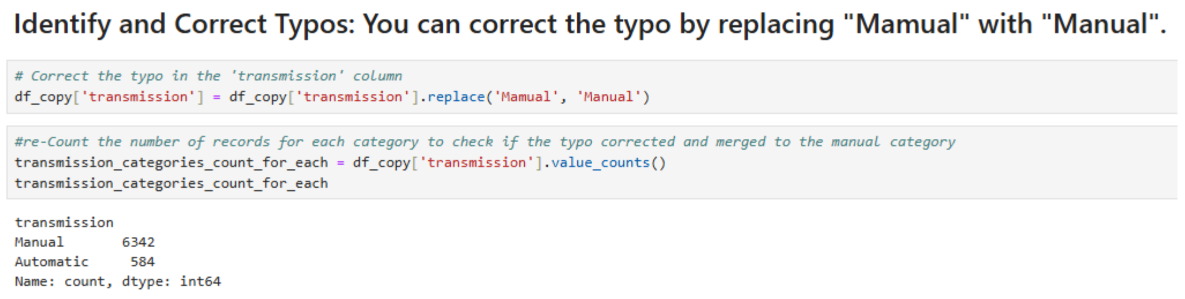
check for transmission

```
transmission_categories = df_copy['transmission'].unique()
transmission_categories
```

```
array(['Manual', 'Mamual', 'Automatic'], dtype=object)
```

```
#Count the number of records for each category
transmission_categories_count_for_each = df_copy['transmission'].value_counts()
transmission_categories_count_for_each
```

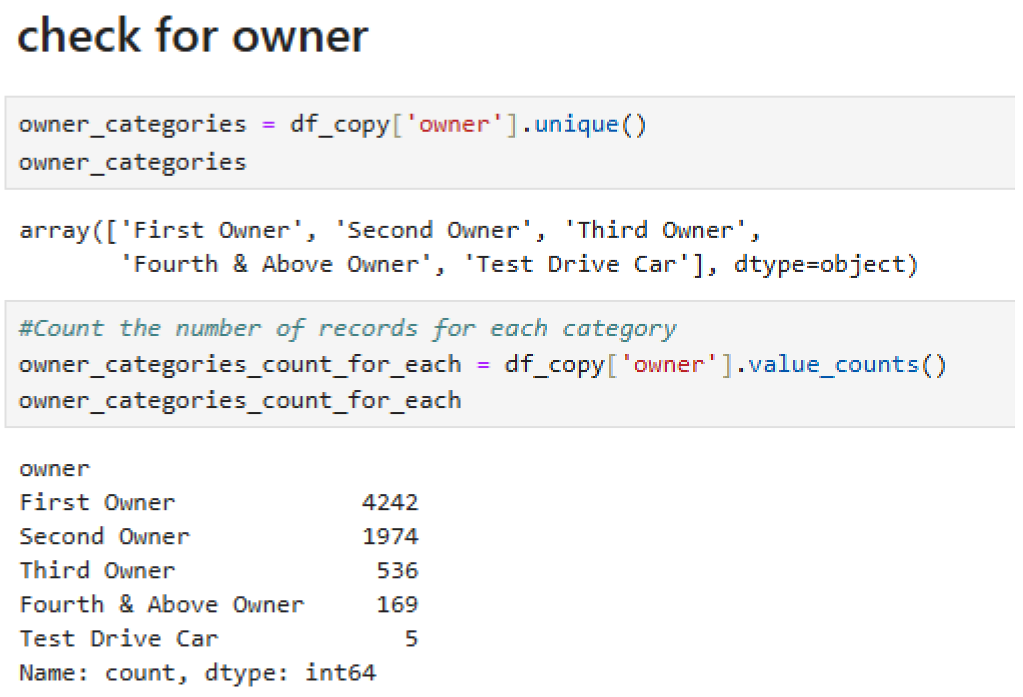
```
transmission
Manual       6341
Automatic     584
Mamual          1
Name: count, dtype: int64
```


Identify and Correct Typos: You can correct the typo by replacing "Mamual" with "Manual".

```
# Correct the typo in the 'transmission' column
df_copy['transmission'] = df_copy['transmission'].replace('Mamual', 'Manual')
```

```
#re-Count the number of records for each category to check if the typo corrected and merged to the manual category
transmission_categories_count_for_each = df_copy['transmission'].value_counts()
transmission_categories_count_for_each
```

```
transmission
Manual       6342
Automatic     584
Name: count, dtype: int64
```


check for owner

```
owner_categories = df_copy['owner'].unique()
owner_categories
```

```
array(['First Owner', 'Second Owner', 'Third Owner',
       'Fourth & Above Owner', 'Test Drive Car'], dtype=object)
```

```
#Count the number of records for each category
owner_categories_count_for_each = df_copy['owner'].value_counts()
owner_categories_count_for_each
```

```
owner
First Owner             4242
Second Owner            1974
Third Owner              536
Fourth & Above Owner     169
Test Drive Car             5
Name: count, dtype: int64
```

categorical data needs to be converted into a numerical format since the model requires numeric inputs. Here are common methods to convert categorical data into a suitable format:

Choosing the Right Encoding Method:

For Non-Ordinal Categorical Data (e.g., apple, nokia, google pixel, tecno):

Use One-Hot Encoding if the number of categories is small.

Consider Binary Encoding if you have a large number of categories and want to reduce dimensionality.

For Ordinal Categorical Data:

Use Label Encoding if there's a meaningful order among categories (e.g., Low, Medium, High).

for our case in this dataset we have no-ordinal categorical data, we should use binary encoding but to avoid complexity of the concept because it create other columns with binary representations let we use label encoding, students can play with binary encoding their self.

```python
from sklearn.preprocessing import LabelEncoder
# Initialize LabelEncoder
label_encoder = LabelEncoder()
```

```python
# Apply LabelEncoder to the 'fuel' column
df_copy['fuel'] = label_encoder.fit_transform(df_copy['fuel'])
df_copy
```

	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage	engine	max_power	seats	brand	model	torque_value	torque_rpm_value
0	2014	450000	145500	1	Individual	Manual	First Owner	23.40	1248.0	74.00	5.0	Maruti	Swift Dzire Vdi	1.902000e+06	2000.0
1	2014	370000	120000	1	Individual	Manual	Second Owner	21.14	1498.0	103.52	5.0	Skoda	Rapid 1.5 Tdi Ambition	2.501500e+10	2000.0
2	2006	158000	140000	3	Individual	Manual	Third Owner	17.70	1497.0	78.00	5.0	Honda	City 2017-2020 Exi	1.248519e+02	NaN
3	2010	225000	127000	1	Individual	Manual	First Owner	23.00	1396.0	90.00	5.0	Hyundai	I20 Sportz Diesel	2.199157e+02	2250.0
4	2007	130000	120000	3	Individual	Manual	First Owner	16.10	1298.0	88.20	5.0	Maruti	Swift Vxi Bsiii	1.132565e+02	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
8121	2013	260000	50000	3	Individual	Manual	Second Owner	18.90	998.0	67.10	5.0	Maruti	Wagon R Vxi Bs Iv With Abs	9.035000e+05	3500.0

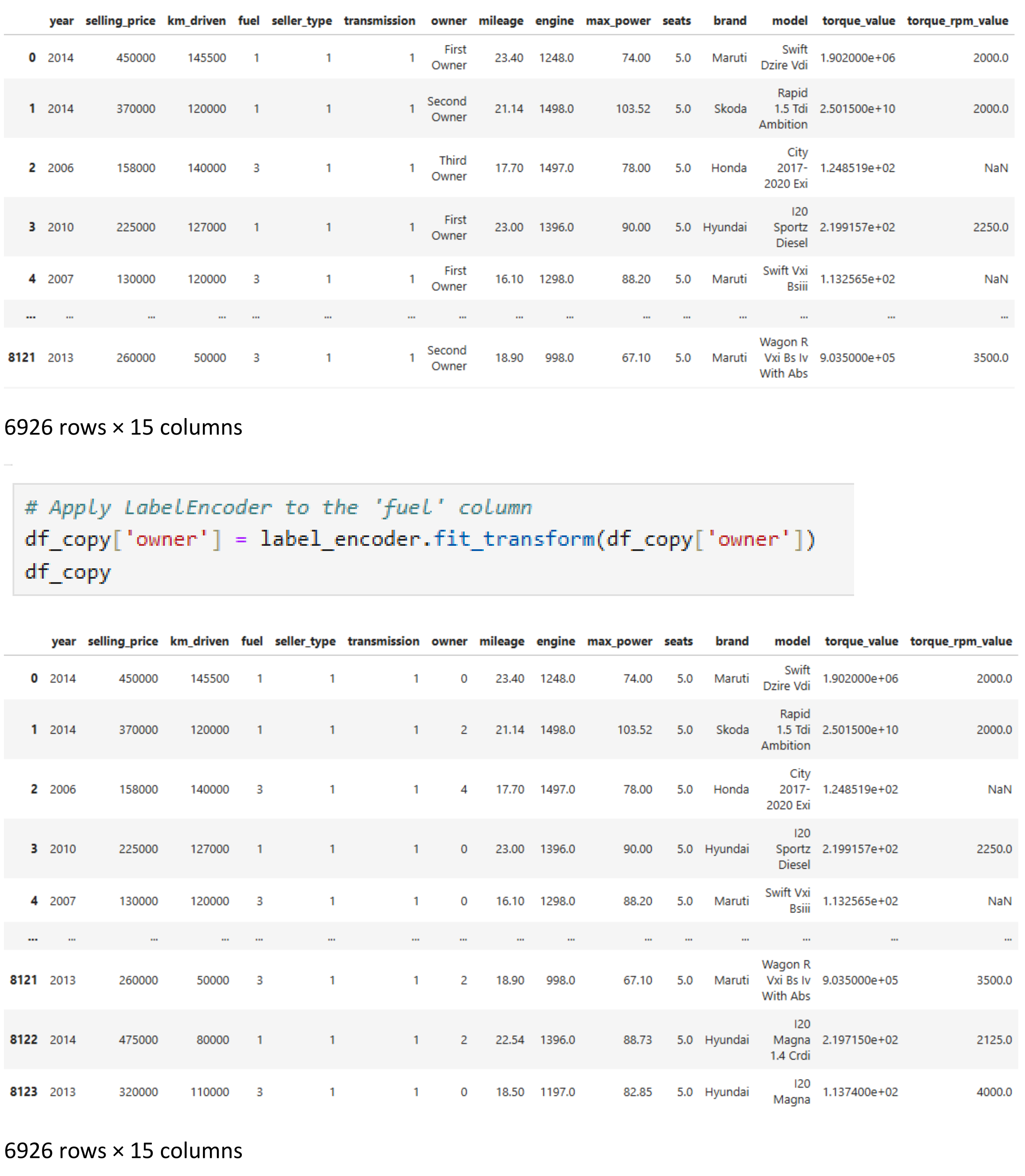
6926 rows × 15 columns

```python
# Apply LabelEncoder to the 'seller_type' column
df_copy['seller_type'] = label_encoder.fit_transform(df_copy['seller_type'])
df_copy
```

	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage	engine	max_power	seats	brand	model	torque_value	torque_rpm_value
0	2014	450000	145500	1	1	Manual	First Owner	23.40	1248.0	74.00	5.0	Maruti	Swift Dzire Vdi	1.902000e+06	2000.0
1	2014	370000	120000	1	1	Manual	Second Owner	21.14	1498.0	103.52	5.0	Skoda	Rapid 1.5 Tdi Ambition	2.501500e+10	2000.0
2	2006	158000	140000	3	1	Manual	Third Owner	17.70	1497.0	78.00	5.0	Honda	City 2017-2020 Exi	1.248519e+02	NaN
3	2010	225000	127000	1	1	Manual	First Owner	23.00	1396.0	90.00	5.0	Hyundai	I20 Sportz Diesel	2.199157e+02	2250.0
4	2007	130000	120000	3	1	Manual	First Owner	16.10	1298.0	88.20	5.0	Maruti	Swift Vxi Bsiii	1.132565e+02	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
8121	2013	260000	50000	3	1	Manual	Second	18.90	998.0	67.10	5.0	Maruti	Wagon R Vxi Bs Iv	9.035000e+05	3500.0

6926 rows × 15 columns

```python
# Apply LabelEncoder to the 'fuel' column
df_copy['transmission'] = label_encoder.fit_transform(df_copy['transmission'])
df_copy
```



	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage	engine	max_power	seats	brand	model	torque_value	torque_rpm_value
0	2014	450000	145500	1	1	1	First Owner	23.40	1248.0	74.00	5.0	Maruti	Swift Dzire Vdi	1.902000e+06	2000.0
1	2014	370000	120000	1	1	1	Second Owner	21.14	1498.0	103.52	5.0	Skoda	Rapid 1.5 Tdi Ambition	2.501500e+10	2000.0
2	2006	158000	140000	3	1	1	Third Owner	17.70	1497.0	78.00	5.0	Honda	City 2017-2020 Exi	1.248519e+02	NaN
3	2010	225000	127000	1	1	1	First Owner	23.00	1396.0	90.00	5.0	Hyundai	I20 Sportz Diesel	2.199157e+02	2250.0
4	2007	130000	120000	3	1	1	First Owner	16.10	1298.0	88.20	5.0	Maruti	Swift Vxi Bsiii	1.132565e+02	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
8121	2013	260000	50000	3	1	1	Second Owner	18.90	998.0	67.10	5.0	Maruti	Wagon R Vxi Bs Iv With Abs	9.035000e+05	3500.0

6926 rows × 15 columns

```
# Apply LabelEncoder to the 'fuel' column
df_copy['owner'] = label_encoder.fit_transform(df_copy['owner'])
df_copy
```

	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage	engine	max_power	seats	brand	model	torque_value	torque_rpm_value
0	2014	450000	145500	1	1	1	0	23.40	1248.0	74.00	5.0	Maruti	Swift Dzire Vdi	1.902000e+06	2000.0
1	2014	370000	120000	1	1	1	2	21.14	1498.0	103.52	5.0	Skoda	Rapid 1.5 Tdi Ambition	2.501500e+10	2000.0
2	2006	158000	140000	3	1	1	4	17.70	1497.0	78.00	5.0	Honda	City 2017-2020 Exi	1.248519e+02	NaN
3	2010	225000	127000	1	1	1	0	23.00	1396.0	90.00	5.0	Hyundai	I20 Sportz Diesel	2.199157e+02	2250.0
4	2007	130000	120000	3	1	1	0	16.10	1298.0	88.20	5.0	Maruti	Swift Vxi Bsiii	1.132565e+02	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
8121	2013	260000	50000	3	1	1	2	18.90	998.0	67.10	5.0	Maruti	Wagon R Vxi Bs Iv With Abs	9.035000e+05	3500.0
8122	2014	475000	80000	1	1	1	2	22.54	1396.0	88.73	5.0	Hyundai	I20 Magna 1.4 Crdi	2.197150e+02	2125.0
8123	2013	320000	110000	3	1	1	0	18.50	1197.0	82.85	5.0	Hyundai	I20 Magna	1.137400e+02	4000.0

6926 rows × 15 columns

Ordering the columns

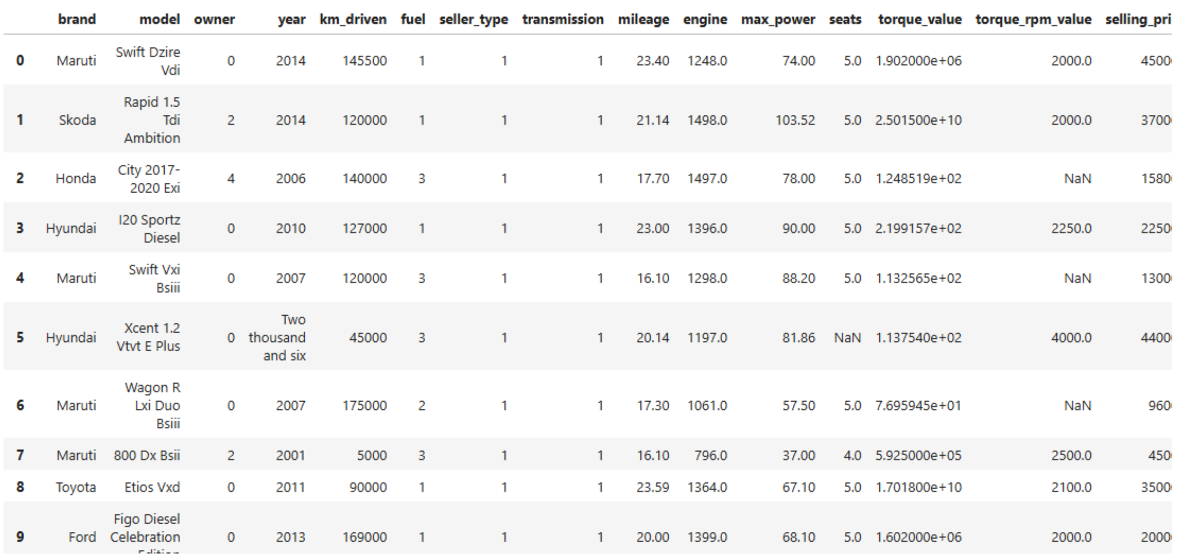
# New column order: Move 'price' to the last position

new_column_order = ['brand', 'model', 'owner','year', 'km_driven', 'fuel', 'seller_type', 'transmission', 'mileage','engine', 'max_power', 'seats', 'torque_value', 'torque_rpm_value', 'selling_price']

# Reorder DataFrame columns

df_copy = df_copy[new_column_order]

df_copy.head(20)



	brand	model	owner	year	km_driven	fuel	seller_type	transmission	mileage	engine	max_power	seats	torque_value	torque_rpm_value	selling_pri
0	Maruti	Swift Dzire Vdi	0	2014	145500	1	1	1	23.40	1248.0	74.00	5.0	1.902000e+06	2000.0	4500
1	Skoda	Rapid 1.5 Tdi Ambition	2	2014	120000	1	1	1	21.14	1498.0	103.52	5.0	2.501500e+10	2000.0	3700
2	Honda	City 2017-2020 Exi	4	2006	140000	3	1	1	17.70	1497.0	78.00	5.0	1.248519e+02	NaN	1580
3	Hyundai	I20 Sportz Diesel	0	2010	127000	1	1	1	23.00	1396.0	90.00	5.0	2.199157e+02	2250.0	2250
4	Maruti	Swift Vxi Bsiii	0	2007	120000	3	1	1	16.10	1298.0	88.20	5.0	1.132565e+02	NaN	1300
5	Hyundai	Xcent 1.2 Vtvt E Plus	0	Two thousand and six	45000	3	1	1	20.14	1197.0	81.86	NaN	1.137540e+02	4000.0	4400
6	Maruti	Wagon R Lxi Duo Bsiii	0	2007	175000	2	1	1	17.30	1061.0	57.50	5.0	7.695945e+01	NaN	960
7	Maruti	800 Dx Bsii	2	2001	5000	3	1	1	16.10	796.0	37.00	4.0	5.925000e+05	2500.0	450
8	Toyota	Etios Vxd	0	2011	90000	1	1	1	23.59	1364.0	67.10	5.0	1.701800e+10	2100.0	3500
9	Ford	Figo Diesel Celebration	0	2013	169000	1	1	1	20.00	1399.0	68.10	5.0	1.602000e+06	2000.0	2000

5. chenging the data type of a year column from object to integer because there is a year written in words on 5 record

```
df_copy.dtypes
```

```
brand                object
model                object
owner                 int32
year                 object
km_driven             int64
fuel                  int32
seller_type           int32
transmission          int32
mileage             float64
engine              float64
max_power           float64
seats               float64
torque_value        float64
torque_rpm_value    float64
selling_price         int64
dtype: object
```

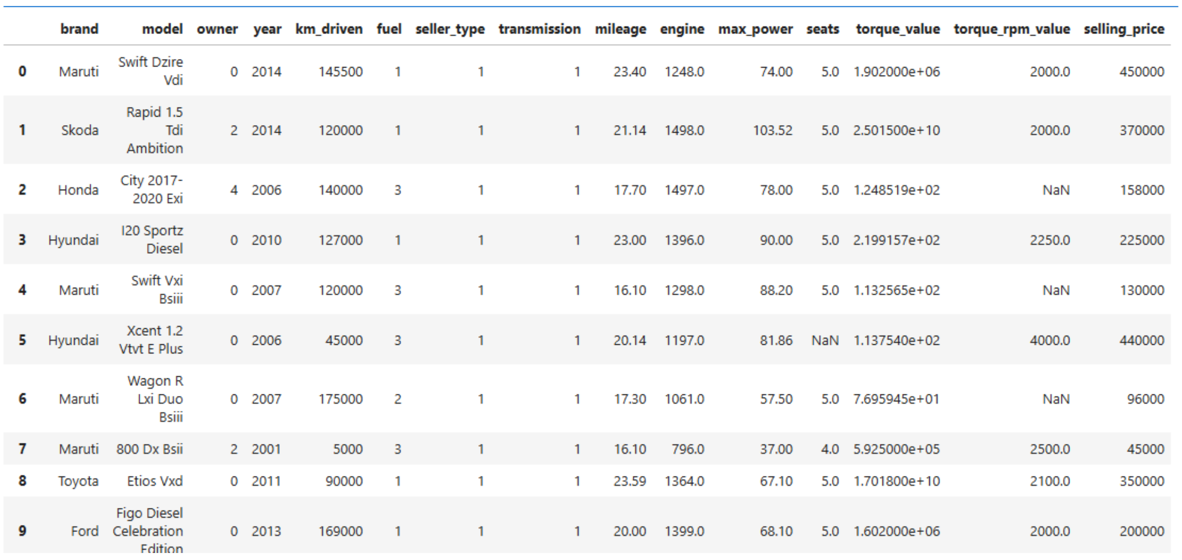
```
pip install word2number
```

Requirement already satisfied: word2number in c:\users\user\anaconda3\lib\site-packages (1.1)Note: you may need to restart the kernel to use updated packages.

# Function to clean the 'year' column and convert word-based values to numeric

from word2number import w2n # For converting words to numbers

```
    def clean_year(value):
        try:
            # Try to convert numeric values directly
            return int(value)
        except ValueError:
            # Convert word-based values using word2number
            try:
                return w2n.word_to_num(value)
            except ValueError:
                # Handle cases where the value cannot be converted
                return None

    # Apply the cleaning function to the 'year' column
df_copy['year'] = df_copy['year'].apply(clean_year)

df_copy.head(20)
```



	brand	model	owner	year	km_driven	fuel	seller_type	transmission	mileage	engine	max_power	seats	torque_value	torque_rpm_value	selling_price
0	Maruti	Swift Dzire Vdi	0	2014	145500	1	1	1	23.40	1248.0	74.00	5.0	1.902000e+06	2000.0	450000
1	Skoda	Rapid 1.5 Tdi Ambition	2	2014	120000	1	1	1	21.14	1498.0	103.52	5.0	2.501500e+10	2000.0	370000
2	Honda	City 2017-2020 Exi	4	2006	140000	3	1	1	17.70	1497.0	78.00	5.0	1.248519e+02	NaN	158000
3	Hyundai	I20 Sportz Diesel	0	2010	127000	1	1	1	23.00	1396.0	90.00	5.0	2.199157e+02	2250.0	225000
4	Maruti	Swift Vxi Bsiii	0	2007	120000	3	1	1	16.10	1298.0	88.20	5.0	1.132565e+02	NaN	130000
5	Hyundai	Xcent 1.2 Vtvt E Plus	0	2006	45000	3	1	1	20.14	1197.0	81.86	NaN	1.137540e+02	4000.0	440000
6	Maruti	Wagon R Lxi Duo Bsiii	0	2007	175000	2	1	1	17.30	1061.0	57.50	5.0	7.695945e+01	NaN	96000
7	Maruti	800 Dx Bsii	2	2001	5000	3	1	1	16.10	796.0	37.00	4.0	5.925000e+05	2500.0	45000
8	Toyota	Etios Vxd	0	2011	90000	1	1	1	23.59	1364.0	67.10	5.0	1.701800e+10	2100.0	350000
9	Ford	Figo Diesel Celebration Edition	0	2013	169000	1	1	1	20.00	1399.0	68.10	5.0	1.602000e+06	2000.0	200000

now the year column have int64 datatype then

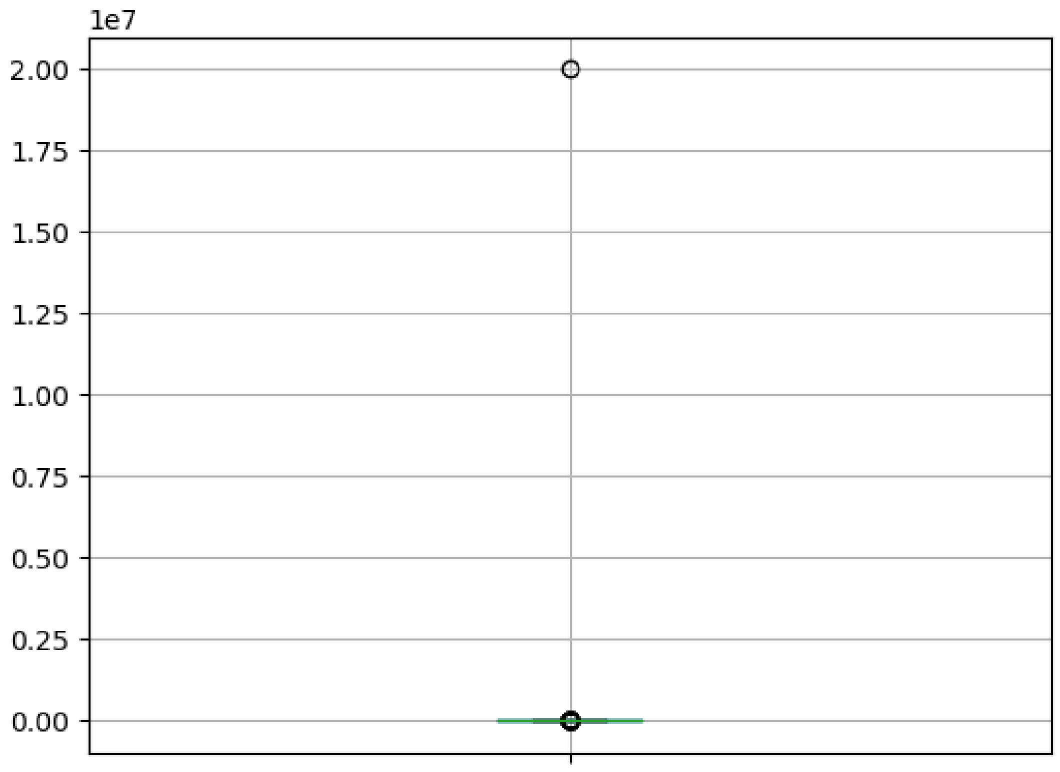
```
df_copy.dtypes
```

```
brand                 object
model                 object
owner                  int32
year                   int64
km_driven              int64
fuel                   int32
seller_type            int32
transmission           int32
mileage              float64
engine               float64
max_power            float64
seats                float64
torque_value         float64
torque_rpm_value     float64
selling_price          int64
dtype: object
```

6. Handling OUTLIERS for each column

Identify outlier by ploting using boxplot

```
import matplotlib.pyplot as plt
df_copy.boxplot(column='year')
plt.show()
```



# Create a histogram for the 'year' column

```
plt.hist(df_copy['year'], bins=20, edgecolor='black')  # Adjust the number of bins as needed

plt.title('Distribution of Year')

plt.xlabel('Year')

plt.ylabel('Frequency')

plt.show()
```

Explanation recall in statistics concepts:

Q1 (25th percentile): The value below which 25% of the data fall.

Q3 (75th percentile): The value below which 75% of the data fall.

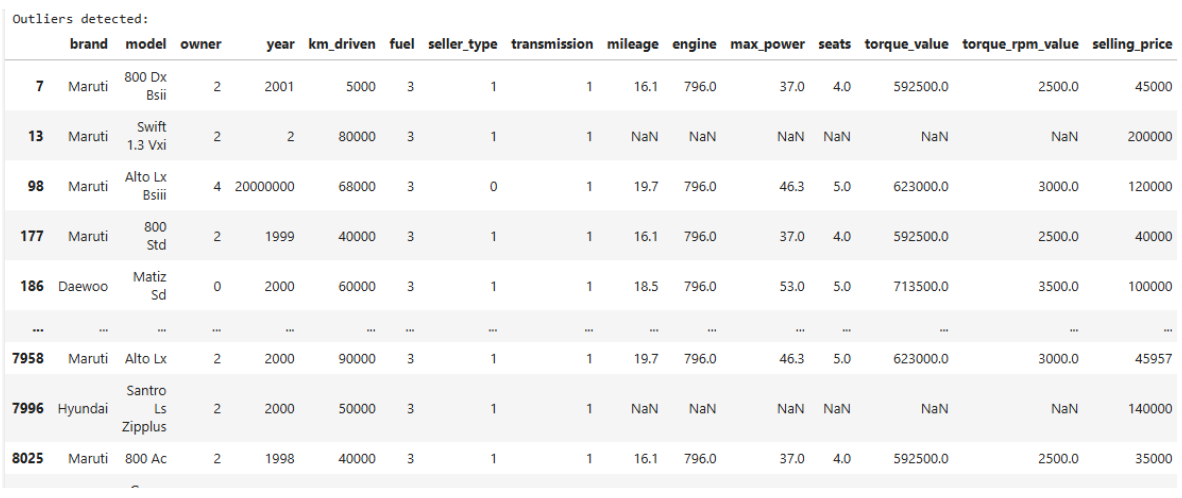
IQR: The range between Q1 and Q3, representing the middle 50% of the data.

Outliers: Data points that fall outside the range

```
# Calculate Q1 (25th percentile) and Q3 (75th percentile)

Q1 = df_copy['year'].quantile(0.25)

Q3 = df_copy['year'].quantile(0.75)


# Calculate IQR (Interquartile Range)

IQR = Q3 - Q1


# Calculate the lower and upper bounds for outliers

lower_bound = Q1 - 1.5 * IQR

upper_bound = Q3 + 1.5 * IQR


# Find the outliers

outliers = df_copy[(df_copy['year'] < lower_bound) | (df_copy['year'] > upper_bound)]


# Display the outliers
```

print("Outliers detected:")

outliers



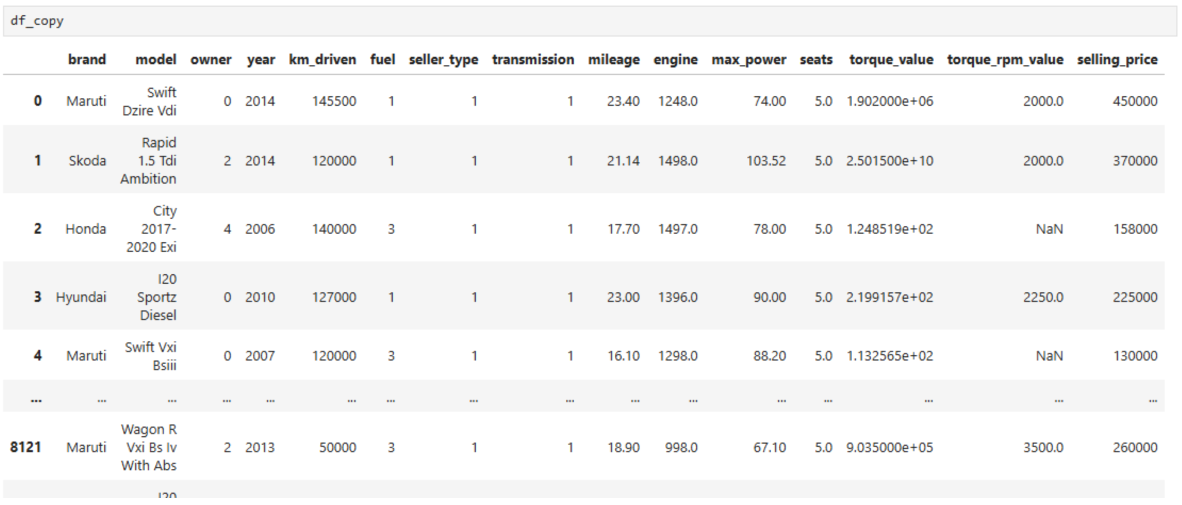
```
Outliers detected:
```

	brand	model	owner	year	km_driven	fuel	seller_type	transmission	mileage	engine	max_power	seats	torque_value	torque_rpm_value	selling_price
7	Maruti	800 Dx Bsii	2	2001	5000	3	1	1	16.1	796.0	37.0	4.0	592500.0	2500.0	45000
13	Maruti	Swift 1.3 Vxi	2	2	80000	3	1	1	NaN	NaN	NaN	NaN	NaN	NaN	200000
98	Maruti	Alto Lx Bsiii	4	20000000	68000	3	0	1	19.7	796.0	46.3	5.0	623000.0	3000.0	120000
177	Maruti	800 Std	2	1999	40000	3	1	1	16.1	796.0	37.0	4.0	592500.0	2500.0	40000
186	Daewoo	Matiz Sd	0	2000	60000	3	1	1	18.5	796.0	53.0	5.0	713500.0	3500.0	100000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
7958	Maruti	Alto Lx	2	2000	90000	3	1	1	19.7	796.0	46.3	5.0	623000.0	3000.0	45957
7996	Hyundai	Santro Ls Zipplus	2	2000	50000	3	1	1	NaN	NaN	NaN	NaN	NaN	NaN	140000
8025	Maruti	800 Ac	2	1998	40000	3	1	1	16.1	796.0	37.0	4.0	592500.0	2500.0	35000

we detect 79 outliers records in our dataset, we are going to remove them

# Filter out the outliers

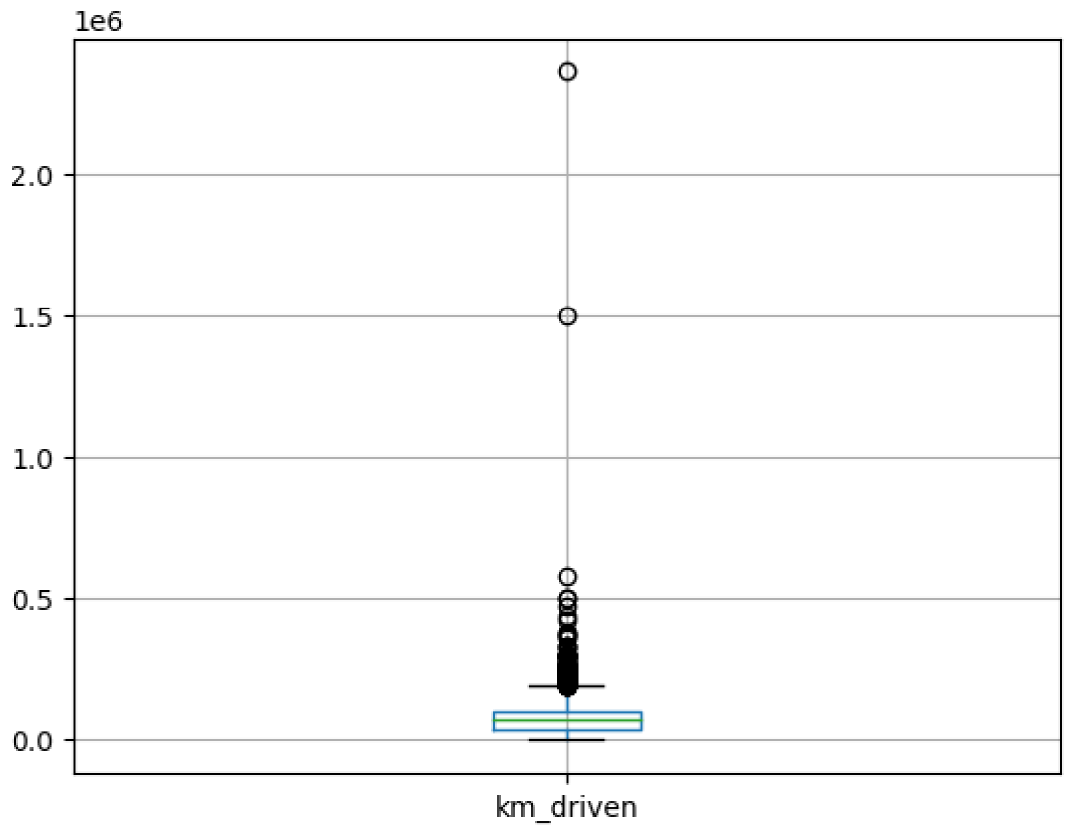
df_copy = df_copy[(df_copy['year'] >= lower_bound) & (df_copy['year'] <= upper_bound)]



```
df_copy
```

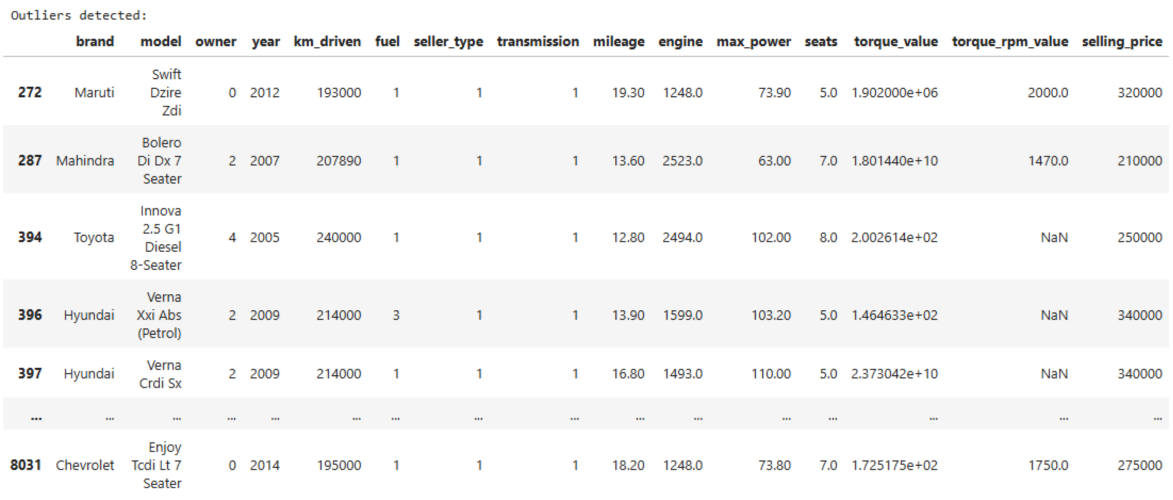
	brand	model	owner	year	km_driven	fuel	seller_type	transmission	mileage	engine	max_power	seats	torque_value	torque_rpm_value	selling_price
0	Maruti	Swift Dzire Vdi	0	2014	145500	1	1	1	23.40	1248.0	74.00	5.0	1.902000e+06	2000.0	450000
1	Skoda	Rapid 1.5 Tdi Ambition	2	2014	120000	1	1	1	21.14	1498.0	103.52	5.0	2.501500e+10	2000.0	370000
2	Honda	City 2017-2020 Exi	4	2006	140000	3	1	1	17.70	1497.0	78.00	5.0	1.248519e+02	NaN	158000
3	Hyundai	I20 Sportz Diesel	0	2010	127000	1	1	1	23.00	1396.0	90.00	5.0	2.199157e+02	2250.0	225000
4	Maruti	Swift Vxi Bsiii	0	2007	120000	3	1	1	16.10	1298.0	88.20	5.0	1.132565e+02	NaN	130000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
8121	Maruti	Wagon R Vxi Bs Iv With Abs	2	2013	50000	3	1	1	18.90	998.0	67.10	5.0	9.035000e+05	3500.0	260000

```
import matplotlib.pyplot as plt
df_copy.boxplot(column='km_driven')
plt.show()
```



```
# Calculate Q1 (25th percentile) and Q3 (75th percentile)
Q1 = df_copy['km_driven'].quantile(0.25)
Q3 = df_copy['km_driven'].quantile(0.75)


# Calculate IQR (Interquartile Range)
IQR = Q3 - Q1


# Calculate the lower and upper bounds for outliers
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
```

```
        # Find the outliers

        outliers = df_copy[(df_copy['km_driven'] < lower_bound) | (df_copy['km_driven']
        > upper_bound)]


        # Display the outliers

        print("Outliers detected:")

outliers
```



```
Outliers detected:
```

	brand	model	owner	year	km_driven	fuel	seller_type	transmission	mileage	engine	max_power	seats	torque_value	torque_rpm_value	selling_price
272	Maruti	Swift Dzire Zdi	0	2012	193000	1	1	1	19.30	1248.0	73.90	5.0	1.902000e+06	2000.0	320000
287	Mahindra	Bolero Di Dx 7 Seater	2	2007	207890	1	1	1	13.60	2523.0	63.00	7.0	1.801440e+10	1470.0	210000
394	Toyota	Innova 2.5 G1 Diesel 8-Seater	4	2005	240000	1	1	1	12.80	2494.0	102.00	8.0	2.002614e+02	NaN	250000
396	Hyundai	Verna Xxi Abs (Petrol)	2	2009	214000	3	1	1	13.90	1599.0	103.20	5.0	1.464633e+02	NaN	340000
397	Hyundai	Verna Crdi Sx	2	2009	214000	1	1	1	16.80	1493.0	110.00	5.0	2.373042e+10	NaN	340000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
8031	Chevrolet	Enjoy Tcdi Lt 7 Seater	0	2014	195000	1	1	1	18.20	1248.0	73.80	7.0	1.725175e+02	1750.0	275000

