UNIT-I

Introduction to EDA and Python Environment

Topics: Introduction to Data Science and EDA, Importance of EDA in Data Science Life Cycle, Setting up Python Environment: Jupiter, Anaconda, VS Code, Introduction to NumPy and Pandas: Arrays, Series, DataFrames, Data loading, viewing, basic operations (info, describe, shape)

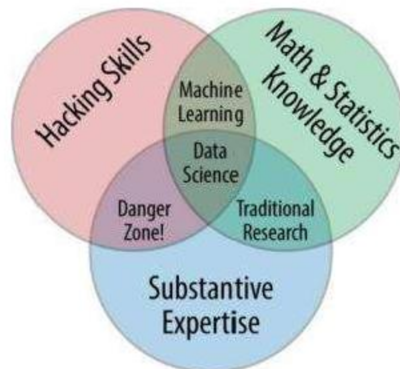
Introduction to Data Science

Data Science is a multidisciplinary field that combines statistical methods, programming skills, and domain knowledge to extract meaningful insights and knowledge from structured and unstructured data. It is at the heart of modern technological advancements, empowering industries to make data-driven decisions and innovate rapidly.

What is Data Science?

Data Science involves the process of collecting, processing, analyzing, and interpreting data to uncover patterns and support decision-making. It draws techniques from fields such as:

- **Statistics** – for understanding data distribution and relationships
- **Computer Science** – for programming and data engineering
- **Mathematics** – for algorithm design and modeling
- **Domain Expertise** – for applying results meaningfully in specific industries



Key Components of Data Science

1. **Data Collection:** Gathering data from various sources such as databases, web APIs, sensors, or spreadsheets.
2. **Data Cleaning and Preprocessing**: Handling missing values, removing noise, and converting raw data into usable formats.
S. **Exploratory Data Analysis (EDA)**: Visualizing and summarizing the main characteristics of data using graphs and statistics.
4. **Data Modeling**: Applying algorithms (e.g., regression, classification, clustering) to find patterns or make predictions.

5. **Model Evaluation**: Assessing the performance of models using metrics like accuracy, precision, recall, or RMSE.
6. **Deployment and Visualization**: Presenting the results in interactive dashboards or embedding models in applications for real-time usage.

Common Tools and Technologies

- **Programming Languages**: Python, R, SQL
- **Libraries/Frameworks**: Pandas, NumPy, Scikit-learn, TensorFlow, Matplotlib, Seaborn
- **Databases**: MySQL, PostgreSQL, MongoDB
- **Big Data Tools**: Hadoop, Spark
- **Cloud Platforms**: AWS, Azure, Google Cloud
- **Visualization Tools**: Tableau, Power BI, Plotly

Applications of Data Science

Data Science is transforming numerous sectors, such as:

- **Healthcare**: Predicting disease, personalized treatments
- **Finance**: Fraud detection, credit risk analysis
- **Retail**: Customer segmentation, recommendation engines
- **Transportation**: Route optimization, predictive maintenance
- **Agriculture**: Crop yield prediction, smart farming
- **Entertainment**: Content recommendations, trend analysis

Data Scientist Roles

- **Data Analyst** – Analyzes and visualizes data for business insights.
- **Machine Learning Engineer** – Builds predictive models and AI systems.
- **Data Engineer** – Designs and maintains data infrastructure and pipelines.
- **Business Intelligence Analyst** – Focuses on KPIs and business metrics.
- **Data Scientist** – A hybrid role that spans all aspects from data wrangling to decision-making.

Data Science Life Cycle

The **Data Science Life Cycle** is a structured approach that outlines the major steps involved in solving a data-driven problem using data science methodologies. Just like the software development life cycle (SDLC), the data science life cycle ensures that data projects are systematic, repeatable, and goal-oriented.

I. Problem Definition

This is the **most critical phase**, as a poorly defined problem leads to poor outcomes.

Key Questions:

- What is the business problem?

- What are the success criteria?
- What value will the solution bring?

Example:
For a bank, the goal might be **predicting loan default** to reduce risk.

2. Data Collection

Once the problem is defined, the next step is to **gather relevant data**.
Sources:

- Databases (MySQL, MongoDB)
- APIs (Twitter API, Weather API)
- Web scraping
- Public datasets (Kaggle, UCI, data.gov.in)
- IoT devices or mobile applications

Activities:

- Identify relevant data sources
- Connect and extract data
- Store in a data warehouse or data lake

3. Data Cleaning and Preprocessing

Raw data is often messy, incomplete, or inconsistent. Cleaning it ensures the quality of analysis.
Tasks Involved:

- Handling **missing values** (imputation or removal)
- Treating **outliers**
- Removing **duplicates**
- Converting **data types**
- Encoding **categorical variables**

Tools:

- Python (Pandas, NumPy)
- R
- SQL

4. Exploratory Data Analysis (EDA)

EDA helps in understanding the underlying patterns in the data and forms the basis for modeling decisions.
Activities:

- Summary statistics (mean, median, mode, etc.)
- Visualization (histograms, scatter plots, box plots)
- Correlation analysis
- Univariate, bivariate, and multivariate analysis

Goals:

- Spot trends and outliers
- Form hypotheses
- Identify data transformations needed

5. Feature Engineering and Selection

This involves creating new input variables (features) and selecting the most relevant ones for modeling.

Techniques:

- Binning, scaling, normalizing
- Creating interaction terms
- Removing highly correlated variables
- Using feature importance scores

Why It's Important:

Good features can dramatically improve the accuracy of models.

6. Model Building

This is the core stage where machine learning or statistical models are trained to learn from the data.

Model Types:

- **Supervised Learning**: Regression, Classification
- **Unsupervised Learning**: Clustering, Dimensionality Reduction
- **Reinforcement Learning**: Agent-based decision making

Process:

- Split data into training and testing sets
- Choose algorithms (e.g., decision trees, SVM, neural networks)
- Train the model on the training data

7. Model Evaluation

The model is assessed for its performance using various metrics, depending on the problem type.

Common Metrics:

- Accuracy, Precision, Recall, F1-score (for classification)
- RMSE, MAE, R2 (for regression)
- Confusion matrix, ROC-AUC curve

Techniques:

- Cross-validation
- Hyperparameter tuning
- Model comparison

8. Model Deployment

Once a model performs well, it's deployed to make real-time or batch predictions.

Deployment Methods:

- REST APIs
- Web applications (Flask, Django)
- Cloud platforms (AWS SageMaker, GCP AI Platform)

Considerations:

- Scalability
- Monitoring and feedback loops
- Version control

9. Monitoring and Maintenance

After deployment, the model must be continuously monitored for **data drift**, **performance degradation**, and **user feedback**.

Activities:

- Retraining with new data
- Logging predictions
- Alerting when thresholds are breached

Introduction to EDA (Exploratory Data Analysis)

Exploratory Data Analysis (EDA) is the initial and critical phase in any data science or machine learning project. It involves analyzing datasets to summarize their main characteristics, often using visual methods. The goal of EDA is to understand the structure, patterns, and anomalies in the data before applying any modeling techniques.

Importance of EDA in the Data Science Life Cycle

Exploratory Data Analysis (EDA) is a **foundational and essential phase** in the Data Science Life Cycle. It serves as the bridge between raw data and actionable insights. While data collection and cleaning prepare the data, EDA allows data scientists to **understand** the data, **ask the right questions**, and **guide modeling decisions**.

I. Understanding the Data

EDA provides a **clear picture** of the dataset's structure, content, and quality. It answers questions like:

- What variables are present?
- What are the distributions?
- Are there missing or erroneous values?

2. Detecting Data Quality Issues

EDA helps in identifying:

- **Missing values**
- **Outliers**
- **Inconsistent data types**
- **Duplicate records**

This ensures that further steps are not built on faulty data.

3. Identifying Patterns and Trends

With EDA, you can discover:

- Relationships between variables (e.g., age and salary)
- Groupings or clusters
- Seasonal or time-based trends

These insights inform **feature selection** and **business understanding**.

4. Feature Engineering Guidance

EDA shows which variables are:

- Most relevant
- Redundant
- Non-informative

This helps in creating or transforming variables to improve model performance.

5. Improves Model Accuracy

A good understanding of data through EDA leads to:

- Better preprocessing
- Informed algorithm selection
- Better tuning of model parameters

All of which enhance **predictive accuracy**.

6. Assumptions Testing

Many models assume certain data characteristics (like linearity, normal distribution). EDA helps:

- Validate those assumptions
- Decide whether transformation (e.g., log-scaling) is needed

7. Visual Storytelling

EDA uses graphs and plots to:

- Communicate findings to non-technical stakeholders
- Create compelling narratives around data
- Support business decisions visually

Setting up Python Environment: Anaconda and Jupiter Environments

(Reference-I)

Setting up Python Environment: VS Code

https://code.visualstudio.com/docs/python/python-tutorial

Introduction to NumPy

NumPy (Numerical Python) is a fundamental package for scientific computing in Python. It provides **efficient arrays**, **mathematical functions**, and **tools for working with large, multi-dimensional data**. It is the foundation for many libraries in data science and machine learning, such as Pandas, SciPy, and TensorFlow.

Installing NumPy

Using pip**:**

pip install numpy

Basic Example

```
import numpy as np
# Create a 10 array
arr = np.array([1, 2, 3, 4, 5])
print(arr)
# Create a 20 array (matrix)
mat = np.array([[1, 2], [3, 4]])
print(mat)
```

Numpy Operations

SNO	Category	Operation / Code	Output
1	Array Creation	np.array([1, 2, S])	[1 2 S]
2	2D Array	np.array([[1, 2], [S, 4]])	[[1 2] [S 4]]
S	Zeros	np.zeros((2, 2))	[[0. 0.] [0. 0.]]
4	Ones	np.ones((1, S))	[[1. 1. 1.]]
5	Full	np.full((2, 2), 7)	[[7 7] [7 7]]
6	Eye (Identity)	np.eye(S)	[[1. 0. 0.] [0. 1. 0.] [0. 0. 1.]]
7	Random floats	np.random.rand(2,2)	[[... ...] [......]]
8	Random integers	np.random.randint(1, 10, (2, 2))	Example: [[4 8] [2 9]]
9	Arange	np.arange(0, 10, 2)	[0 2 4 6 8]
10	Linspace	np.linspace(0, 1, 5)	[0. 0.25 0.5 0.75 1.]
11	Shape	arr.shape	(2, S)
12	Dimensionality	arr.ndim	2
1S	Size	arr.size	6
14	Indexing	arr[1, 2]	value at row 1, col 2
15	Slicing	arr[:, 1]	All rows, column 1
16	Reshape	arr.reshape(S, 2)	Reshaped Sx2 matrix

SNO	Category	Operation / Code	Output
17	Flatten	arr.flatten()	1D array
18	Transpose	arr.T	Transposed matrix
19	Addition	a + b	Element-wise sum
20	Multiplication	a * b	Element-wise product
21	Dot product	np.dot(a, b)	Scalar or matrix result
22	Power	a ** 2	Square of each element
2S	Mean	np.mean(arr)	Mean of elements
24	Sum	np.sum(arr)	Total sum
25	Min/Max	np.min(arr), np.max(arr)	Minimum, Maximum
26	Std Dev	np.std(arr)	Standard deviation
27	Axis sum	np.sum(arr, axis=0)	Column-wise sum
28	Boolean Mask	arr[arr > 25]	Filtered array
29	Logical operators	(arr > 10) & (arr < S0)	Boolean array
S0	Inverse	np.linalg.inv(a)	Matrix inverse
S1	Determinant	np.linalg.det(a)	Scalar value
S2	Eigenvalues	np.linalg.eig(a)	Eigenvalues and vectors
SS	Random seed	np.random.seed(0)	Ensures reproducibility
S4	Broadcasting	a + b (shape-matched)	Automatically expanded
S5	Unique elements	np.unique(arr)	[unique values]
S6	Sort	np.sort(arr)	Sorted array
S7	Where condition	np.where(arr == 10)	Index/indices
S8	Concatenate	np.concatenate([a, b])	Combined 1D array
S9	Vstack	np.vstack([a, b])	Vertical stack
40	Hstack	np.hstack([a, b])	Horizontal stack
41	NaN detection	np.isnan(arr)	Boolean array
42	Inf detection	np.isinf(arr)	Boolean array
4S	Remove NaN	arr[~np.isnan(arr)]	Valid data only

For More Numpy Programs Practice- (Reference-2)

Introduction to Pandas in Python

Pandas is a powerful, flexible, and easy-to-use open-source **data analysis and manipulation library** for Python.
It provides two core data structures:

- Series – One-dimensional labeled array
- DataFrame – Two-dimensional labeled data structure (like an Excel spreadsheet or SQL table)

Importance of Pandas

I. Efficient Data Handling: Provides powerful data structures like DataFrame and Series for easy manipulation of structured/tabular data.
2. Simplifies Data Cleaning & Preprocessing: Offers built-in tools to handle missing values, duplicates, data type conversions, and inconsistencies.
3. Supports Data Analysis & Exploration: Enables filtering, sorting, grouping, and summarizing data for quick insight generation during EDA.
4. Flexible Input/output Capabilities: Supports reading/writing from CSV, Excel, JSON, SQL, and more, making it versatile for real-world data sources.
5. Seamless Integration & High Performance: Works efficiently with large datasets and integrates well with NumPy, Matplotlib, Seaborn, and Scikit-learn.

Installing Pandas

Using **pip**:

```
pip install pandas
```

Basic Example:
Series – One-dimensional Arrays

```
import pandas as pd
s = pd.Series([10, 20, 30, 40])
print(s)

#Output
0     10
1     20
2     30
3     40
dtype: int64
```

DataFrame – Two-dimensional Matrices

```
import pandas as pd
data = {
    'Name': ['Mahesh', 'Paani', 'Suresh'],
    'Age': [25, 30, 35],
    'City': ['Chennai', 'Bangalore', 'Tirupati']
```

```
}
df = pd.0ataFrame(data)
print(df)
```

#Output

```
   Name   Age    City
0  Mahesh  25    Chennai
1  Paani   S0    Bangalore
2  Suresh  S5    Tirupati
```

Pandas Operations

SNO	Operation	Description	Example
1	pd.Series(data)	Create a Series	pd.Series([10, 20, S0])
2	df=pd.DataFrame(data)	Create a DataFrame	pd.DataFrame({'A':[1,2], 'B':[S,4]})
S	df.head()	View first 5 rows	df.head()
4	df.tail(n)	View last *n* rows	df.tail(2)
5	df.shape	Get (rows, columns)	df.shape
6	df.columns	List column names	df.columns
7	df.dtypes	Show data types	df.dtypes
8	df.info()	Summary of DataFrame	df.info()
9	df.describe()	Summary stats for numeric columns	df.describe()
10	df['col']	Access column as Series	df['Age']
11	df[['col1','col2']]	Access multiple columns	df[['Age','Salary']]
12	df.loc[row]	Access row by label	df.loc[1]
1S	df.iloc[row]	Access row by index	df.iloc[0]
14	df[condition]	Filter rows using condition	df[df['Age'] > 25]
15	df.sort_values('col')	Sort by column	df.sort_values('Age')
16	df['new'] = ...	Add new column	df['Tax'] = df['Salary']*0.1
17	df.drop('col', axis=1)	Drop column	df.drop('Age', axis=1)
18	df.drop(index)	Drop row	df.drop(2)
19	df.isnull()	Check for missing values	df.isnull()
20	df.dropna()	Drop missing values	df.dropna()
21	df.fillna(value)	Fill missing values	df.fillna(0)
22	df.duplicated()	Check for duplicates	df.duplicated()
2S	df.drop_duplicates()	Remove duplicates	df.drop_duplicates()
24	df.mean()	Column-wise mean	df.mean()

SNO	Operation	Description	Example
25	df.sum()	Column-wise sum	df.sum()
26	df.groupby('col')	Group by column	df.groupby('City')
27	df.groupby().mean()	Group and aggregate	df.groupby('City')['Salary'].mean()
28	df.apply(func)	Apply function to rows/columns	df['Age'].apply(np.sqrt)
29	df.astype('type')	Change data type	df['Age'].astype(float)
S0	df.rename(columns={})	Rename columns	df.rename(columns={'Age':'Years'})
S1	pd.concat([df1, df2])	Concatenate DataFrames (rows)	pd.concat([df1, df2])
S2	pd.concat([df1, df2], axis=1)	Concatenate DataFrames (columns)	pd.concat([df1, df2], axis=1)
SS	pd.merge(df1, df2, on=...)	Merge DataFrames on key column	pd.merge(df1, df2, on='ID')
S4	df.to_csv('file.csv')	Export to CSV	df.to_csv('output.csv', index=False)
S5	df.plot()	Plot data (line plot by default)	df['Sales'].plot()
S6	df.value_counts()	Frequency of unique values	df['City'].value_counts()
S7	df.nunique()	Count of unique values in each column	df.nunique()
S8	df.sample(n)	Random sample of n rows	df.sample(S)
S9	df.corr()	Correlation matrix	df.corr()
40	df.reset_index()	Reset index to default integers	df.reset_index(drop=True)

Basic Operations

I. Create DataFrame

```
import pandas as pd
data = {
    'Name': ['Mahesh', 'Paani', 'Suresh'],
    'Age': [25, 30, 35],
 'City': ['Chennai', 'Bangalore', 'Tirupati']
}
df = pd.0ataFrame(data)
print(df)
```

#Output

```
   Name   Age    City
0  Mahesh  25    Chennai
1   Paani  S0   Bangalore
2  Suresh  S5    Tirupati
```

2. info()

Syntax: **info()**

Displays a concise summary of the DataFrame.

```
import pandas as pd
data = {
    'Name': ['Mahesh', 'Paani', 'Suresh'],
    'Age': [25, 30, 35],
    'City': ['Chennai', 'Bangalore', 'Tirupati']
}

df = pd.0ataFrame(data)
df.info()
```

#Output

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: S entries, 0 to 2
Data columns (total S columns):
 #  Column  Non-Null Count  Dtype
---- -------- ------------------ -------
 0  Name    S non-null     object
 1  Age     S non-null     int64
 2  City    S non-null     object
dtypes: int64(1), object(2)
memory usage: 200.0+ bytes
```

2. describe()

The describe() function in Pandas is used to generate summary statistics of numerical columns in a DataFrame.

Syntax : **df.describe()**

Metric	Meaning
count	Number of non-null entries
mean	Average of the values
std	Standard deviation
min	Minimum value
25%	1st quartile (25th percentile)
50%	Median (50th percentile)
75%	Srd quartile (75th percentile)
max	Maximum value

```
import pandas as pd
data = {
    'Name': ['Mahesh', 'Paani', 'Suresh'],
    'Age': [25, 30, 35],
    'City': ['Chennai', 'Bangalore', 'Tirupati']
}
df = pd.0ataFrame(data)
print(df.describe())
```

#Output

```
       Age
count  S.000000
mean   S0.000000
std    5.000000
min    25.000000
25%    27.500000
50%    S0.000000
75%    S2.500000
max    S5.000000
```

Note: describe () by default **only includes numeric columns**. To include all columns (including object types like strings), use:

df.describe(include='all')

3. df.shape

df.shape returns a tuple representing the **dimensions** of a DataFrame.ie in terms of **(rows, columns)**

Syntax:

df.shape

Example:

```
import pandas as pd
data = {
   'Name': ['Mahesh', 'Paani', 'Suresh'J,
   'Age': [25, 30, 35J,
   'City': ['Chennai', 'Banglore', 'Tirupati'J
}

df = pd.DataFrame(data)
print(df.shape)
```

#Output
(3, 3)

Output Explanation
S rows → one for each person: Mahesh, Paani, Suresh
S columns → Name, Age, City

For more Pandas Programs Practice-(Reference-3)

Setting up Python Environment
Anaconda and Jupiter Environments
(Reference-1)

Data Science and Machine Learning for Engineering Applications

Python installation, Anaconda-Navigator, and Jupyter notebook: beginner's tutorial

March 8, 2023 - Politecnico di Torino

Introduction

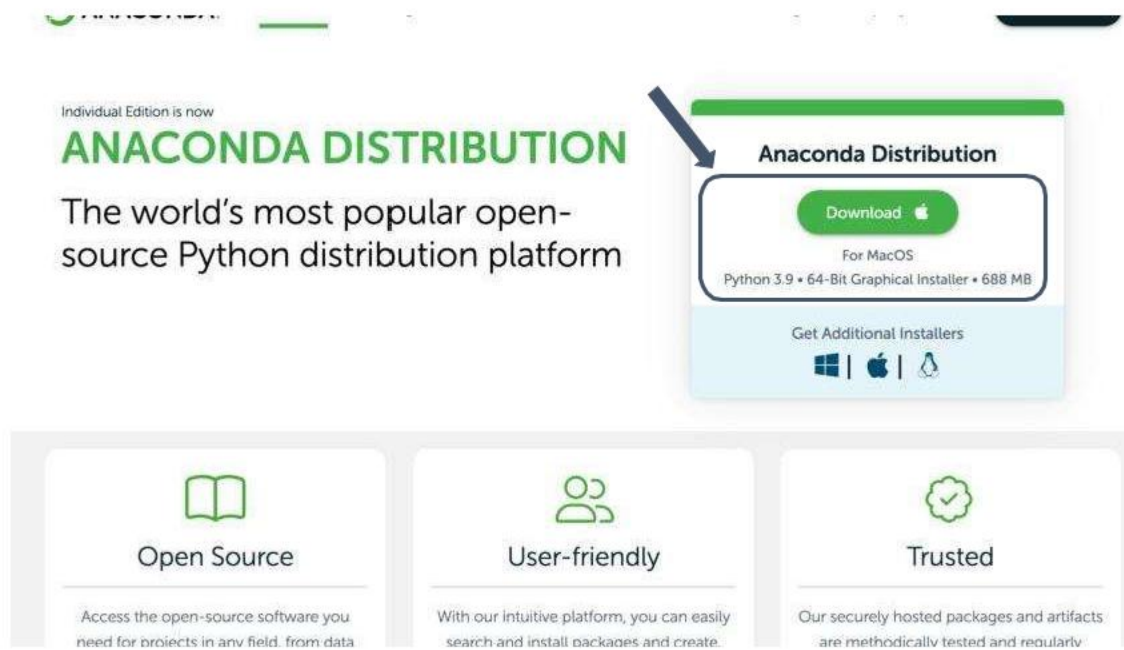
This tutorial will show you how to: i) install Python with Anaconda-Navigator (Section 1); ii) manage virtual environments with Anaconda (Section 2); iii) install python packages (Section 3); iv) use Jupyter Notebook (Section 4).

1 Install Anaconda-Navigator

Anaconda Navigator is a desktop GUI (Graphical User Interface) allowing you to launch applications and manage conda packages and environments without command-line commands. It includes a GUI, Anaconda Navigator, as a graphical alternative to the command line interface. Navigator can search for packages, install them in an environment, run the packages, and update them. The Anaconda guide can be found at the following URL: https://docs.anaconda.com/anaconda/user-guide/getting-started/.

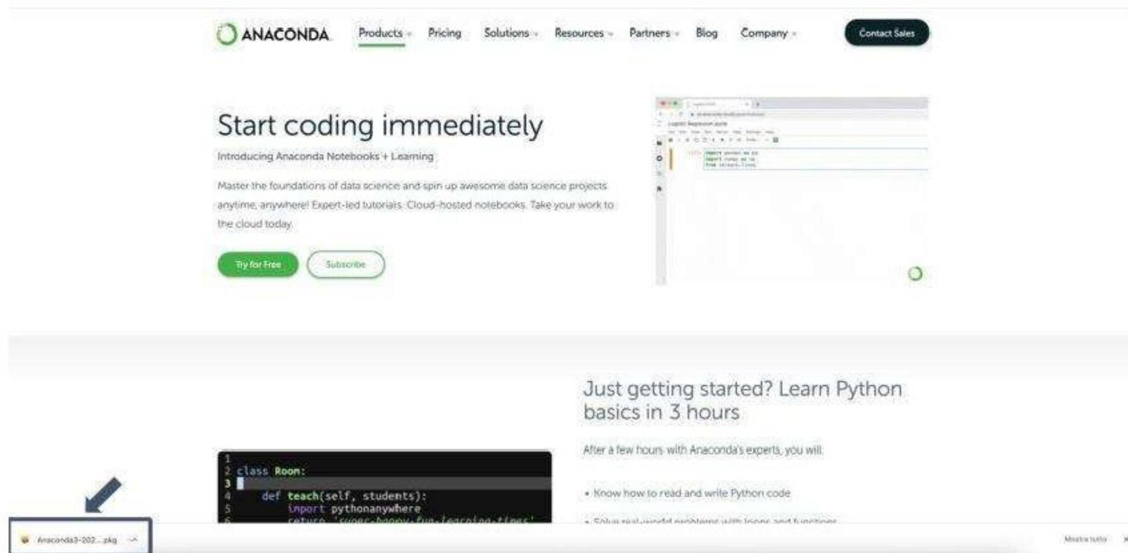
1.1 Download Anaconda-Navigator

From the Anaconda website at the following URL: https://www.anaconda.com/products/distribution, download the installation files for your operating system (i.e., MacOS, Linux, or Windows). Install the latest version of python with Anconda-Navigator. In this case, python 3.9.



1.2 Install Anaconda-Navigator

When the download is finished, double-click on the downloaded file in the bottom left-hand corner of your browser. This will start the installation of Anaconda-Navigator. The installation process depends on your operating system.



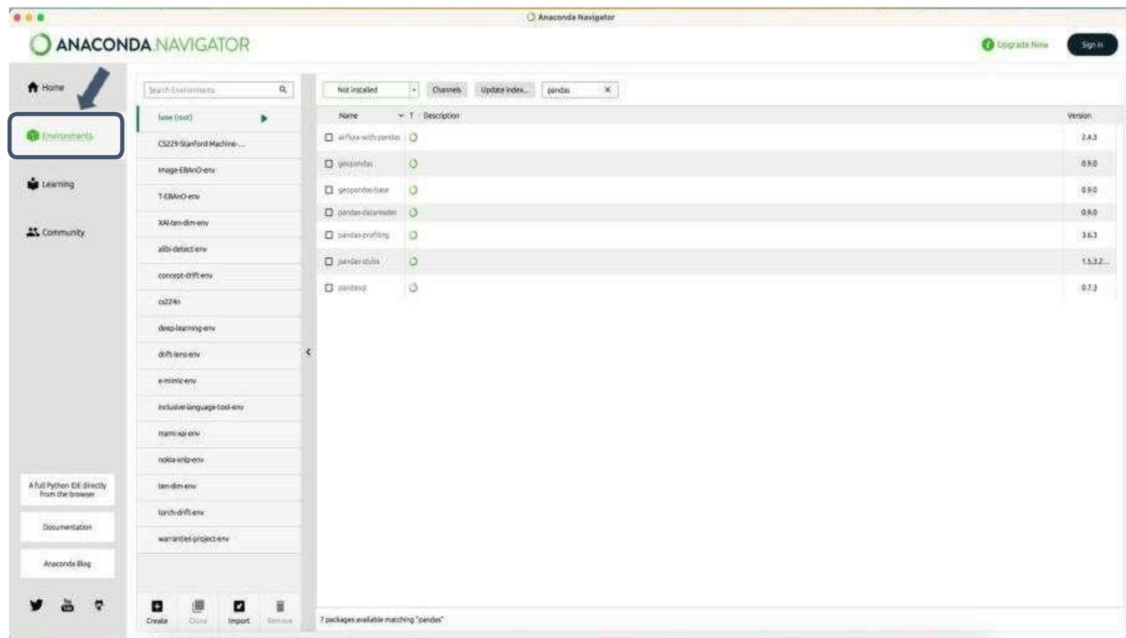
2 Create a virtual environment with Anaconda-Navigator

Python requires a different version for different kinds of applications. The application needs to run on a specific language version because it requires certain dependencies that are present in older versions but change in newer versions. Virtual environments make it easy to separate different applications and avoid problems with different dependencies [4]. Multiple ways of creating an environment include virtualenv, venv, and conda. However, the conda command is the preferred interface for managing installations and virtual environments with the Anaconda Python distribution.

This section shows how to create a virtual environment with Anaconda-Navigator, by exploiting the GUI (without the command line). If you want to learn more about creating a virtual environment with conda entirely with the command line, you can read more on this URL: https://towardsdatascience.com/manage-your-python-virtual-environment-with-conda-a0d2934d5195. This last option can be useful to run complex python projects on a remote server where the GUI is not available. However, it is not required for this course.

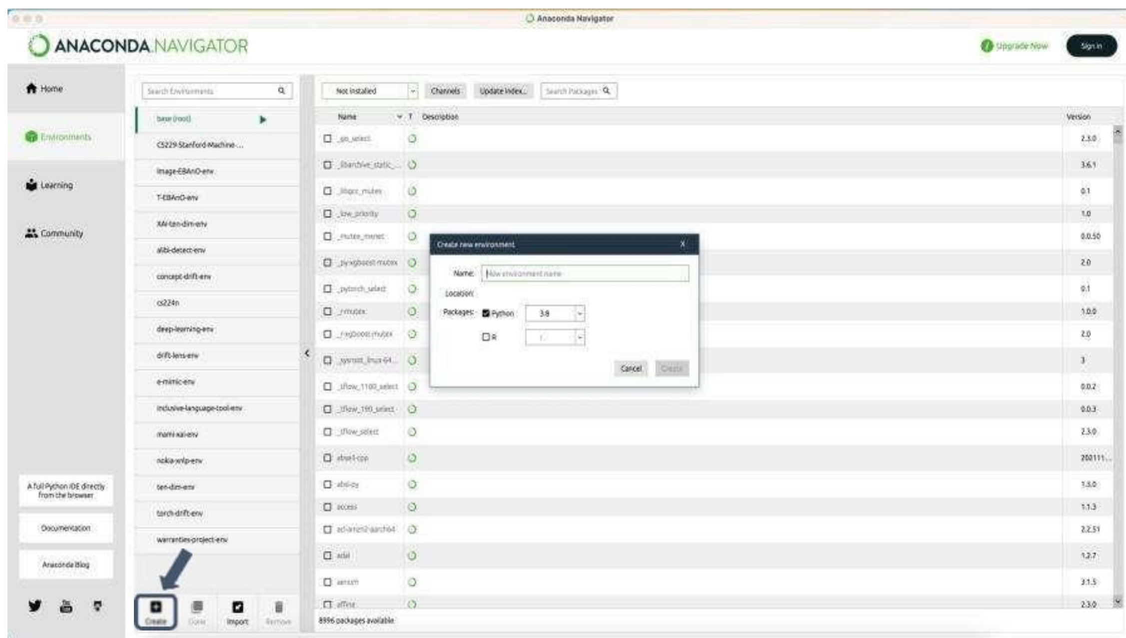
2.1 Select the environments

Click on the "Environments" button from the left menu. It will show the list of all your environments.



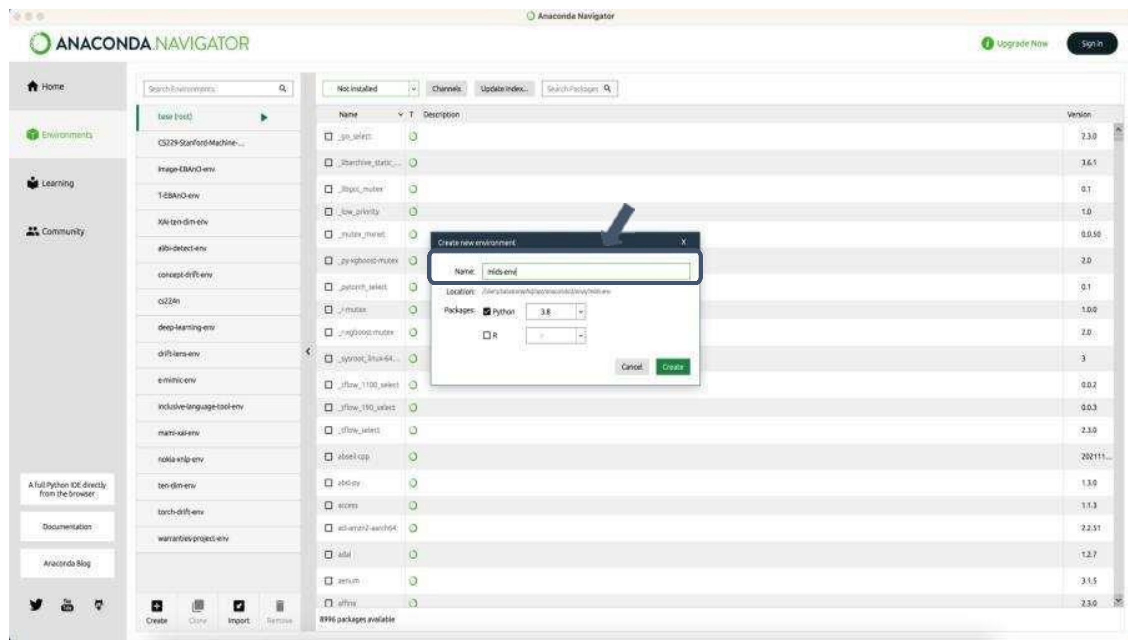
2.2 Create a virtual environment

Click the "`Create`" button in the bottom left-hand corner to create a new virtual environment.



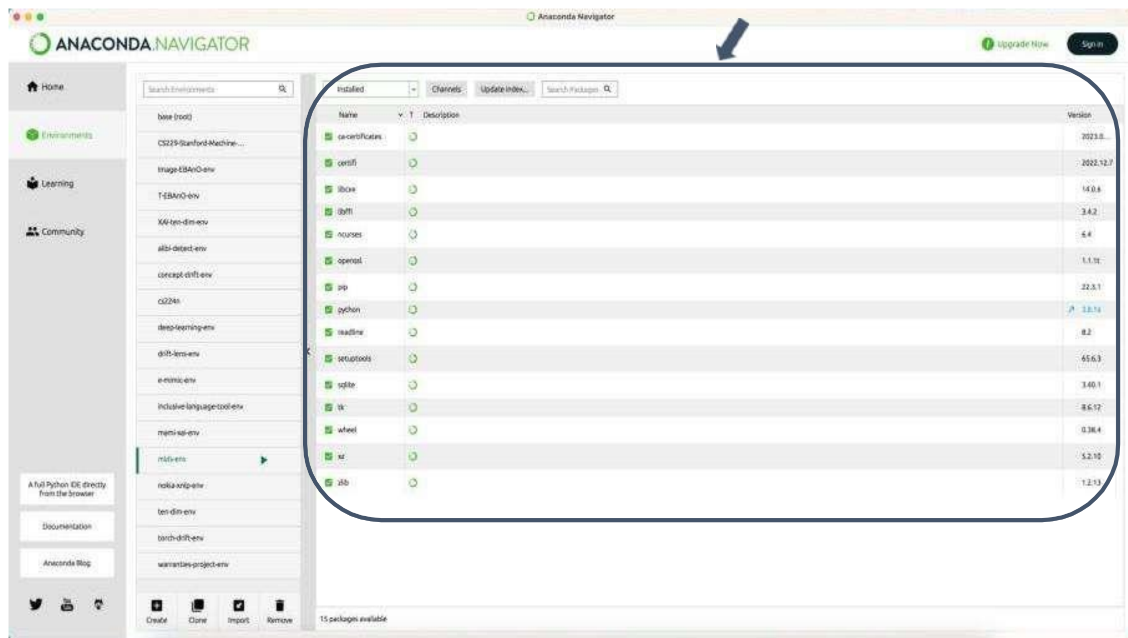
2.3 Choose a new name for your virtual environment

You have to specify the environment name and the Python version. Then, click the "`Create`" button.



2.4 Check the installed packages

Once created a new environment, the list of all installed packages in that environment will be shown. Notice that some packages are already installed.



3 Packages

To install a new package in the virtual environment, you have two options:

- Using the Anaconda-Navigator GUI directly (Section 3.1).