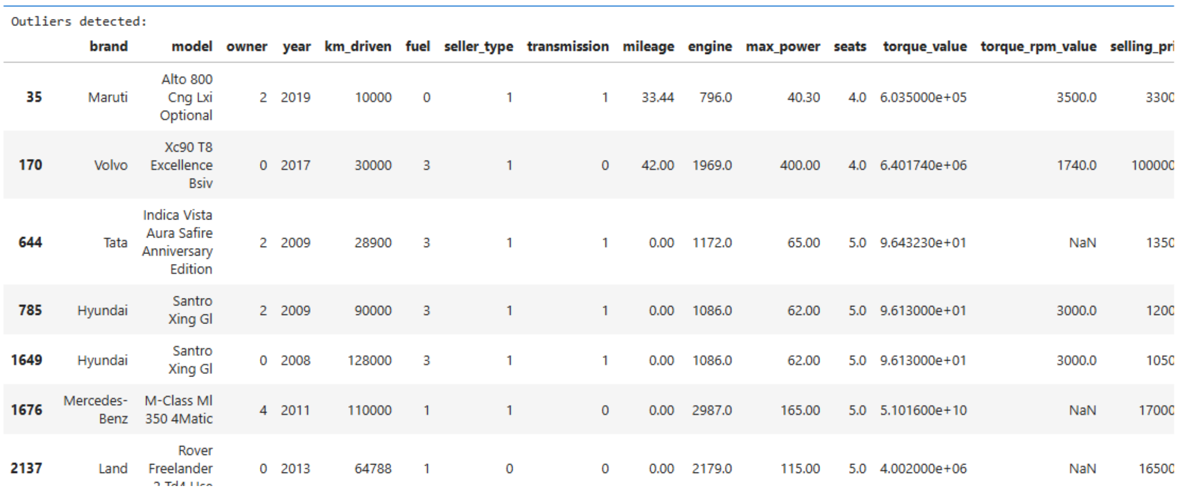
```
        # Calculate Q1 (25th percentile) and Q3 (75th percentile)

        Q1 = df_copy['mileage'].quantile(0.25)

        Q3 = df_copy['mileage'].quantile(0.75)


        # Calculate IQR (Interquartile Range)

        IQR = Q3 - Q1


        # Calculate the lower and upper bounds for outliers

        lower_bound = Q1 - 1.5 * IQR

        upper_bound = Q3 + 1.5 * IQR


        # Find the outliers

        outliers = df_copy[(df_copy['mileage'] < lower_bound) | (df_copy['mileage'] >
```

upper_bound)]

# Display the outliers

print("Outliers detected:")

outliers



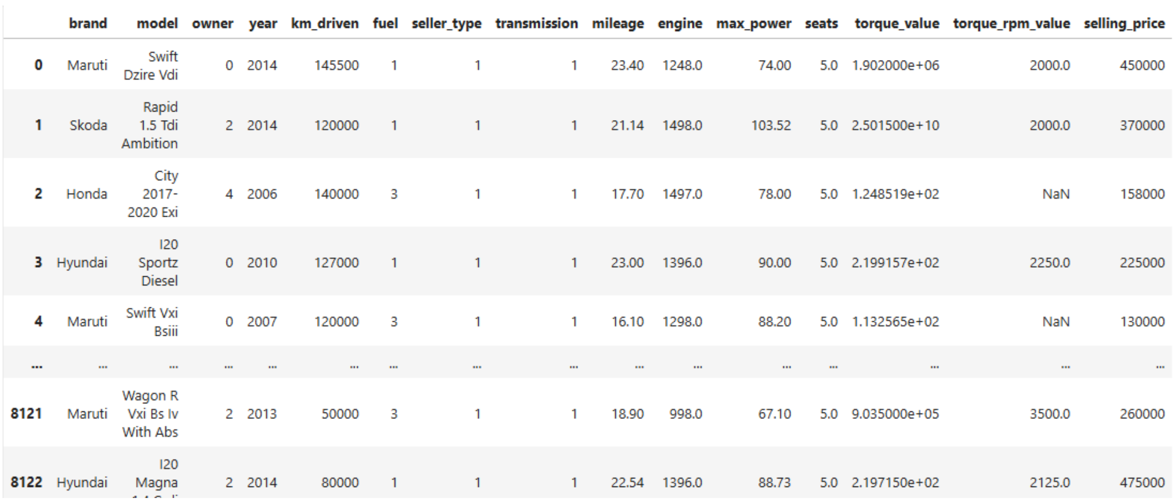
```
Outliers detected:
```

	brand	model	owner	year	km_driven	fuel	seller_type	transmission	mileage	engine	max_power	seats	torque_value	torque_rpm_value	selling_pri
35	Maruti	Alto 800 Cng Lxi Optional	2	2019	10000	0	1	1	33.44	796.0	40.30	4.0	6.035000e+05	3500.0	3300
170	Volvo	Xc90 T8 Excellence Bsiv	0	2017	30000	3	1	0	42.00	1969.0	400.00	4.0	6.401740e+06	1740.0	100000
644	Tata	Indica Vista Aura Safire Anniversary Edition	2	2009	28900	3	1	1	0.00	1172.0	65.00	5.0	9.643230e+01	NaN	1350
785	Hyundai	Santro Xing Gl	2	2009	90000	3	1	1	0.00	1086.0	62.00	5.0	9.613000e+01	3000.0	1200
1649	Hyundai	Santro Xing Gl	0	2008	128000	3	1	1	0.00	1086.0	62.00	5.0	9.613000e+01	3000.0	1050
1676	Mercedes-Benz	M-Class Ml 350 4Matic	4	2011	110000	1	1	0	0.00	2987.0	165.00	5.0	5.101600e+10	NaN	17000
2137	Land	Rover Freelander	0	2013	64788	1	0	0	0.00	2179.0	115.00	5.0	4.002000e+06	NaN	16500

# Filter out the outliers

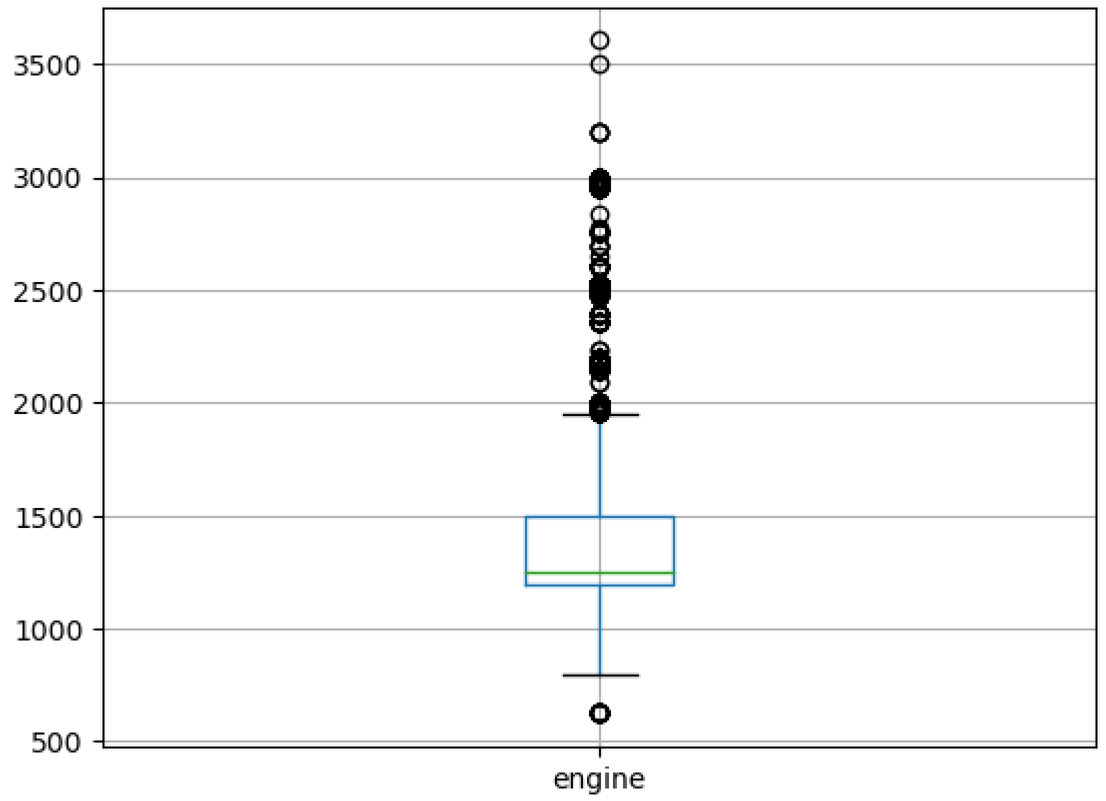
df_copy = df_copy[(df_copy['mileage'] >= lower_bound) & (df_copy['mileage'] <= upper_bound)]

df_copy



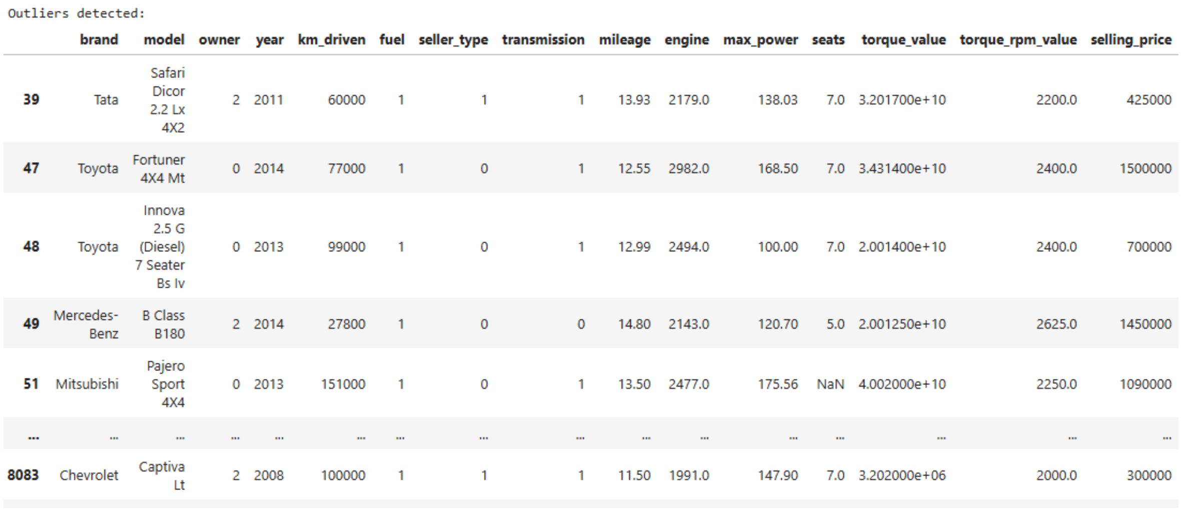
	brand	model	owner	year	km_driven	fuel	seller_type	transmission	mileage	engine	max_power	seats	torque_value	torque_rpm_value	selling_price
0	Maruti	Swift Dzire Vdi	0	2014	145500	1	1	1	23.40	1248.0	74.00	5.0	1.902000e+06	2000.0	450000
1	Skoda	Rapid 1.5 Tdi Ambition	2	2014	120000	1	1	1	21.14	1498.0	103.52	5.0	2.501500e+10	2000.0	370000
2	Honda	City 2017-2020 Exi	4	2006	140000	3	1	1	17.70	1497.0	78.00	5.0	1.248519e+02	NaN	158000
3	Hyundai	I20 Sportz Diesel	0	2010	127000	1	1	1	23.00	1396.0	90.00	5.0	2.199157e+02	2250.0	225000
4	Maruti	Swift Vxi Bsiii	0	2007	120000	3	1	1	16.10	1298.0	88.20	5.0	1.132565e+02	NaN	130000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
8121	Maruti	Wagon R Vxi Bs Iv With Abs	2	2013	50000	3	1	1	18.90	998.0	67.10	5.0	9.035000e+05	3500.0	260000
8122	Hyundai	I20 Magna	2	2014	80000	1	1	1	22.54	1396.0	88.73	5.0	2.197150e+02	2125.0	475000

```
import matplotlib.pyplot as plt
df_copy.boxplot(column='engine')
plt.show()
```



```
# Calculate Q1 (25th percentile) and Q3 (75th percentile)

Q1 = df_copy['engine'].quantile(0.25)

Q3 = df_copy['engine'].quantile(0.75)


# Calculate IQR (Interquartile Range)

IQR = Q3 - Q1


# Calculate the lower and upper bounds for outliers

lower_bound = Q1 - 1.5 * IQR

upper_bound = Q3 + 1.5 * IQR


# Find the outliers
```

```
        outliers = df_copy[(df_copy['engine'] < lower_bound) | (df_copy['engine'] >
        upper_bound)]


        # Display the outliers

        print("Outliers detected:")

outliers
```

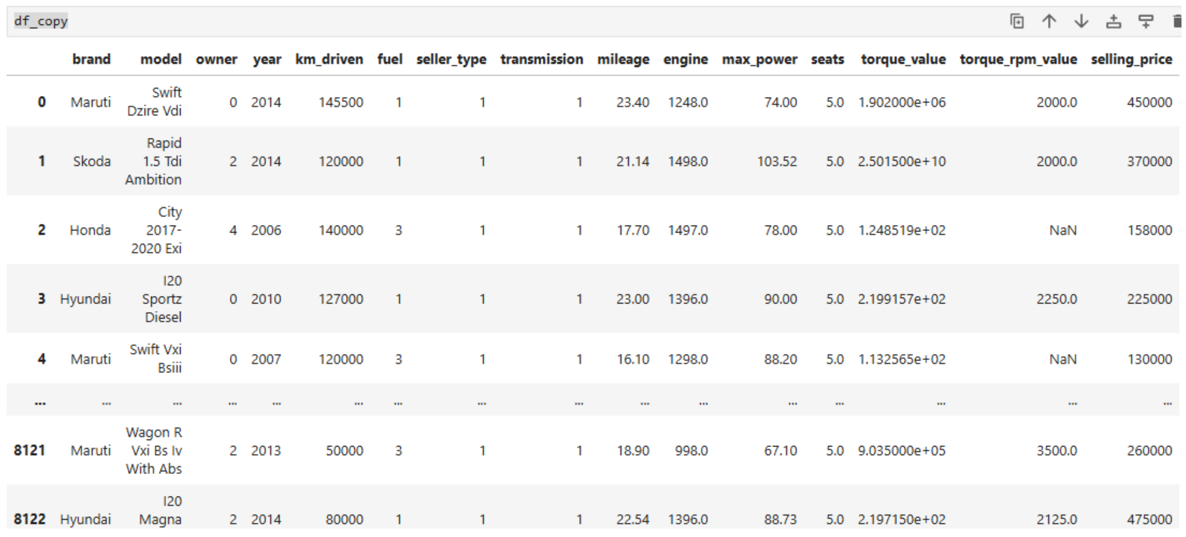


```
Outliers detected:
```

	brand	model	owner	year	km_driven	fuel	seller_type	transmission	mileage	engine	max_power	seats	torque_value	torque_rpm_value	selling_price
39	Tata	Safari Dicor 2.2 Lx 4X2	2	2011	60000	1	1	1	13.93	2179.0	138.03	7.0	3.201700e+10	2200.0	425000
47	Toyota	Fortuner 4X4 Mt	0	2014	77000	1	0	1	12.55	2982.0	168.50	7.0	3.431400e+10	2400.0	1500000
48	Toyota	Innova 2.5 G (Diesel) 7 Seater Bs Iv	0	2013	99000	1	0	1	12.99	2494.0	100.00	7.0	2.001400e+10	2400.0	700000
49	Mercedes-Benz	B Class B180	2	2014	27800	1	0	0	14.80	2143.0	120.70	5.0	2.001250e+10	2625.0	1450000
51	Mitsubishi	Pajero Sport 4X4	0	2013	151000	1	0	1	13.50	2477.0	175.56	NaN	4.002000e+10	2250.0	1090000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
8083	Chevrolet	Captiva Lt	2	2008	100000	1	1	1	11.50	1991.0	147.90	7.0	3.202000e+06	2000.0	300000

1103 rows × 15 columns

```
        # Filter out the outliers

df_copy = df_copy[(df_copy['engine'] >= lower_bound) & (df_copy['engine'] <=
upper_bound)]
```

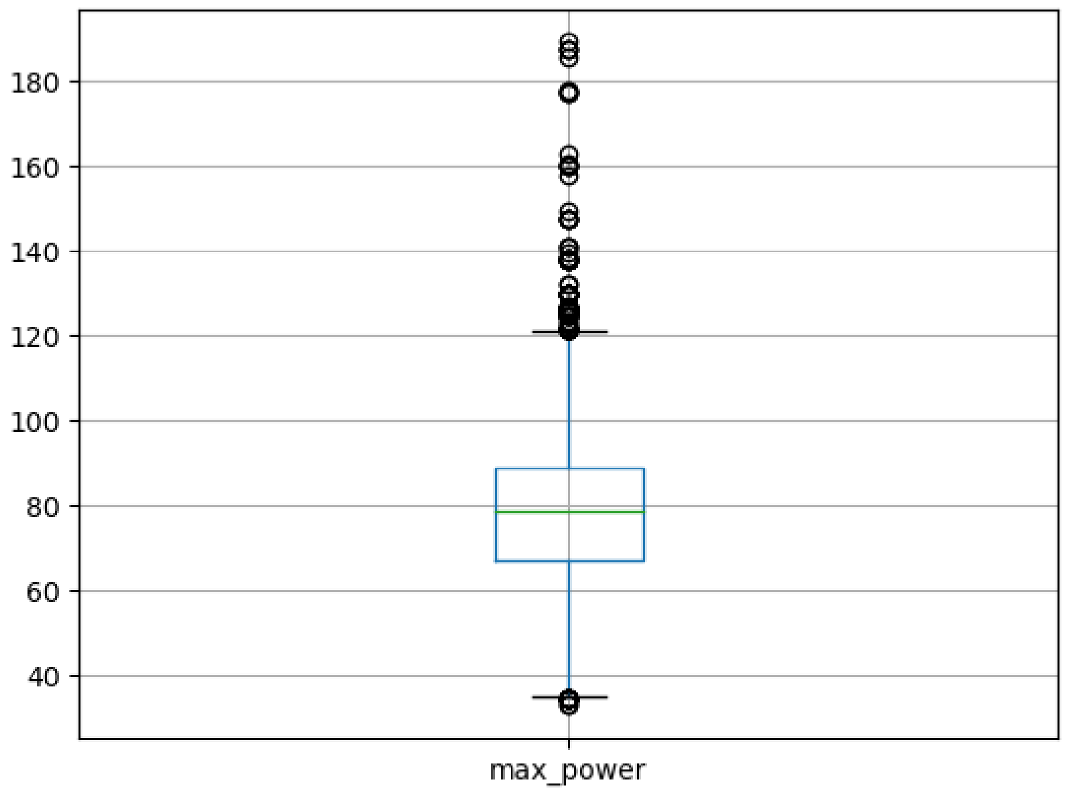


```
df_copy
```

	brand	model	owner	year	km_driven	fuel	seller_type	transmission	mileage	engine	max_power	seats	torque_value	torque_rpm_value	selling_price
0	Maruti	Swift Dzire Vdi	0	2014	145500	1	1	1	23.40	1248.0	74.00	5.0	1.902000e+06	2000.0	450000
1	Skoda	Rapid 1.5 Tdi Ambition	2	2014	120000	1	1	1	21.14	1498.0	103.52	5.0	2.501500e+10	2000.0	370000
2	Honda	City 2017-2020 Exi	4	2006	140000	3	1	1	17.70	1497.0	78.00	5.0	1.248519e+02	NaN	158000
3	Hyundai	I20 Sportz Diesel	0	2010	127000	1	1	1	23.00	1396.0	90.00	5.0	2.199157e+02	2250.0	225000
4	Maruti	Swift Vxi Bsiii	0	2007	120000	3	1	1	16.10	1298.0	88.20	5.0	1.132565e+02	NaN	130000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
8121	Maruti	Wagon R Vxi Bs Iv With Abs	2	2013	50000	3	1	1	18.90	998.0	67.10	5.0	9.035000e+05	3500.0	260000
8122	Hyundai	I20 Magna	2	2014	80000	1	1	1	22.54	1396.0	88.73	5.0	2.197150e+02	2125.0	475000

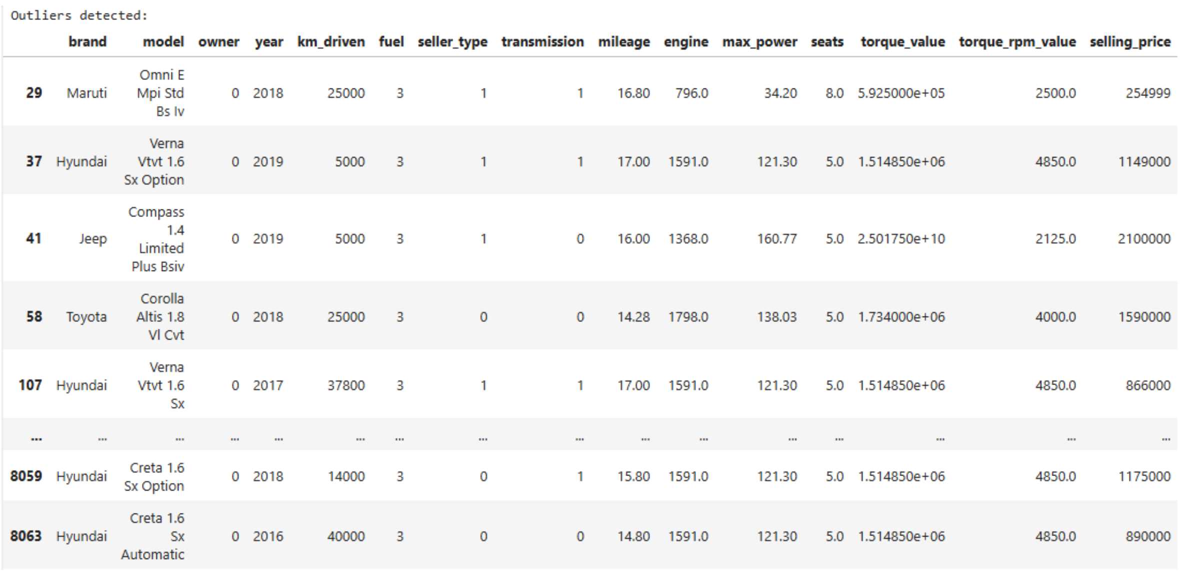
5376 rows × 15 columns

```
import matplotlib.pyplot as plt
df_copy.boxplot(column='max_power')
plt.show()
```



```
# Calculate Q1 (25th percentile) and Q3 (75th percentile)
Q1 = df_copy['max_power'].quantile(0.25)
Q3 = df_copy['max_power'].quantile(0.75)

# Calculate IQR (Interquartile Range)
IQR = Q3 - Q1

# Calculate the lower and upper bounds for outliers
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Find the outliers
```

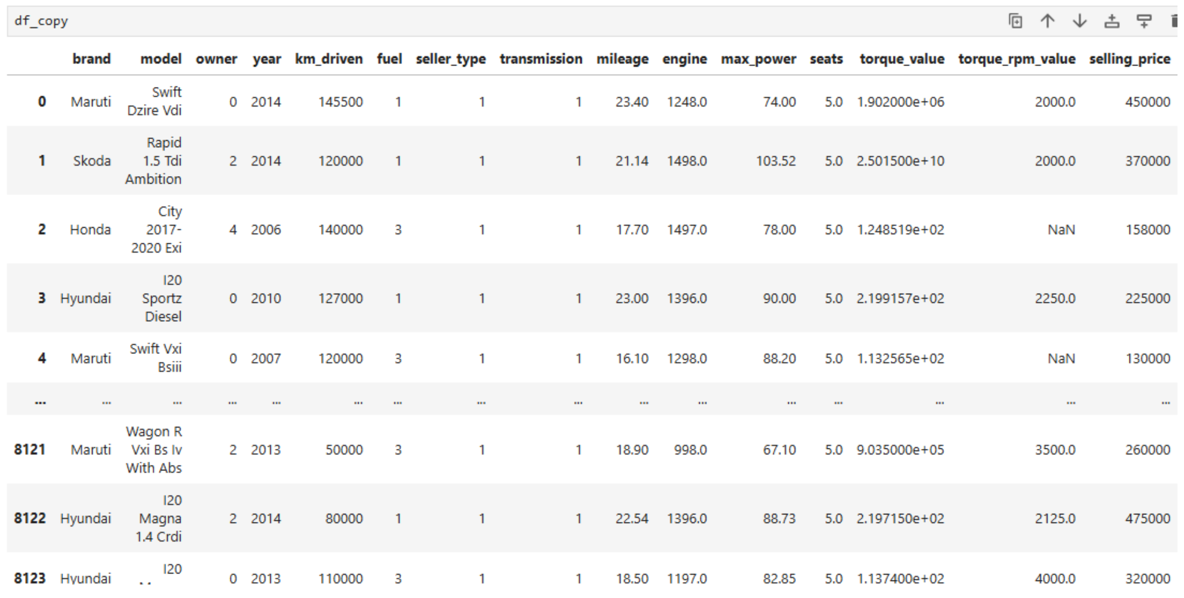
```
        outliers = df_copy[(df_copy['max_power'] < lower_bound) | (df_copy['max_power'] > upper_bound)]


        # Display the outliers

        print("Outliers detected:")

outliers
```



```
Outliers detected:
```

	brand	model	owner	year	km_driven	fuel	seller_type	transmission	mileage	engine	max_power	seats	torque_value	torque_rpm_value	selling_price
29	Maruti	Omni E Mpi Std Bs Iv	0	2018	25000	3	1	1	16.80	796.0	34.20	8.0	5.925000e+05	2500.0	254999
37	Hyundai	Verna Vtvt 1.6 Sx Option	0	2019	5000	3	1	1	17.00	1591.0	121.30	5.0	1.514850e+06	4850.0	1149000
41	Jeep	Compass 1.4 Limited Plus Bsiv	0	2019	5000	3	1	0	16.00	1368.0	160.77	5.0	2.501750e+10	2125.0	2100000
58	Toyota	Corolla Altis 1.8 Vl Cvt	0	2018	25000	3	0	0	14.28	1798.0	138.03	5.0	1.734000e+06	4000.0	1590000
107	Hyundai	Verna Vtvt 1.6 Sx	0	2017	37800	3	1	1	17.00	1591.0	121.30	5.0	1.514850e+06	4850.0	866000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
8059	Hyundai	Creta 1.6 Sx Option	0	2018	14000	3	0	1	15.80	1591.0	121.30	5.0	1.514850e+06	4850.0	1175000
8063	Hyundai	Creta 1.6 Sx Automatic	0	2016	40000	3	0	0	14.80	1591.0	121.30	5.0	1.514850e+06	4850.0	890000

```
        # Filter out the outliers

df_copy = df_copy[(df_copy['max_power'] >= lower_bound) & (df_copy['max_power'] <= upper_bound)]
```

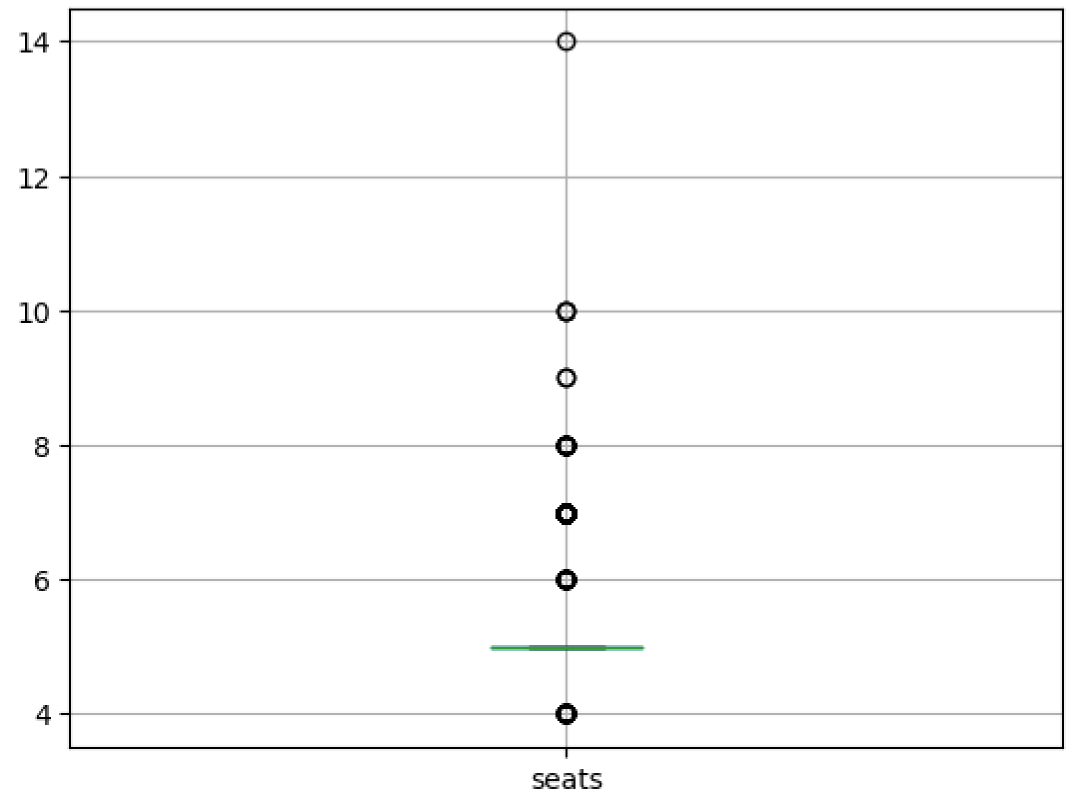


```
df_copy
```

	brand	model	owner	year	km_driven	fuel	seller_type	transmission	mileage	engine	max_power	seats	torque_value	torque_rpm_value	selling_price
0	Maruti	Swift Dzire Vdi	0	2014	145500	1	1	1	23.40	1248.0	74.00	5.0	1.902000e+06	2000.0	450000
1	Skoda	Rapid 1.5 Tdi Ambition	2	2014	120000	1	1	1	21.14	1498.0	103.52	5.0	2.501500e+10	2000.0	370000
2	Honda	City 2017-2020 Exi	4	2006	140000	3	1	1	17.70	1497.0	78.00	5.0	1.248519e+02	NaN	158000
3	Hyundai	I20 Sportz Diesel	0	2010	127000	1	1	1	23.00	1396.0	90.00	5.0	2.199157e+02	2250.0	225000
4	Maruti	Swift Vxi Bsiii	0	2007	120000	3	1	1	16.10	1298.0	88.20	5.0	1.132565e+02	NaN	130000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
8121	Maruti	Wagon R Vxi Bs Iv With Abs	2	2013	50000	3	1	1	18.90	998.0	67.10	5.0	9.035000e+05	3500.0	260000
8122	Hyundai	I20 Magna 1.4 Crdi	2	2014	80000	1	1	1	22.54	1396.0	88.73	5.0	2.197150e+02	2125.0	475000
8123	Hyundai	I20	0	2013	110000	3	1	1	18.50	1197.0	82.85	5.0	1.137400e+02	4000.0	320000

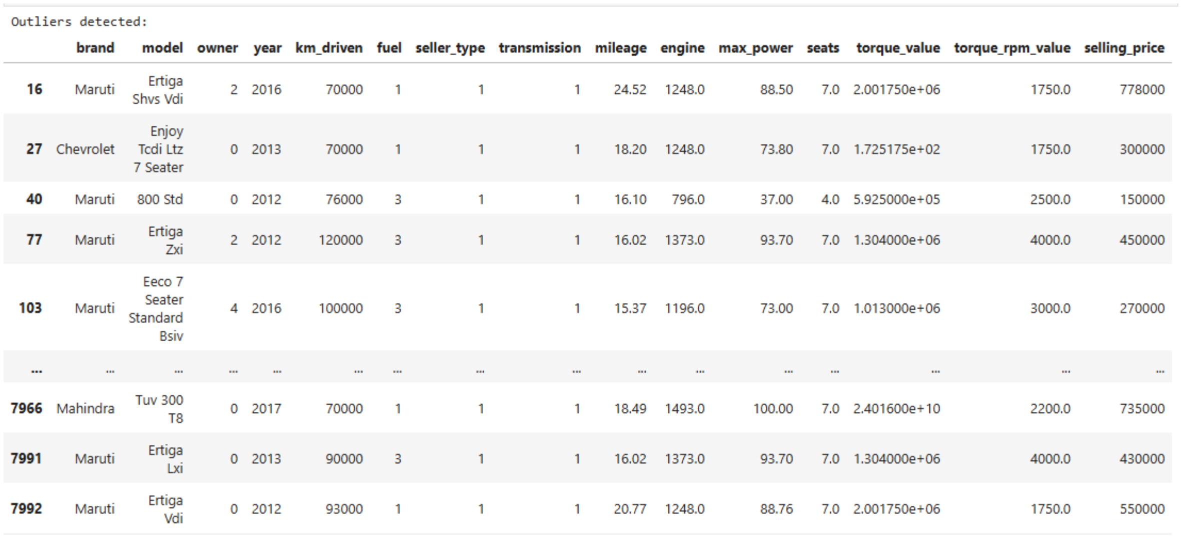
5105 rows × 15 columns

```python
import matplotlib.pyplot as plt
df_copy.boxplot(column='seats')
plt.show()
```



```
# Calculate Q1 (25th percentile) and Q3 (75th percentile)

Q1 = df_copy['seats'].quantile(0.25)

Q3 = df_copy['seats'].quantile(0.75)


# Calculate IQR (Interquartile Range)

IQR = Q3 - Q1


# Calculate the lower and upper bounds for outliers

lower_bound = Q1 - 1.5 * IQR

upper_bound = Q3 + 1.5 * IQR


# Find the outliers
```

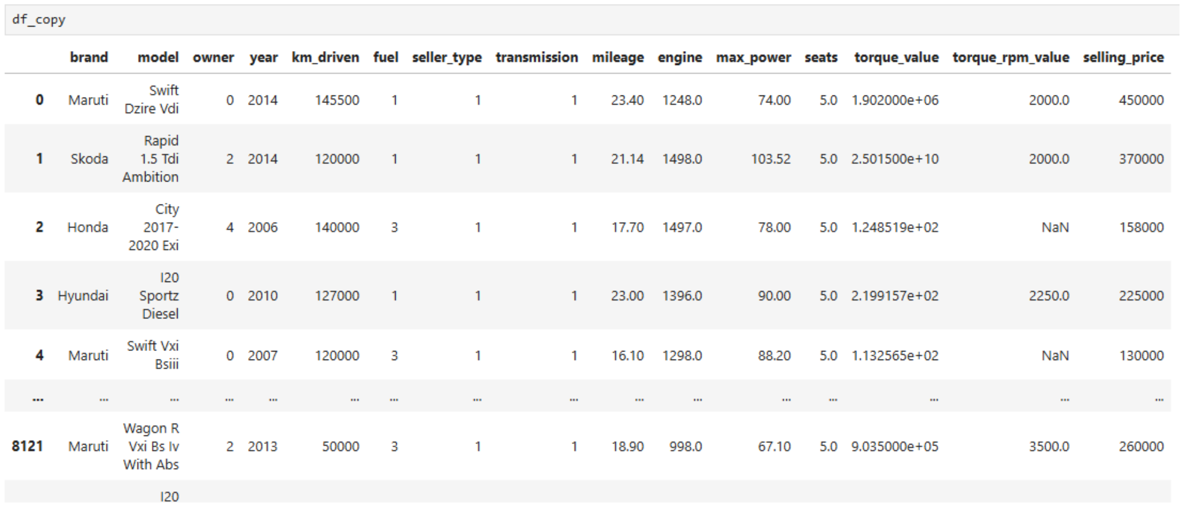
```
        outliers = df_copy[(df_copy['seats'] < lower_bound) | (df_copy['seats'] > upper_bound)]


        # Display the outliers

        print("Outliers detected:")

outliers
```



```
Outliers detected:
```

	brand	model	owner	year	km_driven	fuel	seller_type	transmission	mileage	engine	max_power	seats	torque_value	torque_rpm_value	selling_price
16	Maruti	Ertiga Shvs Vdi	2	2016	70000	1	1	1	24.52	1248.0	88.50	7.0	2.001750e+06	1750.0	778000
27	Chevrolet	Enjoy Tcdi Ltz 7 Seater	0	2013	70000	1	1	1	18.20	1248.0	73.80	7.0	1.725175e+02	1750.0	300000
40	Maruti	800 Std	0	2012	76000	3	1	1	16.10	796.0	37.00	4.0	5.925000e+05	2500.0	150000
77	Maruti	Ertiga Zxi	2	2012	120000	3	1	1	16.02	1373.0	93.70	7.0	1.304000e+06	4000.0	450000
103	Maruti	Eeco 7 Seater Standard Bsiv	4	2016	100000	3	1	1	15.37	1196.0	73.00	7.0	1.013000e+06	3000.0	270000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
7966	Mahindra	Tuv 300 T8	0	2017	70000	1	1	1	18.49	1493.0	100.00	7.0	2.401600e+10	2200.0	735000
7991	Maruti	Ertiga Lxi	0	2013	90000	3	1	1	16.02	1373.0	93.70	7.0	1.304000e+06	4000.0	430000
7992	Maruti	Ertiga Vdi	0	2012	93000	1	1	1	20.77	1248.0	88.76	7.0	2.001750e+06	1750.0	550000

```
        # Filter out the outliers

df_copy = df_copy[(df_copy['seats'] >= lower_bound) & (df_copy['seats'] <= upper_bound)]
```

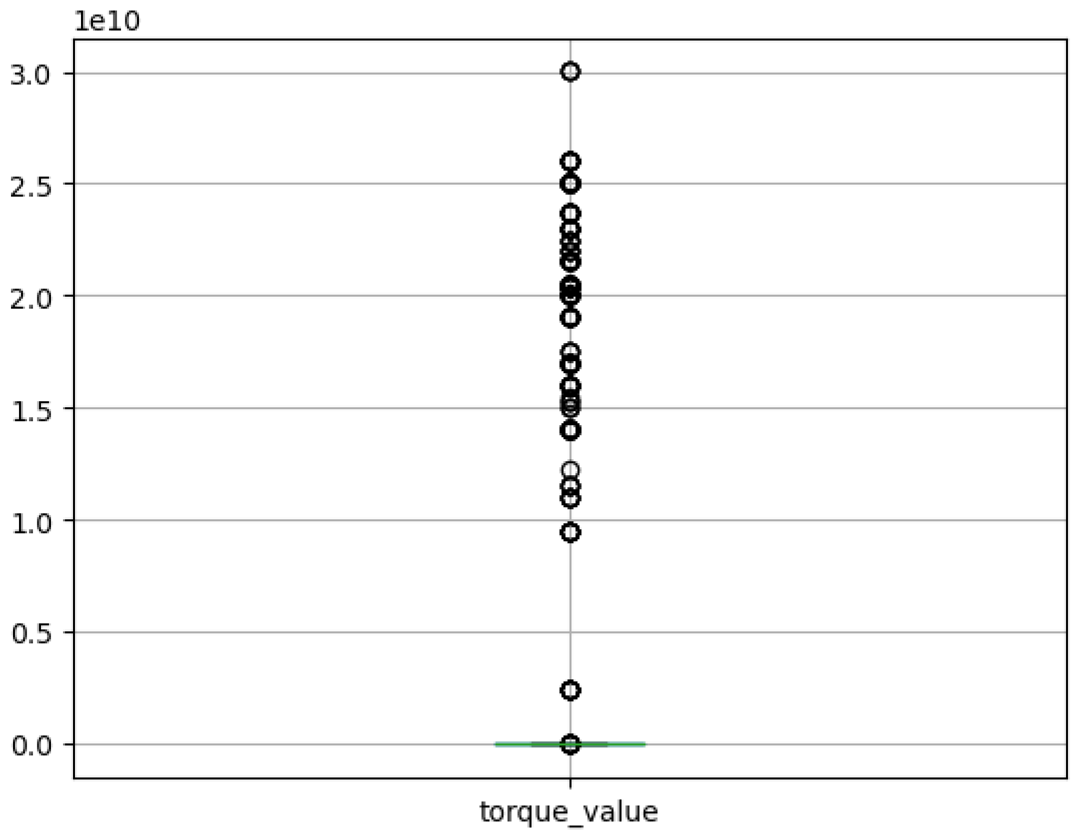


```
df_copy
```

	brand	model	owner	year	km_driven	fuel	seller_type	transmission	mileage	engine	max_power	seats	torque_value	torque_rpm_value	selling_price
0	Maruti	Swift Dzire Vdi	0	2014	145500	1	1	1	23.40	1248.0	74.00	5.0	1.902000e+06	2000.0	450000
1	Skoda	Rapid 1.5 Tdi Ambition	2	2014	120000	1	1	1	21.14	1498.0	103.52	5.0	2.501500e+10	2000.0	370000
2	Honda	City 2017-2020 Exi	4	2006	140000	3	1	1	17.70	1497.0	78.00	5.0	1.248519e+02	NaN	158000
3	Hyundai	I20 Sportz Diesel	0	2010	127000	1	1	1	23.00	1396.0	90.00	5.0	2.199157e+02	2250.0	225000
4	Maruti	Swift Vxi Bsiii	0	2007	120000	3	1	1	16.10	1298.0	88.20	5.0	1.132565e+02	NaN	130000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
8121	Maruti	Wagon R Vxi Bs Iv With Abs	2	2013	50000	3	1	1	18.90	998.0	67.10	5.0	9.035000e+05	3500.0	260000
		I20													

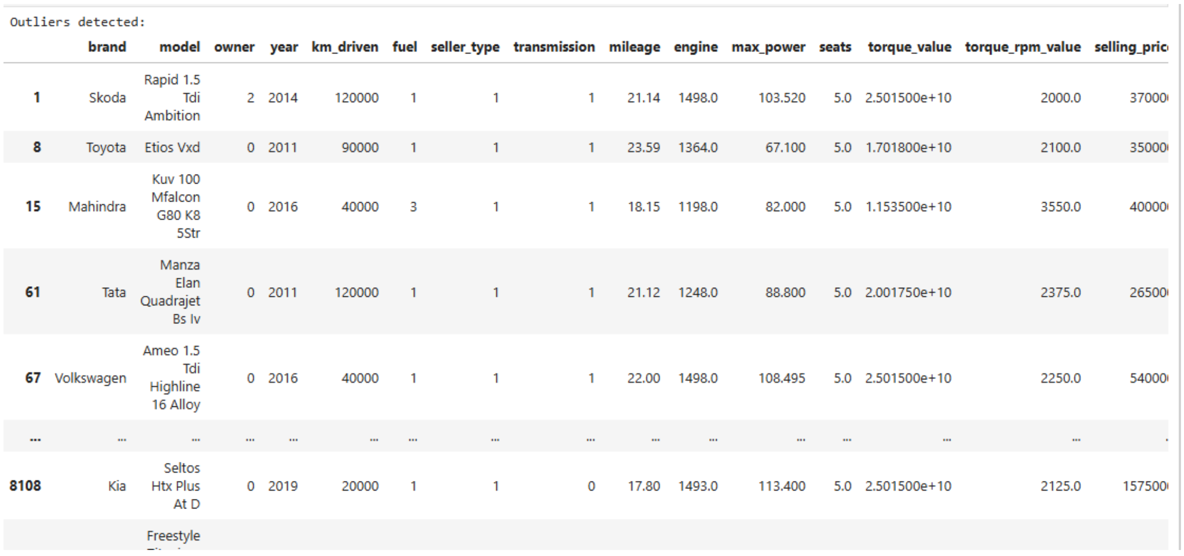
4669 rows × 15 columns

```
import matplotlib.pyplot as plt
df_copy.boxplot(column='torque_value')
plt.show()
```



```
# Calculate Q1 (25th percentile) and Q3 (75th percentile)

Q1 = df_copy['torque_value'].quantile(0.25)

Q3 = df_copy['torque_value'].quantile(0.75)


# Calculate IQR (Interquartile Range)

IQR = Q3 - Q1


# Calculate the lower and upper bounds for outliers

lower_bound = Q1 - 1.5 * IQR

upper_bound = Q3 + 1.5 * IQR
```

```
# Find the outliers

outliers = df_copy[(df_copy['torque_value'] < lower_bound) | (df_copy['torque_value'] > upper_bound)]


# Display the outliers

print("Outliers detected:")

outliers
```



```
Outliers detected:
```

	brand	model	owner	year	km_driven	fuel	seller_type	transmission	mileage	engine	max_power	seats	torque_value	torque_rpm_value	selling_pric
1	Skoda	Rapid 1.5 Tdi Ambition	2	2014	120000	1	1	1	21.14	1498.0	103.520	5.0	2.501500e+10	2000.0	37000
8	Toyota	Etios Vxd	0	2011	90000	1	1	1	23.59	1364.0	67.100	5.0	1.701800e+10	2100.0	35000
15	Mahindra	Kuv 100 Mfalcon G80 K8 5Str	0	2016	40000	3	1	1	18.15	1198.0	82.000	5.0	1.153500e+10	3550.0	40000
61	Tata	Manza Elan Quadrajet Bs Iv	0	2011	120000	1	1	1	21.12	1248.0	88.800	5.0	2.001750e+10	2375.0	26500
67	Volkswagen	Ameo 1.5 Tdi Highline 16 Alloy	0	2016	40000	1	1	1	22.00	1498.0	108.495	5.0	2.501500e+10	2250.0	54000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	.
8108	Kia	Seltos Htx Plus At D	0	2019	20000	1	1	0	17.80	1493.0	113.400	5.0	2.501500e+10	2125.0	157500
		Freestyle													

611 rows × 15 columns

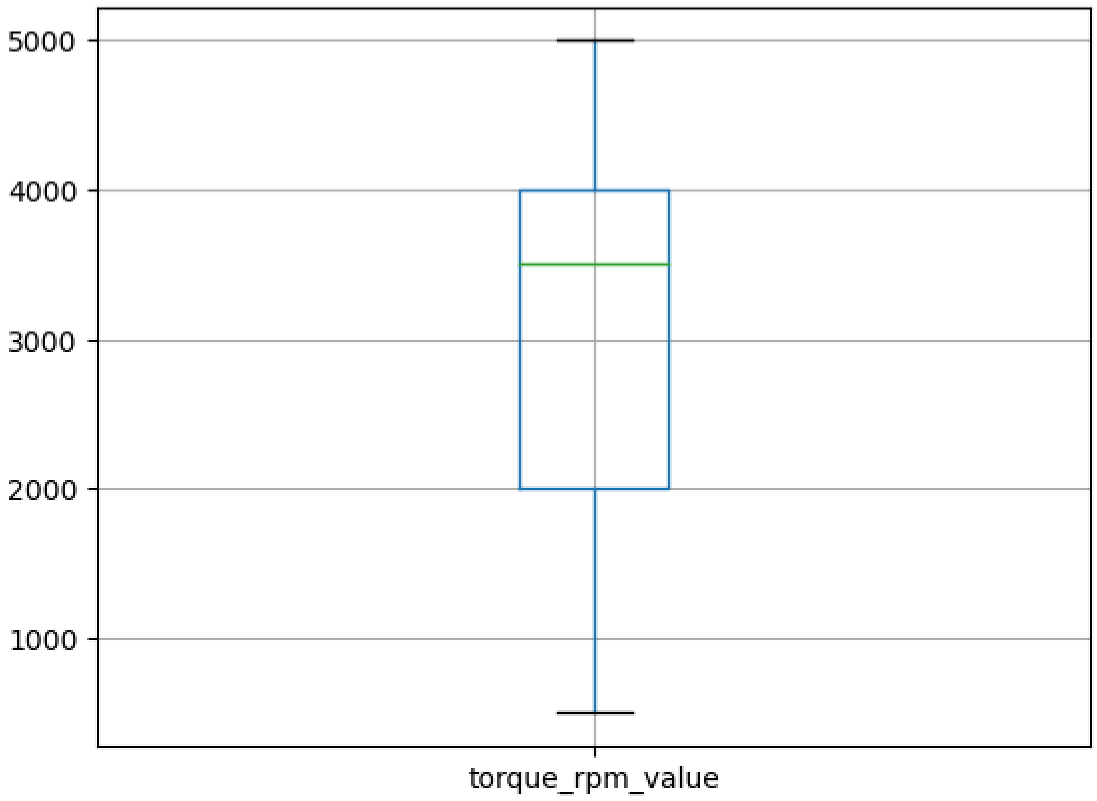
```
# Filter out the outliers

df_copy = df_copy[(df_copy['torque_value'] >= lower_bound) & (df_copy['torque_value'] <= upper_bound)]
```

```
df_copy
```

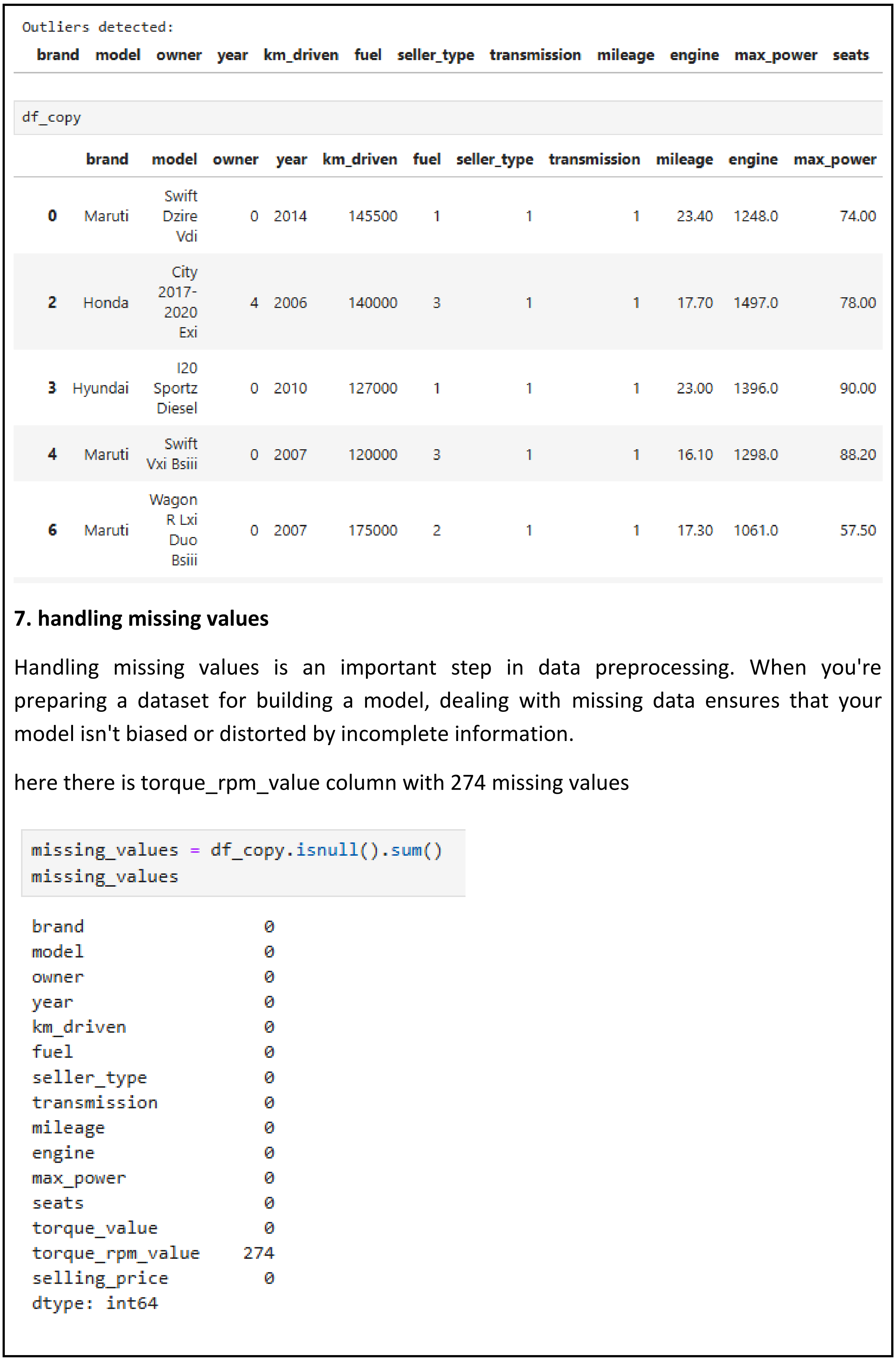
	brand	model	owner	year	km_driven	fuel	seller_type	transmission	mileage	engine	max_power	seats	torque_value
0	Maruti	Swift Dzire Vdi	0	2014	145500	1	1	1	23.40	1248.0	74.00	5.0	1.902000e+06
2	Honda	City 2017-2020 Exi	4	2006	140000	3	1	1	17.70	1497.0	78.00	5.0	1.248519e+02
3	Hyundai	I20 Sportz Diesel	0	2010	127000	1	1	1	23.00	1396.0	90.00	5.0	2.199157e+02
4	Maruti	Swift Vxi Bsiii	0	2007	120000	3	1	1	16.10	1298.0	88.20	5.0	1.132565e+02
6	Maruti	Wagon R Lxi Duo Bsiii	0	2007	175000	2	1	1	17.30	1061.0	57.50	5.0	7.695945e+01
...	...	...	...	...	...	...	...	...	...	...	...	...	...
8120	Hyundai	Santro Xing Gls	0	2008	191000	3	1	1	17.92	1086.0	62.10	5.0	9.613000e+01

torque_rpm_value has no outlier

```python
import matplotlib.pyplot as plt
df_copy.boxplot(column='torque_rpm_value')
plt.show()
```



```
        # Calculate Q1 (25th percentile) and Q3 (75th percentile)

        Q1 = df_copy['torque_rpm_value'].quantile(0.25)

        Q3 = df_copy['torque_rpm_value'].quantile(0.75)


        # Calculate IQR (Interquartile Range)

        IQR = Q3 - Q1


        # Calculate the lower and upper bounds for outliers

        lower_bound = Q1 - 1.5 * IQR

        upper_bound = Q3 + 1.5 * IQR


        # Find the outliers

        outliers = df_copy[(df_copy['torque_rpm_value'] < lower_bound) |
        (df_copy['torque_rpm_value'] > upper_bound)]


        # Display the outliers

        print("Outliers detected:")

outliers
```



```
Outliers detected:
```

brand	model	owner	year	km_driven	fuel	seller_type	transmission	mileage	engine	max_power	seats

```
df_copy
```

	brand	model	owner	year	km_driven	fuel	seller_type	transmission	mileage	engine	max_power
0	Maruti	Swift Dzire Vdi	0	2014	145500	1	1	1	23.40	1248.0	74.00
2	Honda	City 2017-2020 Exi	4	2006	140000	3	1	1	17.70	1497.0	78.00
3	Hyundai	I20 Sportz Diesel	0	2010	127000	1	1	1	23.00	1396.0	90.00
4	Maruti	Swift Vxi Bsiii	0	2007	120000	3	1	1	16.10	1298.0	88.20
6	Maruti	Wagon R Lxi Duo Bsiii	0	2007	175000	2	1	1	17.30	1061.0	57.50

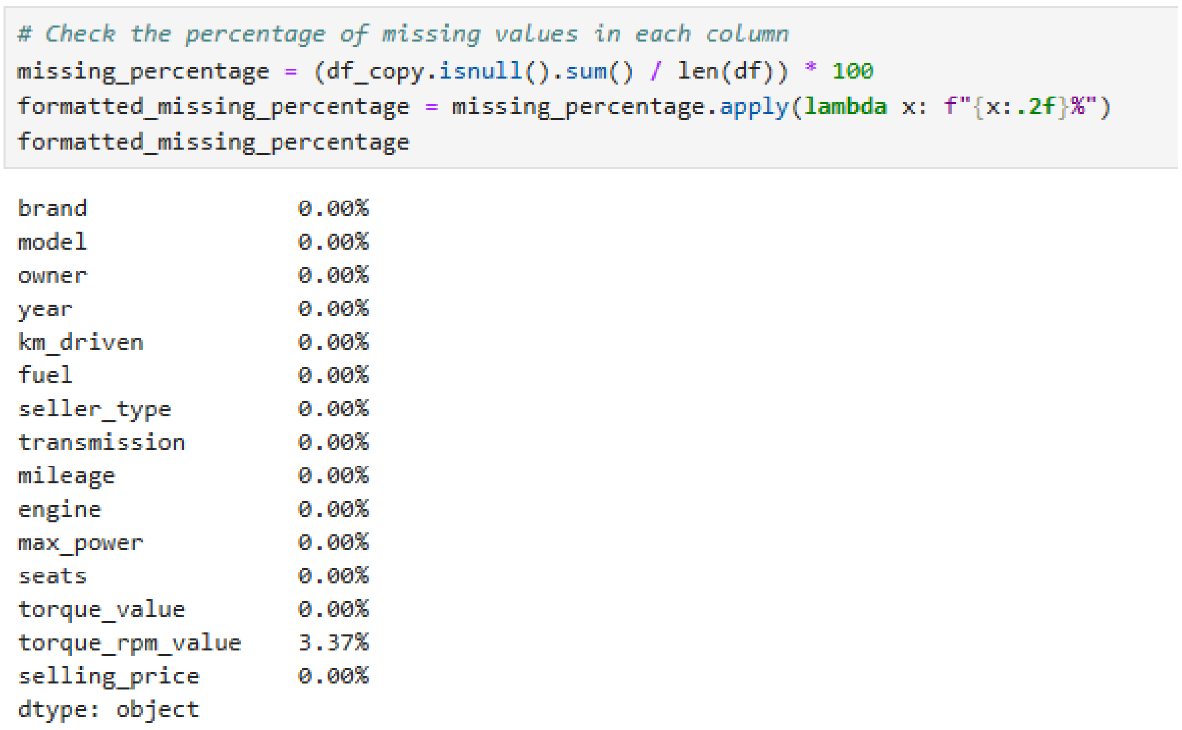
7. handling missing values

Handling missing values is an important step in data preprocessing. When you're preparing a dataset for building a model, dealing with missing data ensures that your model isn't biased or distorted by incomplete information.

here there is torque_rpm_value column with 274 missing values

```
missing_values = df_copy.isnull().sum()
missing_values
```

```
brand                 0
model                 0
owner                 0
year                  0
km_driven             0
fuel                  0
seller_type           0
transmission          0
mileage               0
engine                0
max_power             0
seats                 0
torque_value          0
torque_rpm_value    274
selling_price         0
dtype: int64
```

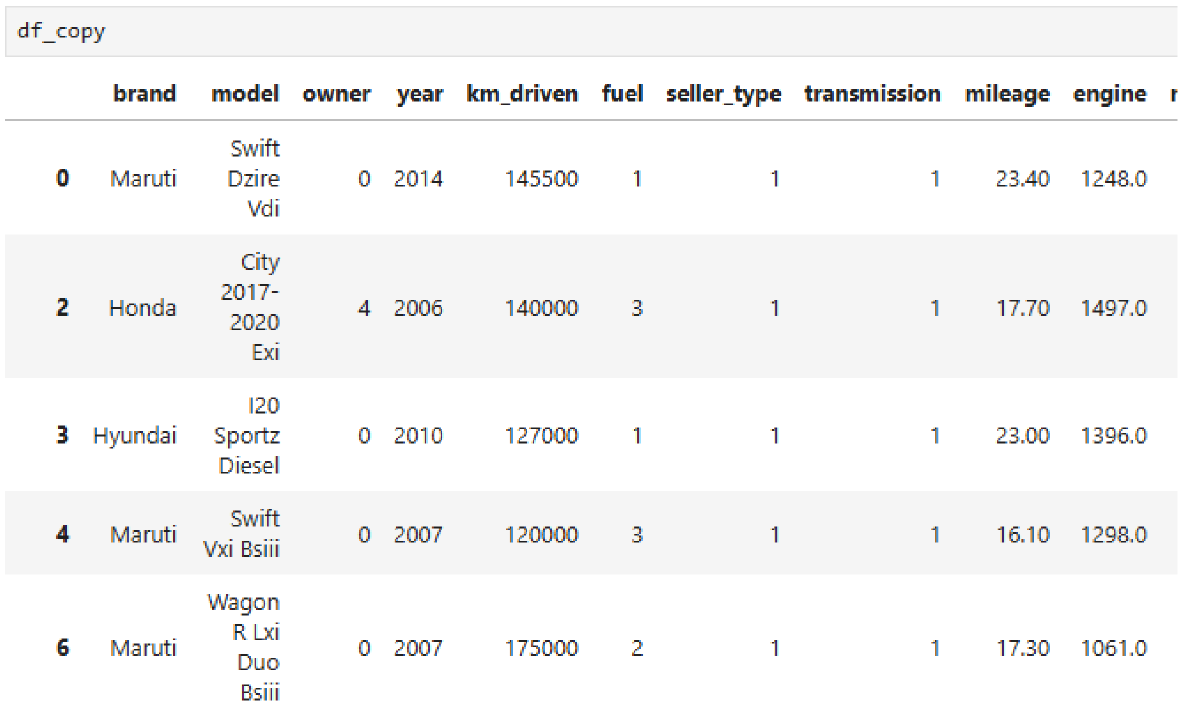


```
# Check the percentage of missing values in each column
missing_percentage = (df_copy.isnull().sum() / len(df)) * 100
formatted_missing_percentage = missing_percentage.apply(lambda x: f"{x:.2f}%")
formatted_missing_percentage
```

```
brand               0.00%
model               0.00%
owner               0.00%
year                0.00%
km_driven           0.00%
fuel                0.00%
seller_type         0.00%
transmission        0.00%
mileage             0.00%
engine              0.00%
max_power           0.00%
seats               0.00%
torque_value        0.00%
torque_rpm_value    3.37%
selling_price       0.00%
dtype: object
```

# Replace missing values with the mean of each column

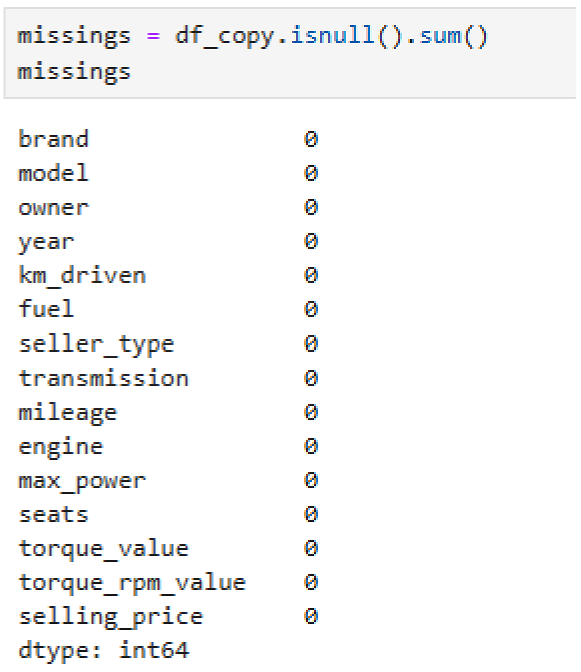
df.fillna(df.mean(), inplace=True)



```
df_copy
```

	brand	model	owner	year	km_driven	fuel	seller_type	transmission	mileage	engine
0	Maruti	Swift Dzire Vdi	0	2014	145500	1	1	1	23.40	1248.0
2	Honda	City 2017-2020 Exi	4	2006	140000	3	1	1	17.70	1497.0
3	Hyundai	I20 Sportz Diesel	0	2010	127000	1	1	1	23.00	1396.0
4	Maruti	Swift Vxi Bsiii	0	2007	120000	3	1	1	16.10	1298.0
6	Maruti	Wagon R Lxi Duo Bsiii	0	2007	175000	2	1	1	17.30	1061.0

Now there is no missing value remaining in our dataset



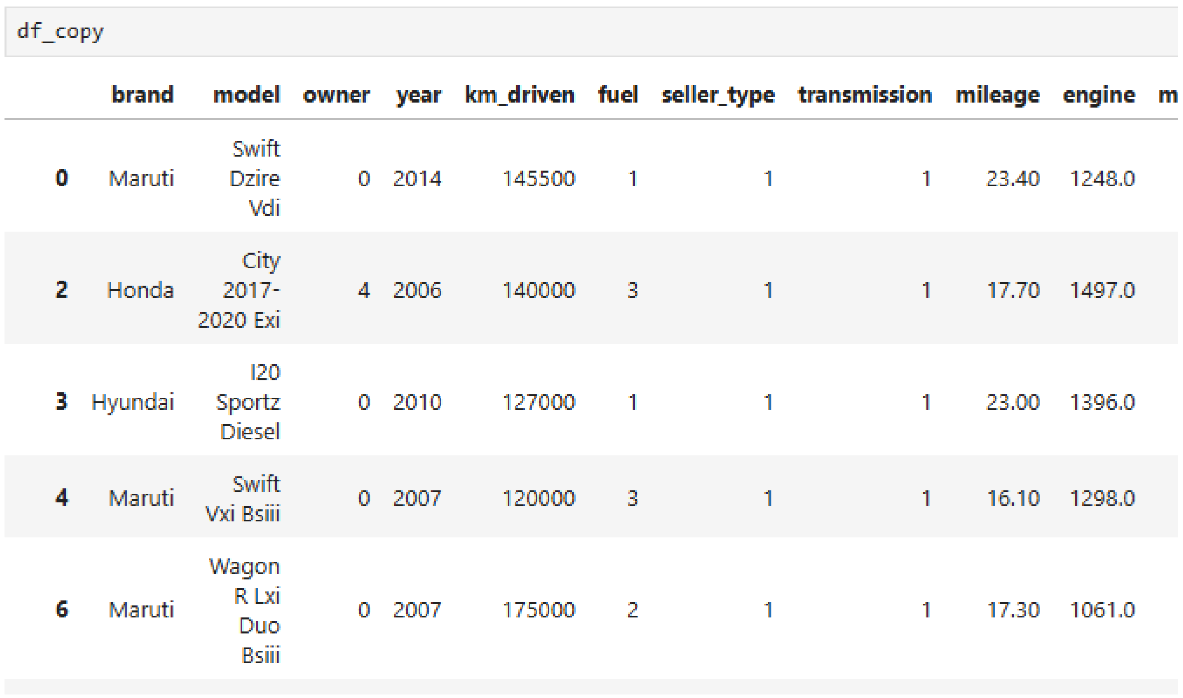
```
missings = df_copy.isnull().sum()
missings
```

```
brand               0
model               0
owner               0
year                0
km_driven           0
fuel                0
seller_type         0
transmission        0
mileage             0
engine              0
max_power           0
seats               0
torque_value        0
torque_rpm_value    0
selling_price       0
dtype: int64
```

8. Normalization, also known as min-max scaling, is a technique that scales the values of numeric features into a specified range, usually between 0 and 1. The idea is to ensure that each feature contributes equally to the model, especially when some features may have much larger values than others.

here we are going to apply it to torque_value, you can even apply it to all the numerical features we have

```
        from sklearn.preprocessing import MinMaxScaler

        # Initialize the MinMaxScaler

        scaler = MinMaxScaler()


        # Normalize the 'torque_value' column using .loc for safe assignment

df_copy.loc[:, 'torque_value'] = scaler.fit_transform(df_copy[['torque_value']])
```



```
df_copy
```

	brand	model	owner	year	km_driven	fuel	seller_type	transmission	mileage	engine	m
0	Maruti	Swift Dzire Vdi	0	2014	145500	1	1	1	23.40	1248.0	
2	Honda	City 2017-2020 Exi	4	2006	140000	3	1	1	17.70	1497.0	
3	Hyundai	I20 Sportz Diesel	0	2010	127000	1	1	1	23.00	1396.0	
4	Maruti	Swift Vxi Bsiii	0	2007	120000	3	1	1	16.10	1298.0	
6	Maruti	Wagon R Lxi Duo Bsiii	0	2007	175000	2	1	1	17.30	1061.0	

Saving the cleaned dataset to be used for machine learning model building.

```
df_copy.to_csv('car_cleaned_dataset.csv', index=False)
```

Practical Activity 1.5.3: Performing data normalization

Task:

1. Read the task bellow:
2. As full-stack, you are asked to go to the computer lab to performing data normalisation on cleaned data on 1.5.2.
3. Refers to provided key reading 1.5.2, perform the task described above.
4. Present your work to the trainer and whole class.

Key readings 1.5.3

Why Normalize Data?

Normalization is important in machine learning because:

It scales numerical values to a common range, preventing large values from dominating smaller ones.

It improves the convergence of gradient-based optimization methods.

It ensures features contribute equally to model learning.

To perform data normalization on the cleaned data stored in car_cleaned_dataset.csv, you can follow the steps below using the pandas and sklearn libraries.

Here's how to do it:

Step 1: Load the Cleaned Data

```
import pandas as pd
# Load the cleaned dataset
car_data = pd.read_csv('car_cleaned_dataset.csv')
```

Step 2: Select Numerical Columns for Normalization

```
# Select numerical columns
numerical_cols = ['selling_price', 'km_driven', 'mileage', 'engine', 'max-power',
'torque', 'seats', 'age']
```

Step 3: Apply Normalization Techniques

1. **Min-Max Scaling (Normalization)**

```
from sklearn.preprocessing import MinMaxScaler
# Initialize MinMaxScaler
min_max_scaler = MinMaxScaler()
# Apply Min-Max Scaling
car_data[numerical_cols]                                                  =
min_max_scaler.fit_transform(car_data[numerical_cols])
```

2. **Z-score Standardization**

```
from sklearn.preprocessing import StandardScaler
# Initialize StandardScaler
standard_scaler = StandardScaler()
# Apply Z-score Standardization
car_data[numerical_cols]                                                    =
standard_scaler.fit_transform(car_data[numerical_cols])
```

3. **Robust Scaling**

```
from sklearn.preprocessing import RobustScaler
# Initialize RobustScaler
robust_scaler = RobustScaler()
# Apply Robust Scaling
car_data[numerical_cols] = robust_scaler.fit_transform(car_data[numerical_cols])
```

Step 4: Save the Normalized Data in Another File

```
# Save the normalized dataset
car_data.to_csv('car_normalized_data.csv', index=False)
```

Summary of the Steps

- ➢ Load the cleaned dataset.
- ➢ Identify the numerical columns that require normalization.
- ➢ Apply three normalization techniques (Min-Max Scaling, Z-score Standardization, and Robust Scaling).
- ➢ Save the normalized dataset to a new CSV file named car_normalized_data.csv.

Which Normalization Method Should You Use among the followings:

- ➢ Min-Max Scaling: Use when you need values between 0 and 1 (e.g., neural networks).
- ➢ Z-score Standardization: Use when data is normally distributed.
- ➢ Robust Scaling: Use when there are outliers.

Practical Activity 1.5.4: Performing data transformation techniques

Task:

1. Read the task bellow:
2. As full-stack, you are asked to go to the computer lab to performing data transformation of normalised data on 1.5.3 in dataset.
3. Refers to provided key reading 1.5.2, perform the task described above.
4. Present your work to the trainer and whole class.

Key readings 1.5.4: Performing data transformation techniques

To perform data transformation on the normalized data stored in car_normalized_data.csv, follow the steps below. This process includes handling missing values, detecting outliers, normalizing and standardizing features, encoding categorical variables, engineering new features, and transforming text data. Here's how to implement it:

Step 1: Load the Normalized Dataset

```
import pandas as pd
# Load the normalized dataset
car_data = pd.read_csv('car_normalized_data.csv')
```

Step 2: Handle Missing Values

```
# Check for missing values
missing_values = car_data.isnull().sum()
# Fill or drop missing values as appropriate
car_data['mileage'].fillna(car_data['mileage'].median(), inplace=True)
car_data.dropna(subset=['name'], inplace=True)  # Dropping rows with critical
missing values
```

Step 3: Detect and Remove Outliers

```
# Using IQR method to remove outliers
Q1 = car_data['selling_price'].quantile(0.25)
Q3 = car_data['selling_price'].quantile(0.75)
IQR = Q3 - Q1
# Remove outliers
car_data = car_data[(car_data['selling_price'] >= (Q1 - 1.5 * IQR)) &
(car_data['selling_price'] <= (Q3 + 1.5 * IQR))]
```

Step 4: Normalize and Standardize Numerical Features

If the data is already normalized, you can skip this step. However, if you want to reapply normalization or standardization, you can do so using the previously mentioned methods.
Here’s an example using Min-Max Scaling:

```
from sklearn.preprocessing import MinMaxScaler
# Initialize MinMaxScaler
min_max_scaler = MinMaxScaler()
# Apply Min-Max Scaling again if needed
car_data[numerical_cols]=
min_max_scaler.fit_transform(car_data[numerical_cols])
```

Step 5: Encode Categorical Variables

```
# One-hot encoding for categorical variables
car_data = pd.get_dummies(car_data, columns=['fuel', 'seller_type',
'transmission'], drop_first=True)
```

Step 6: Engineer New Features

```
# Example: Create new feature for age
car_data['age'] = 2023 - car_data['year']  # Assuming the current year is 2023
```

Step 7: Transform Text Data

```
# Clean text data if necessary (e.g., for the 'name' column)
car_data['name'] = car_data['name'].str.lower().str.strip()
```

Save the Transformed Data

```
# Save the transformed dataset
car_data.to_csv('car_transformed_data.csv', index=False)
```

Summary of Steps:

- ➢ Load the normalized dataset.
- ➢ Handle missing values appropriately.
- ➢ Detect and remove outliers using the IQR method.
- ➢ Normalize and standardize numerical features if necessary.
- ➢ Encode categorical variables with one-hot encoding.
- ➢ Engineer new features based on existing data.
- ➢ Transform text data to ensure consistency.

Points to Remember

- Data cleaning is the process of detecting, correcting, or removing inaccurate, incomplete, or inconsistent data to improve its quality.
- It ensures datasets are accurate, complete, and suitable for analysis, enhancing the reliability of statistical models and machine learning algorithms.
- The key steps include handling missing values through imputation or removal, removing duplicates, standardizing data formats, detecting and treating outliers, encoding categorical variables, and applying feature scaling.
- Text data cleaning, ensuring consistent data types, validating integrity, and documenting changes are essential parts of the process.
- Tools like Pandas, NumPy, Scikit-learn, and visualization libraries like Matplotlib help streamline data cleaning workflows.
- High-quality data possesses several characteristics that improve machine learning performance.
- Accuracy ensures correctness, while completeness minimizes missing values.
- Consistency maintains uniform data representation across sources, and relevance ensures only meaningful data is included.
- Timeliness keeps data up-to-date, and uniformity enforces consistent formats.
- Granularity determines the right level of detail, and diversity ensures representation across different conditions.
- Non-redundancy avoids duplication, and scalability ensures efficient processing.
- Structured, well-documented data with clear validity enhances reliability, leading to better model performance and decision-making.
- Inconsistencies in data arise from variations in date formats, measurement units, typos, and missing values, which can compromise data quality. Cleaning these inconsistencies improves data reliability, enhances model performance, reduces costs, facilitates better decision-making, and ensures compliance with regulatory standards. Techniques include handling missing values through imputation or removal, detecting and eliminating duplicates, standardizing formats, identifying and treating outliers, encoding categorical variables, applying feature scaling, and validating data integrity.
- Common techniques include Min-Max normalization, which scales data between 0 and 1, Z-score normalization, which centers data around the mean with a standard deviation of 1, decimal scaling, which adjusts values based on the highest absolute value, and MaxAbs scaling, which normalizes by the maximum absolute value. Choosing the appropriate normalization method depends on the dataset and machine learning algorithm used.

- **Techniques and steps for data cleaning:**

  Step 1: Load the Dataset
  Step 2: Handling Missing Values
  Step 3: Remove Duplicates
  Step 4: Correct Data Types
  Step 5: Handle Outliers
  Step 7: Encode Categorical Variables
  Step 8: Feature Engineering
  Step 9: Text Data Processing
  Step 10: Final Data Check

- Min-Max Scaling: Use when you need values between 0 and 1 (e.g., neural networks).
- Z-score Standardization: Use when data is normally distributed.
- Robust Scaling: Use when there are outliers.
- Once you want to perform data transformation you follow the following steps.
    1. Load the cleaned dataset
    2. Handle missing values
    3. Detect and remove outliers
    4. Normalize and standardize numerical features
    5. Encode categorical variables
    6. Engineer new features
    7. Transform text data

Application of learning 1.5.

Sarmotor Rwanda has corrected data related to cars in year 2024, as full-stack, you are hired as machine learning expert responsible for performing the followings:

a) Performing data cleaning for inconsistencies rectification

b) Perform data normalisation

c) Perform data transformation

For each activity you have to store the file containing the information of the performed action in independent file.

Learning outcome 1 end assessment

Theoretical assessment

I. **Read the following statements concerning machine learning and circle the letter corresponding to the right answer**

1. The primary goal of machine learning is:
    a) To develop artificial intelligence
    b) To automate complex data analysis
    c) To create software applications
    d) To mimic human intelligence

2. The following are the types of machine learning, EXCEPT:

    a) Supervised Learning
    b) Unsupervised Learning
    c) Reinforcement Learning
    d) Pattern Learning

3. The first step in the machine learning life cycle is:

    a) Data Collection
    b) Model Evaluation
    c) Data Visualization
    d) Model Deployment

4. Machine learning method uses labeled data to train models is:

    a) Unsupervised Learning
    b) Supervised Learning
    c) Semi-supervised Learning
    d) Reinforcement Learning

5. The main difference between Artificial Intelligence, Machine Learning is:

    a) AI is a subset of ML
    b) ML is a subset of AI
    c) ML is superior to AI
    d) None of the above

6. The tool used for creating interactive data visualizations is:

    a) Matplotlib
    b) Seaborn
    c) Plotly
    d) Power BI

7. The following is a data cleaning technique.

    a) Data Normalization
    b) Data Transformation
    c) Data Aggregation
    d) Removing duplicates

8. "V" in Big Data refers to the speed at which data is generated is:

    a) Volume
    b) Variety
    c) Velocity
    d) Veracity

9. A key characteristic of structured data is:

    a) No predefined schema

b) Defined schema and organization
c) Text, image, and video format
d) Generated by social media

10. The following are examples of unstructured data except:

a) Images
b) Audio recordings
c) Excel spreadsheet
d) Videos

II. Select the correct answer from the given choices.

1. _____________ data refers to information that is organized in rows and columns in databases or spreadsheets.
Choices:

a) Unstructured data
b) Semi-structured data
c) Structured data
d) Transactional data

2. _____________ is a popular data visualization library in Python for creating plots like line charts and histograms.
Choices:

a) Matplotlib
b) Seaborn
c) Power BI
d) Tableau

3. _____________ learning requires labeled data to make accurate predictions.
Choices:

a) Supervised Learning
b) Unsupervised Learning
c) Reinforcement Learning
d) Semi-supervised Learning

4. The _____________ plot is ideal for showing the distribution of numerical data by representing their quartiles and potential outliers.
Choices:

a) Scatter Plot
b) Box Plot
c) Bar Chart
d) Line Plot

5. Data **_____________** is essential to ensure that machine learning models do not give too much importance to certain features with larger scales.
   Choices:

   a) Normalization
   b) Aggregation
   c) Transformation
   d) Cleaning

III. Read the following statements concerning machine learning and State whether the following statements are true or false.

1. Supervised learning involves training a model using labeled data.
2. In unsupervised learning, the system tries to find patterns in data without being explicitly told what to look for.
3. Machine learning is a subset of artificial intelligence.
4. Data visualization is not necessary for interpreting machine learning models.
5. The 6 V's of Big Data are Volume, Variety, Velocity, Veracity, Value, and Variability.

IV. Match the terms related to data collection and acquisition in column A with their corresponding meaning in the column B and write the answer in the appropriate place.

Answers	**Column A**	**Column B**
1……	1. Big Data	A) The process of converting raw data into a machine-readable format.
2……	2.Data visualization	B) Ensuring that data is accurate, consistent, and free of errors.
3…..	3.Semi-	C) Large datasets that cannot be processed using traditional

	Structured Data	data processing techniques.
4….	4. Data Cleaning	D) A process used to scale data to a standard range, usually between 0 and 1.
5……	5.Data Transformation	E) Data that has some organizational properties but not as structured as databases.
6….	6.Data Normalization	
7…….	7.Data interpretation	

Practical assessment

As a trainee in machine learning, you are requested by your school to go in computer lab to prepare a robust machine learning environment to measure the student performance, by performing the following tasks:

1) Preparing Machine Learning environment by Installing Python, installing tools and test Environment
2) Collect the student performance dataset
3) Use Types of Data Visualization like Scatter Plots, Line Plots, Bar Charts, Histograms, Box Plots and Heat map plot
4) Choose the best practices for data visualization and interpretation results
5) Perform data cleaning for unnecessary data in dataset
6) Apply data normalization and transformation for preparing dataset.

END

References

Brownlee, J. (2016). Master Machine Learning Algorithms - Discover how they work.

Eric, H., Sinayobye, O., Masabo, E., Ufitinema, C., Murangira, T., Rwibasira, P., . . . Gaurav, B. (2023). An Intelligent Rwandan Arabica Dataset. Technologies, 23.

Hitimana, E., Richard, M., Bajpai, G., Louis, S., & Kayalvizhi, J. (2021). Implementation of IoT Framework with Data Analysis Using Deep Learning Methods for Occupancy Prediction in a Building. Future InternetSystem-Based Coffee Plant Leaf Disease Recognition Using Deep Learning Techniques.

Learning Outcome 2: Develop Machine Learning Model.

Indicative contents
2.1 Description of Machine learning algorithms and applications
2.2 Selection of machine learning algorithm
2.3. Train machine learning model
2.4. Evaluation of machine learning model
2.5 Tuning Hyperparameters

Key Competencies for Learning Outcome 2: Develop Machine Learning Model

Knowledge	Skills	Attitudes
● Description Machine learning algorithms and their applications ● Selecting machine learning algorithm	● Analysing type of data based on dataset. ● Training machine learning model based on dataset ● Evaluating machine learning model based on provided data ● Tuning Hyperparameters based on performance ● Resolving Potential Bias	● Team work ability while training a model ● Being critical thinker in evaluating model. ● Problem-Solving in resolving potential bias. ● Having good Collaboration and Communication

Duration:50 hrs

Learning outcome 2 objectives:

By the end of the learning outcome, the trainees will be able to:

1. Describe correctly Machine learning algorithms and their applications based on model.
2. Choose properly machine learning algorithm based on model
3. Select correctly machine learning algorithm based on model
4. Analyse properly the types of data based on the dataset
5. Train properly machine learning model based on dataset
6. Evaluate correctly machine learning model based on provided data
7. Tune correctly Hyperparameters based on performance
8. Resolve correctly Potential Bias based on test data

Resources

Equipment	**Tools**	**Materials**
● Computer ● Projector Storage Devices	● Datasets ● Python Distribution (CPython, Anaconda), ● Python IDEs (Jupyter Notebook, PyCharm, Visual Studio Code, or Spyder) ● Machine Learning Libraries (NumPy, Pandas, Scikit-Learn, TensorFlow or PyTorch, Matplotlib, Seaborn, Keras, SciPy) ● Cloud platform	● Internet

Indicative content 2.1: Description of Machine Learning Algorithms and Applications

Duration: 6hrs

Theoretical Activity 2.1.1: Description of supervised machine Learning algorithm and application

Tasks:

1: Answer the following questions:

I. Explain the following supervised machine learning algorithms

a) Supervised Learning

b) Linear Regression

c) Logistic Regression

d) Decision Trees

e) Random Forest

f) Support Vector Machine (SVM)

g) k-Nearest Neighbours (KNN)

h) Naive Bayes

II. List down 5 application of supervised machine learning algorithms

2: Provide the answer for the asked questions and write them on papers.