- Using the command line with the conda or pip commands (Section 3.2).

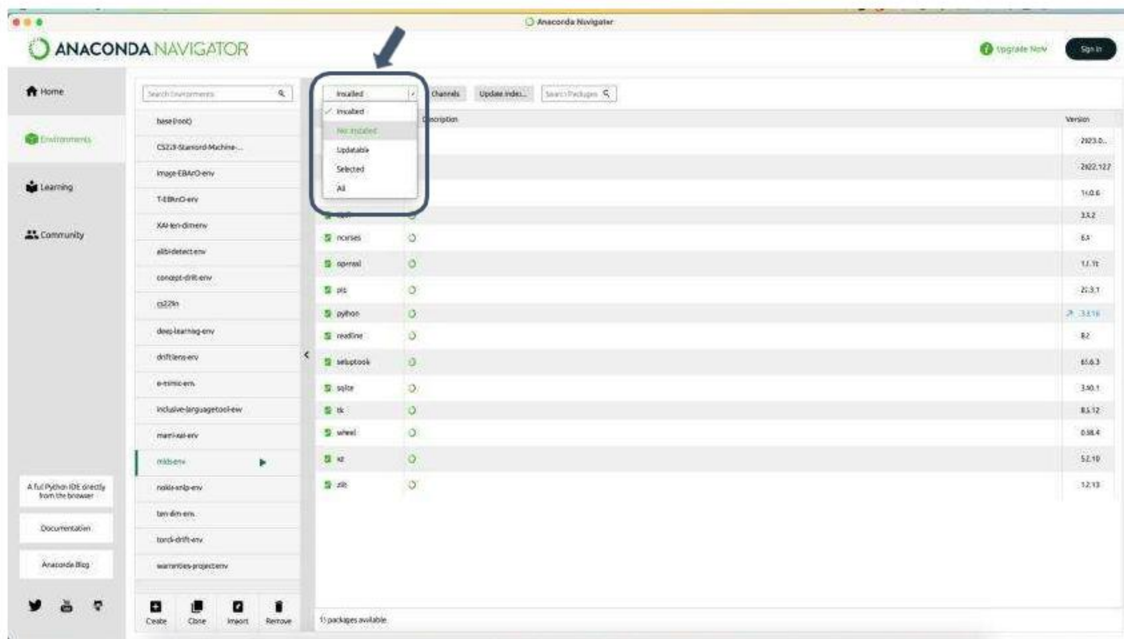
The main difference between conda and the pip package manager is how the package dependencies are managed. When pip installs a package, it also automatically installs any dependent Python packages without checking if these conflict with previously installed packages. Therefore, it will install a package and any of its dependencies regardless of the state of the existing installation. In contrast, conda analyzes the current environment, including everything currently installed and any version limitations specified. It works out how to install a compatible set of dependencies and shows a warning if this cannot be done [5]. Using the Anaconda-Navigator GUI to install a package will exploit the conda package manager. You can learn more about the differences between conda and pip at the following URL: `https://www.anaconda.com/blog/understanding-conda-and-pip`.

3.1 Install a package with the navigator GUI

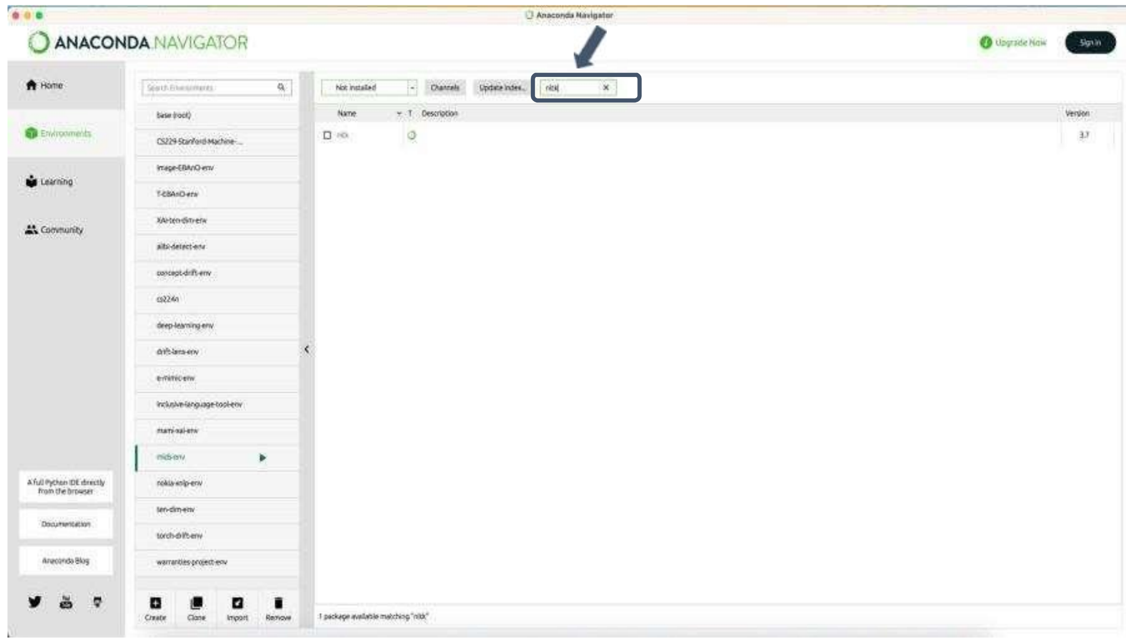
Installing any package through Anaconda-Navigator GUI is straightforward. You have to search for the required package, select a package, and click on "`Apply`" to install it

3.1.1 Search the required package

Select the option "`Not Installed`" in the top-center menu.

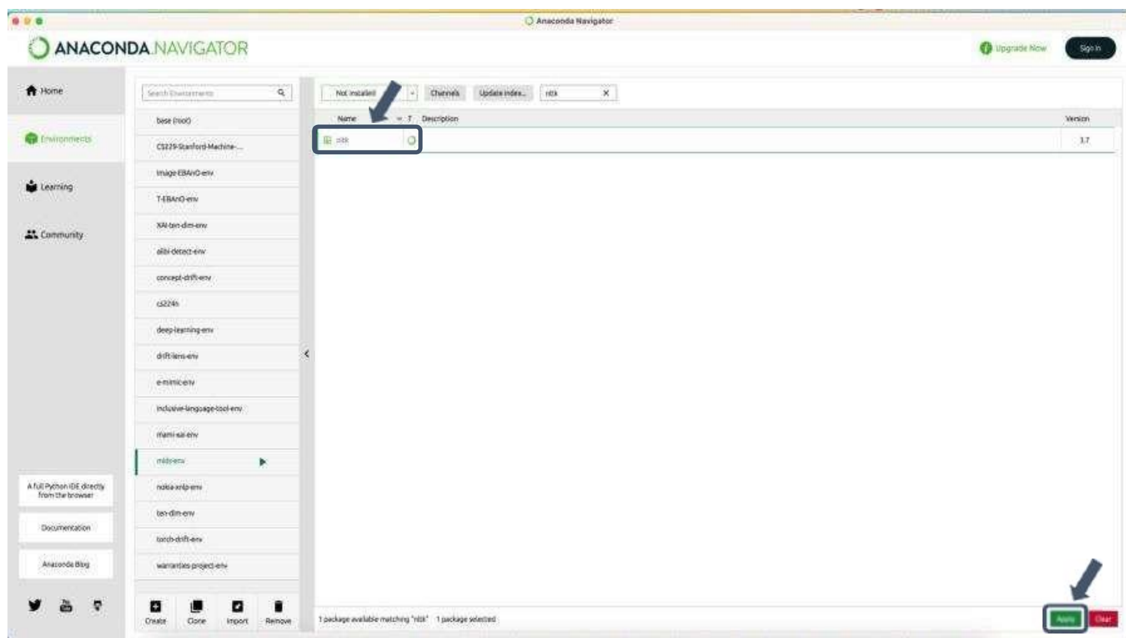


Then search for the package that you want to install by typing the name in the `textbox` (e.g., in this case, NLTK).

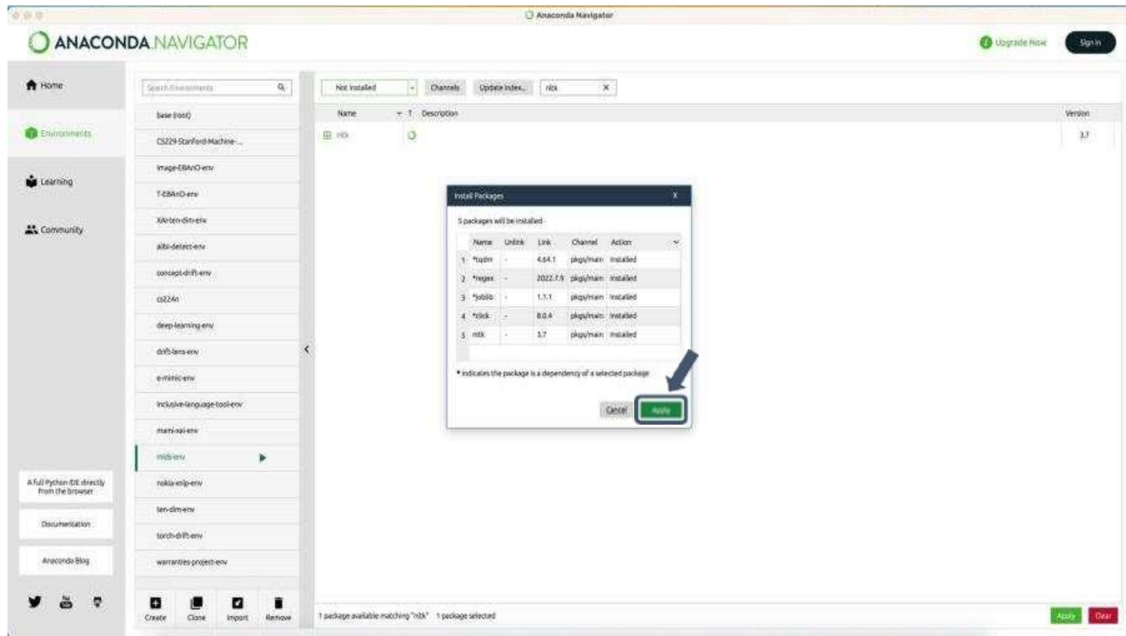


3.1.2 Select and install the required package

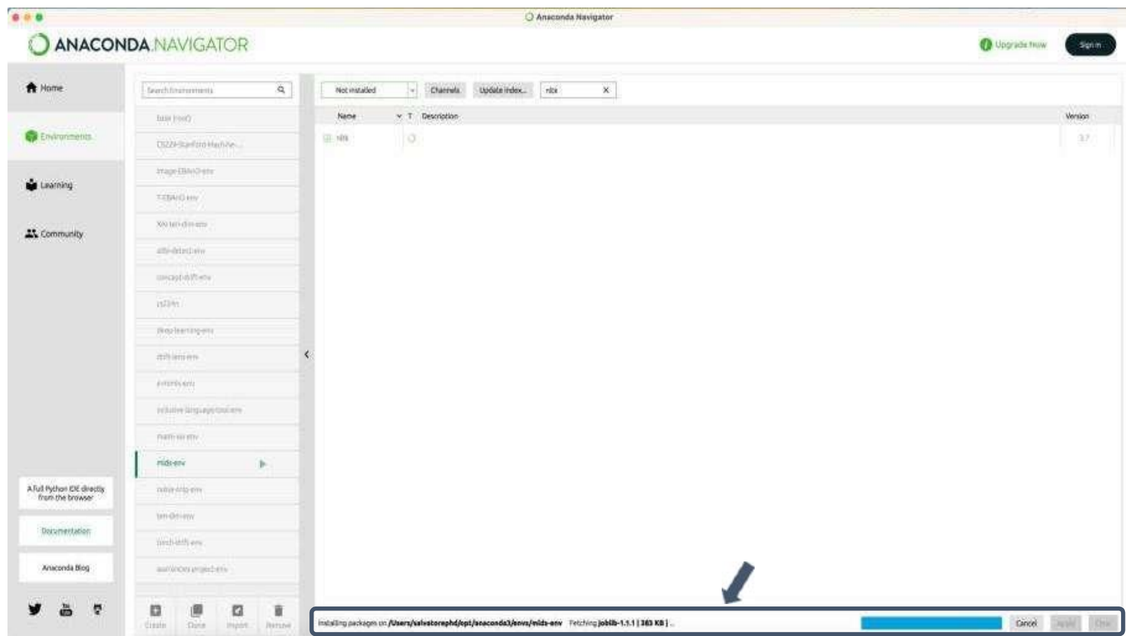
The Anaconda-Navigator will search in the conda repository for all the conda packages matching the typed name. Then, select the wanted package line and click on the "`Apply`" button in the right-hand bottom corner.



It will open a new window with all the dependencies for that package. The conda package manager will install all the dependencies for you. Click the "`Apply`" button to start the package installation.

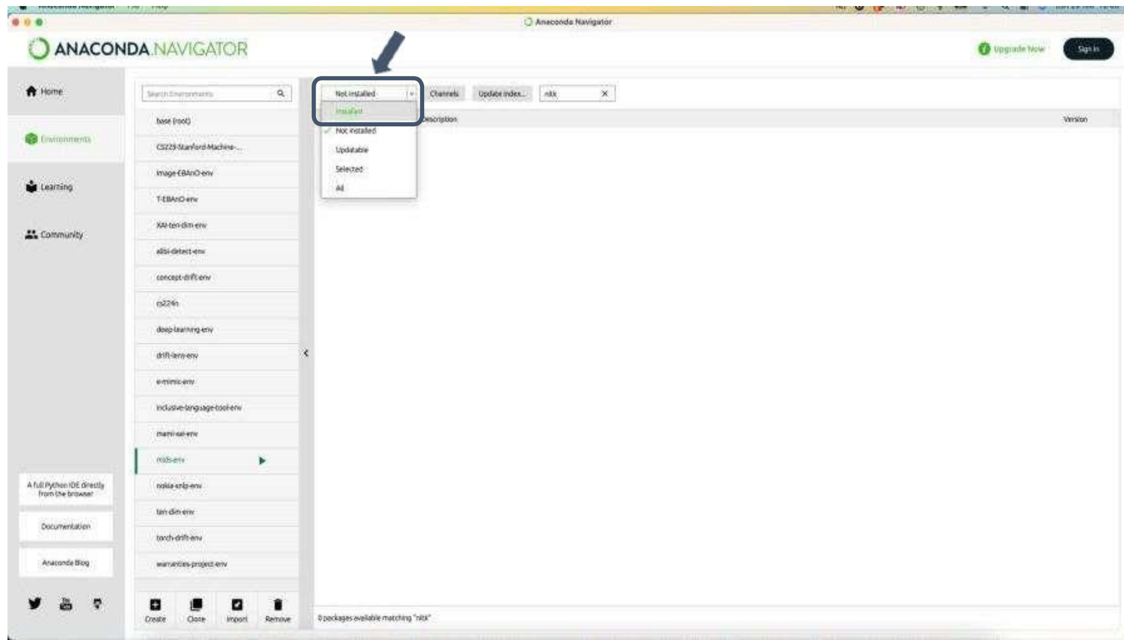


Wait for the download and installation. It could take some minutes.

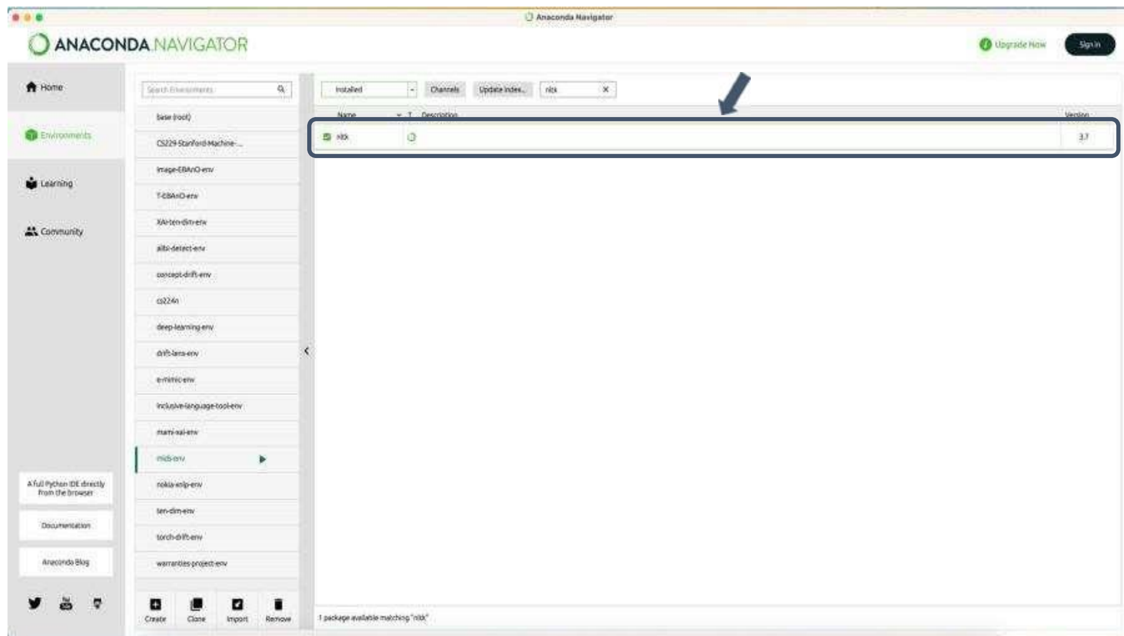


3.1.3 Check the installed package

You can check if the package has been correctly installed by selecting the "`Installed`" selection in the drop-down menu.

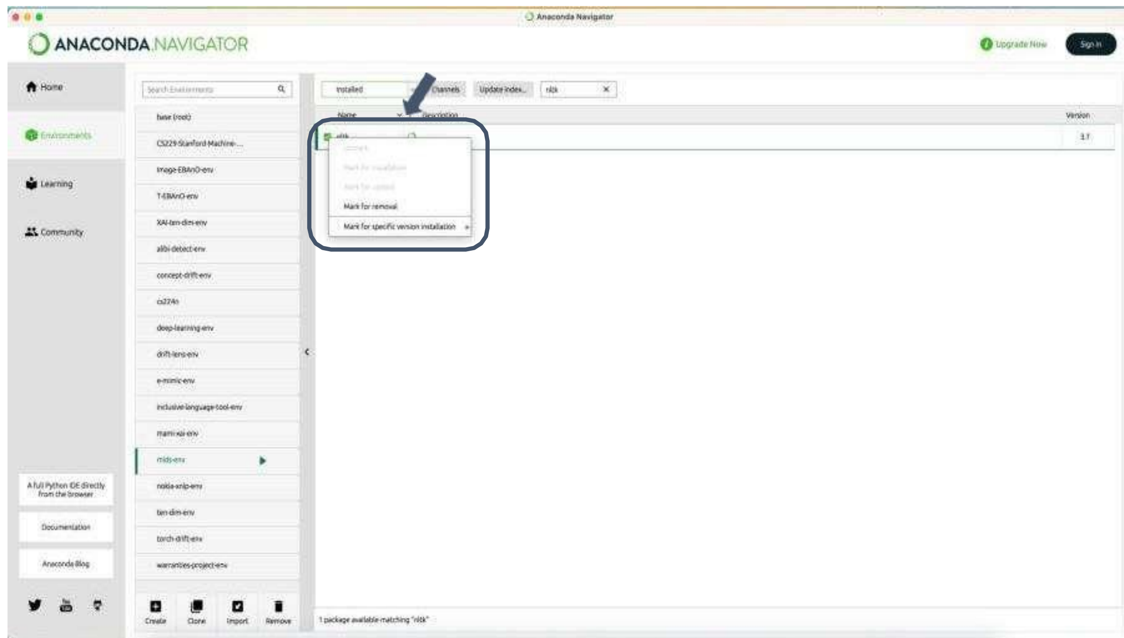


A new line corresponding to the installed package (in this case, NLTK) should appear.

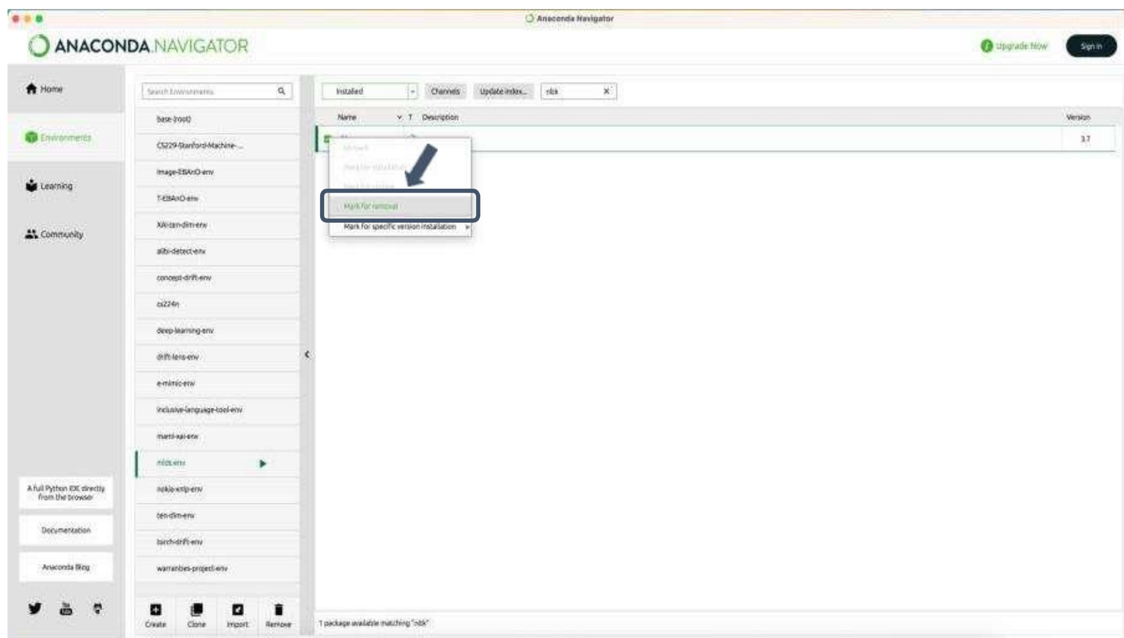


3.1.4 Uninstall the package

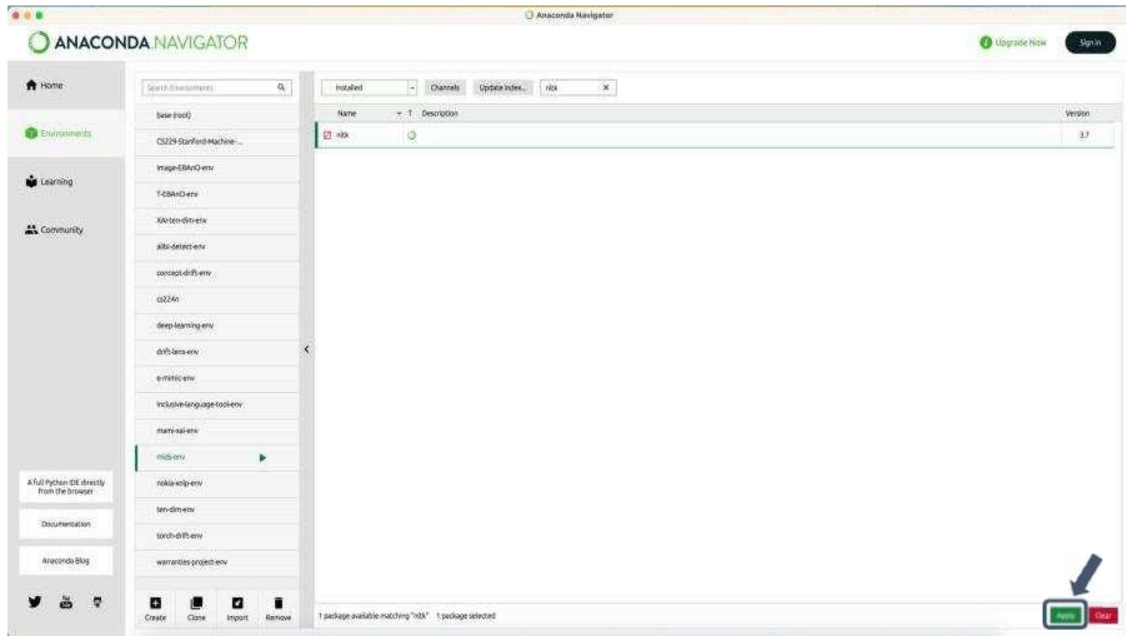
To uninstall a package, click the green √ on the line corresponding to the package you want to remove.



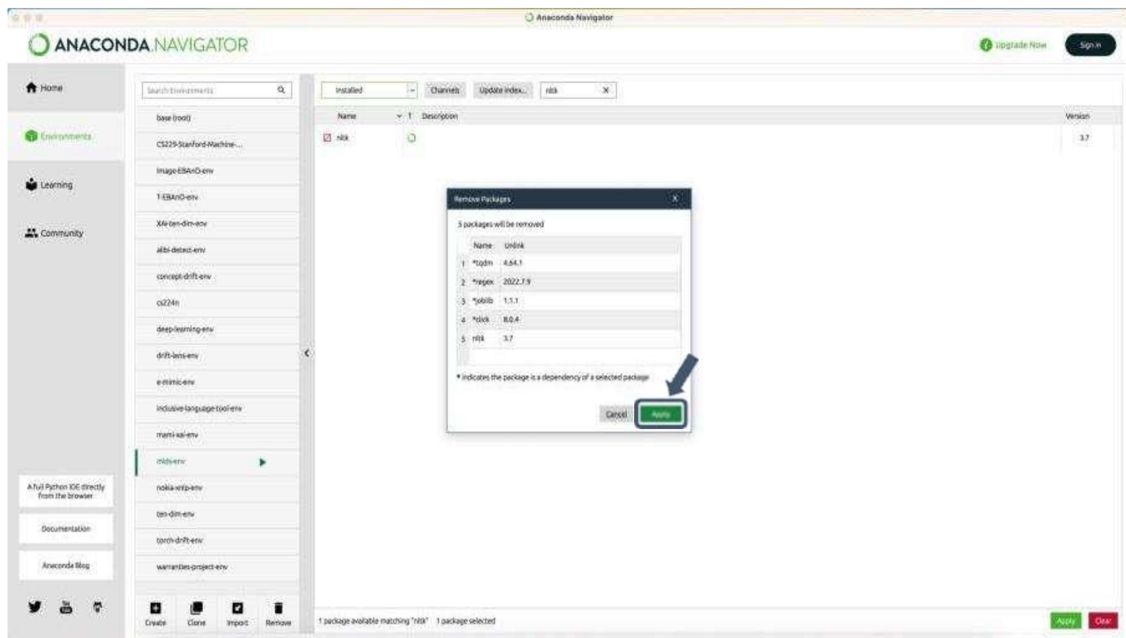
Then, click on the "`Mark for Removal`" option.



The green √ will become a red crossed box. Then, click the "`Apply`" button in the right-hand bottom corner.

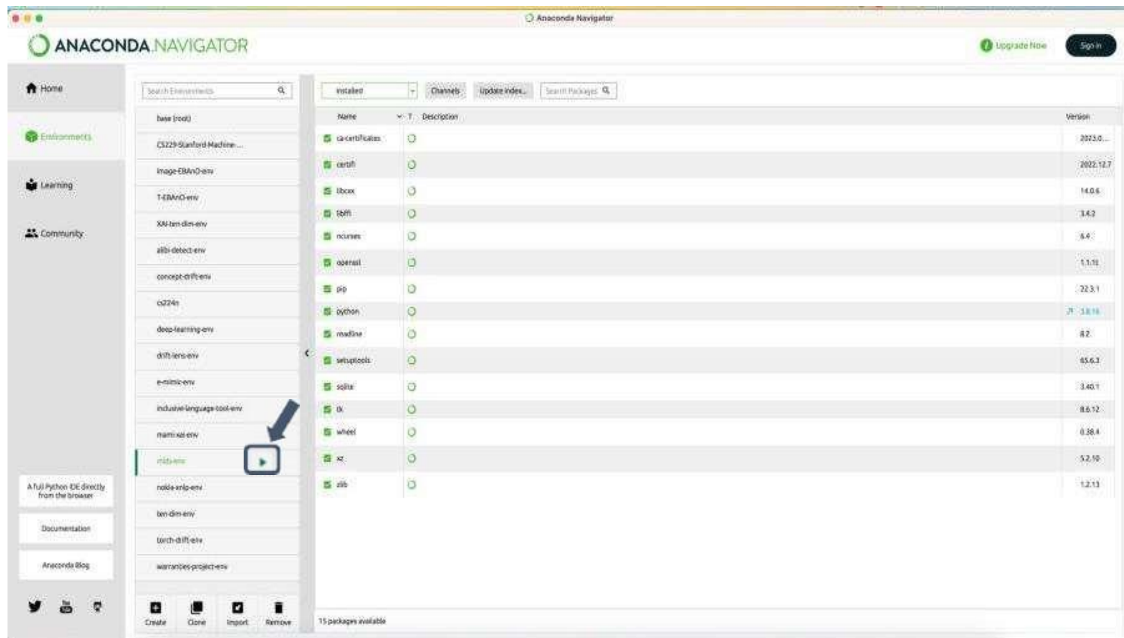


It will open a new window with all the packages that will be removed. Finally, click the "`Apply`" button in the right-hand bottom corner to start the package uninstallation.

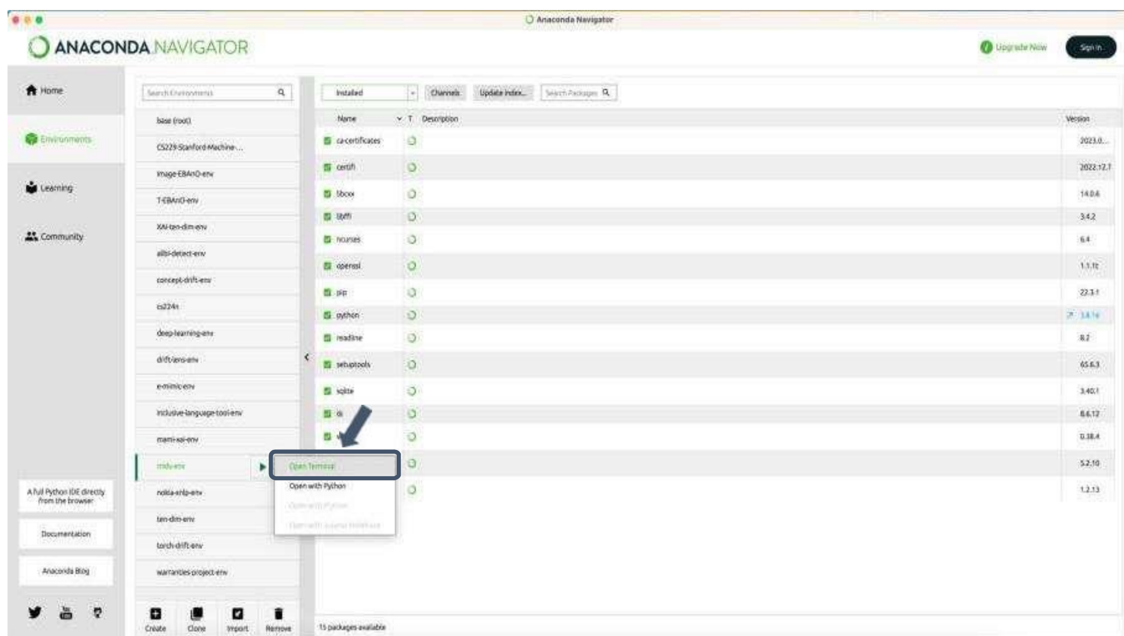


3.2 Install a package with the command line

This Section will show you how to install packages by terminal, with the pip (Section 3.2.1) and conda (Section 3.2.2) commands. To open the terminal for your environment, select the corresponding line and click the green ▷ symbol.



Then, select the "`Open Terminal`" option.



This will open the terminal with the selected environment activated (i.e., if you install a package, it will be installed in the activated environment). You can see the activated environment in the round brackets at the left of the line (e.g., *mlds-env*).

3.2.1 Install a package with the pip command

Some packages could not be available in the conda environment. You can find and install the package with another package manager like pip. To install a package with the pip command, type the command `pip install package-name`, in this case, NLTK. You can find the specific pip command for each package installation on the official documentation websites.