3: Present the findings/answers to the whole class and trainer

4: For more clarification, read the key readings 2.1.1.

Key readings 2.1.1.: Description of Machine learning algorithms and applications

Machine learning algorithms are the backbone of artificial intelligence, enabling computers to learn from data, make predictions, and improve their performance without explicit programming.

- **Supervised machine Learning algorithm and application**

Supervised machine learning algorithms are trained on a labelled dataset, where each data point has an associated output or target variable. The algorithm learns to map inputs to outputs, enabling it to make predictions on new, unseen data.

✓ **Linear Regression**

Linear regression is a supervised machine learning technique used for predicting and forecasting values that fall within a continuous range, such as sales numbers or housing prices. It is a technique derived from statistics and is commonly used to establish a relationship between an input variable (X) and an output variable (Y) that can be represented by a straight line.

In simple terms, linear regression takes a set of data points with known input and output values and finds the line that best fits those points. This line, known as the "**regression line**," serves as a predictive model. By using this line, we can estimate or predict the output value (Y) for a given input value (X).

Linear regression is primarily used for **predictive modelling** rather than categorization. It is useful when we want to understand how changes in the input variable affect the output variable. By analysing the slope and intercept of the regression line, we can gain insights into the relationship between the variables and make predictions based on this understanding.

Y = bx+ c.

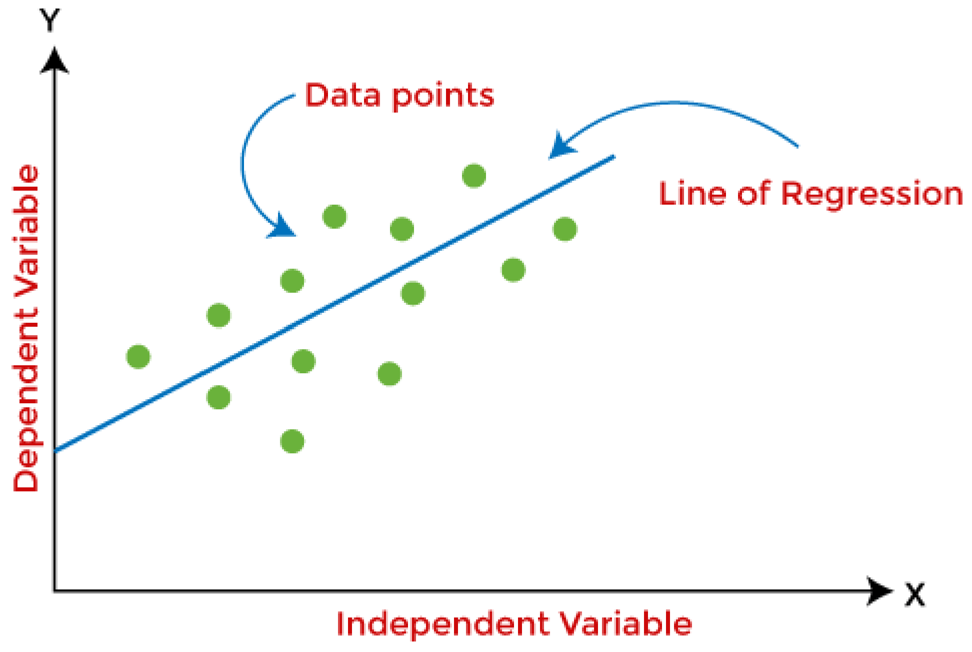


Figure 42 linear regression

The main purpose of the linear regression model is to find the best fit line that best fits the data points.

Types of Linear Regression in Machine Learning

Linear regression is a fundamental supervised learning algorithm used to model the relationship between a dependent variable and one or more independent variables. Here are the main types:

1. Simple Linear Regression:

This is the most basic form, involving a single independent variable (predictor) and a single dependent variable (response).

- **Equation:** y = b0 + b1*x

    y: Dependent variable

    x: Independent variable

    b0: Intercept (value of y when x is 0)

    b1: Slope (rate of change of y with respect to x)

2. Multiple Linear Regression:

It Extends simple linear regression to include multiple independent variables.

- **Equation:** y = b0 + b1*x1 + b2*x2 + ... + bn*xn

    y: Dependent variable

    x1, x2, ..., xn: Independent variables

    b0: Intercept

    b1, b2, ..., bn: Coefficients for each independent variable

✓ **Logistic Regression**

Logistic Regression is a statistical model used to predict a binary outcome (i.e., 0 or 1, true or false, success or failure) based on one or more predictor variables. Logistic regression is a supervised learning algorithm primarily used for classification problems.

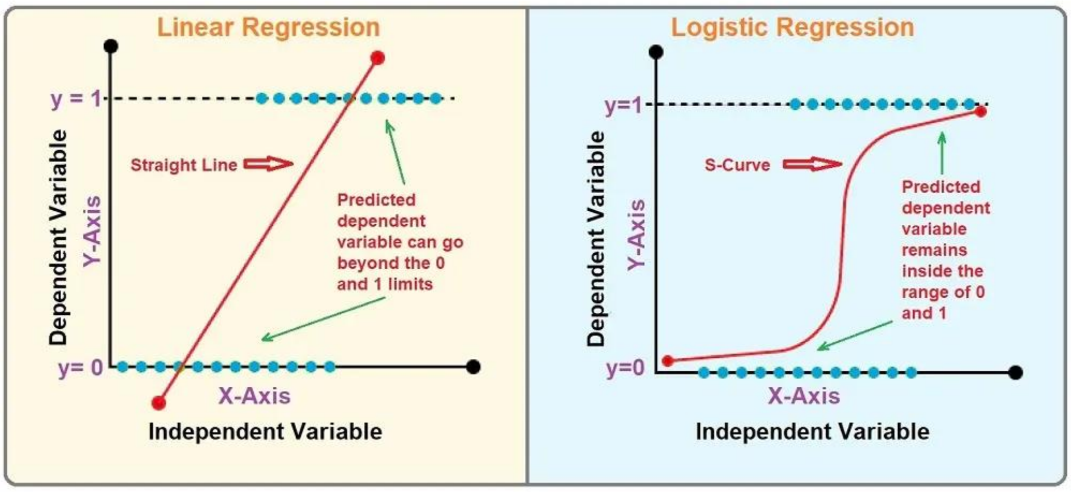


Figure 43 Linear and logistic regression

It's prediction can be like:

- ❖ Yes or No
- ❖ True or False
- ❖ Like or Dislike

- ➢ It uses clues (called features) to figure out the probability of one of these two outcomes.

- ➢ Remember when we tossed a coin in probability, we said it will give us either a tail or head, here the output is binary like the output we have here in logistic that is why we base on the probability range of (0 – 1)

For example:

- ❖ Will they say yes to going on a date?
- ❖ Will the text message get a reply?
- ❖ Is this a love match?

- ➢ Imagine you're trying to guess if your crush likes you back. You observe certain things they do—how they smile at you, how often they talk to you, or if they laugh at your jokes. Based on these observations, you try to predict: **Do they like me? Yes or No?**

This guessing game is a lot like **Logistic Regression**, a type of mathematical model that helps us predict one of two possible outcomes based on various clues or features.

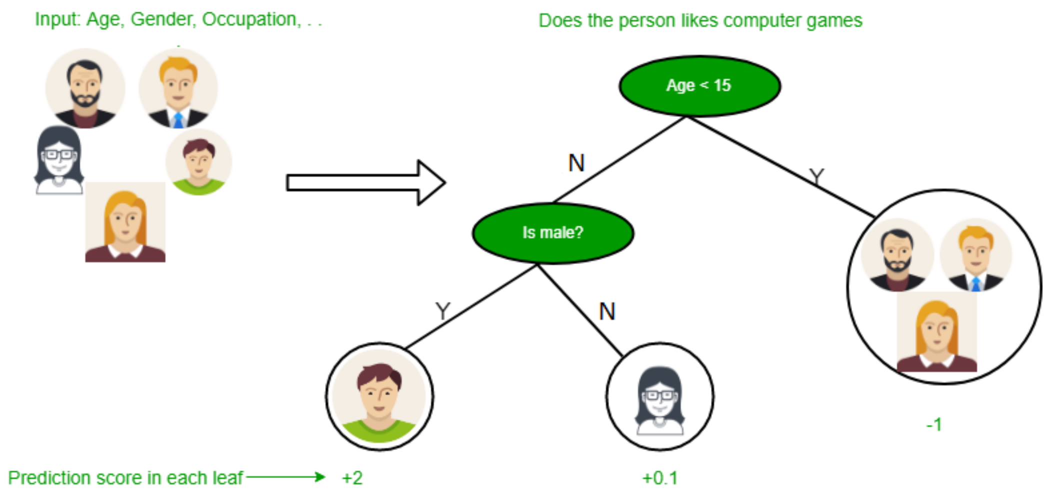
✓ Decision Trees

Decision Tree is a supervised machine learning model used for classification and regression tasks. It splits the data into subsets based on the feature values and makes predictions by following the path from the root of the tree to a leaf node.

Decision trees are a fundamental supervised learning technique widely used in machine learning for both classification and regression tasks.

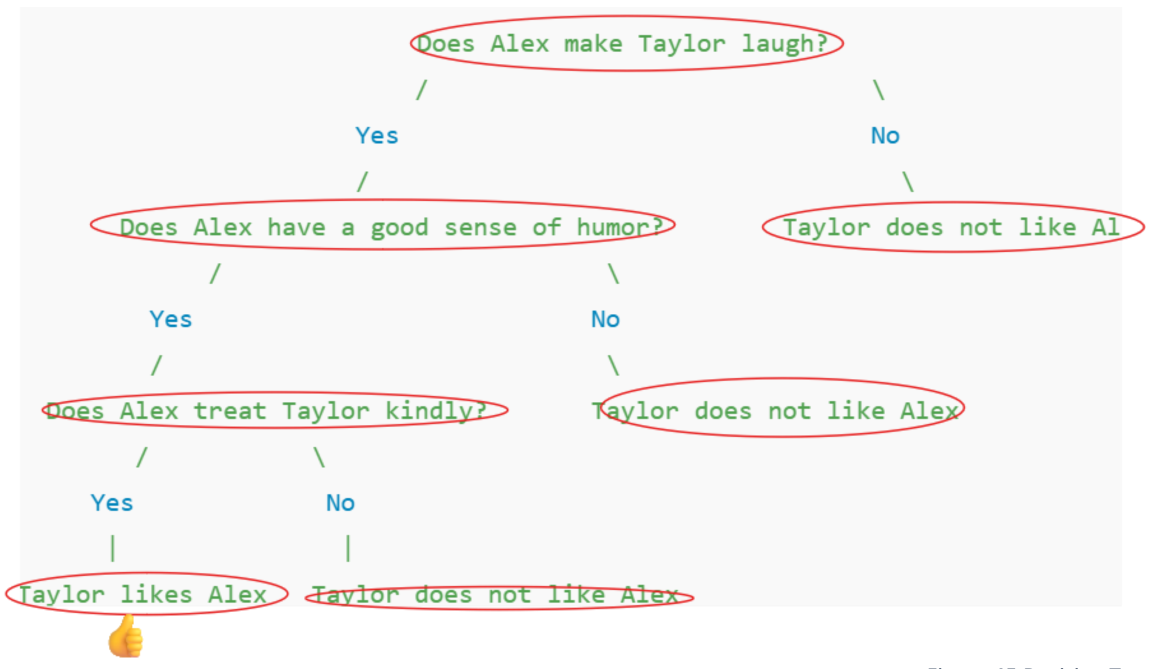
Components of Decision Trees

- ❖ **Root Node**: The top node that represents the entire dataset, which is then split into sub-nodes.
- ❖ **Internal Nodes**: Nodes that test an attribute and split the dataset based on the outcome of that test.
- ❖ **Leaf Nodes**: Terminal nodes that represent final outcomes or predictions.
- ❖ **Branches**: Connections between nodes representing the flow of decisions based on attribute values.


Input: Age, Gender, Occupation, . .
Does the person likes computer games
Age < 15
N
Y
Is male?
Y
N
-1
Prediction score in each leaf
+2
+0.1
Figure 44 Decision tree

Let's say Taylor is trying to decide if they like Alex or not. Taylor might ask a series of questions to help with this decision, like:

- Does Alex make Taylor laugh?
- Does Alex have a good sense of humour?
- Does Alex treat Taylor kindly?



```
                  Does Alex make Taylor laugh?
                 /                              \
               Yes                               No
              /                                    \
   Does Alex have a good sense of humor?     Taylor does not like Al
       /                            \
     Yes                             No
     /                                \
Does Alex treat Taylor kindly?      Taylor does not like Alex
    /              \
  Yes               No
   |                 |
Taylor likes Alex  Taylor does not like Alex
   👍
```

Figure 45 Decision Tree

Each of these questions splits the data in a way that helps Taylor decide whether they like Alex or not.

✓ **Random Forest**

It is a powerful and widely used machine learning algorithm. It's essentially an ensemble of Decision Trees, where multiple trees are built and combined to improve accuracy and reduce overfitting. The idea behind Random Forest is to take advantage of the strength of individual Decision Trees while minimizing their weaknesses.

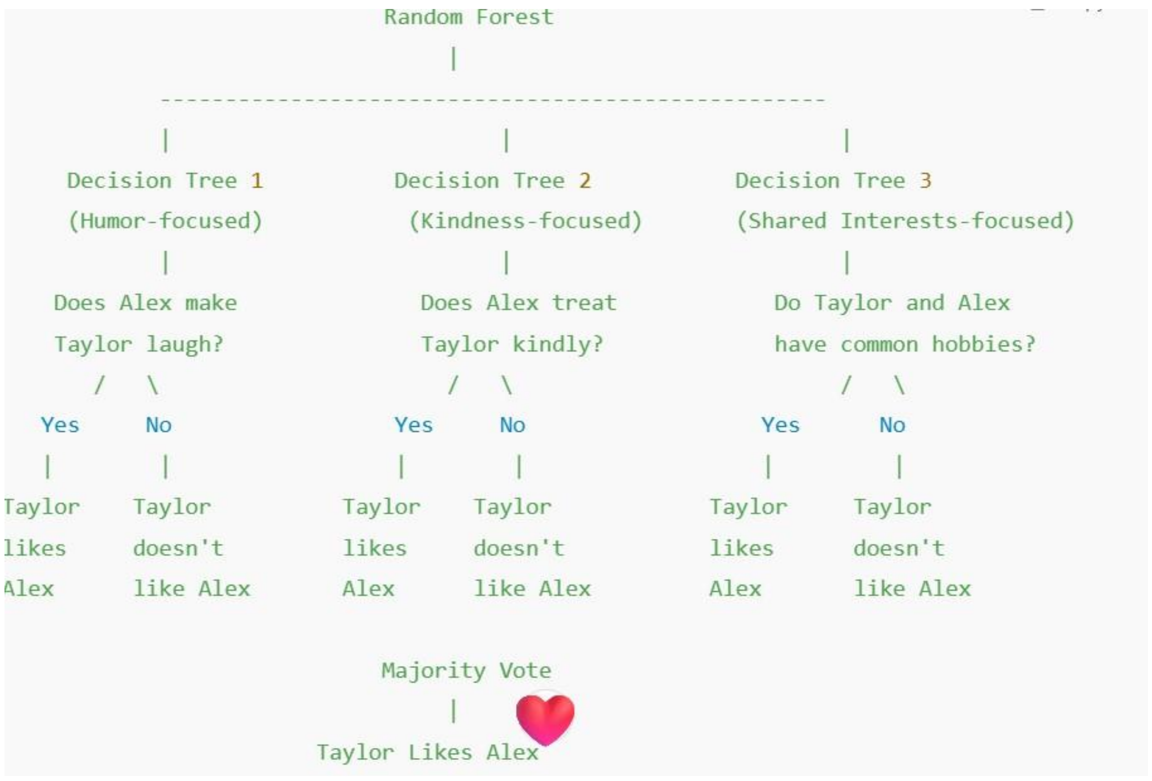
It is a collection (or forest) of Decision Trees, where each tree makes an independent decision or prediction. The final prediction is made by combining the predictions from all the individual trees. It's like asking a group of people for their opinion and then taking the majority vote to make a final decision.

Example Scenario: Deciding if Taylor Likes Alex

Let's consider that Taylor is deciding if they like Alex based on several factors, like humor, kindness, and shared interests. If Taylor is uncertain about whether to like Alex or not, they might ask several friends (each representing a different Decision Tree in the Random Forest).

Here's how Random Forest would work in the romantic scenario:

1. **Taylor's Decision Trees**: Each friend (Decision Tree) has their own opinion about whether Taylor will like Alex. For example, one friend might think humor is the most important, while another might prioritize kindness or shared interests.
2. **Each Friend Votes**:
   - **Friend 1** (Tree 1): Thinks humor is the most important, so if Alex makes Taylor laugh, they vote "Yes." o **Friend 2** (Tree 2): Prioritizes kindness, so if Alex treats Taylor kindly, they vote "Yes."
   - **Friend 3** (Tree 3): Focuses on shared interests, so if they have many common hobbies, they vote "Yes."
3. **Final Decision**: Once all the friends (trees) have voted, the **majority vote** decides the outcome. If most friends say "Yes," then Taylor will likely like Alex. If most say "No," then Taylor might decide not to pursue the relationship.



```
                     Random Forest
                           |
          -------------------------------------------------
          |                        |                        |
     Decision Tree 1          Decision Tree 2          Decision Tree 3
     (Humor-focused)           (Kindness-focused)      (Shared Interests-focused)
          |                        |                        |
    Does Alex make             Does Alex treat             Do Taylor and Alex
    Taylor laugh?              Taylor kindly?              have common hobbies?
        /   \                      /   \                        /   \
   Yes      No                Yes      No                 Yes       No
    |        |                 |        |                  |         |
Taylor    Taylor           Taylor    Taylor            Taylor     Taylor
likes     doesn't          likes     doesn't           likes      doesn't
Alex      like Alex        Alex      like Alex         Alex       like Alex

                     Majority Vote
                           |
                  Taylor Likes Alex
```

Figure 46 Random Forest

- ➢ The Random Forest takes the **majority vote** from the three trees. If most trees say "Yes," then the final prediction is that **Taylor likes Alex**.
- ➢ In this case, even if one tree votes "No," the overall prediction can still be "Yes" if the majority of the trees agree. This highlights how Random Forest combines multiple perspectives to make a more reliable decision.

✓ **Support Vector Machine (SVM)**

- ➢ It is a powerful supervised learning algorithm used for classification and regression tasks. It works by finding the best boundary (or hyperplane) that separates data points into different classes. The goal of SVM is to maximize the margin between the data points and the boundary, ensuring the best separation between classes.

Romantic Scenario: Deciding Compatibility

Let’s imagine Taylor is trying to decide if they should start dating Alex or Jordan.
Taylor has several friends who offer advice based on past experiences, which are split into two categories: successful relationships and not-so-successful relationships. Each friend's advice is based on specific characteristics like shared interests and communication styles.

Data Points: Each relationship experience (either successful or not) is a data point in this

scenario.
Features: Characteristics like communication style and shared interests are the features.
Classes: The two classes are "Successful Relationship" and "Unsuccessful Relationship."
SVM will find the best way to separate these two classes using a line (or hyperplane) that maximizes the distance (margin) between the closest data points of each class. These closest points are called support vectors.

```
        Successful Relationship      Unsuccessful Relationship
              o                            x
               o                          x
              (o)                        (x)
    o                                      x
                    |<------- Margin ------->|
                      \                /
                        \ Hyperplane  /
```

Figure 47 SVM Scenario

➢ In this illustration:

The points labelled (o) and (x) are the support vectors. They are the closest to the hyperplane from each class.

The hyperplane is adjusted based on these support vectors to maximize the margin (the space between the closest points of each class).

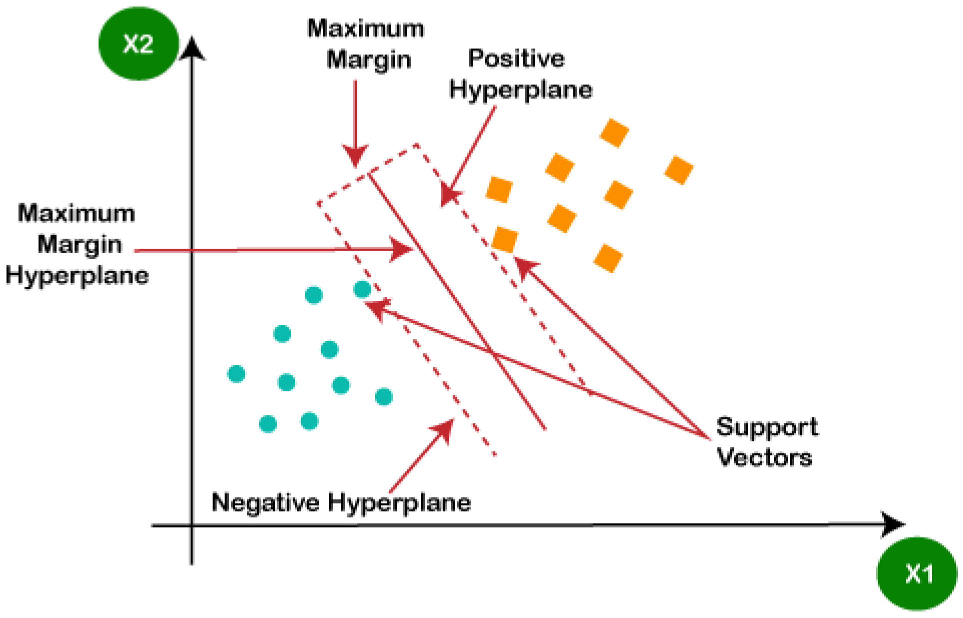


Figure 48 Hyperplane for SVM

✓ **K-Nearest Neighbours (KNN)**

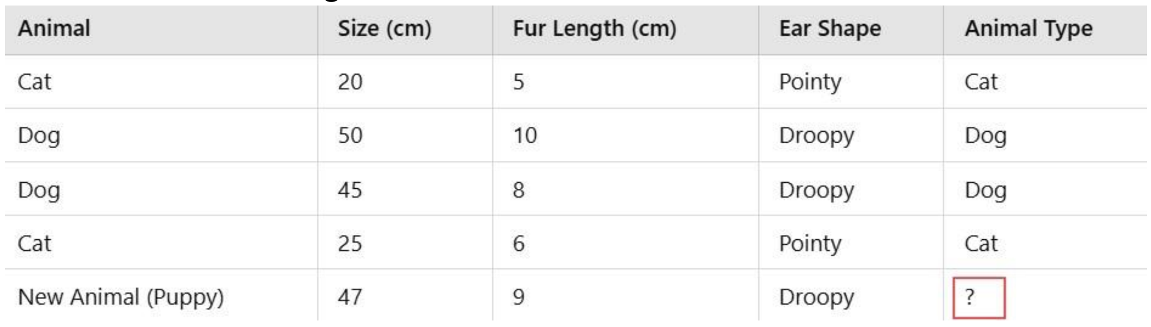
it is a simple, intuitive, and powerful classification algorithm used in machine learning. It works by comparing a new data point (like a new romantic possibility) to existing data points in the dataset, and making predictions based on the majority class (category) of the **k** nearest neighbours. The parameter **k** refers to how many neighbours are considered when making a decision.

Classifying Animals (Cats vs. Dogs)

Scenario: Imagine you're trying to classify animals into either "cat" or "dog" based on certain features like **size**, **fur length**, and **ear shape**.

How KNN Works:

- You are given a new animal (like a new puppy or kitten) and want to predict whether it's a cat or a dog.
- You measure its **size**, **full length**, and **ear shape**.
- You then look at the **k nearest neighbours** (k=3) in the dataset (i.e., other animals you already know) to see which category they belong to.
- If the majority of the three closest animals are **dogs**, the new animal is classified as a **dog**.



Animal	Size (cm)	Fur Length (cm)	Ear Shape	Animal Type
Cat	20	5	Pointy	Cat
Dog	50	10	Droopy	Dog
Dog	45	8	Droopy	Dog
Cat	25	6	Pointy	Cat
New Animal (Puppy)	47	9	Droopy	?

Table 3 KNN Scenario

If we set **k=3**, we check the three closest neighbours (using distance metrics), and we see that 2 out of 3 neighbours are **dogs**, so the new animal will be classified as a **dog**.

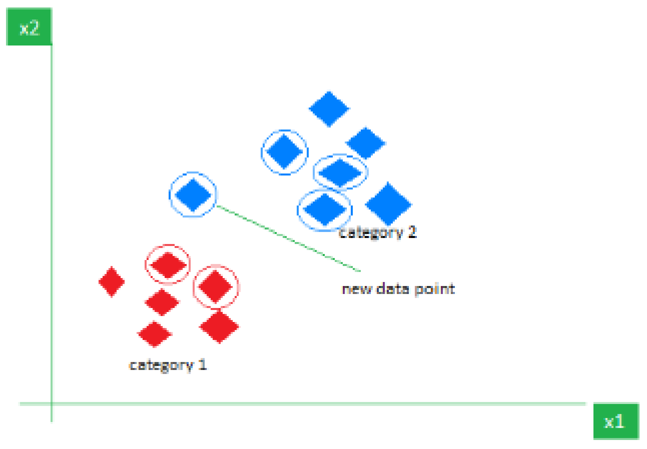


Figure 49 KNN Graph

- ✓ **Naive Bayes**

it is a classification algorithm based on **probability theory** and **Bayes' Theorem**. It's called **"naive"** because it assumes that all features are **independent** from each other, which is often not the case in real life. Despite this "naive" assumption, it performs quite well for many types of problems, especially when the dataset has multiple features and categories.

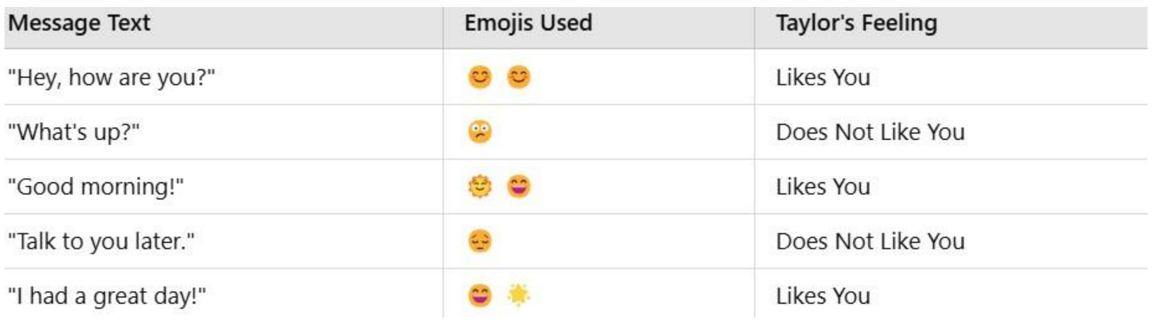
Naive Bayes is commonly used for classification tasks where you want to determine the category or class of a data point. It works by calculating the probability that a given data point belongs to each category and then choosing the class with the highest probability.

Example Scenario: Predicting Whether Someone Likes You

Let's create a romantic scenario to make it more relatable!

Scenario: You want to know whether your crush, **Taylor**, likes your back. Based on their responses to your texts (for example, **word choices** and **emojis**), you can predict whether Taylor likes you or not.

You will create a **Naive Bayes classifier** to predict **Taylor's feelings**. The features could be the words or emojis Taylor uses in their messages, and the classes are **"Likes You"** or **"Does Not Like You"**.



Message Text	Emojis Used	Taylor's Feeling
"Hey, how are you?"	😊 😊	Likes You
"What's up?"	😟	Does Not Like You
"Good morning!"	🌼 😄	Likes You
"Talk to you later."	😔	Does Not Like You
"I had a great day!"	😄 ☀️	Likes You

Table 4 Naive bayes scenario

Train the Model: The Naive Bayes algorithm will first count how often words and emojis appear in the messages classified as "Likes You" and "Does Not Like You." It will calculate the probability of each feature (word or emoji) occurring in each class.

Classify New Data: When you send a new message,

say, “How’s your day going?” with the emoji ,

the classifier will calculate the probability of Taylor liking you or not liking you based on the words and emoji you used.

It will compute:

- ❖ **P(Likes You | Text, Emojis)** = The probability that Taylor likes you given the features in the message (words + emoji).
- ❖ **P(Does Not Like You | Text, Emojis)** = The probability that Taylor does not like you given the features in the message.

The class with the **highest probability** will be selected as the prediction.

Theoretical Activity 2.1.2: Description of Unsupervised Machine Learning algorithm

Tasks:

1: Answer the following questions related to the Description of Unsupervised Machine Learning

I. Explain the following unsupervised machine learning algorithms

a) K-Means Clustering

b) Principal Component Analysis (PCA)

c) Hierarchical Clustering

d) Anomaly Detection

II. List down 5 application unsupervised machine learning algorithms

2: Provide the answer for the asked questions and write them on papers.
3: ask the trainee present the findings to trainer and whole class
4: For more clarification, read the key readings 2.1.2. In addition, ask questions if any.

Key readings 2.1.2. Description of Unsupervised Machine Learning algorithm

- **Unsupervised Learning Algorithms**

Unsupervised Machine learning models implement the learning process opposite to supervised learning, which means it enables the model to learn from the unlabeled data.

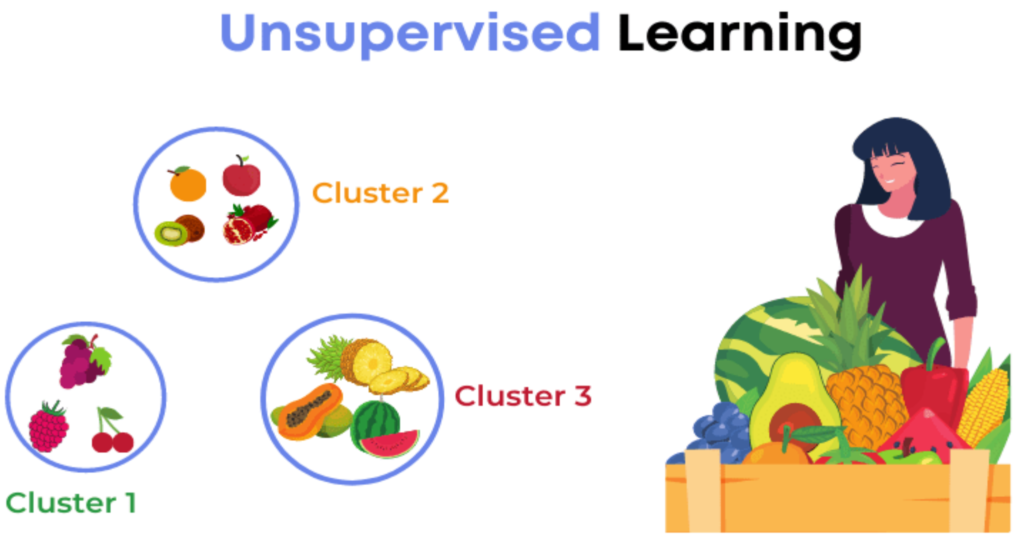


Figure 50 Unsupervised Algorithms

Unsupervised learning aims to discover the dataset's **underlying pattern**, assemble that data according to **similarities**, and express that dataset in a precise format. Let us discuss different unsupervised machine learning algorithms.

✓ **K-Means Clustering**

K-Means Clustering is an unsupervised machine learning algorithm used for partitioning a dataset into distinct groups (clusters) based on similarity.

The objects with the most similarities remain in the same group, and they have no or very few similarities from other groups.

Clustering algorithms can be widely used in different tasks such as **Image segmentation, Statistical data analysis, Market segmentation**, etc. Some commonly used Clustering algorithms are *K-means Clustering, hierarchal Clustering, DBSCAN*, etc.

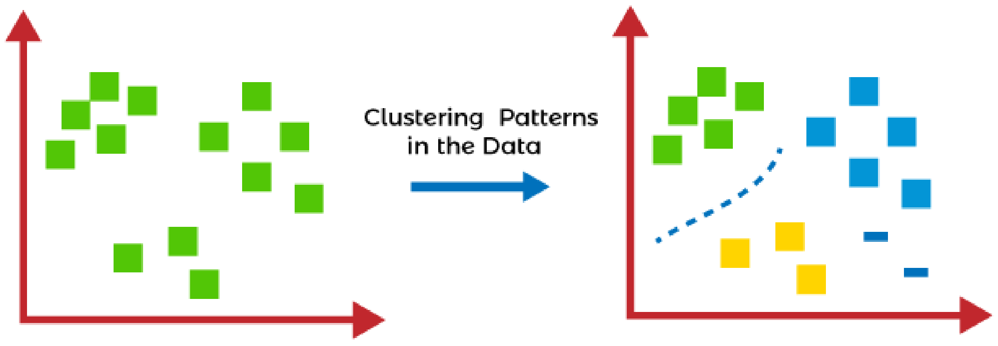


Figure 51 Clustering in K-Means Clustering

- ➢ **Imagine you're a party planner.** You have a list of guests with their addresses, and you want to divide them into groups so you can send out invitations more efficiently.

Here's how K-Means Clustering works:

1. **Choose the number of groups (clusters):** Let's say you want to divide guests into 3 groups based on their location.
2. **Randomly place 3 "party centers" on the map:** These are your initial cluster centers.
3. **Assign each guest to the nearest party center:** This creates your initial groups.
4. **Move each party center to the average location of the guests assigned to it:** This improves the center's position to better represent the group.
5. **Repeat steps 3 and 4:** Guests are reassigned, and centers are moved again. This process continues until the centers stabilize, meaning they no longer move significantly.

The result: You have 3 distinct groups of guests, each centered around a party location. This allows you to easily send out invitations and plan logistics for each group.

In essence, K-Means Clustering groups similar data points together by iteratively finding the centers of those groups.

✓ **Principal Component Analysis (PCA)**

Principal Component Analysis (PCA) is an unsupervised machine learning technique primarily used for dimensionality reduction. It transforms a high-dimensional dataset into a lower-dimensional form while preserving as much variability as possible. PCA is widely used for feature extraction, noise reduction, and data visualization.

- ➢ Let's say you're building a face recognition system. Each face image has thousands of pixels, making it difficult for a computer to process efficiently.

PCA can help by:

1. **Finding the most important facial features:** PCA identifies the key features that distinguish one face from another, such as eye shape, nose size, and jawline.
2. **Reducing the number of pixels:** PCA can represent each face image with a smaller set of numbers, capturing the essence of the face while reducing the amount of data.

This makes face recognition faster and more accurate, as the computer can focus on the most important information.

✓ **Hierarchical Clustering**

In machine learning, clustering is the unsupervised learning technique that groups the data based on similarity between the set of data.

A dendrogram, a tree-like figure produced by hierarchical clustering, depicts the hierarchical relationships between groups. Individual data points are located at the bottom of the dendrogram, while the largest clusters, which include all the data points, are located at the top. In order to generate different numbers of clusters, the dendrogram can be sliced at various heights.

The dendrogram is created by iteratively merging or splitting clusters based on a measure of similarity or distance between data points. Clusters are divided or merged repeatedly until all data points are contained within a single cluster, or until the predetermined number of clusters is attained.

We can look at the dendrogram and measure the height at which the branches of the dendrogram form distinct clusters to calculate the ideal number of clusters. The dendrogram can be sliced at this height to determine the number of clusters.

Types of Hierarchical Clustering

Basically, there are two types of hierarchical Clustering:

1. Agglomerative Clustering
2. Divisive clustering

I. Hierarchical Agglomerative Clustering

It is also known as the bottom-up approach or hierarchical agglomerative clustering (HAC). A structure that is more informative than the unstructured set of clusters returned by flat clustering. This clustering algorithm does not require us to prespecify the number of clusters. Bottom-up algorithms treat each data as a singleton cluster at the outset and then successively agglomerate pairs of clusters until all clusters have been merged into a single cluster that contains all data.

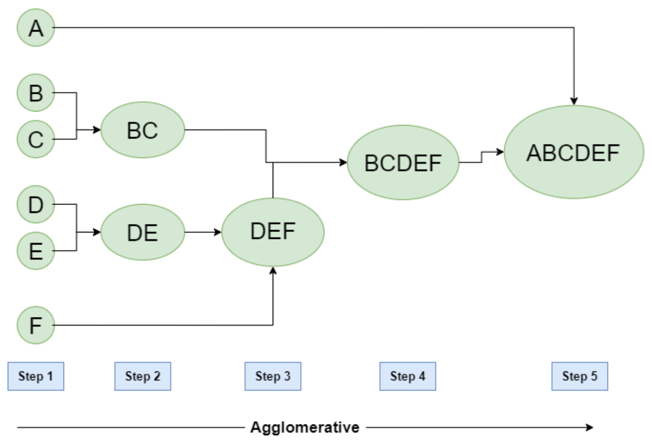


Figure 52Hierarchical Agglomerative Clustering

Steps:

- Consider each alphabet as a single cluster and calculate the distance of one cluster from all the other clusters.
- In the second step, comparable clusters are merged together to form a single cluster. Let's say cluster (B) and cluster (C) are very similar to each other therefore we merge them in the second step similarly to cluster (D) and (E) and at last, we get the clusters [(A), (BC), (DE), (F)]
- We recalculate the proximity according to the algorithm and merge the two nearest clusters([(DE), (F)]) together to form new clusters as [(A), (BC), (DEF)]
- Repeating the same process; The clusters DEF and BC are comparable and merged together to form a new cluster. We're now left with clusters [(A), (BCDEF)].
- At last, the two remaining clusters are merged together to form a single cluster [(ABCDEF)].

II. Hierarchical Divisive clustering

It is also known as a top-down approach. This algorithm also does not require to prespecify the number of clusters. Top-down clustering requires a method for splitting a cluster that contains the whole data and proceeds by splitting clusters recursively until individual data have been split into singleton clusters.

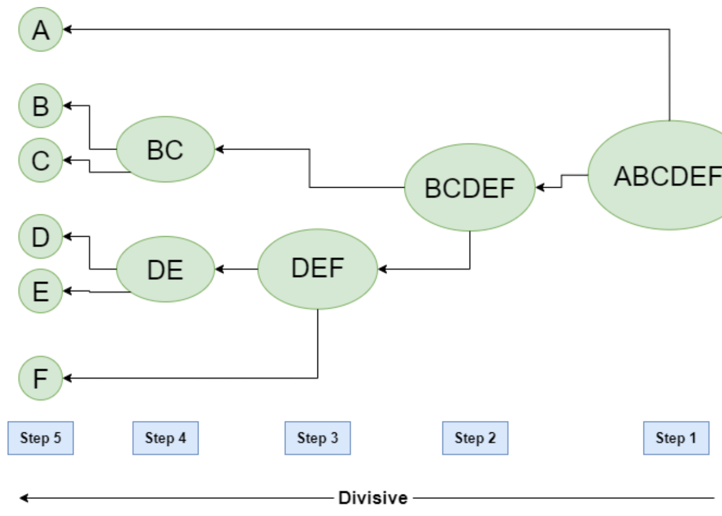


Figure 53Hierarchical Divisive clustering

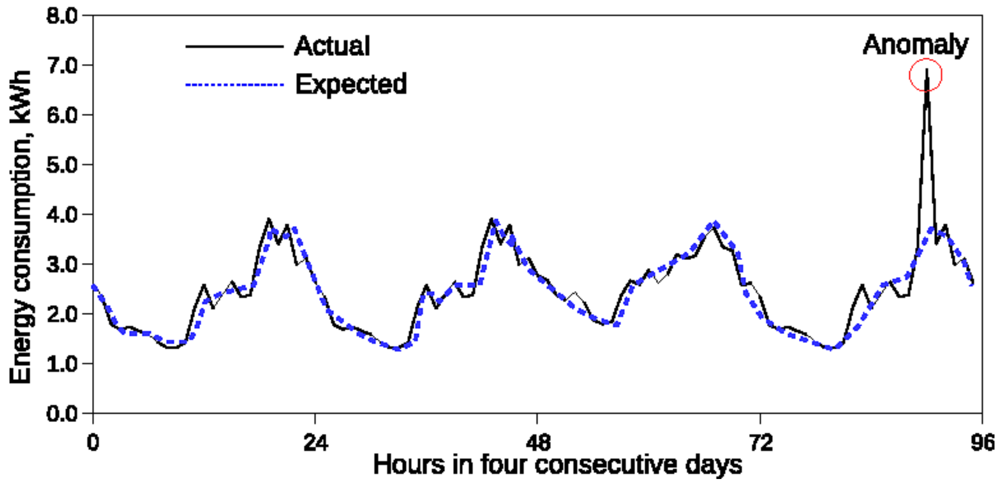
✓ **Anomaly Detection**

nomaly Detection, additionally known as outlier detection, is a technique in records analysis and machine studying that detects statistics points, activities, or observations that vary drastically from the dataset's ordinary behavior. These abnormalities may sign extreme conditions which include mistakes, flaws, or fraud.

Types of Anomalies

- **Point Anomalies:** Single data points that deviate significantly from the norm.
- **Contextual Anomalies:** Data points that are anomalous only in a specific context.
- **Collective Anomalies:** A group of data points that together deviate from the norm.

Below, we can compare predictions of time-series data with the actual occurrence. As seen, the forecast closely follows the actual data until an anomaly occurs. This considerable variation is unexpected, as we see from the past data trend and the model prediction shown in blue. You can train machine learning models can to identify such out-of-distribution anomalies from a much more complex dataset.



Theoretical Activity 2.1.3: Semi-supervised Learning

Tasks:

1: Answer the following questions related to the description of semi-supervised learning algorithm

I. Explain semi-supervised machine learning algorithm

II. List down at least 5 applications of Semi-Supervised Learning

2: Provide the answer for the asked questions and write them on papers.

3: Ask trainee to present their findings to trainer and whole class.

4: For more clarification, read the key readings 2.1.3.

Key readings 2.1.3. Semi-supervised Learning algorithm

✓ **Semi-supervised learning**

Semi-supervised learning is a machine learning technique that falls between supervised and unsupervised learning. It uses a combination of labelled and unlabeled data to train a model. This approach is useful when obtaining labeled data is expensive or time-consuming, but there's plenty of unlabeled data available.

Common Techniques:

- **Self-training:** A simple method where the model predicts labels for unlabeled data and then uses these predictions as additional training examples.
- **Co-training:** Uses multiple models trained on different views of the data. Each model predicts labels for the other model's unlabeled data.
- **Generative models:** These models learn the underlying data distribution and can be used to generate new data points that can be used for training.

Application of Supervised Learning

- **Spam filters:** Many email clients use supervised learning algorithms to filter out spam emails. The algorithms are trained on a dataset of labeled emails (spam and non-spam) and use this information to predict whether a new email is spam or not.
- **Fraud detection:** Many financial institutions use supervised learning algorithms to identify fraudulent activity. The algorithms are trained on a dataset of labeled transactions (fraudulent and non-fraudulent) and use this information to flag potentially fraudulent transactions in real-time.
- **Image classification:** Many image recognition systems, such as those used by social media platforms to automatically tag photos, use supervised learning algorithms to classify images based on their content. The algorithms are trained on a dataset of labeled images (e.g. images of cats and dogs) and use this information to classify new images.

Theoretical Activity 2.1.4: Reinforcement Learning

Tasks:

1: Answer the following questions related to the description of Reinforcement Learning Algorithm

I. Explain Reinforcement Learning

II. List down at least 5 applications of Reinforcement Learning

2: Provide the answer for the asked questions and write them on papers.

3: Ask trainee to present their findings to trainer and whole class.

4: For more clarification, read the key readings 2.1.4

Key readings 2.1.4. Reinforcement learning algorithm

- **Reinforcement learning Algorithm**

  Reinforcement Learning (RL) is a type of machine learning algorithm that focuses on teaching an agent how to behave in an environment by performing actions and receiving feedback in the form of rewards or penalties. It's like training a dog to do tricks by rewarding good behavior and discouraging bad behavior.

Below are some popular algorithms that come under reinforcement learning:

- **Q-Learning**

Q-learning is a model-free reinforcement learning algorithm that helps an agent learn the optimal action-selection policy by iteratively updating Q-values, which represent the expected rewards of actions in specific states.

Applications:

- **Inventory Management:** Determine optimal order quantities to minimize costs while ensuring product availability.
- **Game Playing:** Develop AI agents to play simple games with discrete action spaces (e.g., tic-tac-toe).

- **Deep Q-Networks (DQN)**

This is the type of reinforcement learning algorithm that combines Q-Learning with deep learning techniques. It is used for training agents to make decisions in environments where the action space is discrete.

Applications:

- **Game Playing**
- **Robotics**

- **Neural Network Architectures**

Neural networks are machine learning models that mimic the complex functions of the human brain. These models consist of interconnected nodes or neurons that process data, learn patterns, and enable tasks such as pattern recognition and decision-making.

Neural networks follows a structured, three-stage process:

- **Input Computation**: Data is fed into the network.
- **Output Generation**: Based on the current parameters, the network generates an output.

- **Iterative Refinement**: The network refines its output by adjusting weights and biases, gradually improving its performance on diverse tasks.

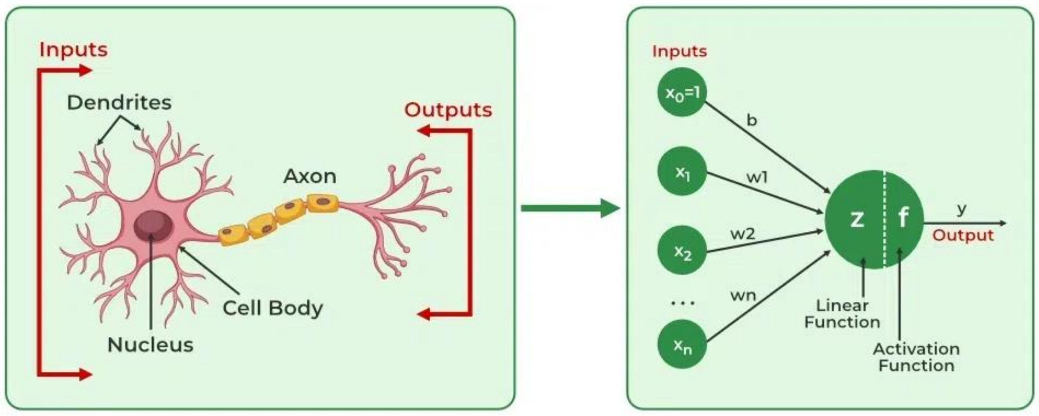


Figure 54Illustrates the analogy between a biological neuron and an artificial neuron, showing how inputs are received and processed to produce outputs in both systems.

Types of Neural Networks

There are *seven* types of neural networks that can be used.

- **Feedforward Networks:** A feedforward neural network is a simple artificial neural network architecture in which data moves from input to output in a single direction.
- **Convolutional Neural Network (CNN):** A Convolutional Neural Network (CNN) is a specialized artificial neural network designed for image processing. It employs convolutional layers to automatically learn hierarchical features from input images, enabling effective image recognition and classification.
- **Recurrent Neural Network (RNN):** An artificial neural network type intended for sequential data processing is called a Recurrent Neural Network (RNN). It is appropriate for applications where contextual dependencies are critical, such as time series prediction and natural language processing, since it makes use of feedback loops, which enable information to survive within the network.
- **Multilayer Perceptron (Artificial Neural Networks):** MLP is a type of feedforward neural network with three or more layers, including an input layer, one or more hidden layers, and an output layer. It uses nonlinear activation functions.

Applications of Neural Network

❖ Animal behaviour, their relationships, and population cycles may be apt for analysis using neural networks.

❖ Evaluation and valuation of property, buildings, automobiles, machinery, etc. should be an easy task if done with the help of a neural network.

- ❖ Criminal sentencing could be anticipated using a large sample of crime details as input and the resulting sentences as output.
- ❖ Data mining, cleaning, and validation could be achieved efficiently if we find out which records suspiciously diverge from the pattern of their peers. This could be done with the help of a neural network.
- ❖ Weather prediction may be possible using a neural network. Inputs would include weather reports from surrounding areas. Output(s) could be the future weather in specific areas based on the input information.
- ❖ Examination of medical issues is an ideal application for neural networks.

Points to Remember

- Description of Supervised Learning algorithms based on labeled data
- Description of unsupervised learning algorithms based on unlabeled data
- Description Semi-supervised Learning uses a mix of labeled and unlabeled data
- Description Reinforcement Learning involves an agent interacting with an environment by performing actions and receiving feedback in the form of rewards or penalties

Application of learning 2.1.

As a trainee in machine learning, you're tasked to select a learning machine algorithm of analysing the student performance based on each academic year. Which best machine learning algorithms based on the nature of the problems you're solving should be used and why?

Indicative content 2.2: Selection of Machine Learning Algorithm

Duration:6 hrs

Practical Activity 2.2.1: Selecting Machine Learning Algorithm

Task: Answer the following questions related to selection of ML Algorithm

1: Read the **Key readings 2.1.1**

2: Refer to the theoretical activity 2.1.1 you are asked to select machine learning algorithm

3: Ask trainee to present their work to trainer and the whole class

4: Ask any clarification to the trainer or colleagues

5: Ask trainee to read the key readings 2.2.1.

Key readings 2.2.1: Selection of machine learning algorithm

- **Selection of machine learning algorithm**

Choosing the right machine learning algorithm involves a series of steps, from understanding the problem to evaluating resource constraints. Here's a structured approach to help you select the most appropriate algorithm:

Step 1: Identify problem

- **Regression**: When the goal is to predict a continuous outcome. For example, predicting house prices based on features like size and location.

- **Classification**: When the goal is to predict categories. For example, classifying emails as spam or not spam.

- **Clustering**: When the goal is to group similar data points together without predefined labels. For example, customer segmentation based on purchasing behaviour.

Step 2: Analyse type of data in a given dataset

- **Independent variable:** Variables used as input features for prediction. They can be continuous, categorical, or a mix. An **independent variable** is a variable that acts as an input to a model and is used to predict or explain the value of the dependent variable.
- **Dependent variable:** The target variable you are trying to predict or classify. In regression, it's continuous; in classification, it's categorical.

Step 3: Resource analysis

When working on machine learning projects, **resource analysis** is crucial to ensure that your hardware and computational resources can handle the complexity and scale of your model.

- **Computational power**

Computational power refers to the ability of a system's processor (CPU or GPU) to perform calculations and execute tasks. It determines how quickly and efficiently your machine learning model can be trained or run inference.

- **Memory limitations**

Memory limitations refer to the capacity of **RAM** (Random Access Memory) or **GPU memory** available for storing data, models, and intermediate computations during the training and inference process.

Step 5: Choose machine learning algorithm to be used

Choosing Machine learning algorithm should be Based on the problem type, dataset properties, and resource constraints

For Regression

Linear regression is used to predict a continuous value by finding **the best-fit straight line between input (independent variable) and output (dependent variable)**

Minimizes the difference between actual values and predicted values using a method called “least squares” to best fit the data.

Predicting a person’s weight based on their height or predicting house prices based on size.

- **For Classification**:

  - **Logistic Regression**: Good for binary classification.
  - **Decision Trees**: Easy to interpret and visualize.
  - **Random Forest**: Handles large datasets well.
  - **Support Vector Machines (SVM)**: Effective in high-dimensional spaces

Points to Remember

- Always follow all steps for Choosing the right machine learning algorithm based on the problem to be solved.

Application of learning 2.2.

As a trainee in machine learning, you're tasked to select the machine learning algorithm among many, which should be the best for developing a machine learning solution for your school platform of analysing the student performance based on each academic year.

Indicative content 2.3: Train a Machine Learning Model

Duration: 6 hrs

Practical Activity 2.3.1: Loading the dataset

Task:

1: You are requested to go to the computer lab and load the dataset by using the following library:

a) Using pandas
b) Using NumPy
c) Using Scikit-Learn
d) Using Seaborn
e) Using Requests and io

2: Apply safety precautions

3: Present out the steps to Load the dataset.

4: Referring to the steps provided, Load the dataset.

5: Present your work to the trainer and whole class

6: Read key reading 2.3.1 and ask clarification where necessary

Key readings 2.3.1 Loading the dataset

- **Loading the dataset**

  Dataset loading refers to the process of importing data from a storage source (e.g., a file, database, or online resource) into a programming environment for analysis, manipulation, and use in applications such as machine learning or statistical modelling.

➢ Different libraries are used to load dataset

1. Using pandas

pandas is widely used for data manipulation and analysis,You can load datasets from various file formats (e.g., CSV, Excel, JSON) using pandas. The pd.read_csv() function, for example, loads data from a local CSV file into a DataFrame, making it easy to explore and manipulate.

```
import pandas as pd
df = pd.read_csv('car_cleaned_dataset.csv')
```

2. Using numpy

numpy is primarily used for numerical operations and creating multidimensional arrays, If your dataset is in raw text or numerical format, you can use numpy.loadtxt() or numpy.genfromtxt() to load it into a NumPy array for further processing.

```
import numpy as np
data = np.loadtxt('data.txt', delimiter=',')
```

3. Using scikit-learn

scikit-learn provides tools for machine learning, including utilities to load sample datasets or pre-existing ones. You can load datasets like iris or digits directly using its built-in functions, or split and preprocess data after loading with other methods.

```
from sklearn.datasets import load_iris
data = load_iris()
X, y = data.data, data.target
```

4. Using seaborn

seaborn is a visualization library that integrates well with pandas for statistical graphics. While not directly used to load datasets, it provides access to built-in sample datasets via seaborn.load_dataset() (e.g., "titanic", "iris"), which are loaded as pandas DataFrames.

```
import seaborn as sns
data = sns.load_dataset('titanic')
```

5. Using requests and io

These libraries are used to fetch data from a web URL when the dataset is hosted online.

- requests: Downloads the content from the URL.
- io.StringIO: Converts the downloaded text data into a format that can be read by pandas.

The fetched content is passed into pandas for processing.

```
import requests
from io import StringIO
url = 'https://example.com/data.csv'
response = requests.get(url)
data = pd.read_csv(StringIO(response.text))
```

Practical Activity 2.3.2: Train a Machine Learning Models

Task:

1: You are requested to go to the computer lab and model.

2: Apply safety precautions.

3: Present out the steps to train a machine learning model.

4: Referring to the steps provided, train a machine learning model.

5: Present your work to the trainer and whole class

6: Read key reading 2.3.2 and ask clarification where necessary

Key readings 2.3.2 Train Machine learning model

- **Split dataset**

Splitting a dataset into **train**, **test**, and **validation** sets is a standard practice in machine learning to ensure that the model generalizes well to unseen data. Here's how each set is used:

1. Purpose of Splits

Train Set: Used to train the model (fitting the model's parameters).

Validation Set: Used during training to tune hyperparameters and evaluate model performance without affecting the training process.

Test Set: Used after training and validation to evaluate the model's performance on completely unseen data.

2. Typical Ratios

Train Set: 60%-70%

Validation Set: 10%-20%

Test Set: 20%-30%

For example: 70% train, 15% validation, 15% test.

3. How to Split the Dataset

You can use **Scikit-Learn**'s train_test_split for this purpose.

```
from sklearn.model_selection import train_test_split
import pandas as pd

# Load the dataset
df = pd.read_csv('car_cleaned_dataset.csv')
```

```
# Split columns properly
df = df.iloc[:, 0].str.split(',', expand=True)
df.columns = [
    'brand', 'model', 'owner', 'year', 'km_driven', 'fuel', 'seller_type',
    'transmission', 'mileage', 'engine', 'max_power', 'seats', 'torque_value',
    'torque_rpm_value', 'selling_price'
]

# Convert relevant columns to numeric
df[['year', 'km_driven', 'mileage', 'engine', 'max_power', 'seats',
    'torque_value', 'torque_rpm_value', 'selling_price']] = df[['year',
    'km_driven', 'mileage', 'engine', 'max_power', 'seats',
    'torque_value',           'torque_rpm_value',           'selling_price']].apply(pd.to_numeric,
errors='coerce')

# Selecting features and target
X = df.drop(columns=['brand', 'model', 'selling_price'])
y = df['selling_price']

# One-hot encode categorical columns
categorical_cols = ['owner', 'fuel', 'seller_type', 'transmission']
X = pd.get_dummies(X, columns=categorical_cols, drop_first=True)

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

Note:

- Reads the dataset from a CSV file named car_cleaned_dataset.csv into a pandas DataFrame (df).

The data is initially loaded as a single column, which will be processed in the next steps.

df.iloc[:, 0].str.split(',', expand=True):

- Takes the first column (which contains all data as a single string per row) and splits it based on commas (,) to create separate columns.
- expand=True ensures that the split values are assigned to new columns in the DataFrame.

- df.columns = [...]:

- Assigns meaningful column names to the dataset for better readability and reference.

- Converts selected columns (mostly numerical ones) from string format to numeric (integers or floats).
- pd.to_numeric(errors='coerce') ensures that any invalid values (such as missing or non-numeric data) are replaced with NaN (not-a-number) instead of causing an error.
- X: Feature matrix containing all columns except brand, model, and selling_price (since brand and model are non-numeric and selling_price is the target variable we want to predict).
- y: Target variable (dependent variable), which contains the car selling price.
- categorical_cols: List of categorical features (non-numeric columns) that need to be converted into numerical format.
- pd.get_dummies(...):

- Converts categorical columns into numerical binary (0/1) values using one-hot encoding.
- drop_first=True avoids multicollinearity by dropping the first category of each categorical feature (e.g., if fuel has values like Petrol, Diesel, Electric, it will create two columns instead of three).

 Splits the dataset into **training** and **testing** subsets.

- X_train, y_train: Used for training the model.
- X_test, y_test: Used for evaluating the model's performance.
- test_size=0.2:
- Reserves **20%** of the data for testing and uses **80%** for training.
  - random_state=42:
- Ensures reproducibility by setting a fixed seed for random splitting.

Initialise the model

In machine learning, model initialization refers to the process of assigning initial values to the model's parameters (weights and biases) before the training process begins. This step is crucial as it can significantly impact the model's training speed, convergence, and overall performance.

First import required library

```
rom sklearn.linear_model import LinearRegression
```

Initialize the Model

```
# Initialize the model
model = LinearRegression()
```

Key Parameters of LinearRegression

When initializing the model, you can specify the following optional parameters:

1. **fit_intercept**:

Whether to calculate the intercept for the model.

Default: True.

Example: LinearRegression(fit_intercept=False)

2. **normalize** *(Deprecated as of Scikit-Learn 1.0)*:

If True, the data will be normalized before fitting.
Use StandardScaler for normalization instead.

3. **copy_X**:

If True, X will be copied; otherwise, it may overwrite the original data.
Default: True.

4. **n_jobs**:

Number of CPU cores to use for computation (-1 uses all available cores).
Default: None.

- **Fit the training data into a model**

In machine learning, fitting a model to training data is the core process where the model learns the underlying patterns and relationships within the data. This involves adjusting the model's internal parameters (weights and biases) to minimize the difference between the model's predictions and the actual target values in the training dataset.

```
from sklearn.linear_model import LinearRegression

# Assuming you have prepared your training data (X_train, y_train)

model = LinearRegression()
model.fit(X_train_scaled, y_train)
```

Refer to this example to get an insight for fitting The training data into a model

```
mport pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import pickle

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

# Split columns properly
df = df.iloc[:, 0].str.split(',', expand=True)
df.columns = [
    'brand', 'model', 'owner', 'year', 'km_driven', 'fuel', 'seller_type',
    'transmission', 'mileage', 'engine', 'max_power', 'seats', 'torque_value',
    'torque_rpm_value', 'selling_price'
]

# Convert relevant columns to numeric
df[['year', 'km_driven', 'mileage', 'engine', 'max_power', 'seats',
    'torque_value', 'torque_rpm_value', 'selling_price']] = df[['year',
    'km_driven', 'mileage', 'engine', 'max_power', 'seats',
    'torque_value',             'torque_rpm_value',             'selling_price']].apply(pd.to_numeric,
errors='coerce')

# Selecting features and target
X = df.drop(columns=['brand', 'model', 'selling_price'])
y = df['selling_price']

# One-hot encode categorical columns
categorical_cols = ['owner', 'fuel', 'seller_type', 'transmission']
X = pd.get_dummies(X, columns=categorical_cols, drop_first=True)

# Splitting data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize numerical features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train linear regression model
model = LinearRegression()
model.fit(X_train_scaled, y_train)
```

Points to Remember

- Loading the dataset, we use different libraries based on the data format.
- While training a model for machine learning ensure that Training, testing, and validation data are splited.
- Fitting the training data into the model require model initialization and data splitting

Application of learning 2.3.

As a trainee in machine learning, you're tasked to train the machine learning model, for developing a machine learning solution for your school platform of analysing the student performance based on each academic year.

by performing the following tasks:

- Load a dataset Using pandas, NumPy, Scikit-Learn, Seaborn and Requests and io
- Split dataset using train set, test set and validation set
- Initialize model
- Fit the training data into a model

Indicative content 2.4: Evaluating Machine Learning Model Based on Provided Data

Duration: 6 hrs

Practical Activity 2.4.1: Evaluating machine learning model based on provided data

Task:

1: You are requested to go to the computer lab to evaluate a trained machine learning model.

2: Apply safety precautions

3: Present out the steps to evaluate a trained machine learning model.

4: Referring to the steps provided, evaluate a trained machine learning model.

5: Present your work to the trainer and whole class

6: Read key reading 2.4.1 and ask clarification where necessary

Key readings 2.4.1: Evaluating machine learning model based on provided data

- **Evaluation of Machine Learning Model**

✓ **Prediction of Results**

Prediction on Test Data:

After training the model on the training set, you need to evaluate how well it generalizes to unseen data. The test set, which the model has not seen during training, is used for this purpose. Predictions on the test data allow you to assess if the model has learned the data effectively.

Example: Using .predict() in Scikit-learn with linear regression to make predictions on the test set.

```
# Predictions and evaluation
y_pred = model.predict(X_test_scaled)
```

Prediction on New Data (Unseen Data):

Once the model is deployed in a real-world application, it will encounter new data that was not part of the training or test datasets. Evaluating the model's performance on completely new data is essential to ensure it maintains accuracy in practical applications.

Example: Deploying the model to predict new student performance data in the school platform.

```
new_data = pd.DataFrame({
                                    # data 1,
                                    # data 2,
                                    # ………..,
                                    #data
                }
        )
new_predictions = model.predict(new_data)
```

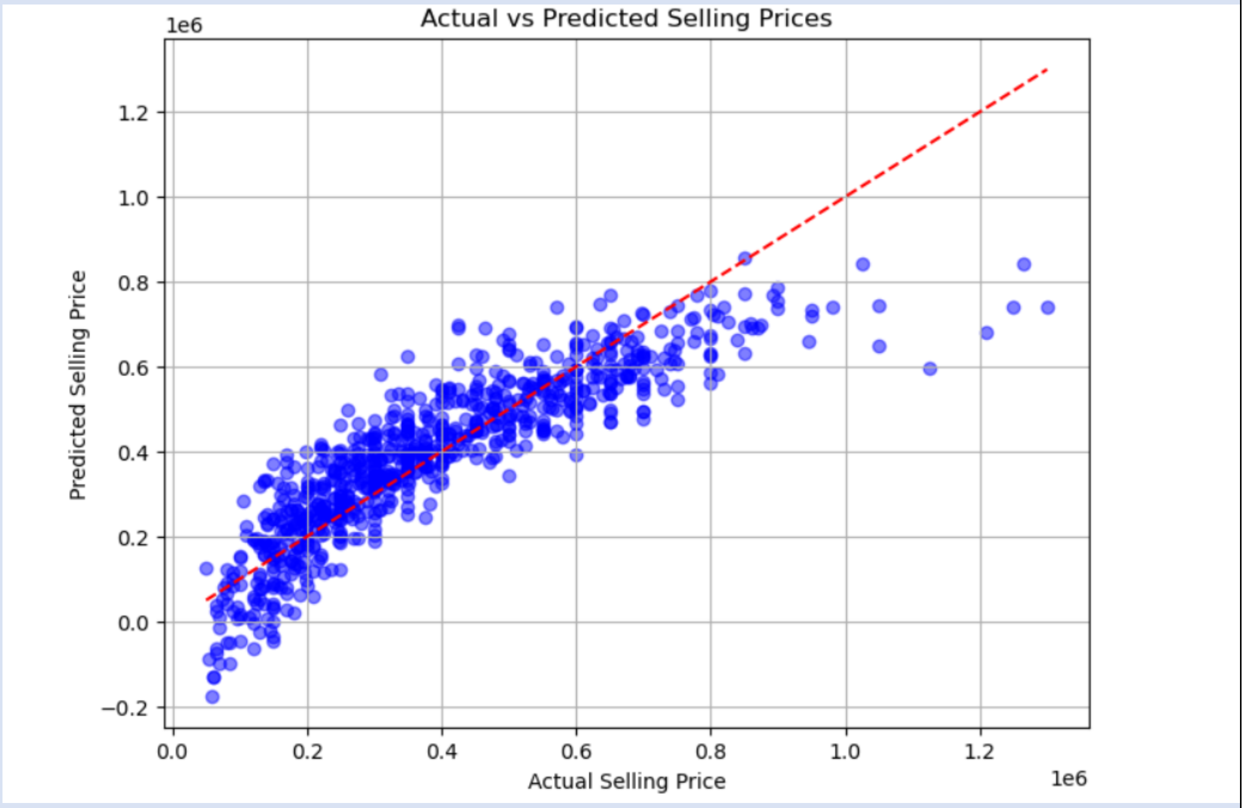
✓ **Visualize Predictions**

Visualizing predictions helps to compare predicted values with actual values, making it easier to spot discrepancies or patterns. Graphical tools like Seaborn or Matplotlib are commonly used to create visuals like scatter plots or line charts.

Example (Regression): Comparing actual vs. predicted student performance scores.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import pickle

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

# Split columns properly
df = df.iloc[:, 0].str.split(',', expand=True)
df.columns = [
    'brand', 'model', 'owner', 'year', 'km_driven', 'fuel', 'seller_type',
    'transmission', 'mileage', 'engine', 'max_power', 'seats', 'torque_value',
    'torque_rpm_value', 'selling_price'
]

# Convert relevant columns to numeric
df[['year', 'km_driven', 'mileage', 'engine', 'max_power', 'seats',
    'torque_value', 'torque_rpm_value', 'selling_price']] = df[['year',
    'km_driven', 'mileage', 'engine', 'max_power', 'seats',
    'torque_value',             'torque_rpm_value',             'selling_price']].apply(pd.to_numeric,
errors='coerce')

# Selecting features and target
X = df.drop(columns=['brand', 'model', 'selling_price'])
y = df['selling_price']

# One-hot encode categorical columns
categorical_cols = ['owner', 'fuel', 'seller_type', 'transmission']
X = pd.get_dummies(X, columns=categorical_cols, drop_first=True)

# Splitting data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize numerical features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train linear regression model
model = LinearRegression()
model.fit(X_train_scaled, y_train)

# Predictions and evaluation
y_pred = model.predict(X_test_scaled)
mse = mean_squared_error(y_test, y_pred)
rmse = mse ** 0.5

print(f'Root Mean Squared Error: {rmse:.2f}')

# Scatter plot: Actual vs Predicted values
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, alpha=0.5, color='blue')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linestyle='--')
plt.xlabel('Actual Selling Price')
plt.ylabel('Predicted Selling Price')
plt.title('Actual vs Predicted Selling Prices')
plt.grid()
plt.show()
```

OUT PUT GRAPH:



✓ **Analyze Evaluation Metrics**

Evaluation metrics are essential for quantifying the model's performance. Depending on whether your task is regression or classification, different metrics should be used:

Regression Metrics:

Accuracy: The ratio of correctly predicted instances to total instances, Measures how well the regression predictions approximate the actual values (ranges from 0 to 1, where 1 is perfect).

Example: How many student performance predictions (pass/fail) were correct.

Root Mean Squared Error (RMSE): The square root of the average squared differences between predicted and actual values,Measures the average magnitude of prediction errors (lower is better)

```
print(f"RMSE: {mean_squared_error(y_test, predictions, squared=False)}")
print(f"Accuracy: {accuracy_score(y_test, predictions)}")
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
import pickle

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

# Split columns properly
df = df.iloc[:, 0].str.split(',', expand=True)
df.columns = [
    'brand', 'model', 'owner', 'year', 'km_driven', 'fuel', 'seller_type',
    'transmission', 'mileage', 'engine', 'max_power', 'seats', 'torque_value',
    'torque_rpm_value', 'selling_price'
]

# Convert relevant columns to numeric
df[['year', 'km_driven', 'mileage', 'engine', 'max_power', 'seats',
    'torque_value', 'torque_rpm_value', 'selling_price']] = df[['year',
    'km_driven', 'mileage', 'engine', 'max_power', 'seats',
    'torque_value',          'torque_rpm_value',          'selling_price']].apply(pd.to_numeric,
errors='coerce')

# Selecting features and target
X = df.drop(columns=['brand', 'model', 'selling_price'])
y = df['selling_price']

# One-hot encode categorical columns
categorical_cols = ['owner', 'fuel', 'seller_type', 'transmission']
X = pd.get_dummies(X, columns=categorical_cols, drop_first=True)

# Splitting data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize numerical features
```

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train linear regression model
model = LinearRegression()
model.fit(X_train_scaled, y_train)

# Predictions and evaluation
y_pred = model.predict(X_test_scaled)
mse = mean_squared_error(y_test, y_pred)
rmse = mse ** 0.5
print(f'Root Mean Squared Error: {rmse:.2f}')
```

OUTPUT:

```
Root Mean Squared Error: 103152.02
```

✓ **Model Interpretation**

Interpreting the model is crucial for understanding how it works and how it makes predictions. This helps in gaining trust in the model and explaining it to stakeholders. The following methods are common in model interpretation, Model interpretation helps us understand the **"why" and "how"** behind the predictions of a machine learning model. It is crucial for evaluating models, identifying potential biases, improving transparency, and building trust in the system.

Key Aspects of Model Interpretation

1. **Global Interpretation**: Understanding the overall behavior of the model across the entire dataset.

Example: Identifying which features have the most influence on predictions.

2. **Local Interpretation**: Understanding why the model made a specific prediction for a particular instance.

Example: Explaining why a student is predicted to pass or fail.

3. **Model Performance Evaluation**: Using metrics to assess how well the model performs on unseen data.

Steps for Model Interpretation

1. **Evaluate Model Performance with Metrics**:
    - **Regression Models**:
        - **R^2 (Coefficient of Determination)**: Measures how well the model

explains variance in the data.

- **RMSE (Root Mean Squared Error)**: Indicates prediction accuracy by measuring average error magnitude.
- **MAE (Mean Absolute Error)**: Measures the average absolute difference between predicted and actual values.

2. **Feature Importance**:

Identify which features contribute the most to the predictions.

3. **Partial Dependence Plot (PDP)**:

Visualizes how a feature impacts predictions while keeping other features constant.
Example: Showing how "hours studied" affects the probability of passing.

4. **Residual Analysis**:

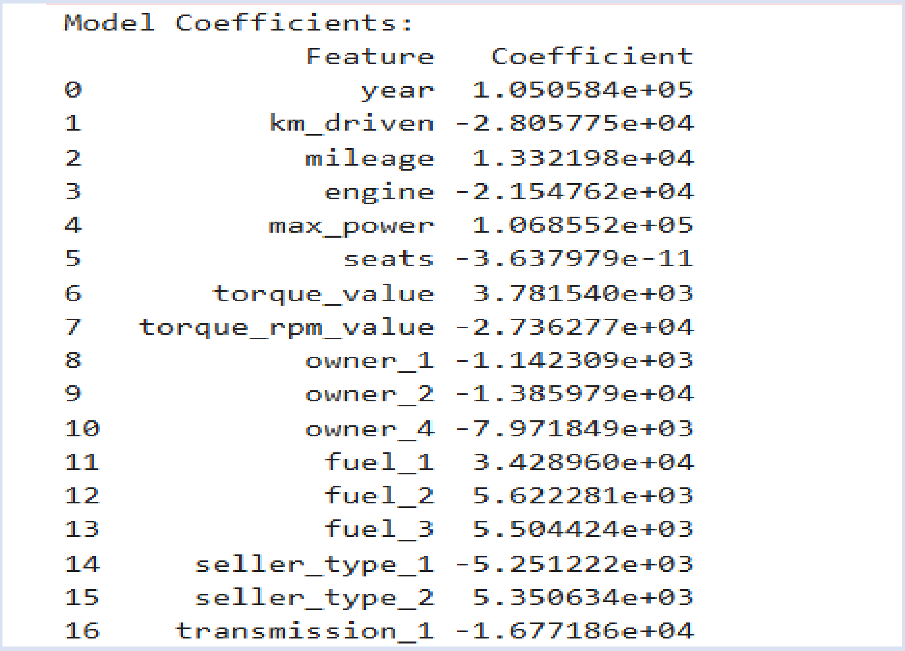
For regression models, analyze residuals (difference between predicted and actual values) to detect patterns or biases.
Example: Residual plots help identify if errors are larger for certain feature ranges.

5. **Model Sensitivity Analysis**:

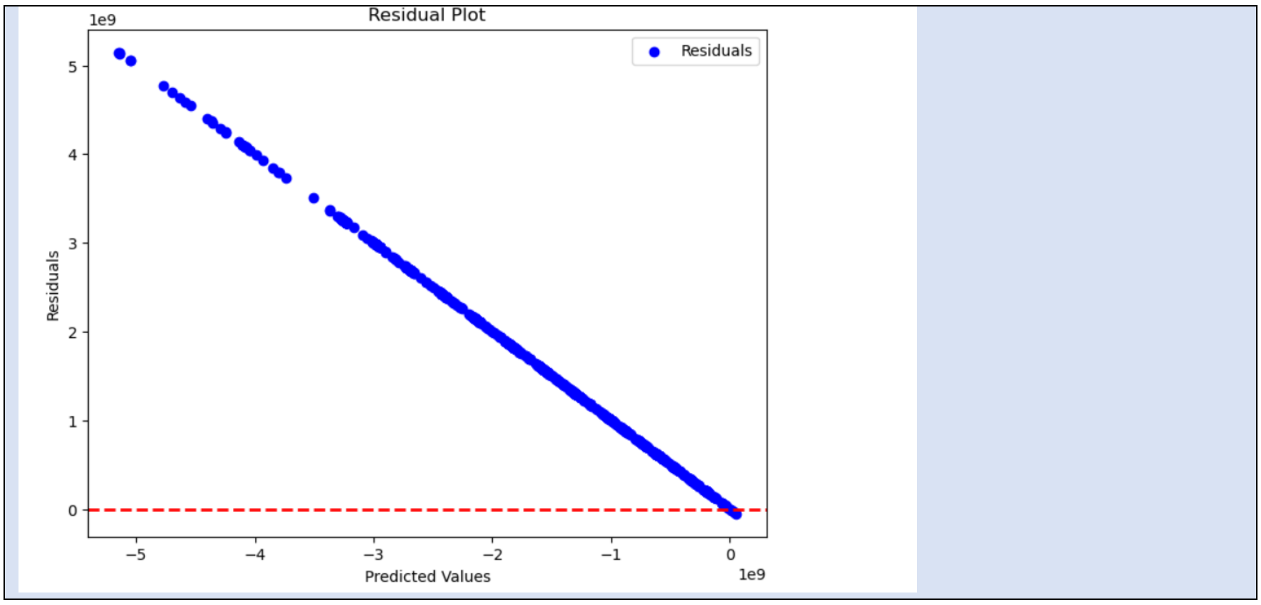
Assess how sensitive the model is to changes in input features.
Example: Testing how much a small change in "hours studied" affects predictions.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
import pickle

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

# Split columns properly
df = df.iloc[:, 0].str.split(',', expand=True)
df.columns = [
    'brand', 'model', 'owner', 'year', 'km_driven', 'fuel', 'seller_type',
    'transmission', 'mileage', 'engine', 'max_power', 'seats', 'torque_value',
    'torque_rpm_value', 'selling_price'
]

# Convert relevant columns to numeric
df[['year', 'km_driven', 'mileage', 'engine', 'max_power', 'seats',
    'torque_value', 'torque_rpm_value', 'selling_price']] = df[['year',
    'km_driven', 'mileage', 'engine', 'max_power', 'seats',
    'torque_value',         'torque_rpm_value',         'selling_price']].apply(pd.to_numeric,
errors='coerce')

# Selecting features and target
X = df.drop(columns=['brand', 'model', 'selling_price'])
y = df['selling_price']

# One-hot encode categorical columns
categorical_cols = ['owner', 'fuel', 'seller_type', 'transmission']
X = pd.get_dummies(X, columns=categorical_cols, drop_first=True)

# Splitting data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize numerical features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train linear regression model
model = LinearRegression()
model.fit(X_train_scaled, y_train)
# Predictions
y_pred = model.predict(X_test)
# Residuals
residuals = y_test - y_pred
# Print model coefficients
coefficients = pd.DataFrame({
    'Feature': X.columns,
    'Coefficient': model.coef_
})

print("Model Coefficients:")
print(coefficients)

# Plot residuals
plt.figure(figsize=(8, 6))
plt.scatter(y_pred, residuals, color='blue', label='Residuals')
plt.axhline(y=0, color='red', linestyle='--', linewidth=2)
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.title('Residual Plot')
plt.legend()
plt.show()
from sklearn.metrics import mean_squared_error
# Evaluate the model
r2 = r2_score(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
print(f"R² Score (Accuracy): {r2:.2f}")
print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")
```

Output:



```
Model Coefficients:
               Feature   Coefficient
0                 year  1.050584e+05
1            km_driven -2.805775e+04
2              mileage  1.332198e+04
3               engine -2.154762e+04
4            max_power  1.068552e+05
5                seats -3.637979e-11
6         torque_value  3.781540e+03
7     torque_rpm_value -2.736277e+04
8              owner_1 -1.142309e+03
9              owner_2 -1.385979e+04
10             owner_4 -7.971849e+03
11              fuel_1  3.428960e+04
12              fuel_2  5.622281e+03
13              fuel_3  5.504424e+03
14       seller_type_1 -5.251222e+03
15       seller_type_2  5.350634e+03
16      transmission_1 -1.677186e+04
```



Points to Remember

- The model's predictions are evaluated on test data to check generalization. It is also tested on new data to ensure it performs well in real-world scenarios.
- Visualization tools like Seaborn and Matplotlib help compare predicted vs. actual values. Common graphs include scatter plots for regression and confusion matrices for classification.
- For regression, key metrics include, RMSE and Accuracy which help evaluate how closely predictions align with actual values.

Application of learning 2.4.

As a trainee in machine learning, you're tasked to evaluate the trained machine learning model solution for your school platform of analysing the student performance based on each academic year by performing the following tasks:

- Predict result on the test data and new data (unseen data)
- Visualize the prediction
- Analyse evaluation metrics using accuracy and Root Mean Square Error (RMSE)
- Interpret the model

Indicative content 2.5: Tuning Hyperparameters

Duration: 6hrs

Theoretical Activity 2.5.1: Tuning Hyperparameters based on performance

Tasks:

1: Answer the following questions related to Tuning Hyperparameters based on performance

i. Define Hyperparameters search space

ii. Choose performance metric

iii. Perform hyperparameter search

2: Provide the answer for the asked questions and write them on papers.

3: Present the findings/answers to the whole class and trainer

4: For more clarification, read the key readings 2.5.1. In addition, ask questions where necessary

Key readings 2.5.1. Tuning Hyperparameters

Hyperparameter tuning is the process of selecting the optimal values for a machine learning model's hyperparameters. Hyperparameters are settings that control the learning process of the model, such as the learning rate, the number of neurons in a neural network, or the kernel size in a support vector machine. The goal of hyperparameter tuning is to find the values that lead to the best performance on a given task.

Tuning hyperparameter involves the following steps:

- **Define Hyperparameters Search Space**

The hyperparameter search space refers to the range or set of values that are considered for each hyperparameter during the tuning process. It defines the possible configurations of hyperparameters that a search algorithm (e.g., Grid Search, Random Search) explores to find the best combination for optimal model performance.

Grid Search Hyperparameter

Grid search is a method for hyperparameter optimization that involves specifying a list of values for each hyperparameter that you want to optimize, and then training a model for each combination of these values. For example, if you want to optimize two hyperparameters, alpha and beta, with grid search, you would specify a list of values for alpha and a separate list of values for the beta.

Randomized Search for Hyperparameter

Randomized search is another method for hyperparameter optimization that can be more efficient than grid search in some cases. With randomized search, instead of specifying a list of values for each hyperparameter, you specify a distribution for each hyperparameter. The randomized search algorithm will then sample values for each hyperparameter from its corresponding distribution and train a model using the sampled values.

- **Choose Performance Metric**

Evaluating the performance of a Machine learning model is one of the important steps while building an effective ML model. To evaluate the performance or quality of the model, different metrics are used, and these metrics are known as performance metrics or evaluation metrics.

Performance Metrics for linear regression

Performance Metrics for Regression

In regression analysis, performance metrics are used to evaluate how well a model predicts continuous outcomes. Here are some commonly used metrics:

1. Root Mean Squared Error (RMSE)

Definition: The square root of the MSE.
Formula:
RMSE = √MSE
Interpretation: Provides an error value in the same units as the target variable, making it easier to interpret.

2. R-squared (Coefficient of Determination)

Definition: Represents the proportion of variance in the dependent variable that is explained by the independent variables in the model.

- **Perform Hyperparameter Search**

Two of the most widely used methods performing hyperparameter search are random search and grid search
1. Random Search
Random search samples hyperparameters from a specified distribution. It is more efficient than grid search as it does not try every combination. Usage: Random search is ideal for discovering hyperparameter combinations that might give a better accuracy. One of the drawback is that it requires more time to execute compared to grid search.

The syntax for Random Search:

```
from sklearn.model_selection import RandomizedSearchCV
from sklearn.linear_model import Lasso

param_dist = {'alpha': [0.1, 1.0, 10.0, 100.0]}
lasso = Lasso()
random_search = RandomizedSearchCV(lasso, param_dist, n_iter=10, cv=5)
random_search.fit(X_train, y_train)
```

2. Grid Search

Grid search is an exhaustive search method where we specify a set of hyperparameters and try all possible combinations. It is simple but can be computationally expensive. It ensures thorough exploration of the parameter space but can be computationally intensive.