Press `enter` on your keyboard to start the download and the installation. It could take some minutes

3.2.2 Install a package with the conda command

Instead, to install a package with the conda command, type the command `conda install package-name`, in this case, NLTK. You can find the specific conda command for each package installation on the official documentation websites.

The terminal will show all the dependencies (i.e., other packages) that will be installed. press `y` and then `enter` to start the download and the installation.

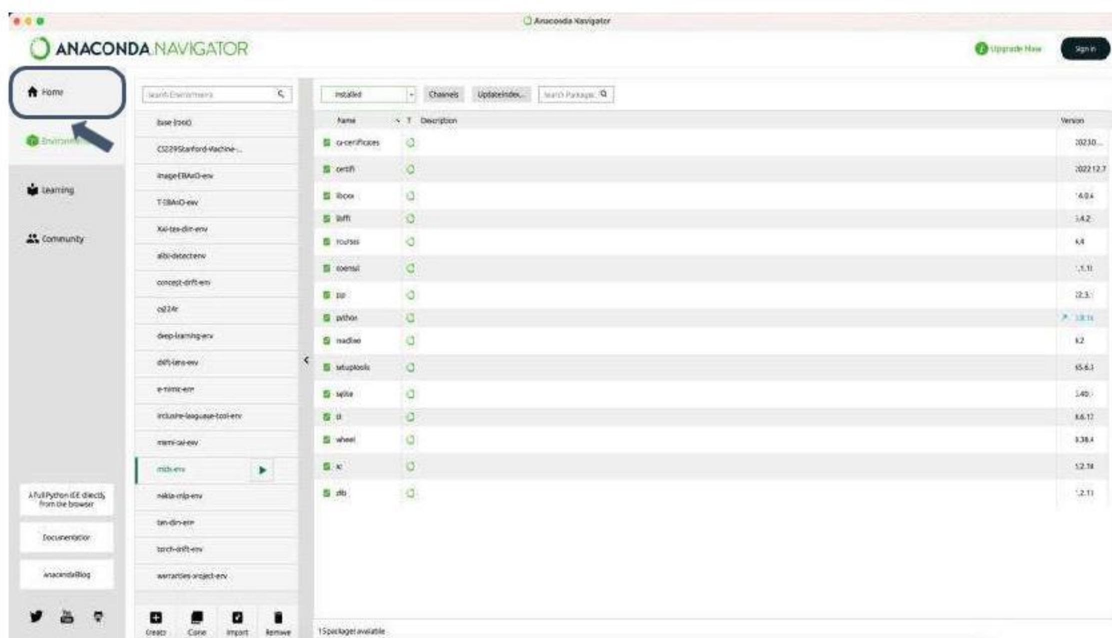
4 Jupyter notebook

Jupyter Notebook is a powerful tool for developing and presenting data science projects interactively. In a Jupyter Notebook document, you can combine code, visualizations, texts, and display outputs. You can find a good guide at the following URL [3]: https://www.dataquest.io/blog/jupyter-notebook-tutorial/.

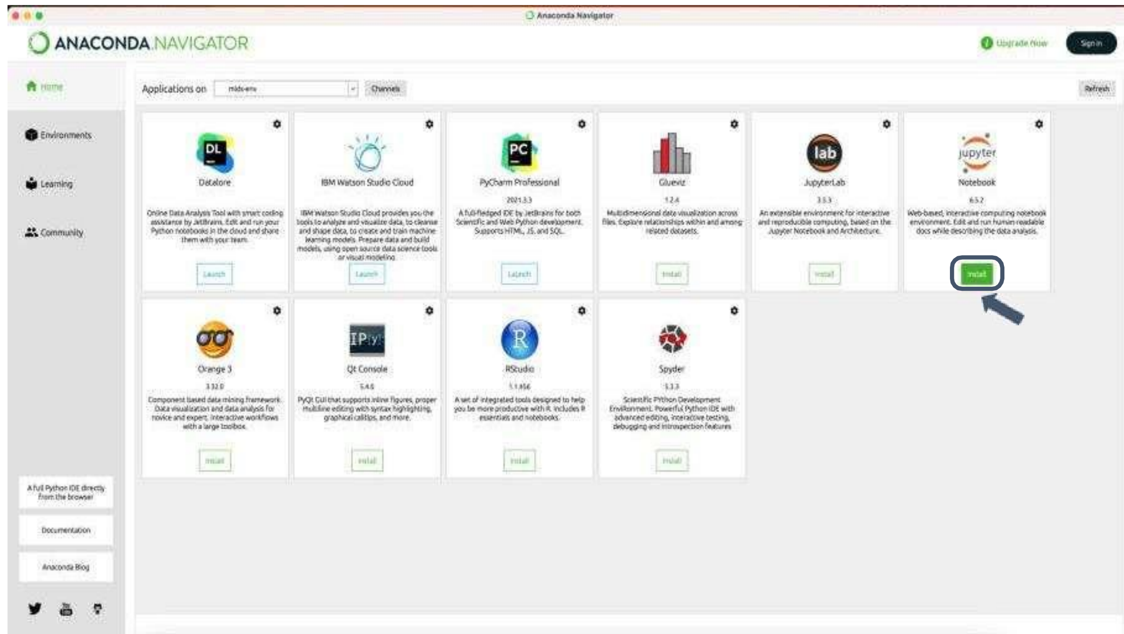
The following sections will show you how to: i) install Jupyter notebook using Anaconda (Section 4.1); ii) launch Jupyter from Anaconda 4.2; iii) create your first Jupyter notebook 4.3; iv) use cells and kernels to effectively exploit Jupyter notebooks (Sections 4.4 and 4.5); v) exploit advanced features of Jupyter notebooks (Section 4.6).

4.1 Install Jupyter

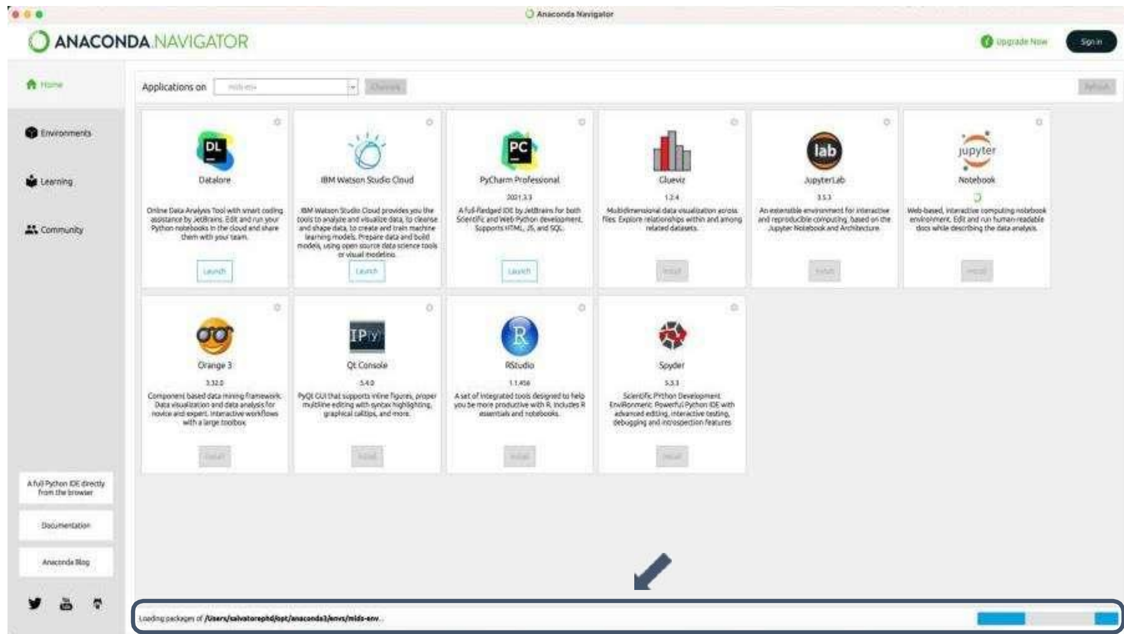
To install Jupyter, you must first go into the `Home` section, which contains all the applications for the current environment. Please check that the created virtual environment is selected (in this case, *mlds-env*).



Then, click the "`Install`" button under the Jupyter application box.



This will start the download and installation process. It may require some minutes.



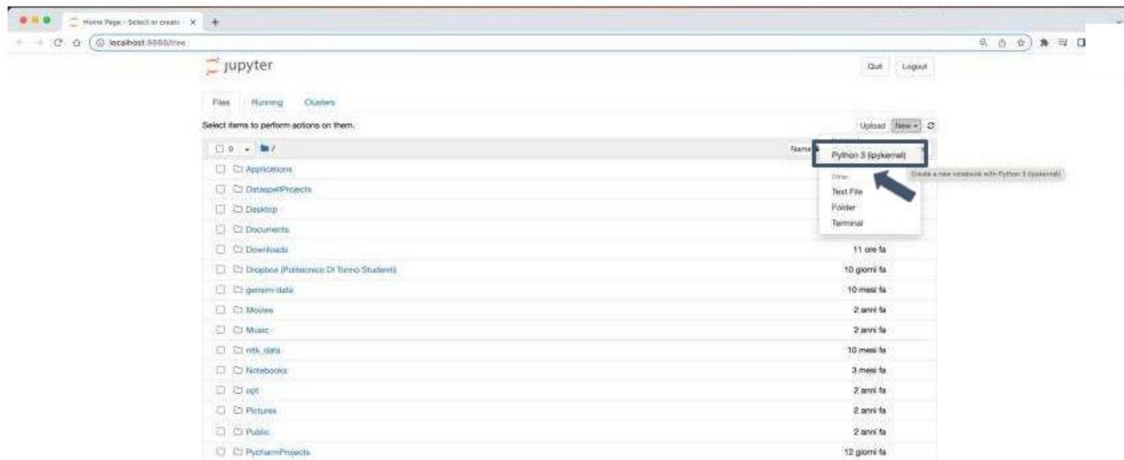
4.2 Launch Jupyter

After the installation, the Anaconda dashboard will show you the "`Launch`" button under the Jupyter Notebook box. Click on "`Launch`" to start Jupyter notebook.

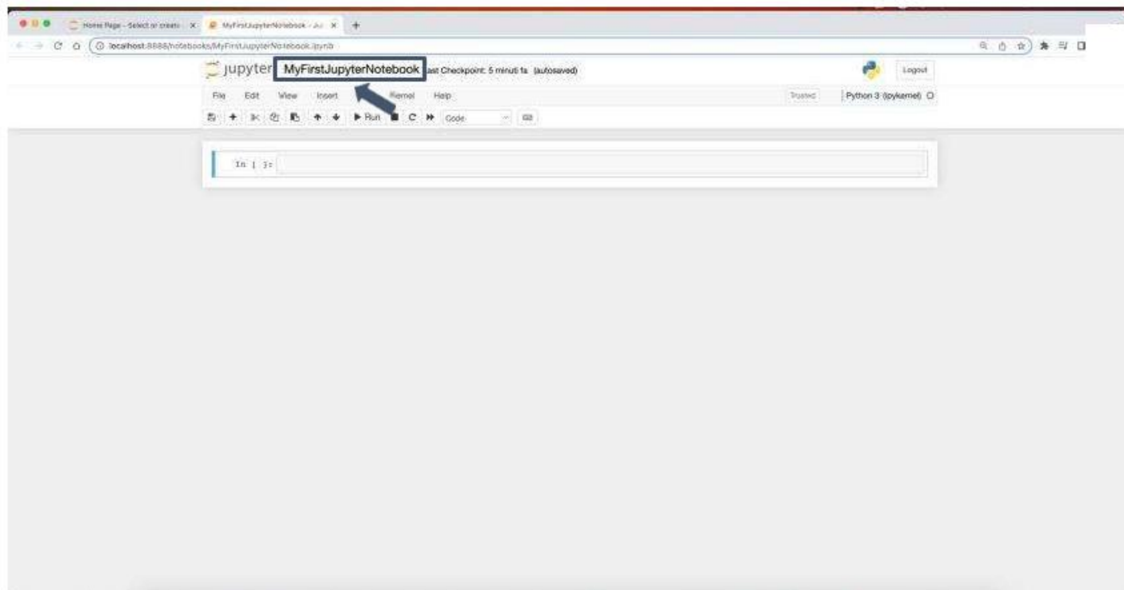
It will open the Notebook Dashboard for exploring, editing, and creating notebooks. Here you can create new folders, notebooks, etc. The URL for the dashboard is *https://localhost:8888/tree*. Localhost is not a website but indicates that the content is run on your local machine.

4.3 Create a new Jupyter notebook

To create your first Jupyter notebook click the "`New`" drop-down button in the top-right menu and select "`Python 3`". This will open your first Jupyter notebook in a new tab. You can open and run multiple notebooks simultaneously in multiple tabs.



It will create a new file `Untitled.ipynb`. Each `.ipynb` file is a text file that describes the contents of your notebook in a format called JSON. Each time you create a new notebook, a new `.ipynb` file will be created. Notice that the notebook extension `.ipynb` is different from the normal python file extension `.py`. Please rename now your filename from the top text box, or, very soon, you will have several `Untitled.ipynb`, `Untitled (1).ipynb` notebooks. The notebook's name should explain the content.



4.3.1 The Jupyter Notebook interface

The two main concepts that you should learn to use notebooks properly are cells and kernels:

- The `cell` is a container for code to be executed or text to be displayed in the notebook by the kernel (Section 4.4).

- The `kernel` is a computational engine that executes the code contained in a notebook document (Section 4.5).

4.4 Notebook cells

Cells compose the body of the notebook. They could contain code, plain text, images, LaTeX, math formulas, etc. There are two main cell types that you should learn:

- Code cells (Section 4.4.1)
- Markdown cells (Section 4.4.2)

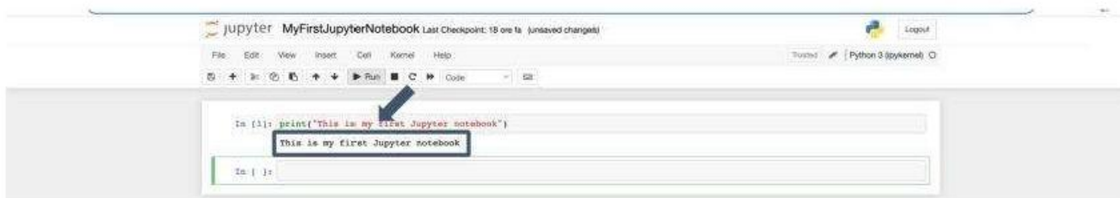
4.4.1 Code cell

Code cells contain code to be executed in the kernel. When the code is run, the notebook displays the output below the code cell that generated it. Note that cells do not have to be executed in order. It is also possible to execute a cell at the end and then one at the beginning of the notebook. The cell type is shown in the drop-down menu. The default type is Code.