The pseudocode for grid search is:

```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import Ridge

param_grid = {'alpha': [0.1, 1.0, 10.0, 100.0]}
ridge = Ridge()
grid_search = GridSearchCV(ridge, param_grid, cv=5)
grid_search.fit(X_train, y_train)
```

1. performing hyperparameter Using RandomizedSearch for Linear Regression

Let's make use of Scikit-Learn's RandomizedSearchCV to search for the best combination of hyperparameter values.

```
from sklearn.model_selection import RandomizedSearchCV
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split

dataset_url = "https://media.geeksforgeeks.org/\
wp-content/uploads/20240617221743/cars.csv"
df_cars = pd.read_csv(dataset_url)

# Independent variables
X = df_cars[["Year", "Kilometers_Driven", "Mileage",
        "Engine", "Power", "Seats"]]

# Dependent variable
```

```
Y = df_cars["Price"]

# Split the data
X_train, X_test, y_train, y_test = train_test_split(
    X, Y, test_size=0.2, random_state=0)

model = LinearRegression()

param_space = {'copy_X': [True,False],
               'fit_intercept': [True,False],
               'n_jobs': [1,5,10,15,None],
               'positive': [True,False]}

random_search = RandomizedSearchCV(model, param_space, n_iter=100, cv=5)
random_search.fit(X_train, y_train)

# Parameter which gives the best results
print(f"Best Hyperparameters: {random_search.best_params_}")

# Accuracy of the model after using best parameters
print(f"Best Score: {random_search.best_score_}")
```

Output:

```
Best Hyperparameters: {'positive': False, 'n_jobs': 1, 'fit_intercept': True, 'copy_X': True}
Best Score: 0.7510045389329963
```

3. **Performing hyperparameter Using Using GridSearch for Linear Regression**

```
from sklearn.model_selection import GridSearchCV

import pandas as pd

from sklearn.linear_model import LinearRegression

from sklearn.metrics import r2_score

from sklearn.model_selection import train_test_split



dataset_url = "https://media.geeksforgeeks.org/\

wp-content/uploads/20240617221743/cars.csv"
```

```
df_cars = pd.read_csv(dataset_url)


# Independent variables
X = df_cars[["Year", "Kilometers_Driven", "Mileage",
          "Engine", "Power", "Seats"]]
# Dependent variable
Y = df_cars["Price"]
# Split the data
X_train, X_test, y_train, y_test = train_test_split(
   X, Y, test_size=0.2, random_state=0)
model = LinearRegression()
param_space = {'copy_X': [True,False], 'fit_intercept': [True,False],
              'n_jobs': [1,5,10,15,None], 'positive': [True,False]}
grid_search = GridSearchCV(model, param_space, cv=5)
grid_search.fit(X_train, y_train)
# Parameter which gives the best results
print(f"Best Hyperparameters: {grid_search.best_params_}")
# Accuracy of the model after using best parameters
print(f"Best Score: {grid_search.best_score_}")
```

Practical Activity 2.5.2: Evaluate performance

Task:

1: You are requested to go to the computer lab to evaluate the machine learning model performance.

2: Apply safety precautions

3: Present out the steps to evaluate the machine learning model performance.

4: Referring to the steps provided, evaluate the machine learning model performance.

5: Present your work to the trainer and whole class

6: Read key reading 2.5.2 and ask clarification where necessary

Key readings 2.5.2. Performance Evaluation

- **Evaluate performance**

  Evaluate the performance of a machine learning model, identify the best parameters through hyperparameter tuning, predict on validation data, and apply evaluation metrics on both validation and test datasets.

- **Steps followed to evaluate Model Performance**

1. Get the Best Parameters

Identify the optimal hyperparameters for the model, typically using methods like Grid Search or Random Search.

Tune hyperparameters (e.g., alpha in Ridge or Lasso regression) to minimize error or maximize the chosen performance metric (e.g., R^2, MSE).

Example: You might find that alpha = 0.1 is the best for Ridge Regression.

2. Get the Best Model

After tuning the hyperparameters, the best model refers to the one trained with those optimal hyperparameters.

After finding the best parameters, use them to train the final model.

Example: If alpha = 0.1 was found to be the best, you would train a Ridge model using this value of alpha.

3. Predict on Validation Data

Use the trained model to make predictions on a validation dataset, which was not seen during training.

Split the dataset into training and validation sets.

Use the best model (trained with optimal hyperparameters) to predict the target values on the validation set.

Example: For a validation set X_val, use model.predict(X_val).

4. Apply Evaluation Metrics on Validation Data

Assess the model's performance on the validation data using metrics like MSE, MAE, R^2, etc.

Use appropriate metrics based on the problem type (e.g., Mean Squared Error (MSE) or R^2 for regression).

Example: For R^2, you can compute r2_score(y_val, y_pred) where y_val is the true target values and y_pred is the predicted values from the model.

5. Apply Evaluation Metrics on Test Data

After the model has been tuned and validated, evaluate it on the test data (data that was never seen during training or validation) to check how well it generalizes.

Apply the trained model on the test set, and compute the same evaluation metrics used for the validation set.

Example: For MSE, use mean_squared_error(y_test, y_pred_test) where y_test is the true values from the test set.

Practical example that summarises above steps:

```
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

# Sample data
from sklearn.datasets import make_regression
X, y = make_regression(n_samples=1000, n_features=10, noise=0.1,
random_state=42)

# Split data into training, validation, and test sets
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3,
random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5,
random_state=42)

# Define model and hyperparameter grid
ridge = Ridge()
param_grid = {'alpha': [0.001, 0.01, 0.1, 1, 10, 100]}

# Grid Search for best parameters
grid_search = GridSearchCV(ridge, param_grid, cv=5,
scoring='neg_mean_squared_error')
grid_search.fit(X_train, y_train)

# Get the best model
best_model = grid_search.best_estimator_

# Predict on validation data
y_val_pred = best_model.predict(X_val)
```

```
# Apply evaluation metrics on validation data
val_mse = mean_squared_error(y_val, y_val_pred)
val_r2 = r2_score(y_val, y_val_pred)

# Predict on test data
y_test_pred = best_model.predict(X_test)

# Apply evaluation metrics on test data
test_mse = mean_squared_error(y_test, y_test_pred)
test_r2 = r2_score(y_test, y_test_pred)

# Print the results
print(f"Best Hyperparameters: {grid_search.best_params_}")
print(f"Validation MSE: {val_mse}, R²: {val_r2}")
print(f"Test MSE: {test_mse}, R²: {test_r2}")
```

Summary of Steps

1. Get the best parameters: Tune hyperparameters with techniques like Grid Search.

2. Get the best model: Use the optimal hyperparameters to train the final model.

3. Predict on validation data: Generate predictions for the validation set.

4. Apply evaluation metrics on validation: Calculate performance metrics (e.g., MSE, R^2) on validation data.

5. Apply evaluation metrics on test data: Evaluate how well the model generalizes to unseen data by applying metrics to the test set.

Practical Activity 2.5.3: Resolving Potential Bias

Task:

1: You are requested to go to the computer lab to Resolve Potential Bias of machine learning model.

2: Apply safety precautions

4: Referring to the key reading 2.5.3 train the model and mention and resolve the bias presents in your model.

5: Present your work to the trainer and whole class

6: Read key reading 2.5.3 and ask clarification where necessary

Key readings 2.5.3: Resolving Potential Bias

When building and evaluating machine learning models like **linear regression**, it's essential to address issues of **bias**, **underfitting**, and **overfitting**, as they can significantly affect the model's performance.

➢ **Bias in Machine Learning**

Bias refers to the error introduced by approximating a real-world problem with a simplified model. A **high bias** model makes strong assumptions and may underfit the data, failing to capture the underlying patterns in the data.

Symptoms of Bias:

- Poor performance on both training and test data (high error).
- The model is too simple to capture the complexity of the data.

How to Resolve Bias:

- **Use a more complex model**: If you're using a simple model like linear regression, consider using more flexible models (e.g., polynomial regression, decision trees, or ensemble methods) to better capture complex relationships.
- **Feature engineering**: Add more relevant features or use polynomial features to allow the model to learn more complex patterns.
- **Increase model complexity**: For linear regression, you might try increasing the degree of the polynomial in polynomial regression.

2. Underfitting

Underfitting occurs when a model is too simple to capture the underlying patterns in the data. This often results from using a model that is too constrained or from having too few features.

Symptoms of Underfitting:

- High bias, poor performance on both training and test data.
- The model is too simplistic and doesn't capture the data trends well.

➢ **How to Resolve Underfitting:**

Increase model complexity: Use more complex models (e.g., polynomial regression or switching to decision trees or neural networks if appropriate).

Add more features: Include more relevant features or perform feature engineering.

Reduce regularization: If you're using regularization (e.g., Ridge or Lasso), try reducing the regularization strength (alpha parameter) to allow the model to fit the training data better.

Improve data quality: Sometimes, poor data preprocessing or insufficient data can cause underfitting. Try cleaning the data, handling missing values, and removing outliers.

3. Overfitting

Overfitting occurs when a model learns the noise or random fluctuations in the training data rather than the actual underlying patterns. This happens when the model is too complex relative to the amount of training data.

Symptoms of Overfitting:

- Very low training error but high test error.
- The model fits the training data too closely, including noise, leading to poor generalization to new data.

➢ **How to Resolve Overfitting:**

Reduce model complexity: Use simpler models or reduce the number of features. For example, in linear regression, use fewer polynomial features or apply feature selection techniques to reduce the dimensionality of the model.

Increase regularization: For models with regularization (e.g., Ridge or Lasso), increasing the regularization strength (alpha in Ridge/Lasso) can help prevent overfitting by penalizing excessively complex models.

Cross-validation: Use techniques like **k-fold cross-validation** to evaluate the model performance across multiple subsets of the data. This helps identify if the model is overfitting the training data.

Use more training data: More data can help the model generalize better and avoid memorizing the training set.

Points to Remember

- The model's predictions are evaluated on test data to check generalization. It is also tested on new data to ensure it performs well in real-world scenarios.
- To find the optimal combination for the best model performance use the best hyperparameter.

- Always evaluate performance by identifying the best parameters and model, predicting on validation data, and applying evaluation metrics on both validation and test data to ensure model effectiveness.
- applying evaluation metrics on both validation and test data to ensure model effectiveness for resolving potential bias.
- For model to perform well should not present any bias.

Application of learning 2.5.

Referring to the trained model in application learning 2.4, you are tasked to evaluate the performance metric and resolve potential bias of machine learning model solution for your school platform of analysing the student performance based on each academic year by performing the following tasks:

i. Tuning Hyperparameter Using GridSearch
ii. Apply Evaluation Metrics on Validation Data
iii. Apply Evaluation Metrics on test Data
iv. Detect the overfitting and underfitting of a model

Learning outcome 2 end assessment

Theoretical assessment

I). Read the following statements concerning machine learning model and circle the letter corresponding to the right answer

Q1: The primary purpose of Linear Regression in machine learning is:

A) Classifying categorical data

B) Predicting continuous values

C) Clustering data points

D) Reducing dimensionality

Q2: Logistic Regression is mainly used for which type of problem?

A) Regression problems

B) Clustering problems

C) Classification problems

D) Dimensionality reduction problems

Q3: The primary goal of a Decision Tree in machine learning is

A) To perform linear regression

B) To group similar data points

C) To split data based on feature values to make decisions

D) To reduce the dimensionality of the dataset

Q4: The key feature of the Random Forest algorithm is:

A) It uses a single decision tree for classification

B) It builds multiple decision trees and combines their predictions

C) It only works for linear data

D) It is used for clustering tasks

Q5: The main goal of Support Vector Machine (SVM) algorithm is:

A) To minimize the distance between data points and the centroid

B) To find the hyperplane that best separates classes

C) To perform feature extraction from the data

D) To find clusters in the data

Q6: K-Nearest Neighbors (KNN) algorithm is used for:

A) Clustering

B) Regression and classification

C) Feature extraction

D) Dimensionality reduction

Q7: The primary goal of K-Means Clustering algorithm is:
A) To classify data points into predefined classes
B) To minimize the distance between data points and their corresponding cluster centroid
C) To reduce the dimensionality of the dataset
D) To detect anomalies in the data

Q8: Which of the following is a limitation of K-Means Clustering?
A) It can only work with categorical data
B) It requires knowing the number of clusters (K) in advance
C) It is sensitive to the size of the dataset
D) It does not require centroids for clustering

Q9: The main purpose of Principal Component Analysis (PCA) is:
A) To perform classification tasks
B) To reduce the dimensionality of the dataset by transforming it into a new coordinate system
C) To find the best hyperplane that separates the data into classes
D) To assign data points to predefined clusters

Q10: What is the purpose of Anomaly Detection in machine learning?
A) To classify data into multiple categories
B) To identify rare items, events, or observations that deviate significantly from the majority of the data
C) To reduce the dimensionality of the dataset
D) To perform regression on the dataset

Q11: Which algorithm is commonly used for Anomaly Detection?
A) Support Vector Machine (SVM)
B) Random Forest
C) Isolation Forest
D) k-Nearest Neighbors (KNN)

Q12: Semi-supervised Learning is?
A) A method where the model learns from labeled data only
B) A method where the model learns from unlabeled data only
C) A method where the model learns from both labeled and unlabeled data
D) A method that combines clustering and regression

Q13: Which of the following is an advantage of Semi-supervised Learning?
A) It requires no labeled data
B) It reduces the need for a large amount of labeled data
C) It always performs better than supervised learning
D) It does not require any data preprocessing

Q14: The primary purpose of the Pandas library in machine learning is:

A) Performing matrix operations
B) Building and training machine learning models
C) Data manipulation and analysis, especially with tabular data
D) visualizing data with charts and graphs

Q15: Which Pandas function is used to load a CSV file into a DataFrames?
A) Pd.load_csv ()
B) pd.read_csv()
C) pd.import_csv()
D) pd.open_csv()

Q16: The primary use of Numpy library in machine learning?
A) Data visualization
B) Numerical computation and array manipulation
C) Building neural networks
D) Data collection from web APIs

Q17. Which of the following is the correct way to create a 1-dimensional Numpy array from a Python list?
A) np.array([1, 2, 3, 4])
B) np.array([1:4])
C) np.newarray(1, 2, 3, 4)
D) np.array_list(1, 2, 3, 4)

Q18: Which function in Scikit-Learn is used to split a dataset into training and testing sets?
A) train_test_split()
B) data_split()
C) train_test()
D) split_data()

Q19: The main purpose of Requests library in machine learning projects is

A) Loading and preprocessing datasets
B) Sending HTTP requests to access data from APIs or web servers
C) Building machine learning models
D) Performing matrix operations

Q20: Which of the following libraries is used primarily for creating arrays and performing fast numerical operations on large datasets?
A) Pandas
B) Seaborn
C) Numpy
D) Scikit-Learn

Answer	Column A	Column B
	a) Feedforward Neural Networks (FNNs)	**1.** A neural network architecture used primarily for sequence data, where the output of one layer depends on the previous output in the sequence. This allows it to capture temporal dependencies in the data.
	b) Random Forest	**2.** A type of neural network that processes data in one direction (from input to output) without looping back, commonly used for tasks such as classification and regression.
	c) Convolutional Neural Networks (CNNs)	**3.** The general term for a network of neurons connected by synapses, typically including an input layer, hidden layers, and an output layer. This structure is used in various fields such as image recognition and natural language processing
	d) Recurrent Neural Networks (RNNs)	**4.** A neural network architecture commonly used for image processing tasks, utilizing convolutional layers to detect spatial features in images
	e) k-Nearest Neighbors (KNN)	
	f) Neural Networks (Artificial Neural Networks - ANNs)	

III. Read the following statements concerning machine learning algorithm and fill in the Gap-Space Questions

1. ____________ is used to predict a continuous value based on one or more input features.

Choices:

A) Classification

B) Regression

C) Clustering

D) Reinforcement Learning

2. In ___________, the goal is to group similar data points together based on their characteristics.

Choices:

A) Regression

B) Clustering

C) Classification

D) Dimensionality Reduction

3. Logistic Regression is a popular algorithm used for ___________ tasks.

Choices:

A) Classification

B) Regression

C) Clustering

D) Reinforcement Learning

4. ___________ is the machine learning task where the model predicts a label from a set of predefined categories.

Choices:

A) Regression

B) Clustering

C) Classification

D) Reinforcement Learning

Iv). Read the following statement-based on machine learning metrics, Answer on the following question by True or False

1. Accuracy is the proportion of true results (both true positives and true negatives) among the total number of cases examined.

2. Precision is the ratio of correctly predicted positive observations to the total predicted positive observations**.**

3. Recall measures the proportion of actual positives that were correctly identified by the model.

4. The F1 Score is the harmonic mean of Accuracy and Precision**.**

5. Mean Absolute Error (MAE) measures the average of the absolute differences between predicted and actual values**.**

6. Root Mean Squared Error (RMSE) penalizes larger errors more than MAE because of the square operation.

7. R Squared score measures how well the independent variables explain the variance in the dependent variable, and it ranges from -∞ to 1.

8. Adjusted R Squared score adjusts for the number of predictors in the model and prevents overfitting**.**

9. In model interpretation, it is important to understand how each independent variable contributes to the model's predictions.

10. Precision is a more important metric than Recall when it is critical to avoid false negatives.

Practical assessment

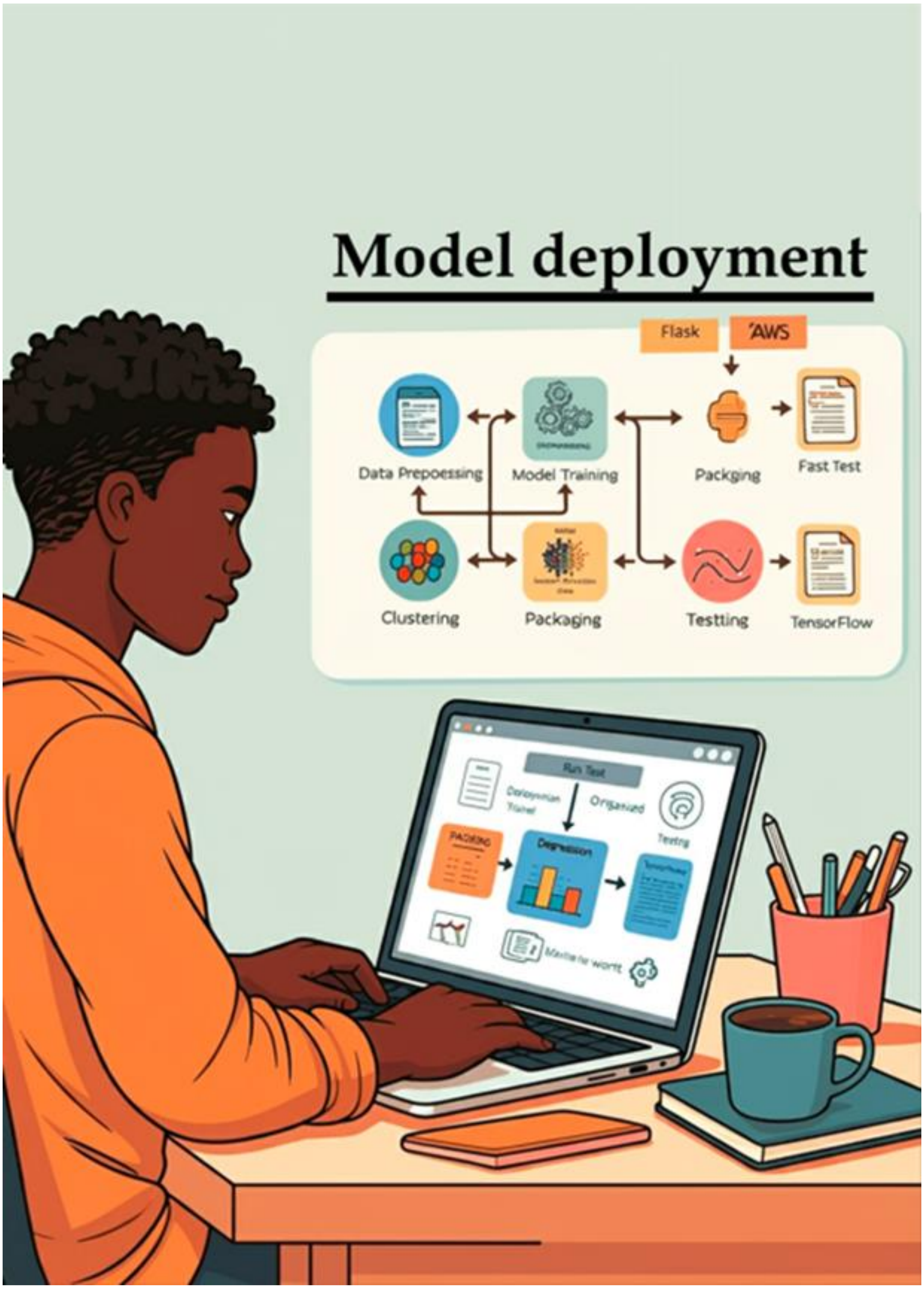
As a machine learning trainee, your responsibility is to create a solution for analysing student performance over academic years from 2000 to 2024 for your school platform. You will work with a dataset containing student performance data and perform the following tasks.:

- Select any machine learning algorithm to be used
- Load a dataset Using pandas, NumPy, Scikit-Learn, Seaborn and Requests and io
- Split dataset using train set, test set and validation set
- Initialize model
- Fit the training data into a model
- Predict result on the test data and new data (unseen data)
- Visualize the prediction
- Analyse evaluation metrics using accuracy, Root Mean Square Error (RMSE)
- Interpret the model
- Tuning Hyperparameter Using GridSearch
- Apply Evaluation Metrics on Validation Data
- Apply Evaluation Metrics on test Data
- Detect the overfitting and under fitting of a model

References

Géron, A. (2019). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts,Tools, and Techniques to Build Intelligent Systems. O'Reilly Media
James, G., Witten, D., Hastie, T., & Tibshirani, R. (2017). An Introduction to Statistical Learning: with Applications in R. Springer.
Müller, A. C., & Guido, S. (2016). Introduction to Machine Learning with Python: A Guide for Data Scientists. O'Reilly Media.
Raschka, S., & Mirjalili, V. (2019). Python Machine Learning: Machine Learning and Deep Learning withPython, scikit-learn, and TensorFlow 2. Packt Publishing.
Sarkar, D., Raghav, B., & Tushar, S. (2017). Practical Machine Learning with Python: A Problem-Solver's Guide to Building Real-World Intelligent Systems.

Learning Outcome 3: Perform Model Deployment



Indicative contents
3.1 Selection of model deployment method
3.2 Integration of model file
3.3 Delivering Prediction to the clients

Key Competencies for Learning Outcome 3: Perform Model Deployment

Knowledge	Skills	Attitudes
• Description of model deployment methods • identifying system specifications • identifying model specification • Description of Integration of model file • Description of API endpoint and data format. • Description of Prediction to the clients	• Analysing different deployment methods • Choosing the best application type and technology stack. • Evaluating the model's requirements and compatibility. • Integrating model with existing systems • Manage data formats and API usage. • Implementing model communication. • Deploying the model and continuously monitor its performance.	• Critical thinker in choosing the best technology • Being Innovative • Being creative in • Having Problem-Solving • Being collaborator and Communicator • Be attentive to Security in deploying the model.

Duration: 20 hrs

Learning outcome 3 objectives:

By the end of the learning outcome, the trainees will be able to:

1. Describe clearly deployment method based on system requirements.
2. Identify correctly system specification based on model deployment.
3. Identify clearly model specification based on system requirement.
4. Integrate effectively model into an existing application architecture.
5. Track/ analyse effectively performance metrics of model based on system requirements.
6. Implement correctly a comprehensive testing model before deployment to ensure reliability and accuracy
7. Describe clearly compliance with relevant industry standards and regulations related to data privacy, security, and ethical AI practices.

Resources

Equipment	**Tools**	**Materials**
● Computer ● Projector ● Storage Devices	● Browser ● Postman ● Internet	● Datasets, Python Distribution (CPython, Anaconda), Python IDEs (Jupyter Notebook, PyCharm, Visual Studio Code, or Spyder), Machine Learning Libraries (NumPy, Pandas, Scikit-Learn, TensorFlow or PyTorch, Matplotlib, Seaborn, Keras, SciPy), Cloud platform, web application framework, web server

Indicative content 3.1: Selection of model deployment method

Duration: 5 hrs

Theoretical Activity 3.1.1: Description of Model Deployment and Specifications

Notes to the trainer:

1: Answer the following questions related to model deployment methods:

i. Define the term Model Deployment in Machine Learning

ii. Discuss on benefit of Model Deployment in Machine Learning

2: Provide the answer for the asked questions and write them on papers.

3: Present the findings to the whole class

4: For more clarification, read the key readings 3.1.1. In addition, ask questions where necessary.

Key readings 3.1.1 Description of Model Deployment and Specifications

- **Description of model deployment methods**

✓ **Definition**

Model deployment in machine learning is the process of integrating a trained machine learning model into an existing production environment where it can take in an input and return an output.

The purpose of deploying your model is so that you can make the predictions from a trained machine learning model available to others.

✓ **Benefits**

Model deployment offers several benefits across various industries and applications:

- **Data-Driven Decision Making:**

Provides actionable insights from data, enabling businesses to make more informed decisions across various departments.

- **Accessibility:**

Models can be accessed via APIs across platforms.

- **Automation**: Models can automate tasks, from detecting anomalies to forecasting trends, reducing manual efforts and improving accuracy.

- **Continuous Learning**: With deployment in production, models can be updated and retrained using fresh data, improving over time and adapting to changes.

- **Enhanced Security**: In fields like fraud detection, deployed models can monitor transactions and activities in real-time, providing instant alerts on suspicious behaviour and enhancing system security.

- **Type of application for deployment**

Before deploying the model, you must first understand the language in which the existing system is built

1. Web application

A web application is accessed through a web browser and typically runs on a server.

Example: E-commerce websites, recommendation systems, chatbots.

Characteristics:

- Accessible from anywhere with internet connectivity.
- Hosted on servers (e.g., cloud or on-premises).

This is possible methods of deployment model through web application:

- **RESTful APIs or GraphQL:**

The model runs on a backend server, and predictions are delivered to the web app via API requests.

- **Serverless Deployment:**

Use cloud functions like AWS Lambda or Google Cloud Functions for handling model inference.

- **Cloud Hosting:**

Host the model on platforms like AWS SageMaker, Azure ML, or Google AI Platform

2. Mobile Application

Applications designed to run on mobile devices (e.g., smartphones, tablets).

Example: Voice assistants, image recognition apps, fitness trackers.

Characteristics:

- Limited processing power and storage.
- Requires lightweight models for efficiency.

This is possible methods of deployment model through web application:

- **On-Device Deployment:**

Deploy a pre-trained and optimized model (e.g., TensorFlow Lite, PyTorch Mobile) to the device for offline use.

- **Cloud-Based Deployment:**

The mobile app sends data to a server hosting the model and receives predictions via APIs.

- **Hybrid Deployment**:

Use on-device inference for simple tasks and cloud inference for complex tasks.

3. Standalone Program

Standalone program: A self-contained application installed and run on a user's device (e.g., PC, laptop). Or A stand-alone program that runs independently on a local machine or a server.

Example: Desktop analytics software, financial forecasting tools.

Characteristics:

- ○ Runs independently of internet connectivity (optional internet integration).
- ○ Can utilize the full power of the user's hardware.

This is possible methods of deployment model through standalone program:

- **Local Deployment:**

Embed the model within the program using libraries like Scikit-learn, TensorFlow, or PyTorch.

- **Packaged Executable:**

Package the program and model together (e.g., with PyInstaller for Python applications).

- **Server-Connected Deployment:**

Allow the program to fetch predictions from a remote model via APIs when needed.

4. **Embedded System**

A dedicated system designed for specific tasks, often integrated into hardware.

Example: Smart home devices, autonomous vehicles, industrial robots.

Characteristics:

- ○ Limited hardware resources (e.g., low power, low memory).
- ○ Often operates in real-time environments.

Possible Deployment Methods through embedded system:

- **On-Device Deployment:**

Use lightweight frameworks like TensorFlow Lite, ONNX, or Caffe2 for edge deployment.

- **Edge Computing:**

Deploy models on edge devices (e.g., NVIDIA Jetson, Raspberry Pi) for localized processing.

Cloud-Assisted Deployment:
Use the cloud for complex tasks while the embedded system handles lightweight inference locally.

Technology (programming languages and frameworks)

The choice of programming languages and frameworks depends heavily on the specific application type, performance requirements, development team expertise, and the desired deployment environment.

Identification of model specifications

- **Model Type:** The specific algorithm used (e.g., linear regression, decision tree, support vector machine, neural network).
- **Hyperparameters**: Parameters that control the learning process of the model (e.g., learning rate, number of layers, regularization strength).
- **Model Architecture:** For complex models like neural networks, the specific architecture (e.g., number of layers, number of neurons per layer, activation functions).
- **Model Size:** The size of the model file (e.g., in megabytes or gigabytes), which can impact deployment and inference speed.
- **Computational Complexity:** The amount of computation required to make a prediction with the model.

Size of the dataset

The size of a dataset in machine learning refers to the number of data points (examples) and features (attributes or variables) that are used to train and test a machine learning model.

Key Aspects of Dataset Size

Number of Data Points (Samples):

- The more data points, the better the model can learn patterns and make accurate predictions.
- Small datasets may lead to overfitting, where the model memorizes the training data instead of generalizing to new data.
- Large datasets can help the model generalize better, but they may also require more computational resources and time for training.

Number of Features (Attributes):

Features represent the different characteristics of the data (e.g., age, salary, weather conditions). The more features a dataset has, the more information the model can use to make decisions.

However, too many features can lead to high-dimensional data, which may increase complexity and lead to issues like overfitting, requiring techniques like feature selection or dimensionality reduction.

- **Memory limitations**

  - **RAM:** The amount of available memory on the deployment platform is crucial for loading the model, processing data, and making predictions.
  - **GPU Memory:** If using a GPU for inference, the amount of GPU memory available will limit the size and complexity of models that can be deployed.
  - **Virtual Memory:** The amount of virtual memory available can also impact performance, especially for large models or datasets.

- **Computing power**

  - **CPU:** The processing power of the CPU will affect the speed of inference.
  - **GPU:** GPUs can significantly accelerate inference for deep learning models.
  - **Specialized Hardware:** Hardware accelerators like TPUs (Tensor Processing Units) can further improve performance for specific tasks

Formatting the Model File

When a machine learning model is trained, it needs to be saved and later deployed for inference or further training. The format in which the model is saved depends on the framework used for building the model, its deployment environment, and interoperability needs. Each format has unique characteristics, strengths, and use cases.

Here are the common model file formats:

- ✓ S**cikit-Learn Model Format:** Scikit-Learn is a popular library for traditional machine learning algorithms such as decision trees, random forests, and SVMs. Models trained in Scikit-Learn are often saved using Python's **pickle** or **joblib** libraries.
  Example: **model.pkl** (Pickle)
  model.joblib (Joblib, more efficient for larger models)
  Advantages:
- Simple and quick to serialize and deserialize models.
- Suitable for quick deployment in Python environments.
  scikit-Learn is limited to Python; not easily interoperable with other platforms or languages. Good for quick prototyping, traditional machine learning tasks, and Python-centric applications.
- ✓ **TensorFlow SavedModel:** TensorFlow is widely used for deep learning, neural networks, and other large-scale machine learning tasks. . Models trained in TensorFlow are often saved using **SavedModel** (Directory with model metadata and weights) Can

be exported to **.h5** for simpler models.

Advantages:

- Platform-independent, can be used for both training and inference.
- Compatible with TensorFlow Serving, TensorFlow.js (for browser deployment), and TensorFlow Lite (for mobile/edge deployment).
- Supports deployment across multiple environments (cloud, web, mobile).

TensorFlow models can be complex and may require a deeper learning curve. Ideal for production environments, large-scale applications, and models requiring deployment across different platforms.

✓ **ONNX (Open Neural Network Exchange)** is an open format designed for model interoperability between different machine learning frameworks like TensorFlow, PyTorch, and Scikit-Learn. File Format **model.onnx** (ONNX model file)

Advantages:

- Allows for easy conversion between different ML frameworks.
- Compatible with a wide range of tools, including cloud-based and edge platforms.
- Optimized for deployment in environments with hardware accelerators, such as GPUs and TPUs.

Not every framework feature is supported in ONNX, so certain models might need simplification. Ideal when working with multiple ML frameworks or when deploying models to environments like IoT devices or different cloud platforms.

✓ **PyTorch PT (TorchScript)** is widely used in research and production for deep learning tasks, offering dynamic computation graphs and easier debugging. File Format **model.pt** or **model.pth** (PyTorch checkpoint file for saving model weights and structure)

Advantages:

- PyTorch's dynamic computation graph makes it easier to debug and modify models.
- Can export models to TorchScript, which makes it possible to run in environments like C++ runtime for production.
- Supported in mobile, cloud, and edge deployment scenarios.

PyTorch may require exporting to other formats (like ONNX) for certain deployment environments. Ideal for research, experimentation, and production environments, especially when using PyTorch-native workflows.

Points to Remember

- Integrating a machine learning model into an existing production environment it’s a crucial thing in machine learning deployment

- while deploy your ML model use the stipulated modes
- While delivering this content, small groups can be used for describing application and technology of model deployment.
- Both memory limitations and computation power play crucial roles in determining how data is processed, models are trained, and which models can be used.
- Applying Common model file formats allow for efficient storage and deployment of trained models.
- While training and saving model, you are always recommended to respect stipulated steps.

Application of learning 3.1

As Student from level, five you are requested to go to computer Lab are given the task with researching and selecting appropriate deployment methods for a machine-learning model, considering factors like cost, performance, scalability, security, and maintenance

Indicative content 3.2: Integration of Model File

Theoretical Activity 3.2.1: Description of model file integration

Tasks:

1: Answer the following questions:

i. Explain the goals of machine learning model integration

ii. What is the importance of checking compatibility

iii. Why do we need to use API endpoint for model integration

2: Provide the answer for the asked questions and write them on papers.

3: Present the findings to the whole class

4: For more clarification, read the key readings 3.2.1. In addition, ask questions where necessary.

Key readings 3.2.1 Description of model file integration

- **Integration goals**

The goal of model file integration is to seamlessly incorporate a trained machine learning model into an application or system for practical use. This involves deploying the model in a production environment, ensuring it is optimized for performance to handle real-time or batch predictions efficiently. It also requires preparing the system for scalability to manage varying loads without performance degradation. Compatibility with existing systems is crucial, as well as ensuring the security of the model and data through proper authentication and encryption

There are important things to care about when we are going to integrate machine learning model into real world application:

✓ **Prediction**

Prediction involves the use of a trained machine learning model to generate outcomes based on new input data. These predictions are the core functionality of the model and can vary depending on the type of problem the model is designed to solve. For instance, in classification tasks, the model predicts the category or label to which an input belongs, such as identifying whether an email is spam or not. In regression tasks, it predicts a continuous value, like forecasting sales revenue based on historical data. In recommendation systems, predictions are used to suggest products, services, or content that a user might find interesting, based on their past behavior or preferences. These predictions are generated by applying the learned patterns from the training phase to new data, providing valuable outputs that applications can use to make decisions or automate processes. Ensuring that these predictions are accurate, timely, and relevant is crucial for delivering meaningful results that enhance the overall user experience and drive business value.

It is like making a smart guess about what will happen in the future based on information you already have. For example, if you've noticed that it usually gets colder in the evening, you might predict that tonight it will be chilly too. In the world of technology, a computer can look at a lot of past information and make a prediction, like whether it will rain tomorrow or what kind of music you might like. It's like when your favorite streaming app suggests a movie you might enjoy – that's a prediction based on what you've watched before! Predictions help you get ready for what's coming next.

- ✓ **Insight**

Insight refers to the meaningful understanding and actionable knowledge derived from the predictions made by a machine learning model. While predictions provide raw data outputs, insights involve analyzing and interpreting these outputs to reveal patterns, trends, and underlying relationships that can inform strategic decisions. For example, insights from customer behavior predictions can help businesses tailor marketing strategies, improve user engagement, and enhance product offerings. Insights gained from trend analysis can help organizations anticipate future demands or market shifts, allowing them to adapt proactively. Additionally, insights can uncover anomalies or outliers, which might indicate potential issues such as fraud or operational inefficiencies. By transforming predictions into insights, businesses can leverage data to make informed decisions, optimize operations, and gain a competitive edge. Effective model integration ensures that insights are easily accessible and actionable, empowering stakeholders to make data-driven decisions that align with their

goals.
it is like the story or meaning you get from looking at data and predictions. Imagine you have a weather app that tells you it will rain tomorrow (that's a prediction). An **insight** would be understanding that if it rains tomorrow, you might need to carry an umbrella, or that it could be a good day for indoor activities. Insights help you make better decisions by explaining what the prediction means for you. So, while a prediction gives you a fact about what might happen, an insight helps you figure out what to do about it.

The integration process aims to ensure that both predictions and insights are not only accurate but also accessible in a form that stakeholders can use to improve outcomes, optimize processes, and enhance user experiences. By effectively integrating a model into a system, businesses can harness the power of data-driven insights to make informed decisions, forecast future trends, and personalize services to meet user needs.

- ✓ **Generate content**

generate content refers to using a trained machine learning model to create new, original data or outputs based on learned patterns. This could include generating text, images, music, or other forms of media. For instance, a text generation model might create articles, summaries, or responses in a conversation, while an image generation model might produce artwork or visual designs. The process involves feeding the model with specific inputs or prompts, and the model then generates content that aligns with the patterns it learned during training. This capability is widely used in applications like chatbots, automated content creation tools, and virtual assistants, where the goal is to produce relevant and engaging content in real-time. Integrating such a model into a system ensures that the content generation is efficient, scalable, and can be tailored to meet the needs of users or the objectives of a business.
Imagine you want to write a story, but instead of writing it yourself, you tell a computer a few details, and it writes the story for you. That’s what generating content is all about. A computer learns from a lot of examples, like stories, songs, or pictures, and then it uses what it learned to make something new. For example, a program could help you write a poem, draw a picture, or even make up a song based on what you like. It’s like having a super creative helper that works really fast!

- ✓ **Data Analysis**

It involves examining and interpreting the data that the machine learning model

uses to make predictions and generate insights. It's the process of understanding the patterns, trends, and relationships in the data before feeding it into the model.
For example, before using a model to predict whether it will rain tomorrow, data analysis might involve looking at past weather data to see how temperature, humidity, and pressure have influenced rainfall in the past. This helps identify what factors are most important for making accurate predictions.

After deploying a machine learning model, data analysis plays a crucial role in monitoring and improving its performance. It helps track the model's predictions and ensures they remain accurate over time; especially as new data is introduced. By analysing the incoming data, you can identify issues such as biases, errors, or changes in data patterns that may affect the model's effectiveness. This ongoing analysis helps detect performance drops, enabling adjustments or retraining to keep the model aligned with real-world conditions. Additionally, data analysis ensures that the data fed into the model remains clean and relevant, optimizing the model's ability to provide valuable insights and predictions.

✓ **Compatibility**

Compatibility in model integration refers to how well a machine learning model interacts with and fits into the existing technology ecosystem, including software, hardware, and infrastructure. When a model is deployed into a real-world system, it needs to communicate properly with other parts of the system (such as databases, front-end applications, or other services) to function correctly. If the model isn't compatible, it can cause errors, slowdowns, or even failures in the system.
There are some cases to take care of while ensuring compatibility success:

✓ **Data Compatibility**

For a model to work well, the data it receives during deployment must be in the right format and structure. The model might expect data to be provided in a specific format (such as a CSV file, JSON, or a database query result). If the data format is inconsistent with the model's requirements, it won't be able to process the information properly, leading to incorrect predictions or errors.
For example, if a model was trained on data where dates are formatted as YYYY-MM-DD and it now receives dates as MM/DD/YYYY, it could fail to interpret the data correctly. Ensuring compatibility with the data is essential to making the model function properly in its new environment.

- ✓ **Software and API Compatibility**

Machine learning models often interact with other software systems or APIs (Application Programming Interfaces). APIs are used to exchange data between the model and other services, like a website or a mobile app. For example, a recommendation system might rely on data from a web application, and the model's outputs (predictions) need to be delivered back through an API.
If the model and the software it's working with have different versions or incompatible technologies, it might cause integration problems. This is why it's important to ensure that the software libraries or frameworks the model uses are compatible with the rest of the system. For instance, if the model was developed using TensorFlow and the system relies on a different machine learning framework, there could be issues with data handling or model execution.

- ✓ **Hardware Compatibility**

Hardware compatibility refers to the model's ability to run efficiently on the system's hardware. Some models, especially deep learning models, require powerful hardware like GPUs (Graphics Processing Units) to function efficiently. If the model is deployed on a machine that doesn't meet the hardware requirements, it may not run at all, or it may run very slowly.
A model needs to be compatible with the available hardware, ensuring that the computational resources are adequate for the model’s size and complexity. For instance, deploying a large image recognition model on a system with limited memory or processing power could cause delays or crashes.

- ✓ **Version Compatibility**

Models and systems often evolve over time, with new versions being developed. Version compatibility ensures that the model and the system can still work together, even as one or both are updated. For example, if the model is updated with new features or algorithms, it needs to be compatible with the existing system. The system should be able to handle these updates without needing a complete overhaul.
This is particularly important in long-term projects where systems are continuously upgraded. A versioning strategy needs to be in place to handle changes in the model, its libraries, or the software it's working with.

- ✓ **Scalability Compatibility**

Scalability refers to the ability of the model to handle increased demand as the system grows. When integrating a model into a system, you must ensure it can scale smoothly, meaning it can continue to perform well as the amount of data,

users, or requests increases.
For example, if a model is serving predictions on an e-commerce platform and the traffic suddenly increases during a sale, the model should be able to handle many more requests without slowing down or crashing. Ensuring compatibility with the system's scaling mechanisms (such as cloud services or load balancing) is essential for maintaining performance.

- ✓ **Cross-Platform Compatibility**

Cross-platform compatibility ensures that the model can function across different devices and environments. For example, the model might need to work both on a web server and on mobile devices, which may have different operating systems and processing capabilities. The model must be able to interact with both platforms without issues, providing consistent performance and results.
This also includes ensuring that the model works across different operating systems, such as Linux, Windows, or macOS, or across cloud environments like AWS, Azure, or Google Cloud.

Interpret API endpoint usage

Understanding API endpoint usage is crucial for facilitating communication between a deployed machine learning model and other system components. API endpoints serve as the access points through which applications can send data to the model, request predictions, and receive results. Each endpoint represents a specific function, such as making predictions or updating model parameters, enabling seamless interaction within a broader software ecosystem.
When a model is integrated into a system, it typically becomes available through a set of API endpoints. For instance, an endpoint like POST /predict allows the system to send input data to the model and retrieve the predicted output. This interaction often involves sending HTTP requests with data payloads formatted in JSON or similar structures. The model processes the data and responds through the endpoint, returning the prediction in a format the application can use.
Ensuring compatibility between the model and the API is essential. The API must handle input data formats and provide outputs that align with the application's requirements. This also includes handling authentication, ensuring that only authorized users can access the model through the API, and managing error handling to address issues like invalid data or system errors gracefully.
In the deployment phase, APIs play a pivotal role in integrating the model into various environments, whether for web applications, mobile apps, or other services. By leveraging API endpoints, developers can build scalable and flexible

systems that utilize machine learning models to enhance functionality, providing insights and predictions to end-users efficiently. Understanding API endpoint usage ensures that the integrated model operates smoothly, delivers accurate results, and enhances the overall system's performance.

Model serving methods

Model serving is the process of making a trained machine learning model available for use in a production environment. It involves deploying the model so it can accept input data, perform computations, and return predictions or insights to users or systems. The goal of model serving is to enable real-time or batch processing of data, allowing applications to leverage the model's capabilities effectively.

Serving Methods

Real-time Serving: The model processes input data and returns predictions almost instantaneously. This method is ideal for applications that require immediate responses, such as recommendation systems or fraud detection.

Batch Serving: The model processes a large volume of input data at once, often scheduled periodically. This method is used in scenarios like generating nightly reports or processing large datasets where real-time responses are not required.

Model loading strategy

It refers to the method by which a machine learning model is loaded into memory and made ready for inference once it has been deployed. An effective loading strategy ensures that the model is available for use with minimal delay and operates efficiently under different workloads.

There are different strategies such as:

Eager Loading:
Eager loading involves loading the machine learning model into memory immediately when the service starts. This strategy ensures that the model is ready to serve predictions as soon as the system is operational, making it ideal

for applications that require high availability and low latency. By having the model preloaded, the first prediction request is handled without any delay, ensuring a seamless user experience. However, eager loading can consume significant memory, especially if multiple models are loaded simultaneously, which may be a concern in resource-limited environments. This strategy is best suited for high-traffic applications where immediate response times are critical.

Lazy Loading:
Lazy loading is a strategy where the model is loaded into memory only when the first prediction request is received. This approach conserves memory resources by delaying the model load until it is actually needed. It is particularly useful for applications that do not receive frequent requests or when managing multiple models that may not all be used simultaneously. While this method reduces initial memory usage, it introduces a delay for the first prediction request, as the model needs to be loaded before any predictions can be made. Lazy loading is suitable for scenarios where occasional delays in initial response time are acceptable.

On-Demand Loading:
On-demand loading involves loading models into memory based on specific triggers, such as user requests or external events. This strategy is highly flexible and efficient for environments with diverse and intermittent model usage, such as multi-tenant systems where different models are required for different users. On-demand loading balances memory usage and response time by dynamically loading and unloading models as needed. However, it requires efficient management to ensure that the loading process does not cause significant delays during peak usage times. This strategy is ideal for applications that need to serve a wide range of models without overburdening system resources.

Preloading with Warm-up:
Preloading with warm-up involves loading the model at startup and then performing a warm-up phase where test or dummy requests are processed to ensure the model is fully operational before serving real requests. This strategy ensures that the model is not only loaded but also primed for optimal performance from the first user interaction. It is especially beneficial in real-time serving environments where any delay in the first prediction is unacceptable. While this method slightly increases startup time, it provides a smooth and efficient user experience by eliminating cold start delays. Preloading with warm-up is well-suited for applications where consistent, high-speed response times are essential.

Model Caching:

Model caching keeps frequently used models in memory or a fast-access storage layer, reducing the time needed to load models for subsequent requests. This strategy is effective in systems with high request volumes for a limited number of models, as it minimizes loading times and improves overall system performance. By storing models in a cache, repeated loading operations are avoided, leading to faster response times. However, efficient cache management is crucial to balance memory usage and determine which models should be retained or evicted. Model caching is ideal for applications that experience frequent and repetitive model usage patterns.

Practical Activity 3.2.2: Integrating of Model with existing systems

1: Perform the following:

i. Test flask python framework successfully installation

ii. Create restful API with flask framework

iii. Avail endpoint to display your model prediction in existing system

iv. Integrate API endpoint created into existing system

v. Use gradio library interface to integrate with model saved?

2: Provide the answer for the asked questions and write them on papers.

3: Present the findings to the whole class

4: For more clarification, read the key readings 3.1.2. In addition, ask questions where necessary.

Key readings 3.2.2 Integrating of Model with existing systems

Installation of flask framework

Flask is widely used for deploying machine learning models because it offers a simple and efficient way to create web services and APIs that serve model predictions. Its lightweight framework allows developers to quickly set up and customize API endpoints to handle input data, pass it to the model, and return predictions, making it highly flexible for various deployment scenarios. Since Flask seamlessly integrates with the Python ecosystem, where most ML models are developed, it provides an intuitive workflow for transitioning from model training to deployment. Its support for JSON data, minimal overhead, and scalability makes it an ideal choice for serving both real-time and batch predictions. Additionally, Flask’s rich ecosystem of extensions and strong community support offer tools and resources that facilitate smooth deployment and maintenance. With built-in development and debugging tools, Flask simplifies testing and refining the deployment process before moving models to production, ensuring efficient and effective model serving.

Virtual environments

Use a virtual environment to manage the dependencies for your project, both in development and in production.

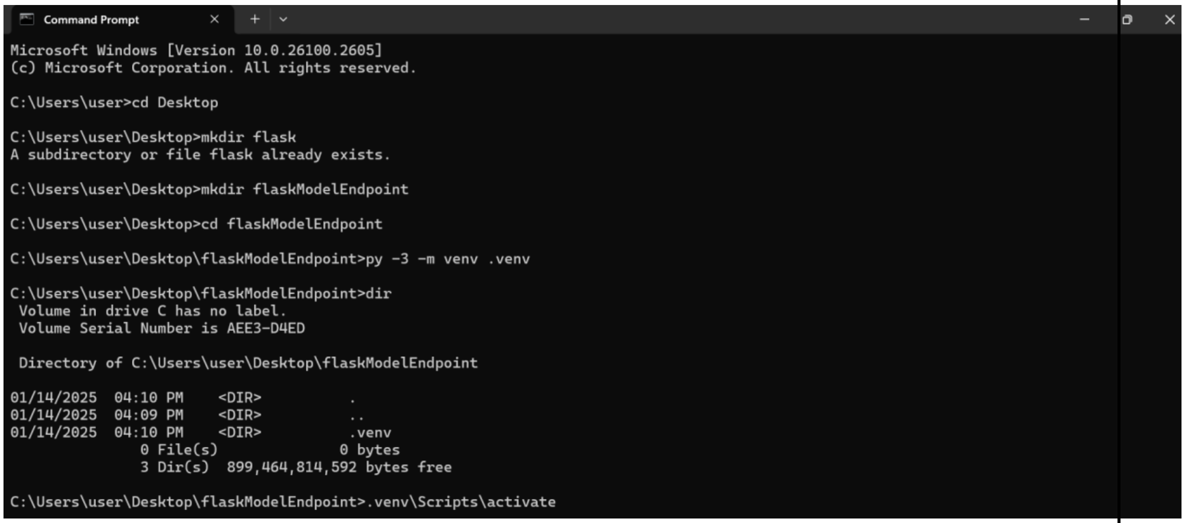
What problem does a virtual environment solve? The more Python projects you have, the more likely it is that you need to work with different versions of Python libraries, or even Python itself. Newer versions of libraries for one project can break compatibility in another project.

Virtual environments are independent groups of Python libraries, one for each project. Packages installed for one project will not affect other projects or the operating system’s packages.

Python comes bundled with the **VENV** Module to create virtual environments.

Activate the environment

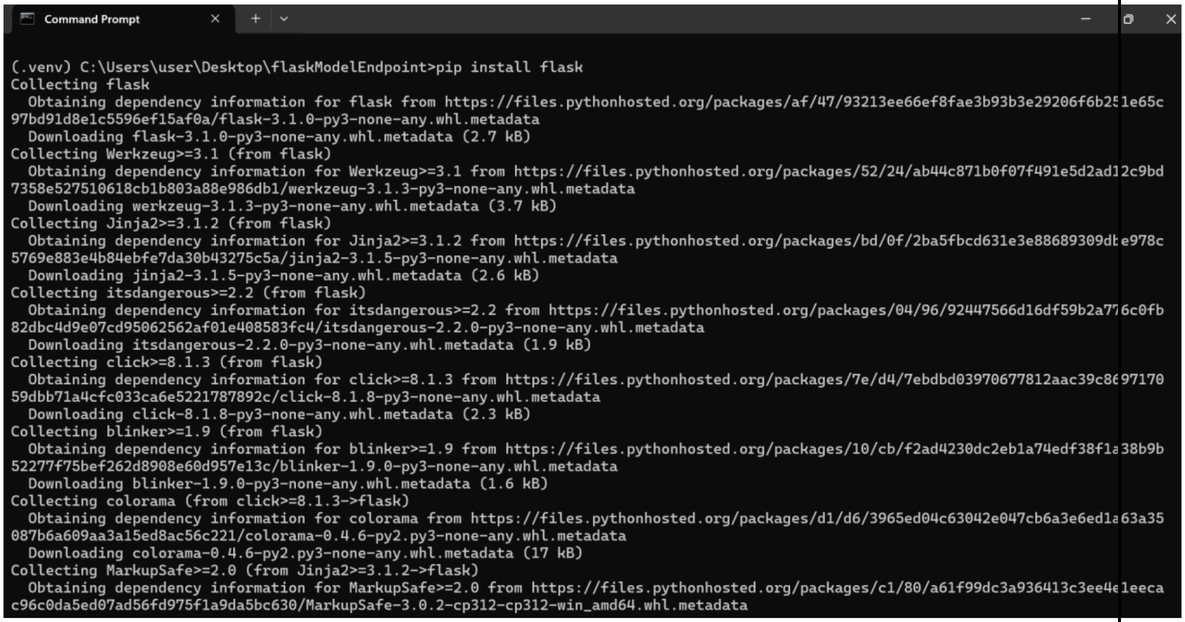
Before you work on your project, activate the corresponding environment:



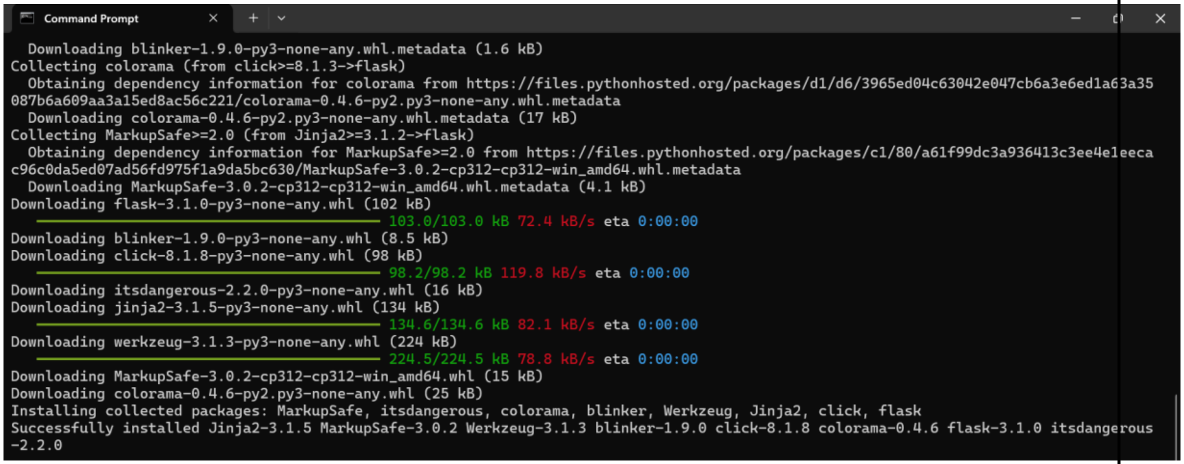
```
Microsoft Windows [Version 10.0.26100.2605]
(c) Microsoft Corporation. All rights reserved.

C:\Users\user>cd Desktop

C:\Users\user\Desktop>mkdir flask
A subdirectory or file flask already exists.

C:\Users\user\Desktop>mkdir flaskModelEndpoint

C:\Users\user\Desktop>cd flaskModelEndpoint

C:\Users\user\Desktop\flaskModelEndpoint>py -3 -m venv .venv

C:\Users\user\Desktop\flaskModelEndpoint>dir
 Volume in drive C has no label.
 Volume Serial Number is AEE3-D4ED

 Directory of C:\Users\user\Desktop\flaskModelEndpoint

01/14/2025  04:10 PM    <DIR>          .
01/14/2025  04:09 PM    <DIR>          ..
01/14/2025  04:10 PM    <DIR>          .venv
               0 File(s)              0 bytes
               3 Dir(s)  899,464,814,592 bytes free

C:\Users\user\Desktop\flaskModelEndpoint>.venv\Scripts\activate
```

Install Flask

Within the activated environment, use the following command to install Flask:



```
(.venv) C:\Users\user\Desktop\flaskModelEndpoint>pip install flask
Collecting flask
  Obtaining dependency information for flask from https://files.pythonhosted.org/packages/af/47/93213ee66ef8fae3b93b3e29206f6b251e65c
97bd91d8e1c5596ef15af0a/flask-3.1.0-py3-none-any.whl.metadata
  Downloading flask-3.1.0-py3-none-any.whl.metadata (2.7 kB)
Collecting Werkzeug>=3.1 (from flask)
  Obtaining dependency information for Werkzeug>=3.1 from https://files.pythonhosted.org/packages/52/24/ab44c871b0f07f491e5d2ad12c9bd
7358e527510618cb1b803a88e986db1/werkzeug-3.1.3-py3-none-any.whl.metadata
  Downloading werkzeug-3.1.3-py3-none-any.whl.metadata (3.7 kB)
Collecting Jinja2>=3.1.2 (from flask)
  Obtaining dependency information for Jinja2>=3.1.2 from https://files.pythonhosted.org/packages/bd/0f/2ba5fbcd631e3e88689309dbe978c
5769e883e4b84ebfe7da30b43275c5a/jinja2-3.1.5-py3-none-any.whl.metadata
  Downloading jinja2-3.1.5-py3-none-any.whl.metadata (2.6 kB)
Collecting itsdangerous>=2.2 (from flask)
  Obtaining dependency information for itsdangerous>=2.2 from https://files.pythonhosted.org/packages/04/96/92447566d16df59b2a776c0fb
82dbc4d9e07cd95062562af01e408583fc4/itsdangerous-2.2.0-py3-none-any.whl.metadata
  Downloading itsdangerous-2.2.0-py3-none-any.whl.metadata (1.9 kB)
Collecting click>=8.1.3 (from flask)
  Obtaining dependency information for click>=8.1.3 from https://files.pythonhosted.org/packages/7e/d4/7ebdbd03970677812aac39c8697170
59dbb71a4cfc033ca6e5221787892c/click-8.1.8-py3-none-any.whl.metadata
  Downloading click-8.1.8-py3-none-any.whl.metadata (2.3 kB)
Collecting blinker>=1.9 (from flask)
  Obtaining dependency information for blinker>=1.9 from https://files.pythonhosted.org/packages/10/cb/f2ad4230dc2eb1a74edf38f1a38b9b
52277f75bef262d8908e60d957e13c/blinker-1.9.0-py3-none-any.whl.metadata
  Downloading blinker-1.9.0-py3-none-any.whl.metadata (1.6 kB)
Collecting colorama (from click>=8.1.3->flask)
  Obtaining dependency information for colorama from https://files.pythonhosted.org/packages/d1/d6/3965ed04c63042e047cb6a3e6ed1a63a35
087b6a609aa3a15ed8ac56c221/colorama-0.4.6-py2.py3-none-any.whl.metadata
  Downloading colorama-0.4.6-py2.py3-none-any.whl.metadata (17 kB)
Collecting MarkupSafe>=2.0 (from Jinja2>=3.1.2->flask)
  Obtaining dependency information for MarkupSafe>=2.0 from https://files.pythonhosted.org/packages/c1/80/a61f99dc3a936413c3ee4e1eeca
c96c0da5ed07ad56fd975f1a9da5bc630/MarkupSafe-3.0.2-cp312-cp312-win_amd64.whl.metadata
```

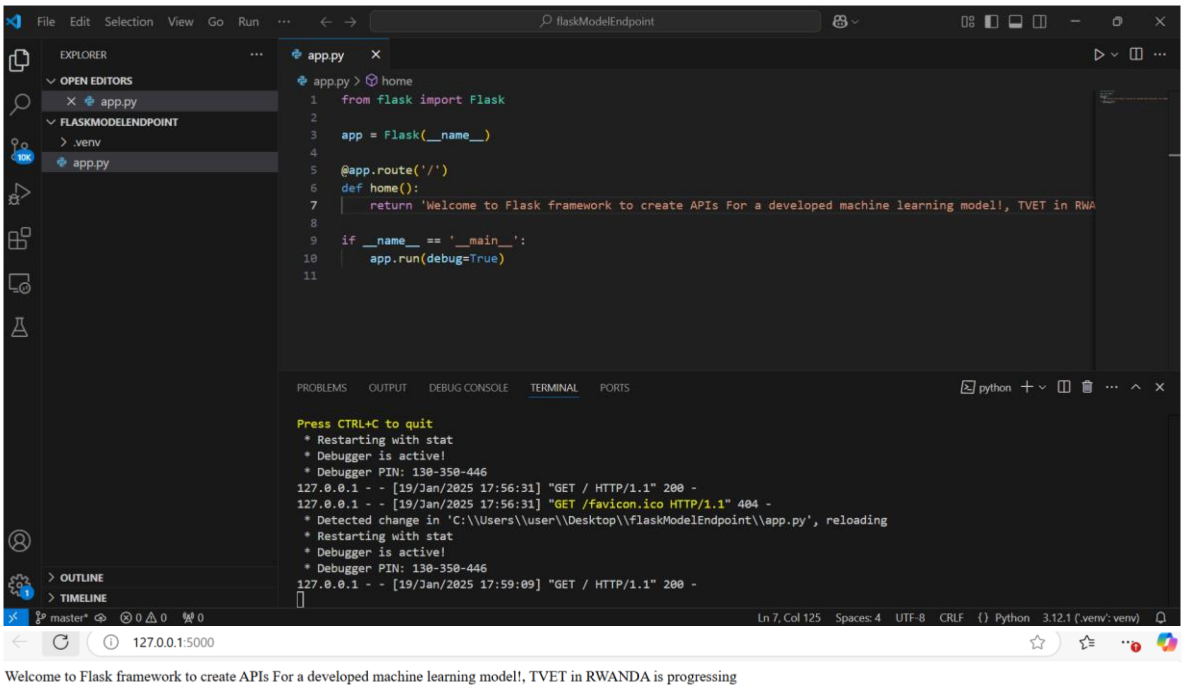


```
  Downloading blinker-1.9.0-py3-none-any.whl.metadata (1.6 kB)
Collecting colorama (from click>=8.1.3->flask)
  Obtaining dependency information for colorama from https://files.pythonhosted.org/packages/d1/d6/3965ed04c63042e047cb6a3e6ed1a63a35
087b6a609aa3a15ed8ac56c221/colorama-0.4.6-py2.py3-none-any.whl.metadata
  Downloading colorama-0.4.6-py2.py3-none-any.whl.metadata (17 kB)
Collecting MarkupSafe>=2.0 (from Jinja2>=3.1.2->flask)
  Obtaining dependency information for MarkupSafe>=2.0 from https://files.pythonhosted.org/packages/c1/80/a61f99dc3a936413c3ee4e1eeca
c96c0da5ed07ad56fd975f1a9da5bc630/MarkupSafe-3.0.2-cp312-cp312-win_amd64.whl.metadata
  Downloading MarkupSafe-3.0.2-cp312-cp312-win_amd64.whl.metadata (4.1 kB)
Downloading flask-3.1.0-py3-none-any.whl (102 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 103.0/103.0 kB 72.4 kB/s eta 0:00:00
Downloading blinker-1.9.0-py3-none-any.whl (8.5 kB)
Downloading click-8.1.8-py3-none-any.whl (98 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 98.2/98.2 kB 119.8 kB/s eta 0:00:00
Downloading itsdangerous-2.2.0-py3-none-any.whl (16 kB)
Downloading jinja2-3.1.5-py3-none-any.whl (134 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 134.6/134.6 kB 82.1 kB/s eta 0:00:00
Downloading werkzeug-3.1.3-py3-none-any.whl (224 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 224.5/224.5 kB 78.8 kB/s eta 0:00:00
Downloading MarkupSafe-3.0.2-cp312-cp312-win_amd64.whl (15 kB)
Downloading colorama-0.4.6-py2.py3-none-any.whl (25 kB)
Installing collected packages: MarkupSafe, itsdangerous, colorama, blinker, Werkzeug, Jinja2, click, flask
Successfully installed Jinja2-3.1.5 MarkupSafe-3.0.2 Werkzeug-3.1.3 blinker-1.9.0 click-8.1.8 colorama-0.4.6 flask-3.1.0 itsdangerous
-2.2.0
```

To begin setting up your Flask application, start by creating a new Python file named app.py in your project directory. This file will serve as the main entry point for your Flask application. Once the file is created, open it in your preferred text editor or IDE and write a basic Flask application. For example, you can initialize Flask by importing the Flask class, creating an instance of it, and defining a simple route that returns a message like

"**Welcome to Flask framework to create APIs for a developed machine learning model! TVET in RWANDA is progressing**"

when accessed. This basic setup allows you to quickly test and run your Flask application locally, laying the groundwork for further development and deployment.



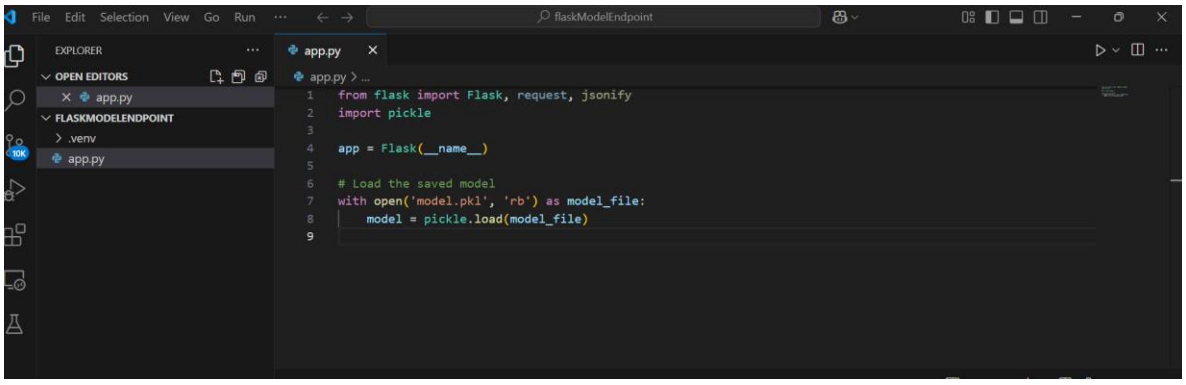
```python
from flask import Flask

app = Flask(__name__)

@app.route('/')
def home():
    return 'Welcome to Flask framework to create APIs For a developed machine learning model!, TVET in RWA

if __name__ == '__main__':
    app.run(debug=True)

```

```
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 130-350-446
127.0.0.1 - - [19/Jan/2025 17:56:31] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [19/Jan/2025 17:56:31] "GET /favicon.ico HTTP/1.1" 404 -
 * Detected change in 'C:\\Users\\user\\Desktop\\flaskModelEndpoint\\app.py', reloading
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 130-350-446
127.0.0.1 - - [19/Jan/2025 17:59:09] "GET / HTTP/1.1" 200 -
```

Welcome to Flask framework to create APIs For a developed machine learning model!, TVET in RWANDA is progressing

Loading model saved

Loading a saved machine learning model is a crucial step in deploying it for real-world applications. Once a model is trained and tested, it is typically saved in a serialized format, such as a Pickle (.pkl) or Joblib file, which preserves the model's structure and learned parameters. Loading the model involves reading this serialized file back into memory, enabling the model to make predictions on new data. This process allows the model to be reused without retraining, saving time and computational resources. In a Flask application, loading the model is usually done at the start of the app to ensure it is ready to serve predictions as soon as requests are received.



```python
from flask import Flask, request, jsonify
import pickle

app = Flask(__name__)

# Load the saved model
with open('model.pkl', 'rb') as model_file:
    model = pickle.load(model_file)
```

Identification Data Format for API Input

When designing an API, it's crucial to define the format for the input data that the API will accept. Two common formats are JSON and Form Data. Below are explanations and examples for each format, typical to handling data for a machine learning model deployed using Flask.

JSON (JavaScript Object Notation)

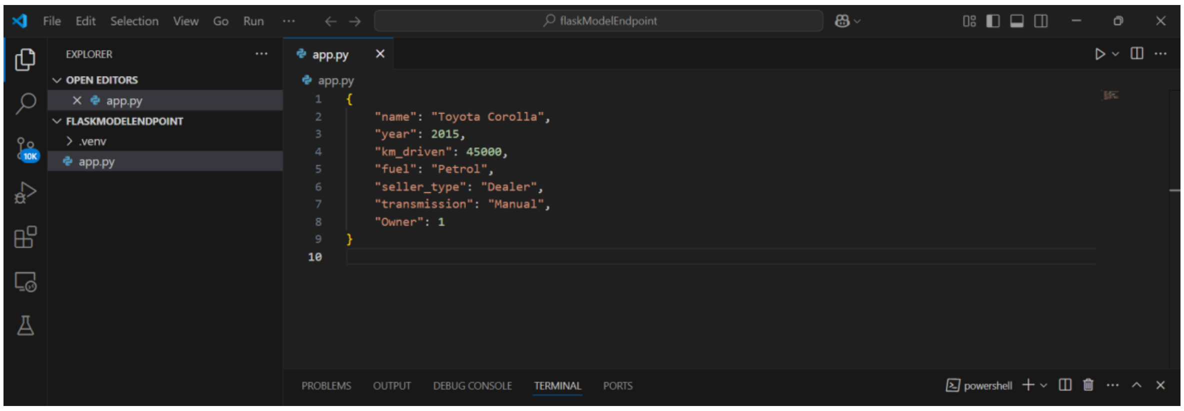
JSON is a lightweight data-interchange format that's easy for humans to read and write, and easy for machines to parse and generate. It is widely used in APIs for data exchange.

Structure:

- JSON data is structured as key-value pairs.
- It supports various data types like strings, numbers, arrays, and objects.

Example:

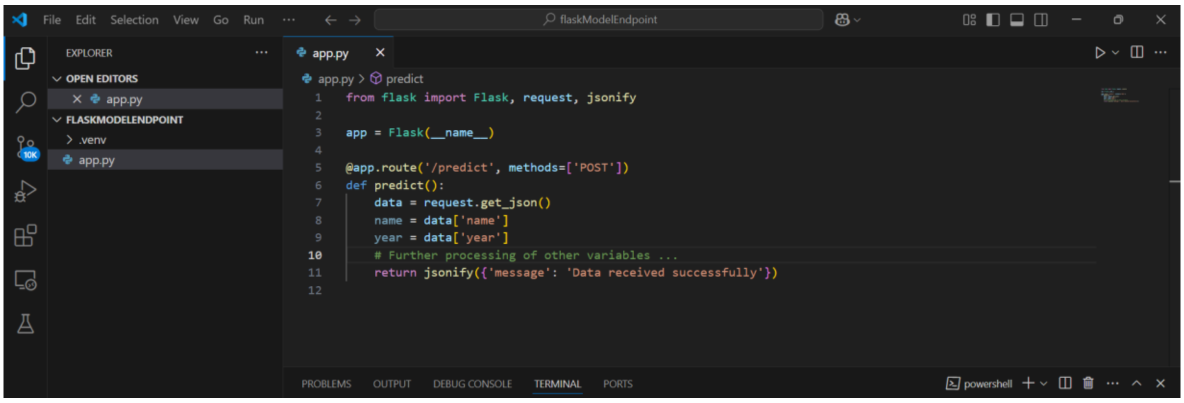
For a car price prediction API, the input data in JSON format might look like this:



```json
{
    "name": "Toyota Corolla",
    "year": 2015,
    "km_driven": 45000,
    "fuel": "Petrol",
    "seller_type": "Dealer",
    "transmission": "Manual",
    "Owner": 1
}
```

Flask Handling:

To handle JSON data in Flask, you use the request.get_json() method.



```python
from flask import Flask, request, jsonify

app = Flask(__name__)

@app.route('/predict', methods=['POST'])
def predict():
    data = request.get_json()
    name = data['name']
    year = data['year']
    # Further processing of other variables ...
    return jsonify({'message': 'Data received successfully'})

```

Form Data

Form Data is used when input data is submitted through an HTML form. This format is key-value pairs, similar to JSON but typically used for simple forms and less complex data structures.

Structure:

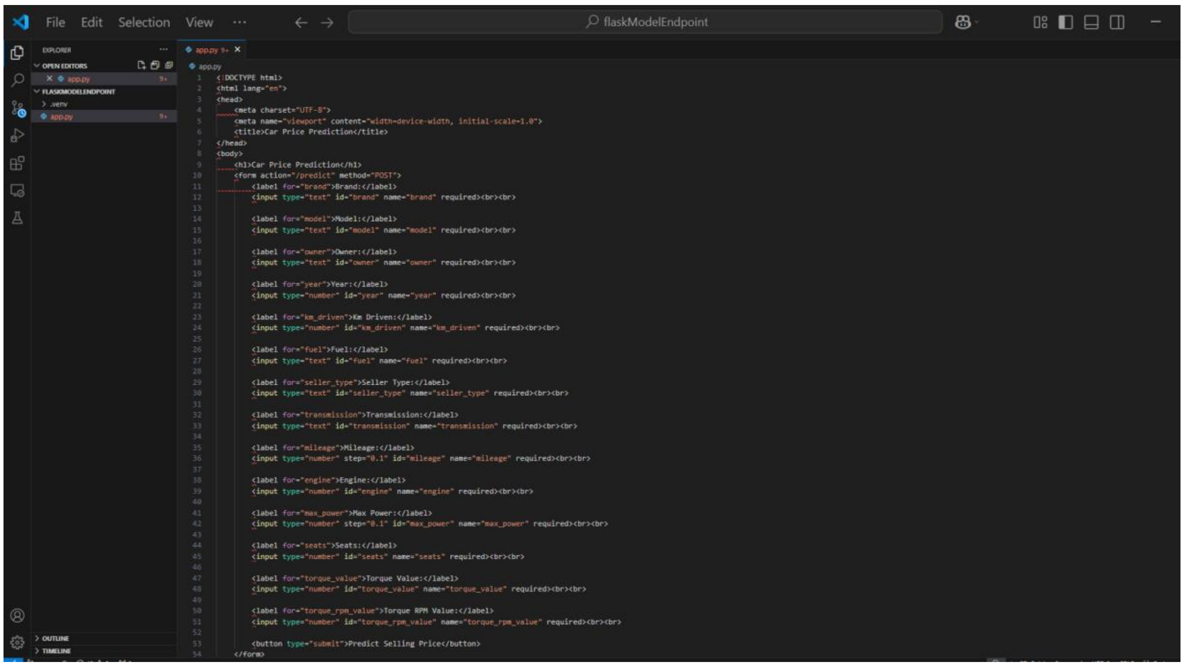
- Form data is sent as application/x-www-form-urlencoded or multipart/form-data.
- Each key-value pair corresponds to an input field in an HTML form.

Example:

For the same car price prediction scenario, form data might be structured as:

brand=Honda&model=Civic&owner=0&year=2018&km_driven=30000&fuel=1&seller_type=1&transmission=1&mileage=20.4&engine=1498&max_power=118&seats=5&torque_value=300&torque_rpm_value=1750

Flask Handling:



```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Car Price Prediction</title>
</head>
<body>
    <h1>Car Price Prediction</h1>
    <form action="/predict" method="POST">
        <label for="brand">Brand:</label>
        <input type="text" id="brand" name="brand" required><br><br>

        <label for="model">Model:</label>
        <input type="text" id="model" name="model" required><br><br>

        <label for="owner">Owner:</label>
        <input type="text" id="owner" name="owner" required><br><br>

        <label for="year">Year:</label>
        <input type="number" id="year" name="year" required><br><br>

        <label for="km_driven">Km Driven:</label>
        <input type="number" id="km_driven" name="km_driven" required><br><br>

        <label for="fuel">Fuel:</label>
        <input type="text" id="fuel" name="fuel" required><br><br>

        <label for="seller_type">Seller Type:</label>
        <input type="text" id="seller_type" name="seller_type" required><br><br>

        <label for="transmission">Transmission:</label>
        <input type="text" id="transmission" name="transmission" required><br><br>

        <label for="mileage">Mileage:</label>
        <input type="number" step="0.1" id="mileage" name="mileage" required><br><br>

        <label for="engine">Engine:</label>
        <input type="number" id="engine" name="engine" required><br><br>

        <label for="max_power">Max Power:</label>
        <input type="number" step="0.1" id="max_power" name="max_power" required><br><br>

        <label for="seats">Seats:</label>
        <input type="number" id="seats" name="seats" required><br><br>

        <label for="torque_value">Torque Value:</label>
        <input type="number" id="torque_value" name="torque_value" required><br><br>

        <label for="torque_rpm_value">Torque RPM Value:</label>
        <input type="number" id="torque_rpm_value" name="torque_rpm_value" required><br><br>

        <button type="submit">Predict Selling Price</button>
    </form>
```

Implementing Communication with HTTP Requests and Responses

In the context of integrating a machine learning model into a web application using Flask, you need to implement communication between the client (e.g., a web browser, mobile app, or another service) and the server (where the Flask app is running). This communication is facilitated through HTTP requests and responses.

1. HTTP Requests

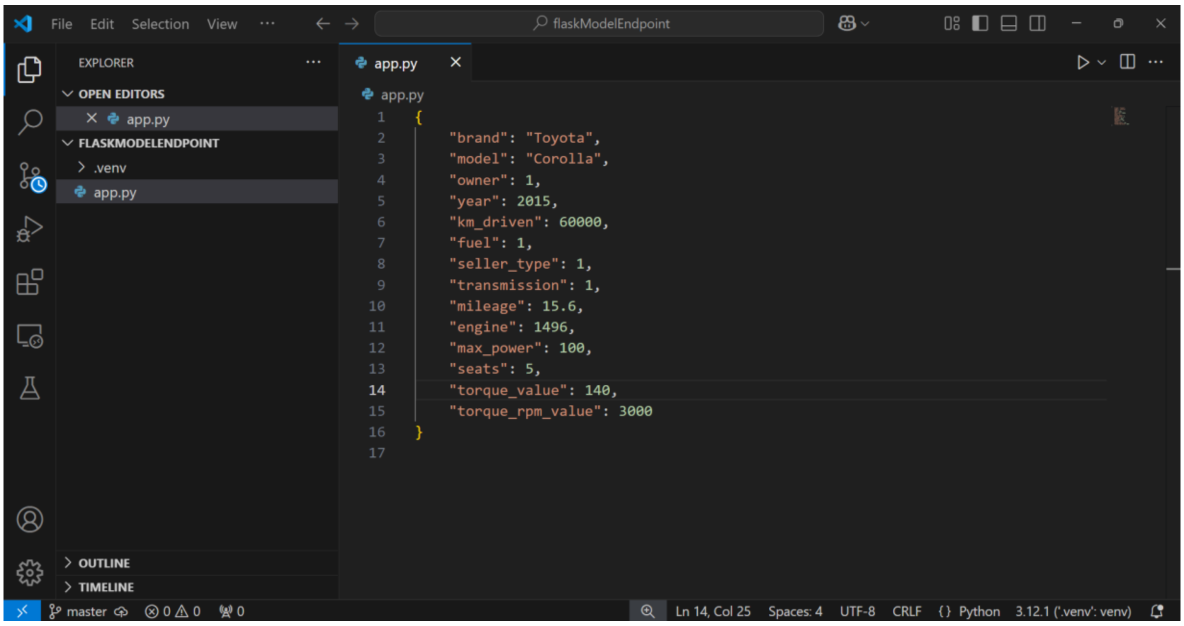
An HTTP request is made by the client to ask the server to perform some action, like submitting data or requesting information. The client can send different types of HTTP requests to interact with the server, including:

Common Types of HTTP Requests:

- GET: Used to request data from the server. This is typically used for retrieving information, such as displaying a webpage or fetching model predictions.
- POST: Used to send data to the server. This is often used for submitting form data, making predictions, or uploading files.
- PUT: Used to update existing resources on the server.
- DELETE: Used to delete resources on the server.

For example, Send a Test Request: Use a tool like Postman or cURL to test the /predict endpoint.

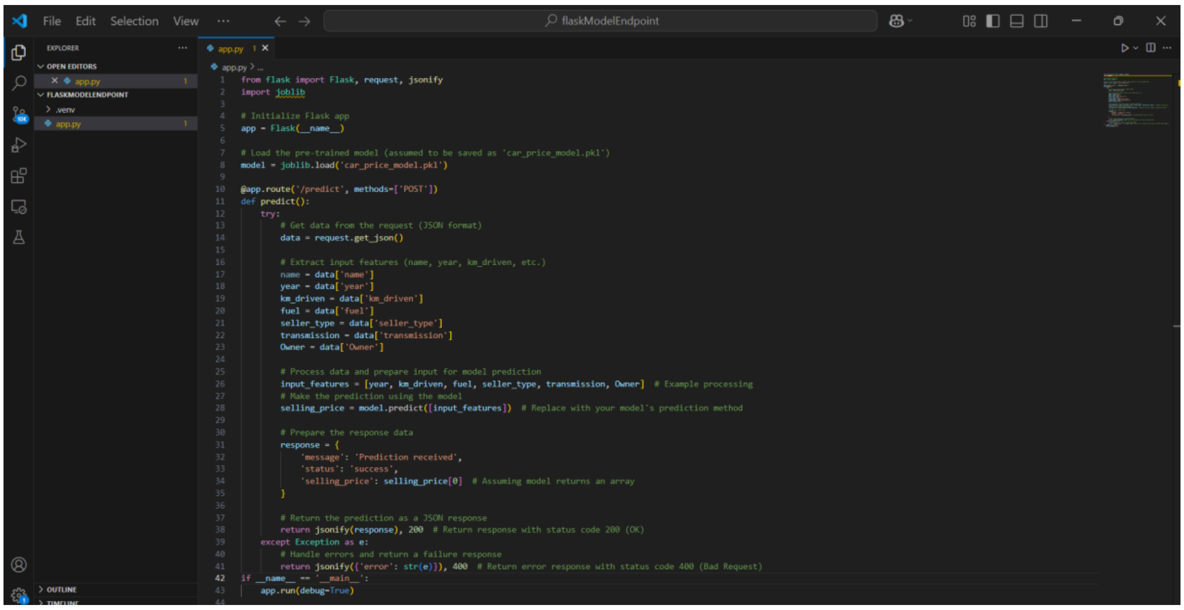
Sample Request (JSON format):



```
{
    "brand": "Toyota",
    "model": "Corolla",
    "owner": 1,
    "year": 2015,
    "km_driven": 60000,
    "fuel": 1,
    "seller_type": 1,
    "transmission": 1,
    "mileage": 15.6,
    "engine": 1496,
    "max_power": 100,
    "seats": 5,
    "torque_value": 140,
    "torque_rpm_value": 3000
}
```

HTTP Responses (Server-Side)

On the server side (Flask), you need to handle incoming requests, make predictions based on the input data, and then return an appropriate HTTP response.