You can run a Code cell by:

- clicking the "`Run`" button
- pressing `ctr + enter`
- pressing `maiusc + enter` (in this case, it also goes to the next cell)

In this case, the execution of the cell will print the string "`This is my first Jupyter notebook`" as output. Each cell could produce an output.

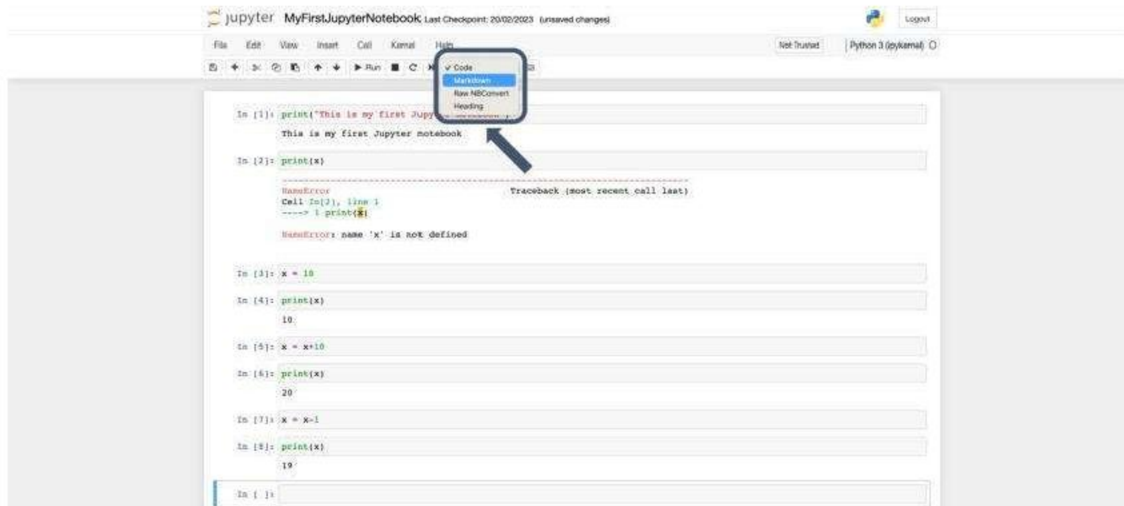


The following cell will create a new variable called x and assigns the values of 10 to x. In this case, no output is produced by the cell.

You should use the print function to output the value of *x*.

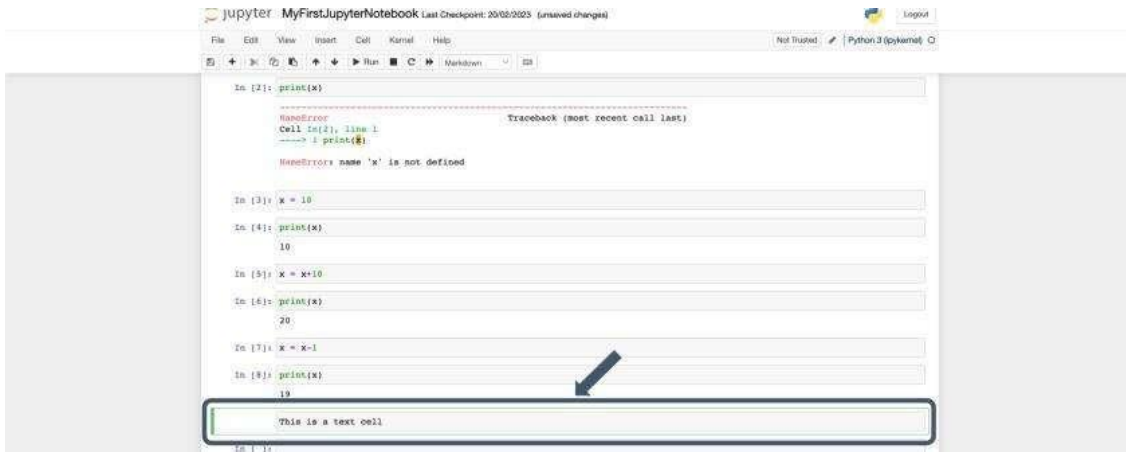
4.4.2 Markdown cell

Markdown cells contain text formatted using Markdown [2] and displays its output in-place when the Markdown cell is run. Markdown is a lightweight markup language that you can use to add formatting elements to plaintext text documents. This cheat sheet will cover the most common elements (cheatsheet). To define a Markdown cell, select the cell and click on the "`Markdown`" option in the top drop-down menu.

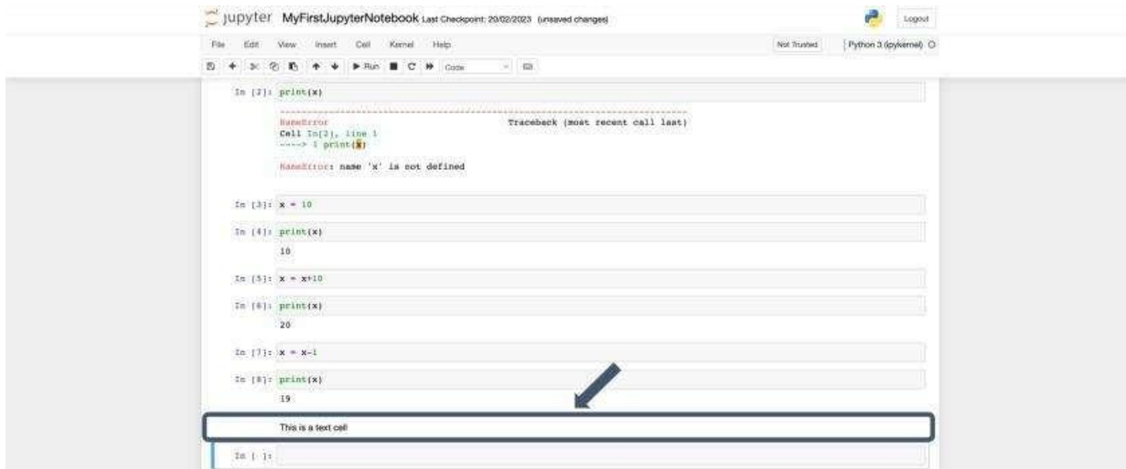


4.4.3 Text Markdown cell

You can write plaintext in a cell. This text is not a code. Therefore, it will not be really executed. You can also empathize text with bold or *italic* with `**bold**` and `*italic*` respectively.

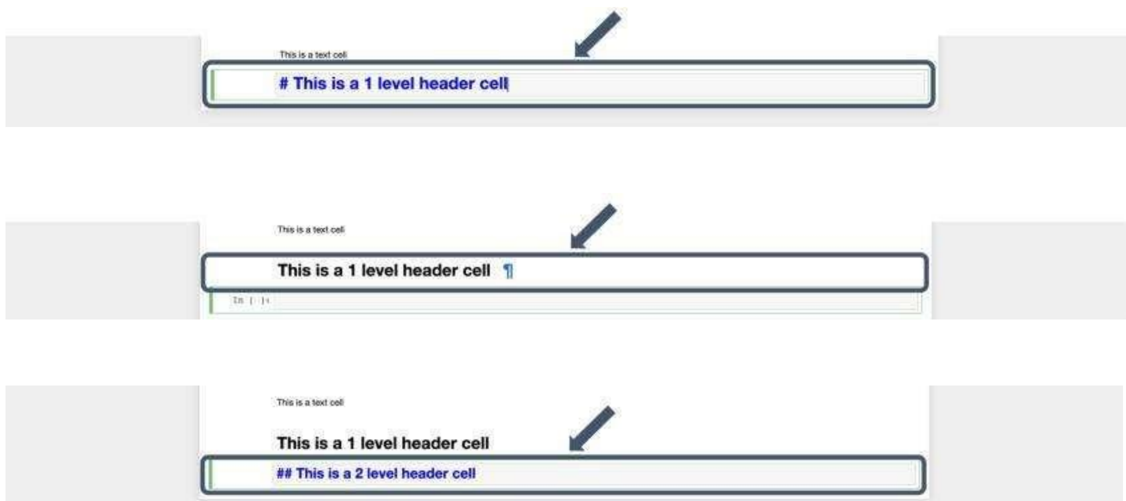


If you run the cell containing plaintext, it will be displayed formatted as output. This can add narrative to your Jupyter notebook.



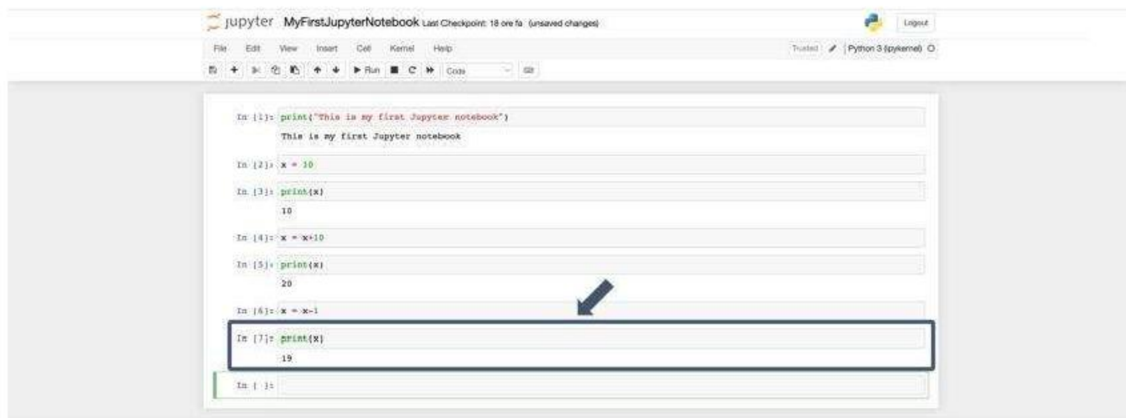
4.4.4 Heading Markdown cell

You can also add first, second, and third-level headings.



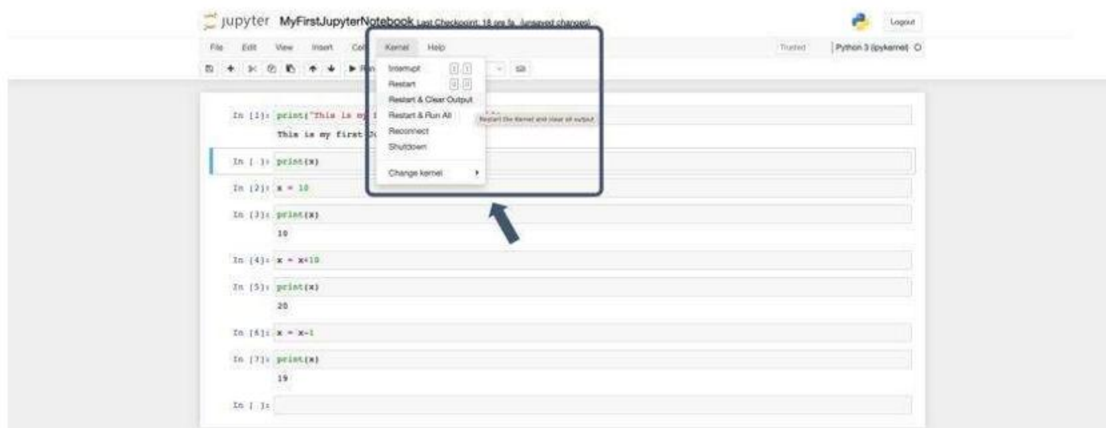
4.5 Notebook kernel

When you run a code cell, that code is executed within the kernel, and the outputs are returned to the cells to be displayed. The kernel's state persists over time between cells. It pertains to the document as a whole and not individual cells. For example, if you import libraries in one cell, they will be available in another. If you define the value of a variable in one cell, the variable's value also persists for the other cells.

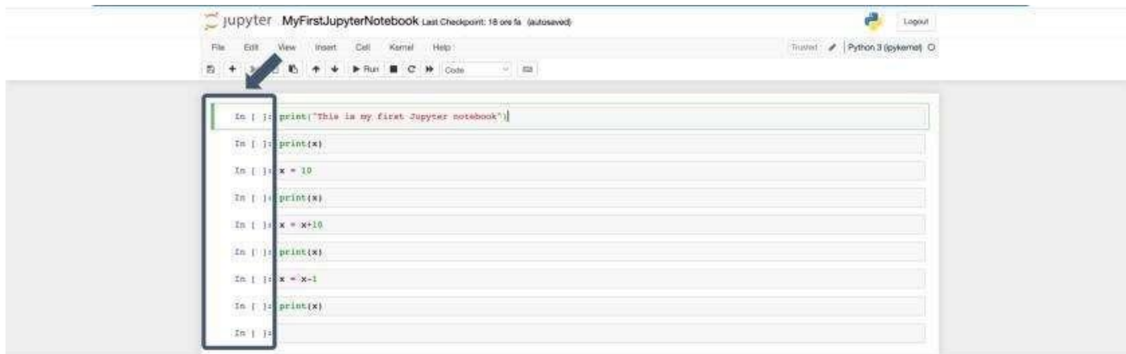


4.5.1 Restarting a kernel

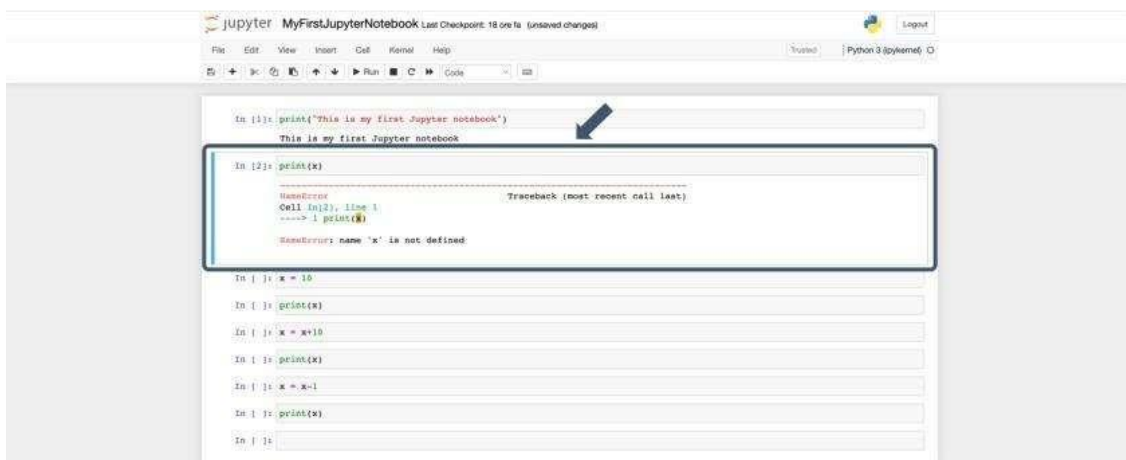
If you restart the kernel, the notebook's status is deleted. After the kernel's restart, all the values of your variables are reset. To restart the kernel, click the "`Kernel`" button in the top menu. Then, select "`Restart & Clear Output`". This will restart your kernel and clear all the outputs in the cells. You can also select "`Restart & Run All`" to restart your kernel and run all cells in order.



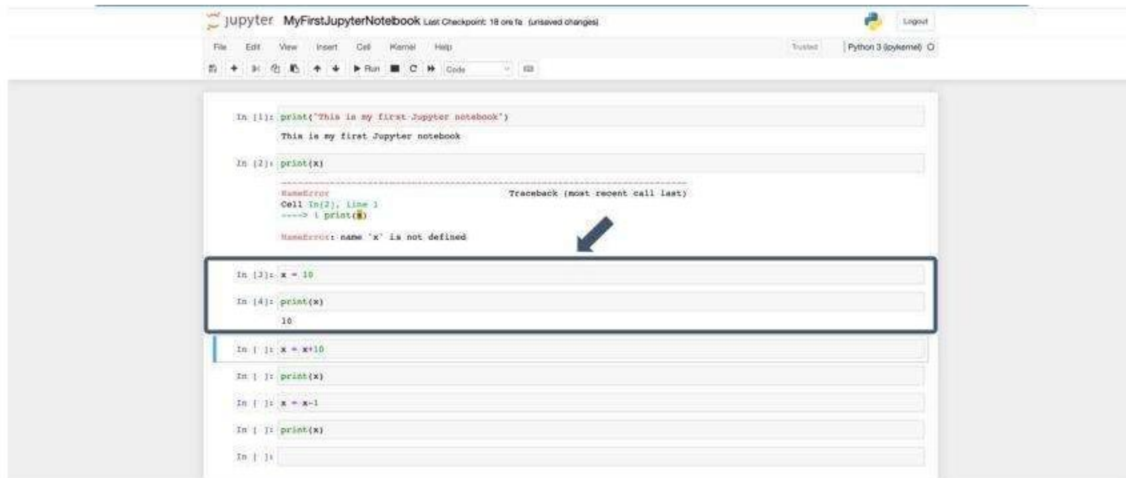
Restarting the kernel clears all the cells' outputs and initializes the run identification number of each cell.



What do you think will happen if you now print the value of the variable x again? It will raise an error message because restarting the kernel caused a reset of the notebook status and, consequently, all the previously defined variables.



Therefore, you should define x again to print its value.



4.6 Jupyter notebook advanced tips, tricks, and shortcuts

More advanced tips and commands such as keyboard shortcuts, pretty display, executing shell commands, using LaTeX could be found here [1] (https://www.dataquest.io/blog/jupyter-notebook-tips-tricks-shortcuts/)

References

[1] Dataquest. *28 Jupyter notebook tips, tricks, and shortcuts*. Feb. 2023. url: https://www.dataquest.io/blog/jupyter-notebook-tips-tricks-shortcuts/.

[2] *Markdown guide*. url: https://www.markdownguide.org/.

[3] Benjamin Pryke. *How to use Jupyter Notebook: A beginner's tutorial*. Feb. 2023. url: https://www.dataquest.io/blog/jupyter-notebook-tutorial/.

[4] *Set up virtual environment for python using anaconda*. Apr. 2022. url: https://www.geeksforgeeks.org/set-up-virtual-environment-for-python-using-anaconda/.

[5] *What is Anaconda?: Domino data science dictionary*. url: https://www.dominodatalab.com/data-science- dictionary/anaconda#:~:text=Anaconda%20Navigator%20is%20included%20in, the%20packages%20and%20update%20them..

Numpy Programs Practice
(Reference-2)

In [2]:

```
#NdArray:The Heart of the Library
import numpy as np
a=np.array([1,2,3])
print('My Array is:',a)
print('Type:',a.dtype)
print('No.of dimenstions:',a.ndim)
print('Size:',a.size)
print('Shape:',a.shape)
print('ItemSize:',a.itemsize)
```

```
My Array is: [1 2 3]
Type: int32
No.of dimenstions: 1
Size: 3
Shape: (3,)
ItemSize: 4
```

In [3]:

```
import numpy as np
a=np.array([[1,2,3],[4,5,6]])
print('My Array is:',a)
print('Type:',a.dtype)
print('No.of dimenstions:',a.ndim)
print('Size:',a.size)
print('Shape:',a.shape)
print('ItemSize:',a.itemsize)
```

```
My Array is: [[1 2 3]
 [4 5 6]]
Type: int32
No.of dimenstions: 2
Size: 6
Shape: (2, 3)
ItemSize: 4
```

In [44]:

```
import numpy as np
a=np.array([[1,2,3],[4,5,6],[7,8,9]])
print('My Array is:',a)
print('Type:',a.dtype)
print('No.of dimenstions:',a.ndim)
print('Size:',a.size)
print('Shape:',a.shape)
print('ItemSize:',a.itemsize)
b=np.array([(1,2,3),(4,5)])
print(b)
print(b.shape)
c=np.array([[[1,2,3],[4,5,6],[7,8,9]]])
print(c)
print(c.shape)
```

```
My Array is: [[1 2 3]
 [4 5 6]
 [7 8 9]]
Type: int32
No.of dimenstions: 2
Size: 9
Shape: (3, 3)
ItemSize: 4
[(1, 2, 3) (4, 5)]
(2,)
```

```
  [7 8 9]]]
(1, 3, 3)
```

```
C:\Users\Lenovo\AppData\Local\Temp/ipykernel_8164/687956672.py:9: VisibleDeprecation
Warning: Creating an ndarray from ragged nested sequences (which is a list-or-tuple
of lists-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If y
ou meant to do this, you must specify 'dtype=object' when creating the ndarray.
  b=np.array([(1,2,3),(4,5)])
```

In [19]:

```
b=np.array([['a','b'],['c','d']],dtype='U1')
print(b)
print(b.dtype)
print(b.dtype.name)
b=np.array([[1,2],[3,4]],dtype=complex)
print(b)
print(b.dtype)
print(b.dtype.name)
```

```
[['a' 'b']
 ['c' 'd']]
<U1
str32
[[1.+0.j 2.+0.j]
 [3.+0.j 4.+0.j]]
complex128
complex128
```

In [23]:

```
c=np.zeros((3,3))
print(c)
d=np.ones((3,3))
print(d)
e=np.full((3,3),7)
print(e)
f=np.eye(3)
print(f)
g=np.random.random((3,3))
print(g)
```

```
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
[[7 7 7]
 [7 7 7]
 [7 7 7]]
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
[[0.94112564 0.55562834 0.45522268]
 [0.71367114 0.31149873 0.41317839]
 [0.12033261 0.12924145 0.39662475]]
```

In [32]:

```
arr1=np.arange(0,5)
print('arr1:',arr1)
arr2=np.arange(1,6)
print('arr2:',arr2)
arr3=np.arange(1,9,2)
print('arr3:',arr3)
arr4=np.arange(1,9,1.5)
print('arr4:',arr4)
```

```
print('arr5:',arr5)
arr6=np.linspace(0,10,6)
print('arr6:',arr6)
arr7=np.linspace(0,10,5)
print('arr7:',arr7)
```

```
arr1: [0 1 2 3 4]
arr2: [1 2 3 4 5]
arr3: [1 3 5 7]
arr4: [1.  2.5 4.  5.5 7.  8.5]
arr5: [1. 3. 5. 7. 9.]
arr6: [ 0.  2.  4.  6.  8. 10.]
arr7: [ 0.   2.5  5.   7.5 10. ]
```

In [43]:

```
arr1=np.arange(1,5)
print('arr1:',arr1)
arr2=np.arange(1,5).reshape(2,2)
print('arr2:',arr2)
arr3=np.arange(1,7).reshape(2,3)
print('arr3:',arr3)
arr4=np.arange(1,7).reshape(1,6)
print('arr4:',arr4)
arr5=np.arange(1,7).reshape(6,1)
print('arr5:',arr5)
arr6=np.arange(1,10).reshape(3,3)
print('arr6:',arr6)
```

```
arr1: [1 2 3 4]
arr2: [[1 2]
 [3 4]]
arr3: [[1 2 3]
 [4 5 6]]
arr4: [[1 2 3 4 5 6]]
arr5: [[1]
 [2]
 [3]
 [4]
 [5]
 [6]]
arr6: [[1 2 3]
 [4 5 6]
 [7 8 9]]
```

In [52]:

```
#Basic Operations
arr1=np.arange(1,5).reshape(2,2)
print('arr1:',arr1)
print('Addition:',arr1+4)
print('Subtraction:',arr1-4)
print('Multiplication:',arr1*2)
print('Division:',arr1//2)
arr1+=1
print('Increment:',arr1)
arr1-=1
print('Decrement:',arr1)
arr2=np.sqrt(arr1)
print('arr2:',arr2)
arr3=np.sin(arr1)
print('arr3:',arr3)
```

```
arr1: [[1 2]
 [3 4]]
```

```
 [7 8]]
Subtraction: [[-3 -2]
 [-1  0]]
Multiplication: [[2 4]
 [6 8]]
Division: [[0 1]
 [1 2]]
Increment: [[2 3]
 [4 5]]
Decrement: [[1 2]
 [3 4]]
arr2: [[1.         1.41421356]
 [1.73205081 2.        ]]
arr3: [[ 0.84147098  0.90929743]
 [ 0.14112001 -0.7568025 ]]
```

In [58]:

```
a=np.arange(1,10).reshape(3,3)
print('A:',a)
b=np.ones((3,3))
print('B:',b)
print('Element  wise  product:',a*b)
print('Matrix Product:',np.dot(a,b)) #a.dot(b)
print('Matrix Product:',b.dot(a))    #np.dot(b,a)
```

```
A: [[1 2 3]
 [4 5 6]
 [7 8 9]]
B: [[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
Element wise product: [[1. 2. 3.]
 [4. 5. 6.]
 [7. 8. 9.]]
Matrix Product: [[ 6.  6.  6.]
 [15. 15. 15.]
 [24. 24. 24.]]
Matrix Product: [[12. 15. 18.]
 [12. 15. 18.]
 [12. 15. 18.]]
```

In [61]:

```
#Aggregate Functions
a=np.array([3.3,4.5,1.2,5.7,0.3])
print('Sum:',a.sum())  #np.sum(a)
print('Min:',a.min())
print('Max:',a.max())
print('Mean:',a.mean())
print('Std:',a.std())
```

```
Sum: 15.0
Min: 0.3
Max: 5.7
Mean: 3.0
Std: 2.0079840636817816
```

In [67]:

```
#indexing
arr1=np.arange(10,20)
print('arr1:',arr1)
print('arr1[4]:',arr1[4])
print('arr1[-1]:',arr1[-1])
print('arr1[-3]:',arr1[-3])
print('arr1[[1,3,5]]:',arr1[[1,3,5]])
```

```
print('arr2:',arr2)
print('arr2[1,2]:',arr2[1,2])
```

```
arr1: [10 11 12 13 14 15 16 17 18 19]
arr1[4]: 14
arr1[-1]: 19
arr1[-3]: 17
arr1[[1,3,5]]: [11 13 15]
arr2: [[10 11 12]
 [13 14 15]
 [16 17 18]]
arr2[1,2]: 15
```

In [74]:

```
#Slicing
arr1=np.arange(10,19)
print('arr1:',arr1)
print('arr1[1:5]:',arr1[1:5])
print('arr1[3:]',arr1[3:])
print('arr1[:3]',arr1[:3])
print('arr1[1:6:2]:',arr1[1:6:2])
print('arr1[::2]:',arr1[::2])
print('arr1[::-1]:',arr1[::-1])
```

```
arr1: [10 11 12 13 14 15 16 17 18]
arr1[1:5]: [11 12 13 14]
arr1[3:] [13 14 15 16 17 18]
arr1[:3] [10 11 12]
arr1[1:6:2]: [11 13 15]
arr1[::2]: [10 12 14 16 18]
arr1[::-1]: [18 17 16 15 14 13 12 11 10]
```

In [87]:

```
arr1=np.arange(10,19).reshape(3,3)
print('arr1:',arr1)
print('arr1[0,:]:',arr1[0,:])
print('arr1[:,0]:',arr1[:,0])
print('arr1[0:1,0]:',arr1[0:1,0])
print('arr1[0:2,0]:',arr1[0:2,0])
print('arr1[1,0:1]:',arr1[1,0:1])
print('arr1[2,0:2]:',arr1[2,0:2])
print('arr1[0:2,0:2]:',arr1[0:2,0:2])
print('arr1[0:2,0:1]:',arr1[0:2,0:1])
print('arr1[[1,2],0:2]:',arr1[[1,2],0:2])
print('arr1[0:2,[1,2]]:',arr1[0:2,[1,2]])
```

```
arr1: [[10 11 12]
 [13 14 15]
 [16 17 18]]
arr1[0,:]: [10 11 12]
arr1[:,0]: [10 13 16]
arr1[0:1,0]: [10]
arr1[0:2,0]: [10 13]
arr1[1,0:1]: [13]
arr1[2,0:2]: [16 17]
arr1[0:2,0:2]: [[10 11]
 [13 14]]
arr1[0:2,0:1]: [[10]
 [13]]
arr1[[1,2],0:2]: [[13 14]
 [16 17]]
arr1[0:2,[1,2]]: [[11 12]
 [14 15]]
```

In [89]:
```
A=np.arange(1,13).reshape(3,4)
print('A:',A)
B=A[:2,1:3]
print('B:',B)
```

```
A: [[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
B: [[2 3]
 [6 7]]
```

In [4]:
```
#Iterating
A=np.array([1,2,3])
for i in A:
    print(i)
```

```
1
2
3
```

In [10]:
```
B=np.arange(9).reshape(3,3)
print('dimension wise.....')
for row in B:
    print(row)
print('element wise....')
for row in B.flat:
    print(row)
```

```
dimension wise.....
[0 1 2]
[3 4 5]
[6 7 8]
element wise....
0
1
2
3
4
5
6
7
8
```

In [17]:
```
print('Calculating mean column wise with out loops')
C=np.apply_along_axis(np.mean, axis=0, arr=B)
print(C)
print('Calculating mean row wise with out loops')
D=np.apply_along_axis(np.mean, axis=1, arr=B)
print(D)
print('Calculating using userdefined function as row wise with out loops')
def func(x):
    return  x/2
E=np.apply_along_axis(func, axis=1, arr=B)
print(E)
```

```
Calculating mean column wise with out loops
[3. 4. 5.]
Calculating mean row wise with out loops
[1. 4. 7.]
Calculating using userdefined function as row wise with out loops
[[0.  0.5 1. ]
```

```
 [1.5 2.  2.5]
 [3.  3.5 4. ]]
```

In [18]:

```
#Conditions and Boolean Arrays
A=np.random.random((4,4))
print(A)
```

```
[[0.33689498 0.34679002 0.95346854 0.59504664]
 [0.64676871 0.69980148 0.60078716 0.42258213]
 [0.88424987 0.87863602 0.04897219 0.3844116 ]
 [0.14712574 0.1573832  0.47599646 0.78034266]]
```

In [19]:

```
A<0.5
```

Out[19]:

```
array([[ True,  True, False, False],
       [False, False, False,  True],
       [False, False,  True,  True],
       [ True,  True,  True, False]])
```

In [21]:

```
print(A[A<0.5])
```

```
[0.33689498 0.34679002 0.42258213 0.04897219 0.3844116  0.14712574
 0.1573832  0.47599646]
```

In [22]:

```
#Shape Manipulation
A=np.random.random(12)
print(A)
```

```
[0.58858325 0.90342922 0.63430374 0.36035659 0.94974123 0.89264921
 0.45228281 0.0065825  0.79497221 0.20206592 0.07753275 0.38876482]
```

In [24]:

```
B=A.reshape(3,4)
print(B)
```

```
[[0.58858325 0.90342922 0.63430374 0.36035659]
 [0.94974123 0.89264921 0.45228281 0.0065825 ]
 [0.79497221 0.20206592 0.07753275 0.38876482]]
```

In [28]:

```
A=np.random.random(12)
print(A)  #one dimenstional
print('rehaping with out reshape()')
A.shape=(3,4)
print(A) #Two dimenstional
```

```
[0.47880727 0.4316619  0.60610411 0.45576162 0.24660146 0.14038998
 0.17779918 0.44203801 0.05851803 0.37803854 0.53552628 0.93900051]
rehaping with out reshape()
[[0.47880727 0.4316619  0.60610411 0.45576162]
 [0.24660146 0.14038998 0.17779918 0.44203801]
 [0.05851803 0.37803854 0.53552628 0.93900051]]
```

In [30]:

```
print('Conerting from two dimenstional to one dimenstional')
B=A.ravel()     #or A.shape=(12)
print(B)
```

```
Conerting from two dimenstional to one dimenstional
[0.47880727 0.4316619  0.60610411 0.45576162 0.24660146 0.14038998
 0.17779918 0.44203801 0.05851803 0.37803854 0.53552628 0.93900051]
```

In [43]:
```
A=np.random.random(12)
print(A)  #one dimenstional
print('rehaping with out reshape()')
A.shape=(3,4)
print(A) #Two dimenstional
print('Transpose of A is :')
B=A.transpose()    #or B=A.T
print(B)
```

```
[0.5106013  0.80169685 0.61101172 0.16518259 0.54730156 0.31882147
 0.12172248 0.7776016  0.00681168 0.42560675 0.63091191 0.13026897]
rehaping with out reshape()
[[0.5106013  0.80169685  0.61101172  0.16518259]
 [0.54730156 0.31882147  0.12172248  0.7776016 ]
 [0.00681168 0.42560675  0.63091191  0.13026897]]
Transpose of A is :
[[0.5106013  0.54730156  0.00681168]
 [0.80169685 0.31882147  0.42560675]
 [0.61101172 0.12172248  0.63091191]
 [0.16518259 0.7776016   0.13026897]]
```

In [49]:
```
#Array Manipulation
#Joining Arrays
A=np.ones((3,3))
B=np.zeros((3,3))
print('A:',A)
print('B:',B)
print('Horizontal stacking')
C=np.hstack((A,B))
print(C)
print('Vertical stacking')
D=np.vstack((A,B))
print(D)
print('Column stacking')
a=np.array([1,2,3])
b=np.array([4,5,6])
c=np.array([7,8,9])
col_arr=np.column_stack((a,b,c))
print(col_arr)
print('Row stacking')
row_arr=np.row_stack((a,b,c))
print(row_arr)
```

```
A: [[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
B: [[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
Horizontal stacking
[[1. 1. 1. 0. 0. 0.]
 [1. 1. 1. 0. 0. 0.]
 [1. 1. 1. 0. 0. 0.]]
Vertical stacking
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
```

In [54]:

```python
print('A:',A)
print('Horizontal Splitting.......')
[B,C]=np.hsplit(A,2)
print('B:',B)
print('C:',C)
print('Vertical Splitting.......')
[B,C]=np.vsplit(A,2)
print('B:',B)
print('C:',C)
print('Non Symmetrical Splitting(Column wise).......')
[A1,A2,A3] = np.split(A,[1,3],axis=1)
print('A1:',A1)
print('A1:',A2)
print('A1:',A3)
print('Non Symmetrical Splitting(Row wise).......')
[A1,A2,A3] = np.split(A,[1,3],axis=0)
print('A1:',A1)
print('A1:',A2)
print('A1:',A3)
```

```
A: [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]
Horizontal Splitting....
B: [[ 0  1]
 [ 4  5]
 [ 8  9]
 [12 13]]
C: [[ 2  3]
 [ 6  7]
 [10 11]
 [14 15]]
Vertical Splitting....
B: [[0 1 2 3]
 [4 5 6 7]]
C: [[ 8  9 10 11]
 [12 13 14 15]]
Non Symmetrical Splitting(Column wise)....
A1: [[ 0]
 [ 4]
 [ 8]
 [12]]
A1: [[ 1  2]
 [ 5  6]
 [ 9 10]
 [13 14]]
A1: [[ 3]
 [ 7]
 [11]
 [15]]
Non Symmetrical Splitting(Row wise)....
A1: [[0 1 2 3]]
A1: [[ 4  5  6  7]
 [ 8  9 10 11]]
A1: [[12 13 14 15]]
```

In [57]:

```python
#General Concepts
#copy or views of Objects
A=np.array([1,2,3])
print('A:',A)
B=A
```

```
A[0]=6
print('A:',A)
print('B:',B)
print('Using Copy()')
A=np.array([1,2,3])
print('A:',A)
B=A.copy()
print('B:',B)
A[0]=6
print('A:',A)
print('B:',B)
```

```
A: [1 2 3]
B: [1 2 3]
A: [6 2 3]
B: [6 2 3]
Using Copy()
A: [1 2 3]
B: [1 2 3]
A: [6 2 3]
B: [1 2 3]
```

In [65]:

```
#Vectorization
A=np.arange(4).reshape(2,2)
B=np.arange(4).reshape(2,2)
print('A:',A)
print('B:',B)
print('Multiplication of A and B is\n',A*B)

#Broadcasting
A = np.arange(16).reshape(4, 4)
b = np.arange(4)
print('A:',A)
print('b:',b)
print('Summation of A and  b is:\n',A+b)

m = np.arange(6).reshape(3, 1, 2)
n = np.arange(6).reshape(3, 2, 1)
print('m:',m)
print('n:',n)
print('Summation of m and  n is:\n',m+n)
```

```
A: [[0 1]
 [2 3]]
B: [[0 1]
 [2 3]]
Multiplication of A and B is
 [[0 1]
 [4 9]]
A: [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]
b: [0 1 2 3]
Summation of A and  b is:
 [[ 0  2  4  6]
 [ 4  6  8 10]
 [ 8 10 12 14]
 [12 14 16 18]]
m: [[[0 1]]

 [[2 3]]
```

```
 [[4 5]]]
n: [[[0]
  [1]]

 [[2]
  [3]]

 [[4]
  [5]]]
Summation of m and  n is:
 [[[ 0  1]
  [ 1  2]]

 [[ 4  5]
  [ 5  6]]

 [[ 8  9]
  [ 9 10]]]
```

In [70]:

```
#Structured Arrays
structured = np.array([(1, 'First', 0.5, 1+2j),(2, 'Second', 1.3,2-2j), (3, 'Third',
print('Structured Arrays is:')
structured
```

```
Structured Arrays is:
```

Out[70]:

```
array([(1, b'First', 0.5, 1.+2.j), (2, b'Second', 1.3, 2.-2.j),
       (3, b'Third', 0.8, 1.+3.j)],
      dtype=[('f0', '<i2'), ('f1', 'S6'), ('f2', '<f4'), ('f3', '<c8')])
```

In [71]:

```
print('Record1 is:')
print(structured[1])
```

```
Record1 is:
(2, b'Second', 1.3, 2.-2.j)
```

In [72]:

```
print('Column Record1 is:')
print(structured['f1'])
```

```
Column Record1 is:
[b'First' b'Second' b'Third']
```

In [73]:

```
structured  =  np.array([(1,'First',0.5,1+2j),(2,'Second',1.3,2-2j),(3,'Third',0.8,1+3
                        dtype=[('id','i2'),('position','a6'),('value','f4'),('complex'
print('Structured Arrays with specific columns is:')
structured
```

```
Structured Arrays with specific columns is:
```

Out[73]:

```
array([(1, b'First', 0.5, 1.+2.j), (2, b'Second', 1.3, 2.-2.j),
       (3, b'Third', 0.8, 1.+3.j)],
      dtype=[('id', '<i2'), ('position', 'S6'), ('value', '<f4'), ('complex', '<c
8')])
```

In [74]:

```
print('Record1 is:')
print(structured[1])
```

```
Record1 is:
(2, b'Second', 1.3, 2.-2.j)
```

In [75]:

```
print('Column Record1 is:')
```

```
Column Record1 is:
[b'First' b'Second' b'Third']
```

In [76]:

```
structured.dtype.names = ('id','order','value','complex')
print('Structured Arrays after column change is:')
structured
```

```
Structured Arrays after column change is:
```

Out[76]:

```
array([(1, b'First', 0.5, 1.+2.j), (2, b'Second', 1.3, 2.-2.j),
       (3, b'Third', 0.8, 1.+3.j)],
      dtype=[('id', '<i2'), ('order', 'S6'), ('value', '<f4'), ('complex', '<c8')])
```

In [77]:

```
print('Column Record1 is:')
print(structured['order'])
```

```
Column Record1 is:
[b'First' b'Second' b'Third']
```

In [82]:

```
#Reading and Writing Array Data on Files
#Saving Data in Binary Files
data=np.arange(12).reshape(4,3)
print('Data is:\n',data)
print('Saving data to saved_data.npy file......... ')
np.save('saved_data',data)
mydata=np.load('saved_data.npy')
print('Loaded data is:\n',mydata)
```

```
Data is:
 [[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
Saving data to saved_data.npy file.....
Loaded data is:
 [[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
```

In [93]:

```
#Reading Files with Tabular Data
#create sample.csv file which contains as follows
#id,name,email,phone
#3001,ABC,abc@gmail,com,9999999999
#3002,XYZ,xyz@gmail,com,8888888888
#3003,BBC,bbc@gmail,com,7777777777
#1241,PNR,pnr2830@gmail,com,9966052830
data = np.genfromtxt('sample.csv', delimiter=',', names=True)
print('Data form Sample.csv file is:\n',data)
data = np.genfromtxt('sample.csv', delimiter=',', names=True,dtype=('i2,S6,S20,i8'))
print('Data form Sample.csv file is:\n',data)
```

```
Data form Sample.csv file is:
 [(3001., nan, nan, 1.00000000e+10) (3002., nan, nan, 8.88888889e+09)
 (3003., nan, nan, 7.77777778e+09) (1241., nan, nan, 9.96605283e+09)]
Data form Sample.csv file is:
 [(3001, b'ABC', b'abc@gmail.com', 9999999999)
 (3002, b'XYZ', b'xyz@gmail.com', 8888888888)
 (3003, b'BBC', b'bbc@gmail.com', 7777777777)
 (1241, b'PNR', b'pnr2830@gmail.com', 9966052830)]
```

```
print('Employee email ids.......')
print(data['email'])
```

```
Employee email ids....
[b'abc@gmail.com'  b'xyz@gmail.com'  b'bbc@gmail.com'  b'pnr2830@gmail.com']
```

In [108…

```
#Extra Concepts
#where()
A=np.array([1,2,3,4])
B=np.where(A==3)
print('Index of 3 is:')
print(B)
C=np.where(A%2==0)
print('Indexes of Even Numbers.......')
print(C)

#searchsorted()
A=np.array([1,7,8,9])
B=np.searchsorted(A,7)
print('Index of 7 is:')
print(B)

B=np.searchsorted(A,7,side='right')
print('Index of 7 is:')
print(B)

#sort()
A=np.array([1,3,2,5,4,6])
print('Sorted elements are:',np.sort(A))

B=np.array([False,True,True,False,False])
print('Sorted elements are:',np.sort(B))

C=np.array(['banana','cherry','apple'])
print('Sorted elements are:',np.sort(C))

#filtering arrays
A=np.array([41,42,45,46,43,40])
filter_arr=[False,True,True,False,False,True]
B=A[filter_arr]
print('Filtering Array is:\n',B)
```

```
Index of 3 is:
(array([2], dtype=int64),)
Indexes of Even Numbers....
(array([1, 3], dtype=int64),)
Index of 7 is:
1
Index of 7 is:
2
Sorted elements are: [1 2 3 4 5 6]
Sorted elements are: [False False False  True  True]
Sorted elements are: ['apple' 'banana' 'cherry']
Filtering Array is:
 [42 45 40]
```

In [109…

```
#create a filter array which contains greater than 42 in the above array
A=np.array([41,42,45,46,43,40])
filter_arr=[]
for ele in A:
    if ele >42:
```

```
        filter_arr.append(False)
print('Filter is:\n',filter_arr)
new_arr=A[filter_arr]
print('Filter array is:\n',new_arr)
```

```
Filter is:
 [False, False, True, True, True, False]
Filter array is:
 [45 46 43]
```

In [110…

```
#create a filter array which contains only even numbers in the above array
A=np.array([41,42,45,46,43,40])
filter_arr=[]
for ele in A:
    if ele%2==0:
        filter_arr.append(True)
    else:
        filter_arr.append(False)
print('Filter is:\n',filter_arr)
new_arr=A[filter_arr]
print('Filter array is:\n',new_arr)
```

```
Filter is:
 [False, True, False, True, False, True]
Filter array is:
 [42 46 40]
```

In [111…

```
#create a filter array which contains greater than 42 in the above array
A=np.array([41,42,45,46,43,40])
filter_arr=A>42
print('Filter is:\n',filter_arr)
new_arr=A[filter_arr]
print('Filter array is:\n',new_arr)
```

```
Filter is:
 [False False  True  True  True False]
Filter array is:
 [45 46 43]
```

In [112…

```
#create a filter array which contains only even numbers in the above array
A=np.array([41,42,45,46,43,40])
filter_arr=A%2==0
print('Filter is:\n',filter_arr)
new_arr=A[filter_arr]
print('Filter array is:\n',new_arr)
```

```
Filter is:
 [False  True False  True False  True]
Filter array is:
 [42 46 40]
```

In [ ]:

Pandas Programs Practice
(Reference-3)

In [3]:

```
import pandas as pd
s=pd.Series([1,2,3])
print(s)
```

```
0    1
1    2
2    3
dtype: int64
```

In [4]:

```
import pandas as pd
s=pd.Series([1,2,3],index=['a','b','c'])
print(s)
```

```
a    1
b    2
c    3
dtype: int64
```

In [9]:

```
import pandas as pd
s=pd.Series([1,2,3],index=['a','b','c'])
print(s.values)
print(s.index)
print(s['c'],s[1],s[2])
```

```
[1 2 3]
Index(['a', 'b', 'c'], dtype='object')
3 2 3
```

In [10]:

```
import pandas as pd
s=pd.Series([15,-2,3,1])
print(s[0:2])
```

```
0    15
1    -2
dtype: int64
```

In [12]:

```
import pandas as pd
s=pd.Series([15,-2,3,1],index=['x','y','z','w'])
print(s[0:2])
print(s[['y','w']])
```

```
x    15
y    -2
dtype: int64
y   -2
w    1
dtype: int64
```

In [13]:

```
import pandas as pd
s=pd.Series([15,-2,3,1],index=['x','y','z','w'])
s[1]=0
print(s)
```

```
x    15
y     0
z     3
w     1
dtype: int64
```

In [15]:

```
import pandas as pd
s=pd.Series([15,-2,3,1],index=['x','y','z','w'])
ser=s
print(ser)
```

```
x    15
y    -2
z     3
w     1
dtype: int64
```

In [16]:

```
import pandas as pd
s=pd.Series([15,-2,3,1],index=['x','y','z','w'])
ser=pd.Series(s)
print(ser)
```

```
x    15
y    -2
z     3
w     1
dtype: int64
```

In [18]:

```
import numpy as np
arr=np.array([1,2,3,4])
s1=pd.Series(arr)
print(s1)
arr[2]=-5
print(arr)
print(s1)
```

```
0    1
1    2
2    3
3    4
dtype: int32
[ 1  2 -5  4]
0    1
1    2
2   -5
3    4
dtype: int32
```

In [19]:

```
import pandas as pd
ser=pd.Series([5,-2,3,4])
print(ser>2)
print(ser[ser>2])
```

```
0     True
1    False
2     True
3     True
dtype: bool
0    5
2    3
3    4
dtype: int64
```

In [20]:

```
ser=pd.Series([10,11,5,8,3],index=['a','b','c','d','e'])
print(ser>6)
print(ser[ser>6])
```

```
a     True
b     True
c    False
d     True
e    False
dtype: bool
a    10
b    11
d     8
dtype: int64
```

In [21]:

```
ser=pd.Series([10,11,5,8,3],index=['a','b','c','d','e'])
print(ser+4)
```

```
a    14
b    15
c     9
d    12
e     7
dtype: int64
```

In [25]:

```
ser=pd.Series([10,11,-5,8,3],index=['a','b','c','d','e'])
print(np.log(ser))
```

```
a     2.302585
b     2.397895
c          NaN
d     2.079442
e     1.098612
dtype: float64

C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\series.py:853: Runt
imeWarning: invalid value encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

In [23]:

```
ser=pd.Series([10,11,5,8,3],index=['a','b','c','d','e'])
print(ser/2)
```

```
a     5.0
b     5.5
c     2.5
d     4.0
e     1.5
dtype: float64
```

In [24]:

```
ser=pd.Series([1,0,2,1,2,3],index=['white','white','red','green','yellow','blue'])
print(ser)
```

```
white     1
white     0
red       2
green     1
yellow    2
blue      3
dtype: int64
```

In [4]:

```
import pandas as pd
ser=pd.Series([1,0,2,1,2,3],index=['white','white','red','green','yellow','blue'])
print(ser)
print(ser.unique())
print(ser.value_counts())
print(ser.isin([1,3]))
```

```
white     1
white     0
red       2
green     1
yellow    2
blue      3
dtype: int64
[1 0 2 3]
2    2
1    2
3    1
0    1
dtype: int64
white      True
white     False
red       False
green      True
yellow    False
blue       True
dtype: bool
```

In [5]:

```
#not a number method
import numpy as np
import pandas as pd
ser=pd.Series([5,-2,np.nan,4,0])
print(ser)
```

```
0    5.0
1   -2.0
2    NaN
3    4.0
4    0.0
dtype: float64
```

In [8]:

```
ser=pd.Series([8,-3,np.nan,4,9])
print(ser)
print(ser.isnull())
print(ser.notnull())
#filtering the values
print(ser[ser.notnull()])
```

```
0    8.0
1   -3.0
2    NaN
3    4.0
4    9.0
dtype: float64
0    False
1    False
2     True
3    False
4    False
dtype: bool
0     True
1     True
2    False
3     True
4     True
dtype: bool
0    8.0
1   -3.0
3    4.0
4    9.0
dtype: float64
```

In [9]:

```
#series as dictionary
mydict={'red':2000,'yellow':500,'blue':300}
ser1=pd.Series(mydict)
print(ser1)
```

```
red       2000
yellow     500
blue       300
dtype: int64
```

In [14]:

```
mydict={'red':2000,'yellow':500,'blue':300}
colors=['red','yellow','blue','green']
ser1=pd.Series(mydict,index=colors)
print(ser1)
print(ser1.notnull())
print(ser1[ser1.notnull()])
```

```
red       2000.0
yellow     500.0
blue       300.0
green        NaN
dtype: float64
red        True
yellow     True
blue       True
green     False
dtype: bool
red       2000.0
yellow     500.0
blue       300.0
dtype: float64
```

In [20]:

```
#operations between series
mydict={'red':2000,'yellow':500,'blue':300}
mydict1={'red':500,'yellow':700,'blue':500,'green':400}
ser=pd.Series(mydict)
ser1=pd.Series(mydict1)
print(ser)
print(ser1)
print("ser1+ser",ser1+ser,sep='\n')
```

```
red       2000
yellow     500
blue       300
dtype: int64
red       500
yellow    700
blue      500
green     400
dtype: int64
ser1+ser
blue       800.0
green        NaN
red       2500.0
yellow    1200.0
dtype: float64
```

In [17]:

```
import numpy as np
frame1=pd.DataFrame([[6,3,2,1],[2,1,4,6],[1,2,0,3],[4,0,1,2]],
                    index=['red','blue','green','yellow'],
                    columns=['pen','ball','paper','pencil'])
print(frame1)
#sorting
print(frame1.sort_values(by='ball'))
```

```
        pen  ball  paper  pencil
red       6     3      2       1
blue      2     1      4       6
green     1     2      0       3
yellow    4     0      1       2
        pen  ball  paper  pencil
yellow    4     0      1       2
blue      2     1      4       6
green     1     2      0       3
red       6     3      2       1
```

In [28]:

```
#data frame
import pandas as pd
mydict={'color':['red','yellow','green','blue','orange'],
       'object':['ball','pen','book','scale','mug'],
       'price':[1.5,2.4,1.3,3.6,4.8]}
frame=pd.DataFrame(mydict)
print(frame)
print('changing index values')
frame=pd.DataFrame(mydict,index=['one','two','three','four','five'])
print(frame)
print('specifying the columns')
frame=pd.DataFrame(mydict,index=['one','two','three','four','five'],
                  columns=['object','price'])
print(frame)
```

```
    color object  price
0     red   ball    1.5
1  yellow    pen    2.4
2   green   book    1.3
3    blue  scale    3.6
4  orange    mug    4.8
changing index values
        color object  price
one       red   ball    1.5
two    yellow    pen    2.4
three   green   book    1.3
four     blue  scale    3.6
five   orange    mug    4.8
specifying the columns
      object  price
one     ball    1.5
two      pen    2.4
three   book    1.3
four   scale    3.6
five     mug    4.8
```

In [46]:

```
mydict={'color':['red','yellow','green','blue','orange'],
        'object':['ball','pen','book','scale','mug'],
        'price':[1.5,2.4,1.3,3.6,4.8]}
frame=pd.DataFrame(mydict,index=['one','two','three','four','five'])
print(frame)
print()
print(frame['color'],end="\n")
print(frame.price)
print(frame.columns,frame.index,sep="\n")
print(frame.values)
```

```
        color object  price
one       red   ball    1.5
two    yellow    pen    2.4
three   green   book    1.3
four     blue  scale    3.6
five   orange    mug    4.8

one         red
two      yellow
three     green
four       blue
five     orange
Name: color, dtype: object
one      1.5
two      2.4
three    1.3
four     3.6
five     4.8
Name: price, dtype: float64
Index(['color', 'object', 'price'], dtype='object')
Index(['one', 'two', 'three', 'four', 'five'], dtype='object')
[['red' 'ball' 1.5]
 ['yellow' 'pen' 2.4]
 ['green' 'book' 1.3]
 ['blue' 'scale' 3.6]
 ['orange' 'mug' 4.8]]
```

In [41]:

```
frame1=pd.DataFrame([[4,'fox'],[2,'kangaroo'],[4,'deer'],[8,'spider'],[np.nan,'snake'
]],
                   columns=['no_oflegs','animal'],
                   index=[0,1,2,3,4])
print(frame1)
```

```
   no_oflegs    animal
0        4.0       fox
1        2.0  kangaroo
2        4.0      deer
3        8.0    spider
4        NaN     snake
```

In [47]:

```
frame3=pd.DataFrame(np.arange(16).reshape(4,4),
                    index=['red','yellow','green','blue'],
                    columns=['ball','pen','book','mug'])
print(frame3)
```

```
        ball  pen  book  mug
red        0    1     2    3
yellow     4    5     6    7
green      8    9    10   11
blue      12   13    14   15
```

In [50]:

```
data={'colors':['red','yelow','green','blue','orange'],
      'object':['ball','pen','book','scale','mug'],
      'price':[1.5,2.4,1.3,3.6,4.8]}
frame=pd.DataFrame(data)
print(frame)
print(frame.loc[2])
print(frame.loc[[1,3]])
```

```
   colors object  price
0     red   ball    1.5
1   yelow    pen    2.4
2   green   book    1.3
3    blue  scale    3.6
4  orange    mug    4.8
colors     green
object      book
price        1.3
Name: 2, dtype: object
  colors object  price
1  yelow    pen    2.4
3   blue  scale    3.6
```

In [54]:

```
#slicing frame object
data={'colors':['red','yelow','green','blue','orange'],
     'object':['ball','pen','book','scale','mug'],
     'price':[1.5,2.4,1.3,3.6,4.8]}
frame=pd.DataFrame(data)
print(frame)
print(frame[0:1])
print(frame[1:3])
print(frame['colors'][3])
print(frame['object'][2])
```

```
   colors object  price
0     red   ball    1.5
1   yelow    pen    2.4
2   green   book    1.3
3    blue  scale    3.6
4  orange    mug    4.8
  colors object  price
0    red   ball    1.5
  colors object  price
1  yelow    pen    2.4
2  green   book    1.3
blue
book
```

In [63]:

```
#assigning values to the frame
data={'colors':['red','yelow','green','blue','orange'],
     'object':['ball','pen','book','scale','mug'],
     'price':[1.5,2.4,1.3,3.6,4.8]}
frame=pd.DataFrame(data)
print(frame)
print()
frame.index.names=['idx']
frame.columns.names=['item']
print(frame)
#adding new column to the existing data frames
frame['count']=10
print(frame)
frame['count']=[10,12,15,16,18]
print(frame)
ser=pd.Series(np.arange(10,16))
frame['newser']=ser
print(frame)
```

```
   colors object  price
0     red   ball    1.5
1   yelow    pen    2.4
2   green   book    1.3
3    blue  scale    3.6
4  orange    mug    4.8

item   colors object  price
idx
0         red   ball    1.5
1       yelow    pen    2.4
2       green   book    1.3
3        blue  scale    3.6
4      orange    mug    4.8
item   colors object  price  count
idx
0         red   ball    1.5     10
1       yelow    pen    2.4     10
2       green   book    1.3     10
3        blue  scale    3.6     10
4      orange    mug    4.8     10
item   colors object  price  count
idx
0         red   ball    1.5     10
1       yelow    pen    2.4     12
2       green   book    1.3     15
3        blue  scale    3.6     16
4      orange    mug    4.8     18
item   colors object  price  count  newser
idx
0         red   ball    1.5     10      10
1       yelow    pen    2.4     12      11
2       green   book    1.3     15      12
3        blue  scale    3.6     16      13
4      orange    mug    4.8     18      14
```

In [66]:

```
data={'colors':['red','yelow','green','blue','orange'],
      'object':['ball','pen','book','scale','mug'],
      'price':[1.5,2.4,1.3,3.6,4.8]}
frame=pd.DataFrame(data)
print(frame)
frame['count']=[10,12,15,16,18]
print(frame)
del frame['count']
print(frame)
```

```
   colors object  price
0     red   ball    1.5
1   yelow    pen    2.4
2   green   book    1.3
3    blue  scale    3.6
4  orange    mug    4.8
   colors object  price  count
0     red   ball    1.5     10
1   yelow    pen    2.4     12
2   green   book    1.3     15
3    blue  scale    3.6     16
4  orange    mug    4.8     18
   colors object  price
0     red   ball    1.5
1   yelow    pen    2.4
2   green   book    1.3
3    blue  scale    3.6
4  orange    mug    4.8
```

In [72]:

```
data={'colors':['red','yelow','green','blue','orange'],
      'object':['ball','pen','book','scale','mug'],
      'price':[1.5,2.4,1.3,3.6,4.8]}
frame=pd.DataFrame(data)
print(frame)
print(frame.isin([1.5,'pen']))
print(frame[frame.isin([1.5,'pen'])])
```

```
   colors object  price
0     red   ball    1.5
1   yelow    pen    2.4
2   green   book    1.3
3    blue  scale    3.6
4  orange    mug    4.8
   colors  object  price
0   False   False   True
1   False    True  False
2   False   False  False
3   False   False  False
4   False   False False
  colors object  price
0    NaN    NaN    1.5
1    NaN    pen    NaN
2    NaN    NaN    NaN
3    NaN    NaN    NaN
4    NaN    NaN    NaN
```