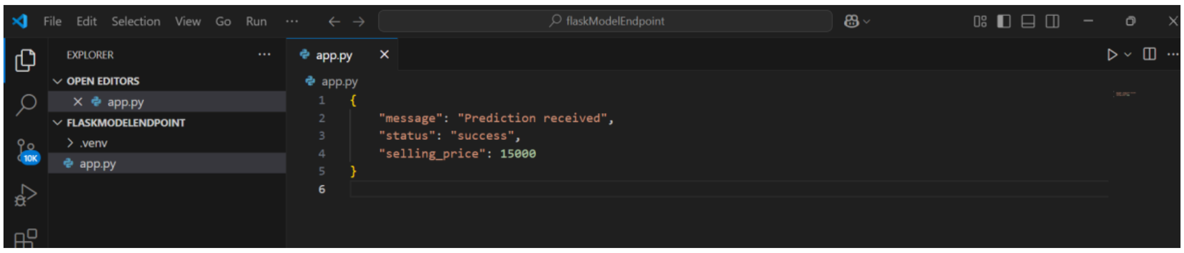


```python
from flask import Flask, request, jsonify
import joblib

# Initialize Flask app
app = Flask(__name__)

# Load the pre-trained model (assumed to be saved as 'car_price_model.pkl')
model = joblib.load('car_price_model.pkl')

@app.route('/predict', methods=['POST'])
def predict():
    try:
        # Get data from the request (JSON format)
        data = request.get_json()

        # Extract input features (name, year, km_driven, etc.)
        name = data['name']
        year = data['year']
        km_driven = data['km_driven']
        fuel = data['fuel']
        seller_type = data['seller_type']
        transmission = data['transmission']
        Owner = data['Owner']

        # Process data and prepare input for model prediction
        input_features = [year, km_driven, fuel, seller_type, transmission, Owner]  # Example processing
        # Make the prediction using the model
        selling_price = model.predict([input_features])  # Replace with your model's prediction method

        # Prepare the response data
        response = {
            'message': 'Prediction received',
            'status': 'success',
            'selling_price': selling_price[0]  # Assuming model returns an array
        }

        # Return the prediction as a JSON response
        return jsonify(response), 200  # Return response with status code 200 (OK)
    except Exception as e:
        # Handle errors and return a failure response
        return jsonify({'error': str(e)}), 400  # Return error response with status code 400 (Bad Request)
if __name__ == '__main__':
    app.run(debug=True)
```

In this code:

- The Flask app listens for POST requests at the /predict endpoint.
- It uses request.get_json() to extract data from the incoming JSON request.
- The prediction is made using the model (model.predict()), and the result is returned as a JSON response using jsonify().
- If there's an error, a 400 status code with the error message is returned.

If the client sends a valid request with the necessary data, the server might return a JSON response like this:



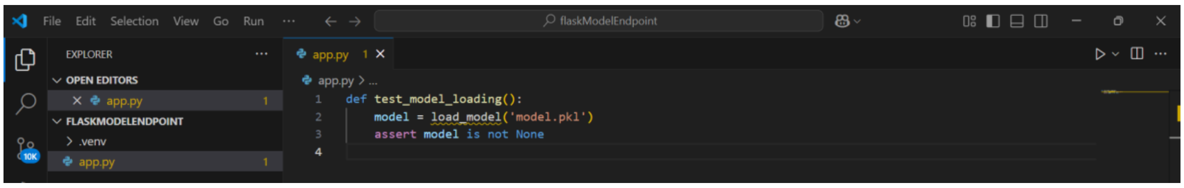
```json
{
    "message": "Prediction received",
    "status": "success",
    "selling_price": 15000
}
```

Testing thoroughly deployment

Thorough testing in model deployment ensures that the system is reliable, efficient, and secure. This involves evaluating individual components to confirm they perform as intended, such as the functions responsible for loading the model and making predictions. It also includes testing how different parts of the system interact, ensuring seamless communication and data flow between the application and the model. Simulating real-world scenarios helps to confirm that the system behaves as expected under typical user conditions, while performance testing assesses its responsiveness and capacity under normal and peak loads. Stress testing challenges the system's limits to check its stability during high usage, and security testing protects against vulnerabilities, ensuring data safety and privacy. Involving end-users in the testing process validates that the system meets their needs and expectations, and continuous testing after updates ensures that new changes do not disrupt existing functionalities. Through these comprehensive testing efforts, developers can deliver a robust, high-performing, and secure deployment that meets user expectations and maintains reliability over time.

Unit testing

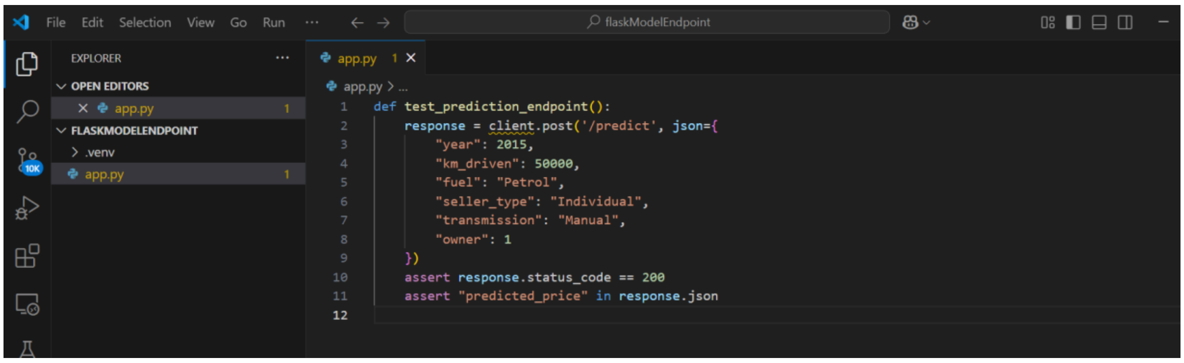
Unit testing is a fundamental part of software development, focusing on verifying the functionality of individual components or functions in isolation. In the context of deploying machine learning models, unit tests ensure that specific parts of the code, such as the model loading function or prediction logic, operate correctly before integrating them into the broader system. By testing these units independently, developers can identify and fix bugs early in the development process, reducing the risk of errors in the final deployment. Unit tests also serve as a safeguard against future changes, helping maintain the integrity of the code by quickly alerting developers to issues introduced by updates or new features. Through consistent unit testing, the overall quality and reliability of the application are significantly enhanced, leading to more stable and predictable performance in production environments.



```
def test_model_loading():
    model = load_model('model.pkl')
    assert model is not None
```

Integration Testing

Integration testing is a crucial phase in software development where individual units or components are combined and tested as a group to verify that they work together as expected. In the context of deploying machine learning models, integration tests ensure that the interactions between different parts of the system—such as data preprocessing, model inference, and post-processing logic—are seamless and function correctly when integrated. These tests identify issues that might not be apparent in unit testing, such as data flow problems, API communication errors, or incompatibilities between modules. By conducting integration tests, developers can catch integration-related bugs early, preventing them from affecting the overall functionality of the application. Integration tests are essential for validating the system's behavior in real-world scenarios, ensuring that different components function cohesively and efficiently in production environments. They also provide confidence when scaling or modifying the system, as they verify that changes to one part of the system do not break other interconnected components.



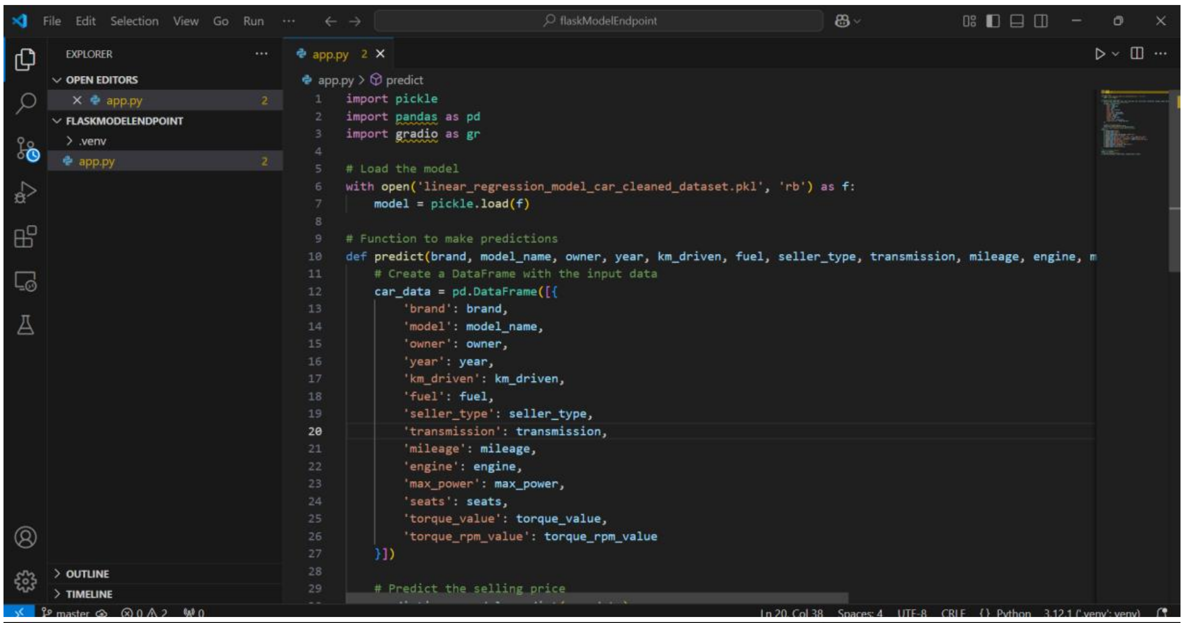
```python
def test_prediction_endpoint():
    response = client.post('/predict', json={
        "year": 2015,
        "km_driven": 50000,
        "fuel": "Petrol",
        "seller_type": "Individual",
        "transmission": "Manual",
        "owner": 1
    })
    assert response.status_code == 200
    assert "predicted_price" in response.json
```

Simplified Production Model Deployment for Networking Professionals Using Gradio

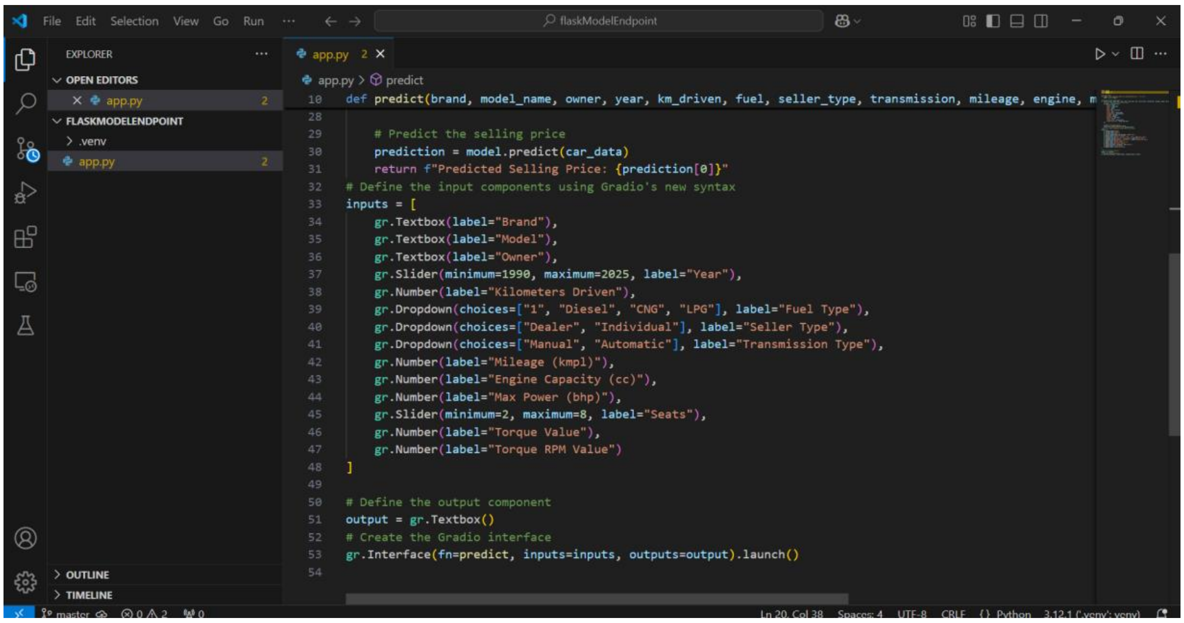
Deploying a model to production using the Gradio library is a powerful and streamlined approach, especially for professionals in the networking domain who may not have extensive experience in software development, cloud hosting, or deployment. Gradio simplifies the process of creating interactive web applications for machine learning models, making it a valuable shortcut for networking specialists who need to deploy models without delving deeply into programming or DevOps tasks.

Gradio allows you to quickly create a user-friendly interface, where users can interact with models through components like textboxes, sliders, and buttons. By simply loading the trained model and writing a function to process user inputs and generate predictions, you can create a web-based interface that makes the model accessible. This approach abstracts much of the backend complexity involved in traditional deployment processes, such as setting up cloud servers, writing extensive code for the front-end, and managing DevOps workflows.

For networking professionals with significant experience in their domain but limited software development or deployment expertise, Gradio offers a practical and efficient solution. It provides the necessary tools to deploy machine learning models to production with minimal effort, making it much easier for those in networking to focus on the practical applications of the models rather than the intricacies of programming and infrastructure management. This enables faster prototyping and testing in production environments, ensuring quicker feedback and continuous improvement of the models.



```python
import pickle
import pandas as pd
import gradio as gr

# Load the model
with open('linear_regression_model_car_cleaned_dataset.pkl', 'rb') as f:
    model = pickle.load(f)

# Function to make predictions
def predict(brand, model_name, owner, year, km_driven, fuel, seller_type, transmission, mileage, engine, m
    # Create a DataFrame with the input data
    car_data = pd.DataFrame([{
        'brand': brand,
        'model': model_name,
        'owner': owner,
        'year': year,
        'km_driven': km_driven,
        'fuel': fuel,
        'seller_type': seller_type,
        'transmission': transmission,
        'mileage': mileage,
        'engine': engine,
        'max_power': max_power,
        'seats': seats,
        'torque_value': torque_value,
        'torque_rpm_value': torque_rpm_value
    }])

    # Predict the selling price
```



```python
def predict(brand, model_name, owner, year, km_driven, fuel, seller_type, transmission, mileage, engine, m

    # Predict the selling price
    prediction = model.predict(car_data)
    return f"Predicted Selling Price: {prediction[0]}"
# Define the input components using Gradio's new syntax
inputs = [
    gr.Textbox(label="Brand"),
    gr.Textbox(label="Model"),
    gr.Textbox(label="Owner"),
    gr.Slider(minimum=1990, maximum=2025, label="Year"),
    gr.Number(label="Kilometers Driven"),
    gr.Dropdown(choices=["1", "Diesel", "CNG", "LPG"], label="Fuel Type"),
    gr.Dropdown(choices=["Dealer", "Individual"], label="Seller Type"),
    gr.Dropdown(choices=["Manual", "Automatic"], label="Transmission Type"),
    gr.Number(label="Mileage (kmpl)"),
    gr.Number(label="Engine Capacity (cc)"),
    gr.Number(label="Max Power (bhp)"),
    gr.Slider(minimum=2, maximum=8, label="Seats"),
    gr.Number(label="Torque Value"),
    gr.Number(label="Torque RPM Value")
]

# Define the output component
output = gr.Textbox()
# Create the Gradio interface
gr.Interface(fn=predict, inputs=inputs, outputs=output).launch()
```

Gradio provides a variety of output components that can be used to display the results. The type of output component you choose will depend on the nature of your prediction. For predicting car selling price, a textbox is appropriate because it will display the result as plain text.

Here's how you define the output component in Gradio:

output = gr.Textbox(label="Predicted Selling Price")

In this example:

- gr.Textbox() creates a simple textbox.
- The label parameter is used to assign a name or description to the textbox that will be displayed above the output area.

Returning the Output from the Prediction Function

Once the user provides the input, Gradio calls the predict function to process the input data and generate a prediction. In this case, the model will predict the selling price based on the input features, and the result will be returned as the output.

return f"Predicted Selling Price: {prediction[0]}"

In this function:

- The model processes the input data and generates a prediction.
- The result (prediction[0]) is formatted as a string, which will be displayed inside the output textbox.

Once the prediction is returned, Gradio automatically populates the output component (the textbox in this case) with the predicted value. For example:

- *The output will look like: "Predicted Selling Price: $12,500"*

This is a user-friendly way of presenting the model's result in plain text, making it easy for users to read and understand.

You can enable live updates in Gradio to display the output immediately as the user adjusts the input values, without waiting for them to click a submit button. This is especially useful for continuous predictions based on changing input data.

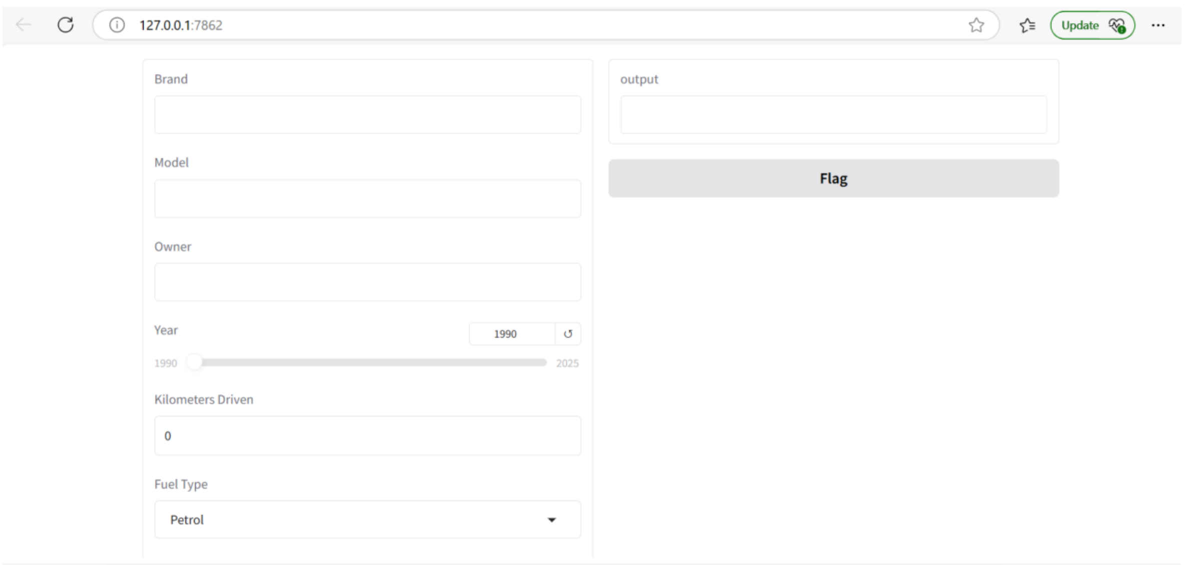
Example with live mode:

gr.Interface(fn=predict, inputs=inputs, outputs=output, live=True).launch()

This will automatically update the predicted selling price as the user modifies input values.

After deploying the model, the output can also be shared with others. When you launch the Gradio interface with the share=True option, you get a public URL where the model and its outputs are accessible by anyone with the link:

*gr.Interface(fn=predict, inputs=inputs, outputs=output, live=True).launch(**share=True**)*


127.0.0.1:7862

Brand

Model

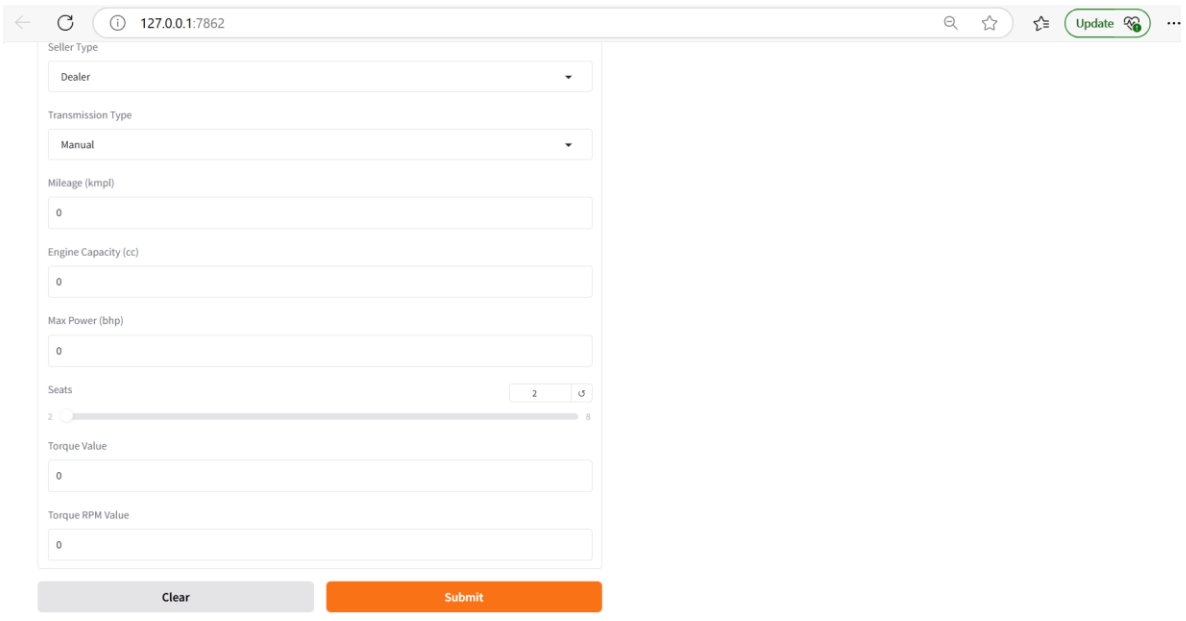
Owner

Year 1990

Kilometers Driven 0

Fuel Type Petrol

output


127.0.0.1:7862

Seller Type Dealer

Transmission Type Manual

Mileage (kmpl) 0

Engine Capacity (cc) 0

Max Power (bhp) 0

Seats 2

Torque Value 0

Torque RPM Value 0

Monitor performance

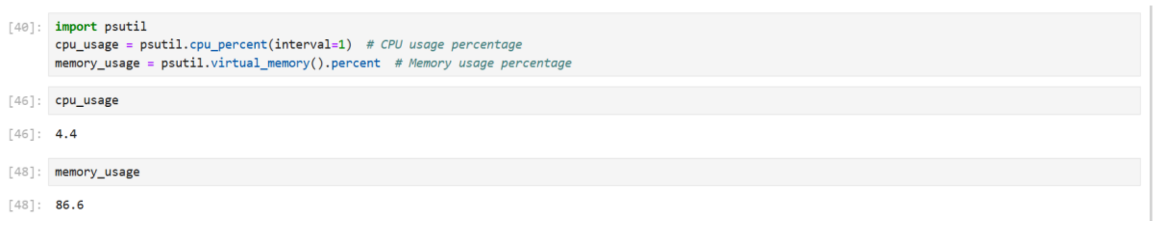
When deploying machine learning models using Gradio, it's essential to monitor the performance of the system to ensure that it is running efficiently, providing accurate predictions, and meeting user expectations. Monitoring helps in identifying areas for optimization, detecting errors, and ensuring the model operates effectively under varying conditions.

System Resource Monitoring

Monitoring system resources is critical to ensure that the infrastructure supporting the machine learning model can handle the computational load. Key system metrics to track include CPU usage, memory usage, disk space, and network latency. High CPU or memory usage can indicate inefficient model computation or server overload, leading to slow responses or potential crashes. Monitoring disk space helps prevent issues with saving model data, temporary files, or logs. Similarly, network latency can affect the time it takes to process requests from users and deliver predictions, negatively impacting the user experience.

Tools such as operating system utilities (e.g., top, htop), as well as cloud-based monitoring services like AWS CloudWatch, provide valuable insights into system performance. These tools help detect performance bottlenecks, ensure resource availability, and optimize the system to handle increasing workloads as usage grows.

Example :



```
[40]: import psutil
      cpu_usage = psutil.cpu_percent(interval=1)  # CPU usage percentage
      memory_usage = psutil.virtual_memory().percent  # Memory usage percentage

[46]: cpu_usage

[46]: 4.4

[48]: memory_usage

[48]: 86.6
```

Model Performance Monitoring

Model performance monitoring focuses on ensuring that the machine learning model continues to make accurate predictions. Over time, a model's performance may degrade due to various reasons such as changes in data patterns or the occurrence of model drift. Monitoring involves evaluating the prediction accuracy, detecting potential model drift, and analyzing the model's response time.

Prediction accuracy can be assessed by comparing the model's output with ground truth labels, where available. Tracking metrics such as accuracy, precision, recall, or error rates is essential for understanding the model's reliability. Model drift occurs when the underlying data distribution shifts, causing the model to produce incorrect predictions. Continuous monitoring helps detect such shifts early, prompting model retraining or updates.

Response time is another critical metric, especially for real-time applications. Slow response times can signal computational inefficiencies or infrastructure bottlenecks, impacting the user experience. Regularly assessing these metrics ensures that the model remains effective and delivers timely predictions.

API Requests Tracking

API requests are the interactions between the deployed model and external clients, whether they are users or other systems. Tracking API requests helps ensure that the system can handle the load, identify popular usage patterns, and understand user behavior.

Response Times

Response time refers to the time it takes for the API to process a request and return a prediction to the user. It's one of the most critical aspects of user experience, as long delays can frustrate users and hinder the effectiveness of the model. Monitoring response times ensures that the system provides timely predictions.

Model Accuracy Tracking

Tracking the accuracy of the machine learning model is crucial for understanding how well it is performing in production. While the model may have been trained and validated on a separate dataset, real-world performance can differ. Regularly tracking accuracy ensures that the model remains reliable and provides high-quality predictions.

Delivering Prediction to the clients

Delivering predictions to clients involves integrating the machine learning model's API into applications, formatting the results for easy understanding, and handling errors effectively. By ensuring smooth communication with the model and providing meaningful, well-formatted predictions, you improve the user experience and make the system more reliable and efficient. Proper error handling further ensures that users are informed about issues, minimizing frustration and enhancing overall satisfaction.
This is done even below while integrating model.

Points to Remember

- Always follow high-level outline to guide you through the process of integrating a model file with Your Existing Application as required.
- Create the API structure, and test it using HTTP methods for RESTful API operations.
- Ensure performance monitoring with tools like Prometheus and Grafana, secure the API with HTTPS, authentication (e.g., OAuth2, JWT), and rate limiting. Track API health using centralized logging (e.g., ELK Stack), and maintain a feedback loop by storing predictions for retraining and tracking performance over time.
- Choose an appropriate model serving method and use Docker for containerization alongside hosting platforms like cloud services or Kubernetes for scalability.
- Set up performance monitoring with tools like Prometheus and Grafana to track API requests, response times, and resource utilization, and test load capacity with tools like JMeter.
- Secure the API using HTTPS, proper authentication (e.g., OAuth2, JWT), and implement rate limiting to prevent misuse.

- Monitor errors and log requests using centralized logging systems like the ELK Stack to track system health and troubleshoot issues.
- Maintain a feedback loop by storing predictions and actual outcomes for model retraining and performance tracking over time.

Application of learning 3.2.

You are tasked with deploying a TensorFlow model to predict customer sentiment in real-time for a customer service application that handles thousands of users daily. The model must be deployed using an appropriate method, such as real-time API serving, to meet the application's need for quick predictions, while considering memory, computing power, and system constraints. Once deployed, you must monitor system performance during peak hours, including API response times, requests, and model accuracy.
Tasks:

1. Choose a suitable deployment method (real-time API serving) and justify based on real-time requirements.
2. Integrate the TensorFlow SavedModel into the web application via an API to serve predictions in real-time.
3. Monitor and optimize system performance during peak hours, tracking API requests, response times, and model accuracy.
4. Implement performance monitoring tools and address potential slowdowns using tools like Prometheus and Grafana.

Indicative content 3.3 Delivering Prediction to the clients

Duration: 5 hrs

Theoretical Activity 3.3.1: Describing API Integration for Model Predictions

Tasks:

1: Answer the following questions related to understanding model integration:

i. What is the role of an API in integrating machine learning models into applications?
ii. How does an API handle the input and output data when serving model predictions?
iii. What are common issues that may occur during API-based model predictions, and how can they be resolved?
iv. Why is it important to format predictions correctly when delivering them to a client application?
v. What error-handling strategies should be implemented in an API to ensure seamless delivery of predictions to clients?

2: Provide the answer for the asked questions and write them on papers.

3: Present the findings to the whole class

4: For more clarification, read the key readings 3.2.1. In addition, ask questions where necessary.

Key readings 3.3.1: Describing API Integration for Model Predictions

- **Describing API Integration for Model Predictions**

Model integration in machine learning refers to the process of embedding a machine learning model into a larger software system or application. It allows the model to be used in real-time for predictions or analysis by integrating it with the APIs, databases, or other software components. This process typically involves connecting the model to external systems, which can be used to retrieve data, make predictions, or provide insights, and then feedback the results to users or other systems.
Model integration is essential for making machine learning models accessible to end-users via user interfaces, software, or web applications, such as mobile apps, dashboards, or

automated systems.

- ✓ **Benefits of Model Integration in Machine Learning**

Model integration offers several benefits, including:

- **Automation of Predictions**: With model integration, predictions can be automated, enabling users to get real-time results without manual intervention.
- **Scalability**: Integrated models can scale across multiple platforms, allowing large-scale operations, such as handling multiple requests from different users in parallel.
- **Improved Decision-Making**: By embedding the model into an operational system, organizations can leverage data-driven insights to improve decision-making processes, such as detecting fraud or recommending products.
- **Increased Accessibility**: When integrated into a user-facing application, a machine learning model becomes easily accessible to non-technical users, allowing businesses to deliver predictive insights directly to stakeholders.
- **Real-time Performance**: Integration allows real-time predictions, which is useful for applications like financial trading platforms, recommendation systems, or smart devices where fast response times are crucial.
- **Efficiency**: Automation reduces manual effort, enabling efficient processing of large datasets or frequent transactions.

- ✓ **Key Concepts in Model Integration for Machine Learning:**

- **Predictions/Insights**:
  - **Predictions**: Machine learning models can predict outcomes based on input data. For instance, a model integrated into a customer service system might predict customer churn by analysing historical interaction data. API integration allows systems to automatically send relevant data to the model and receive the predicted outcome (e.g., the likelihood of a customer leaving).
  - **Insights**: The integration of models can generate deeper insights, such as identifying patterns in large datasets. For example, a model analysing student performance data might uncover trends, such as the influence of certain subjects on overall academic performance. These insights help educators or administrators make informed decisions.
- **Content Generation**: Content generation involves using machine learning models to automatically produce relevant content. For example, natural language processing (NLP) models can generate automated reports, product descriptions, or even personalized email messages based on user input.

  Example: A news app might integrate a machine learning model to generate article summaries, or an e-commerce platform could automatically create personalized product descriptions for users based on their browsing behaviour.
- **Data Analysis**: Machine learning models integrated into an application can process and analyse large volumes of data, identifying patterns or generating forecasts.

For example, a retail company could integrate a model that predicts future sales based on historical sales data, enabling them to optimize stock and inventory levels.

Example: A financial application might integrate a predictive model to analyse market trends and advise users on investment opportunities.

Practical Activity 3.3.2: implementing API Integration and Error Handling for a Machine Learning Model

Task:

1: you are requested to go to the computer lab to implement API Integration and Error Handling for a Machine Learning Model

2: Apply safety precautions

3: Read key reading 3.3.2 and ask clarification where necessary

4: Present out the steps to implementing API Integration and Error Handling for a Machine Learning Model.

5: Referring to the steps provided, implement API Integration and Error Handling for a Machine Learning Model.

6: Present your work to the trainer and whole class.

Key readings 3.3.2: Implementing API Integration and Error Handling for a Machine Learning Model

- **API Integration and Error Handling for a Machine Learning Model**

implementing API Integration and Error Handling for a Machine Learning Model
In this practical activity, we need build a simple REST API using Flask that serves a machine learning model for making predictions. This include error handling to ensure the system behaves correctly when invalid inputs are received.

Prerequisites
Python installed on your machine.
Install the necessary Python libraries

✓ **Steps for Implementing API Integration and Error Handling for a Machine**

Learning Model

Step 1. Train a Simple Model

Start by training a simple machine learning model, saving it, and then creating an API to serve this model.

```
# train_model.py
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
import joblib

# Load dataset
data = load_iris()
X = data['data']
y = data['target']

# Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train a model
model = LogisticRegression(max_iter=200)
model.fit(X_train, y_train)

# Save the model
joblib.dump(model, 'iris_model.pkl')
print("Model saved successfully!")
```

This script trains a Logistic Regression model on the Iris dataset and saves the trained model using joblib.

Step 2. Create the Flask API

Once the model is trained and saved, we'll create a Flask API that loads the model and allows users to make predictions by sending requests.

```
# api.py
from flask import Flask, request, jsonify
import joblib
import numpy as np
```

```
# Initialize Flask app
app = Flask(__name__)

# Load the trained model
model = joblib.load('iris_model.pkl')

@app.route('/')
def home():
    return "Welcome to the Iris Prediction API!"

# Prediction endpoint
@app.route('/predict', methods=['POST'])
def predict():
    try:
        # Get the JSON input from the client
        data = request.json
        features = data.get('features', None)

        if features is None:
            return jsonify({'error': 'No features provided'}), 400

        # Convert input features to numpy array
        features = np.array(features).reshape(1, -1)

        # Validate input data length
        if features.shape[1] != 4:
            return jsonify({'error': 'Incorrect number of features. Expecting 4'}), 400

        # Make prediction
        prediction = model.predict(features)[0]
        return jsonify({'prediction': int(prediction)})

    except Exception as e:
        return jsonify({'error': str(e)}), 500

# Error handling for invalid routes
@app.errorhandler(404)
def not_found_error(error):
    return jsonify({'error': 'Route not found'}), 404
```

```
if __name__ == '__main__':
    app.run(debug=True)
```

- ✓ Loading the model: The model is loaded using **joblib.load().**
- ✓ Endpoints:
  - **GET /:** This serves a simple welcome message.
  - **POST /predict:** This endpoint accepts a JSON request containing features and returns the predicted class.
- ✓ Error Handling:
  - Input Validation: We ensure that the features are provided and that they have the correct dimensions.
  - General Error Handling: The try-except block catches any unexpected errors during the prediction process.
  - 404 Error Handling: A custom error handler for routes that do not exist.

Step 3. Testing the API

You can test this API using Postman or cURL. Sample request (JSON format):

```
curl -X POST http://127.0.0.1:5000/predict \
-H "Content-Type: application/json" \
-d '{"features": [5.1, 3.5, 1.4, 0.2]}'
```

Expected Response:

json

```
{
  "prediction": 0
}
```

Testing Error Scenarios:

1. **Missing Features:**

```
curl -X POST http://127.0.0.1:5000/predict \
-H "Content-Type: application/json" \
-d '{}'
```

Response:

json

```
{
  "error": "No features provided"
}
```

2. **Incorrect Number of Features:**

```
curl -X POST http://127.0.0.1:5000/predict \
-H "Content-Type: application/json" \
-d '{"features": [5.1, 3.5]}'
```

Response:

json

```
{
  "error": "Incorrect number of features. Expecting 4"
}
```

3. **Invalid Route:**

```
curl -X GET http://127.0.0.1:5000/invalid
```

json

```
{
  "error": "Route not found"
}
```

Step 4. Run the API

To start the Flask API, run the following command in your terminal:

```
python api.py
```

The API will be served at **http://127.0.0.1:5000/.**

Points to Remember

- Use APIs as bridges between machine learning models and client applications to enable seamless communication and real-time interaction by exposing model functionalities as services.
- Ensure proper data handling by structuring input data (e.g., JSON or form data) and formatting output predictions in a user-friendly way for client consumption.
- Format predictions correctly to provide clear, interpretable, and consistent data that enhances the user experience and reduces confusion
- implementing API Integration and Error Handling for a Machine Learning Model is crucial for enhancing performance of the model.
- Plan for future updates by using version control in your API
- Design the API to handle multiple requests concurrently, considering load balancing if needed for large-scale applications.

Application of learning 3.3.

Mietech LTD is a house seller company located in Gasabo district, it used to sell house through application. The task involves using an API to serve predictions based on user inputs such as location, square footage, and the number of rooms. You need to ensure that the predictions are correctly formatted for display and handle errors like invalid input or server issues, providing clear feedback to users. The APIs will facilitate interaction between the application and the model, focusing on prediction delivery and error management for a seamless user experience. You are assigned a task to integrate a machine learning model for predicting house prices into a real estate web (application).

Learning outcome 3 end assessment

Theoretical assessment

I. Read the following statements concerned to the machine learning deployment and circle the letter corresponding to the right answer

1) The Primary benefit of using a web application for model deployment is?

A) Limited accessibility

B) Real-time interaction with users

C) High maintenance costs

D) Requires extensive hardware

2) Common format for model files is?

A) CSV

B) XML

C) TensorFlow SavedModel

D) TXT

3) API stand for?

A) Application Programming Interface

B) Automated Programming Instruction

C) Application Process Integration

D) Advanced Programming Interface

4) Integrating a machine learning model, data format that is commonly used for sending input data to the API is?

A) HTML

B) PDF

C) JSON

D) CSV

5) **Model deployment method which is NOT used in Machine learning is ?**

A) Real-time inference

B) Batch processing

C) Data scraping

D) Streaming inference

6) **Main purpose of tracking API requests is?**

A) To increase the application size

B) To monitor user behaviour

C) To assess model performance and response times

D) To improve the front-end interface

7) **In which situation would you typically use a mobile app for model deployment?**

A) For applications needing heavy computational resources

B) For users requiring offline access and notifications

C) For data-heavy processing tasks

D) For machine-to-machine communication

8) **Common HTTP response code for a successful API request is?**

A) 404

B) 200

C) 500

D) 400

9) **The crucial thing for ensuring the compatibility of the model with the existing system is?**

A) Model size

B) Model accuracy

C) Programming language and framework

D) Data collection methods

10) Primary goal of integrating the model file into an application is?

A) To generate random data

B) To analyse historical data

C) To provide predictions or insights

D) To store large datasets

11. Which of the following are common methods for delivering predictions to clients in machine learning?

a) REST APIs
b) Batch Processing
c) Real-time Streaming
d) All of the above

12. Which method is typically used for large volumes of data that do not require immediate predictions?
a) Real-time Streaming
b) Batch Processing
c) REST APIs
d) Cloud Functions

13. Which method is best suited for applications where clients need to interact with the model in real-time, such as chatbots or recommendation systems?

a) Batch Processing
b) Real-time Streaming
c) REST APIs
d) Email

14. Which of the following factors should be considered when choosing a prediction delivery method?
a) Latency requirements
b) Data volume
c) Security and privacy
d) Cost
e) All of the above

19. What is the role of APIs in delivering machine learning predictions?

a) APIs enable clients to interact with the model programmatically.
b) APIs are only used for batch processing.
c) APIs are not relevant for real-time predictions.
d) APIs are mainly used for internal model management.

II. Read the following terms as used in machine learning model deployment Match the terms in Column A with their corresponding descriptions in Column B and provide the answer in appropriate place.

Answer	Column A	Column B
1.............	1. What format is commonly used to save and integrate machine learning models?	A. Enables communication between the model and external applications.
2............	2. What is the role of APIs in model integration?	B. h5, .pkl, .joblib, and .onnx formats
3.............	3. What library is commonly used to save Python-based ML models?	C. ONNX
4.............	4. Which framework supports saving models as .onnx for cross-platform use?	D. joblib and pickle
5..............	5. What is the purpose of serialization in model integration?	E. Flask or FastAPI
6..............	6. What tool is commonly used to serve machine learning models as REST APIs?	F. Loading the saved model back into memory for inference.
7.............	7. What does model deserialization involve?	G. Converts a model into a file or byte stream for storage and transfer.
8............	8. What environment factors must be considered during integration?	H. Docker
9..............	9. How can a TensorFlow model be saved for integration?	I. Python version, library versions, and dependencies.
10...........	10. What platform is widely used for deploying containerized ML models?	J. Using the model.save () function to create .h5 or SavedModel files.

III. Read the following statements related to the deployment of machine learning and answer by t true if the statement is correct or false if the statement is incorrect.

1) The main advantage of using a standalone program for model deployment is its ease of integration with existing web applications.
2) Scikit-Learn is a commonly used framework for deploying machine learning models.
3) Monitoring performance involves tracking API requests, response times, and model accuracy.
4) Batch processing is the only method available for deploying machine learning models.
5) Proper formatting of predictions is unnecessary if the model is accurate.
6) Platforms offer high scalability and flexibility for deploying machine learning models.
7) On-premise deployment provides the highest level of control over the model and its environment.
8) Edge devices are typically used for high-latency applications where real-time processing is critical.
9) Cost is a significant factor to consider when choosing a deployment method.
10) Security is not a major concern when deploying machine learning models in the cloud.

Practical assessment

Accenture is a top data analytics-consulting firm that works with clients from 40+ industries. You are hired to deploy a machine-learning model to provide real-time predictions for patient health outcomes. In this application-based learning scenario, select a suitable model from Scikit-Learn, TensorFlow, or PyTorch, such as one that predicts the likelihood of hospital readmission or disease progression based on patient data.

Tasks:

I. including the required dataset characteristics, a
II. chose the best application type and technology stack
III. design and implement an API using a web framework of their choice,
IV. generates predictions, and returns well-formatted responses,
V. Implement strategies for monitoring performance in healthcare applications.

END

References

Amershi, S., Begel, A., Bird, C., DeLine, R., Gall, H., Kamar, E., & Zimmermann, T. (2019). Software engineering for machine learning: A case study. *IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. https://doi.org/10.1109/ICSE-SEIP.2019.00042

Kraska, T., Beutel, A., Chi, E. H., & Dean, J. (2020). The case for learned index structures. *Communications of the ACM, 63*(12), 105–113. https://doi.org/10.1145/3380971

Olston, C., Fiedel, N., Gorovoy, K., Harmsen, J., Lao, L., Ramesh, S., & Soergel, D. (2017). TensorFlow-Serving: Flexible, high-performance ML serving. *arXiv preprint arXiv:1712.06139*. https://arxiv.org/abs/1712.06139

Raschka, S., & Mirjalili, V. (2017). Python machine learning: Machine learning and deep learning with Python, scikit-learn, and TensorFlow 2 (2nd ed.). Packt Publishing.

Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., & Young, M. (2015). Hidden technical debt in machine learning systems. *Advances in Neural Information Processing Systems*, 28, 2503–2511. https://papers.nips.cc/paper/5656-hidden-technical-debt-in-machine-learning-systems

Zhang, Z., Lipton, Z. C., Li, M., & Smola, A. J. (2020). Dive into deep learning. *MIT Press*. https://d2l.ai

RTB | RWANDA TVET BOARD

October 2024