In [4]:

```
#filtering values
import pandas as pd
import numpy as np
data={'colors':['red','yelow','green','blue','orange'],
     'object':['ball','pen','book','scale','mug'],
     'price':[1.5,2.4,1.3,3.6,4.8]}
frame=pd.DataFrame(data)
print(frame)
#print(frame[frame > 2.5])
```

```
   colors object  price
0     red   ball    1.5
1   yelow    pen    2.4
2   green   book    1.3
3    blue  scale    3.6
4  orange    mug    4.8
```

In [11]:

```
#nested dictionary
nestdict={'red':{2011:12,2013:15},
         'blue':{2011:14,2012:13,2013:16},
         'green':{2011:10,2012:17,2013:18}}
frame1=pd.DataFrame(nestdict)
print(frame1)
#transposition of frame
print(frame1.T)
print(frame1.index.is_unique)
```

```
       red  blue  green
2011  12.0    14     10
2013  15.0    16     18
2012   NaN    13     17
       2011  2013  2012
red    12.0  15.0   NaN
blue   14.0  16.0  13.0
green  10.0  18.0  17.0
True
```

In [6]:

```
ser1=pd.Series([5,0,3,8,4],index=['red','blue','yellow','white','green'])
print(ser1)
print(ser1.idxmin())
print(ser1.idxmax())
```

```
red       5
blue      0
yellow    3
white     8
green     4
dtype: int64
blue
white
```

In [10]:

```
ser2=pd.Series(np.arange(6),index=['white','white','blue','green','green','yellow'])
print(ser2)
print(ser2['white'])
print(ser2.index.is_unique)
```

```
white     0
white     1
blue      2
green     3
green     4
yellow    5
dtype: int32
white    0
white    1
dtype: int32
False
```

In [12]:

```
#other functionality on indexing
#reindexing
ser3=pd.Series([2,5,7,4],index=['one','two','three','four'])
print(ser3)
ser3.reindex(['three','four','five','one'])
```

```
one      2
two      5
three    7
four     4
dtype: int64
```

Out[12]:

```
three    7.0
four     4.0
five     NaN
one      2.0
dtype: float64
```

In [13]:

```
ser4=pd.Series([1,5,6,3],index=[0,3,5,6])
print(ser4)
print(ser4.reindex(range(6)))
#using ffill or bfill
print(ser4.reindex(range(6),method='ffill'))
print(ser4.reindex(range(6),method='bfill'))
```

```
0    1
3    5
5    6
6    3
dtype: int64
0    1.0
1    NaN
2    NaN
3    5.0
4    NaN
5    6.0
dtype: float64
0    1
1    1
2    1
3    5
4    5
5    6
dtype: int64
0    1
1    5
2    5
3    5
4    6
5    6
dtype: int64
```

In [1]:

```
#on dataframes
import pandas as pd
data={'color':['red','green','blue'],
     'object':['pen','ball','pencil'],
     'price':[20,15,14]}
frame3=pd.DataFrame(data)
print(frame3)
print(frame3.reindex(range(3),columns=['color','count','object','price']))
print(frame3.reindex(range(3),method='ffill',columns=['color','count','object','price'
]))
```

```
   color  object  price
0    red     pen     20
1  green    ball     15
2   blue  pencil     14
   color  count  object  price
0    red    NaN     pen     20
1  green    NaN    ball     15
2   blue    NaN  pencil     14
   color  count  object  price
0    red    red     pen     20
1  green  green    ball     15
2   blue   blue  pencil     14
```

In [3]:

```
#dropping
import pandas as pd
import numpy as np
ser5=pd.Series(np.arange(4),index=['red','green','blue','white'])
print(ser5)
print('dropping index green')
print(ser5.drop('green'))
print(ser5.drop(['red','blue']))
```

```
red      0
green    1
blue     2
white    3
dtype: int32
dropping index green
red      0
blue     2
white    3
dtype: int32
green    1
white    3
dtype: int32
```

In [4]:

```
#dropping in dataframes
frame4=pd.DataFrame(np.arange(16).reshape(4,4),
                    index=['r','b','g','w'],
                    columns=['ball','pen','pencil','book'])
print(frame4)
print('dropping indexes blue , green')
print(frame4.drop(['b','g']))
print('dropping columns')
print(frame4.drop(['pen','pencil'],axis=1))
```

```
   ball  pen  pencil  book
r     0    1       2     3
b     4    5       6     7
g     8    9      10    11
w    12   13      14    15
dropping indexes blue , green
   ball  pen  pencil  book
r     0    1       2     3
w    12   13      14    15
dropping columns
   ball  book
r     0     3
b     4     7
g     8    11
w    12    15
```

In [7]:

```
#alignment
s1=pd.Series([3,2,5,1],index=['white','yellow','green','blue'])
s2=pd.Series([1,4,7,2,1],index=['white','yellow','black','green','brown'])
print('s1 series are:',s1,sep='\n')
print('s2 series are:',s2,sep='\n')
print('s1+s2',s1+s2,sep='\n')
```

```
s1 series are:
white     3
yellow    2
green     5
blue      1
dtype: int64
s2 series are:
white     1
yellow    4
black     7
green     2
brown     1
dtype: int64
s1+s2
black     NaN
blue      NaN
brown     NaN
green     7.0
white     4.0
yellow    6.0
dtype: float64
```

In [2]:

```
import pandas as pd
import numpy as np

frame5=pd.DataFrame(np.arange(16).reshape(4,4),index=['red','blue','yellow','green'],co
lumns=['ball','pen','pencil','book'])
print('frame5:',frame5,sep='\n')
frame6=pd.DataFrame(np.arange(12).reshape(4,3),index=['blue','green','white','yellow'],
columns=['pen','ball','mug'])
print('frame6:',frame6,sep='\n')
print('frame5+frame6:',(frame5+frame6),sep='\n')
```

```
frame5:
        ball  pen  pencil  book
red        0    1       2     3
blue       4    5       6     7
yellow     8    9      10    11
green     12   13      14    15
frame6:
        pen  ball  mug
blue      0     1    2
green     3     4    5
white     6     7    8
yellow    9    10   11
frame5+frame6:
        ball  book  mug   pen  pencil
blue     5.0   NaN  NaN   5.0     NaN
green   16.0   NaN  NaN  16.0     NaN
red      NaN   NaN  NaN   NaN     NaN
white    NaN   NaN  NaN   NaN     NaN
yellow  18.0   NaN  NaN  18.0     NaN
```

In [3]:

```
print(frame5.add(frame6))
```

```
        ball  book  mug   pen  pencil
blue     5.0   NaN  NaN   5.0     NaN
green   16.0   NaN  NaN  16.0     NaN
red      NaN   NaN  NaN   NaN     NaN
white    NaN   NaN  NaN   NaN     NaN
yellow  18.0   NaN  NaN  18.0     NaN
```

In [4]:

```
s5=pd.Series(np.arange(4),index=['ball','pen','pencil','book'])
print('s5:',s5,sep='\n')
print('frame5:',frame5,sep='\n')
print('frame5+s5:',(frame5+s5),sep='\n')
```

```
s5:
ball      0
pen       1
pencil    2
book      3
dtype: int32
frame5:
        ball  pen  pencil  book
red        0    1       2     3
blue       4    5       6     7
yellow     8    9      10    11
green     12   13      14    15
frame5+s5:
        ball  pen  pencil  book
red        0    2       4     6
blue       4    6       8    10
yellow     8   10      12    14
green     12   14      16    18
```

In [8]:

```
def fun(x):
    return x.max()-x.min()
frame=pd.DataFrame(np.arange(16).reshape(4,4))
print(frame)
print(frame.apply(fun,axis=0))#row wise
#print(frame)
```

```
    0   1   2   3
0   0   1   2   3
1   4   5   6   7
2   8   9  10  11
3  12  13  14  15
0    12
1    12
2    12
3    12
dtype: int64
```

In [12]:

```
fun=lambda.x : x.max()-x.min()
print(frame.apply(fun))
```

```
  File "<ipython-input-12-cb0b16fc1950>", line 1
    fun=lambda(x) : x.max()-x.min()
              ^
SyntaxError: invalid syntax
```

In [1]:

```
import pandas as pd
import numpy as np
frame1=pd.DataFrame([[1,3,5,7],[2,4,6,8],[1,5,6,2],[3,6,8,9]],
                    columns=['ball','pen','pencil','book'],
                    index=['red','green','yellow','blue'])
def create(x):
    return pd.Series([x.max(),x.min()],index=['max','min'])
frame1.apply(create)
```

Out[1]:

	ball	pen	pencil	book
max	3	6	8	9
min	1	3	5	2

In [4]:

```
def Findsum(y):
    return pd.Series(y.sum(),index=['sum'])
print(frame1.apply(Findsum))
```

```
     ball  pen  pencil  book
sum     7   18      25    26
```

In [3]:

```
import pandas as pd
import numpy as np
frame1=pd.DataFrame([[1,3,5,7],[2,4,6,8],[1,5,6,2],[3,6,8,9]],
                    columns=['ball','pen','pencil','book'],
                    index=['red','green','yellow','blue'])
print(frame1.sum())
print(frame1.mean())
print(frame1.describe())
```

```
ball       7
pen       18
pencil    25
book      26
dtype: int64
ball      1.75
pen       4.50
pencil    6.25
book      6.50
dtype: float64
           ball       pen    pencil      book
count  4.000000  4.000000  4.000000  4.000000
mean   1.750000  4.500000  6.250000  6.500000
std    0.957427  1.290994  1.258306  3.109126
min    1.000000  3.000000  5.000000  2.000000
25%    1.000000  3.750000  5.750000  5.750000
50%    1.500000  4.500000  6.000000  7.500000
75%    2.250000  5.250000  6.500000  8.250000
max    3.000000  6.000000  8.000000  9.000000
```

In [8]:

```
#sorting and ranking
ser1=pd.Series([5,0,3,8,4],index=['red','green','yellow','blue','white'])
print(ser1)
print(ser1.sort_index(ascending=True))
print(ser1.sort_index(ascending=False))
print(ser1.sort_values(ascending=True))
```

```
red       5
green     0
yellow    3
blue      8
white     4
dtype: int64
blue      8
green     0
red       5
white     4
yellow    3
dtype: int64
yellow    3
white     4
red       5
green     0
blue      8
dtype: int64
green     0
yellow    3
white     4
red       5
blue      8
dtype: int64
```

In [13]:

```
frame1=pd.DataFrame(np.arange(16).reshape(4,4),index=['red','blue','green','yellow'],
                   columns=['pen','ball','paper','pencil'])
print(frame1.sort_index(ascending=True))
print(frame1.sort_index(axis=1))
print(frame1.sort_values(by='ball'))
```

```
        pen  ball  paper  pencil
blue      4     5      6       7
green     8     9     10      11
red       0     1      2       3
yellow   12    13     14      15
        ball  paper  pen  pencil
red        1      2    0       3
blue       5      6    4       7
green      9     10    8      11
yellow    13     14   12      15
        pen  ball  paper  pencil
red       0     1      2       3
blue      4     5      6       7
green     8     9     10      11
yellow   12    13     14      15
```

In [15]:

```
#ranking
ser1=pd.Series([5,0,3,8,4],index=['red','green','yellow','blue','white'])
print(ser1)
print(ser1.rank())
print(ser1.rank(ascending=False))
```

```
red       5
green     0
yellow    3
blue      8
white     4
dtype: int64
red       4.0
green     1.0
yellow    2.0
blue      5.0
white     3.0
dtype: float64
red       2.0
green     5.0
yellow    4.0
blue      1.0
white     3.0
dtype: float64
```

In [25]:

```
frame=pd.DataFrame([[4,'fox'],[2,'kangaroo'],[4,'deer'],[8,'spider'],[np.nan,'snake']],
                  columns=['number_legs','animal'],index=[0,1,2,3,4])
print(frame)
frame['default_rank']=frame['number_legs'].rank()
print(frame)
```

```
   number_legs    animal
0          4.0       fox
1          2.0  kangaroo
2          4.0      deer
3          8.0    spider
4          NaN     snake
   number_legs    animal  default_rank
0          4.0       fox           2.5
1          2.0  kangaroo           1.0
2          4.0      deer           2.5
3          8.0    spider           4.0
4          NaN     snake           NaN
```

In [8]:

```
#covariance and correlation
import pandas as pd
import numpy as np
ser1=pd.Series([5,2,3],index=['red','green','blue'])
print(ser1)
print(ser1.var())
frame1=pd.DataFrame([[10,15,7,2,16],[13,0,7,4,1]],
                    index=['commercial watched','product purchase'])
print(frame1)
print(frame1.var())
print('covariance:',frame1.cov(),sep='\n')
print('correlation:',frame1.corr(),sep='\n')
```

```
red      5
green    2
blue     3
dtype: int64
2.3333333333333335
                     0   1  2  3   4
commercial watched  10  15  7  2  16
product purchase    13   0  7  4   1
0      4.5
1    112.5
2      0.0
3      2.0
4    112.5
dtype: float64
covariance:
      0      1    2     3      4
0   4.5  -22.5  0.0   3.0  -22.5
1 -22.5  112.5  0.0 -15.0  112.5
2   0.0    0.0  0.0   0.0    0.0
3   3.0  -15.0  0.0   2.0  -15.0
4 -22.5  112.5  0.0 -15.0  112.5
correlation:
     0    1   2    3    4
0  1.0 -1.0 NaN  1.0 -1.0
1 -1.0  1.0 NaN -1.0  1.0
2  NaN  NaN NaN  NaN  NaN
3  1.0 -1.0 NaN  1.0 -1.0
4 -1.0  1.0 NaN -1.0  1.0
```

In [12]:

```
import pandas as pd
import numpy as np
ser1=pd.Series([6,7,np.nan,4],index=['red','green','blue','yellow'])
print('series 1:',ser1,sep='\n')
ser1['green']=None
print(ser1)
print(ser1.dropna())
#print(ser1)
```

```
series 1:
red       6.0
green     7.0
blue      NaN
yellow    4.0
dtype: float64
red       6.0
green     NaN
blue      NaN
yellow    4.0
dtype: float64
red       6.0
yellow    4.0
dtype: float64
```

In [17]:

```
# using how option
frame1=pd.DataFrame([[6,np.nan,4.0],[np.nan,np.nan,np.nan],[2.2,np.nan,5]],index=['red'
,'green','blue'],
                   columns=['ball','pen','pencil'])
print('frame1:',frame1,sep='\n')
#print(frame1.dropna(how=all))
#filling the nan position values using fillna()
frame1.fillna(0)
```

```
frame1:
       ball  pen  pencil
red     6.0  NaN     4.0
green   NaN  NaN     NaN
blue    2.2  NaN     5.0
```

Out[17]:

	ball	pen	pencil
red	6.0	0.0	4.0
green	0.0	0.0	0.0
blue	2.2	0.0	5.0

In [21]:

```
#hierarchical indexing and levelling
mser=pd.Series(np.random.random(8),
               index=[['w','w','w','b','b','r','r','r'],
                      ['up','down','left','up','down','up','down','left']])
print(mser)
print('indexes:',mser.index,sep='\n')
print(mser['w'])
print(mser['w','up'])
print(mser[:,'up'])
print('converting series into dataframe using unstack() method')
print(mser.unstack())
```

```
w  up       0.694212
   down     0.405391
   left     0.351115
b  up       0.200439
   down     0.154843
r  up       0.728985
   down     0.644616
   left     0.625199
dtype: float64
indexes:
MultiIndex([('w',   'up'),
            ('w', 'down'),
            ('w', 'left'),
            ('b',   'up'),
            ('b', 'down'),
            ('r',   'up'),
            ('r', 'down'),
            ('r', 'left')],
           )
up      0.694212
down    0.405391
left    0.351115
dtype: float64
0.6942119950056037
w    0.694212
b    0.200439
r    0.728985
dtype: float64
converting series into dataframe using unstack() method
       down      left        up
b  0.154843       NaN  0.200439
r  0.644616  0.625199  0.728985
w  0.405391  0.351115  0.694212
```

In [22]:

```
mframe=pd.DataFrame(np.arange(16).reshape(4,4),
                    index=['red','blue','yellow','white'],
                    columns=['ball','pen','pencil','book'])
print('mframe:',mframe,sep='\n')
print('converting dataframe into series using stack() method')
print(mframe.stack())
```

```
mframe:
        ball  pen  pencil  book
red        0    1       2     3
blue       4    5       6     7
yellow     8    9      10    11
white     12   13      14    15
converting dataframe into series using stack() method
red     ball       0
        pen        1
        pencil     2
        book       3
blue    ball       4
        pen        5
        pencil     6
        book       7
yellow  ball       8
        pen        9
        pencil    10
        book      11
white   ball      12
        pen       13
        pencil    14
        book      15
dtype: int32
```

In [30]:

```
#creating hierachical indexing for dataframe
import pandas as pd
import numpy as np
mframe=pd.DataFrame(np.arange(16).reshape(4,4),
                    index=[['white','white','red','red'],['up','down','up','down']],
                    columns=[['pen','pen','pencil','pencil'],[1,2,1,2]])
print('mframe:',mframe,sep='\n')
#adding headers to the indexes
mframe.index.names=['colors','direction']
mframe.columns.names=['object','id']
print(mframe)
#swapping of indexes
print(mframe.swaplevel('colors','direction'))
#sorting values
mframe.sort_index(level='colors') #error coming
#summation
print(mframe.sum(level='colors'))
```

```
mframe:
           pen     pencil
             1   2      1   2
white up     0   1      2   3
      down   4   5      6   7
red   up     8   9     10  11
      down  12  13     14  15
object            pen     pencil
id                  1   2      1   2
colors direction
white  up           0   1      2   3
       down         4   5      6   7
red    up           8   9     10  11
       down        12  13     14  15
object            pen     pencil
id                  1   2      1   2
direction colors
up        white     0   1      2   3
down      white     4   5      6   7
up        red       8   9     10  11
down      red      12  13     14  15
object pen     pencil
id       1   2      1   2
colors
white    4   6      8  10
red     20  22     24  26
```

In [17]:

```
csvfile=pd.read_csv('data.csv')
print(csvfile)
```

```
     id     name  marks
0  3001   afreen    9.0
1  3002  keerthi    9.5
2  3003    priya    9.3
```

In [18]:

```
csvfile=pd.read_csv('data.csv',sep=',')
print(csvfile)
```

```
     id     name  marks
0  3001   afreen    9.0
1  3002  keerthi    9.5
2  3003    priya    9.3
```

In [19]:

```
csvfile=pd.read_csv('data.csv',names=['sid','snames'])
print(csvfile)
```

```
          sid snames
id        name  marks
3001    afreen      9
3002   keerthi    9.5
3003     priya    9.3
```

In [21]:

```
import pandas as pd
csvfile=pd.read_csv('sample2.csv',index_col=['color','direction'])
print(csvfile)
```

```
                 it1  it2
color direction
red   up           1    2
      down         2    3
white up           3    4
      left         5    6
      down         7    8
```

In [22]:

```
txtfile=pd.read_table('mydata.txt',sep='\s+')
print(txtfile)
```

```
   rollno name  percent
0    3001    X       93
1    3002    Y       95
2    3003    Z       91
```

In [23]:

```
txtfile=pd.read_table('mydata.txt',sep='\D+')
print(txtfile)
```

```
   Unnamed: 0  Unnamed: 1
0        3001          93
1        3002          95
2        3003          91

C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: Parser
Warning: Falling back to the 'python' engine because the 'c' engine does n
ot support regex separators (separators > 1 char and different from '\s+'
are interpreted as regex); you can avoid this warning by specifying engine
='python'.
  """Entry point for launching an IPython kernel.
```

In [ ]:

```

```

UNIT-II

Data Wrangling and Preprocessing

Topics: Handling Missing Data (mean, median, drop, interpolation), Dealing with Duplicates, Outliers, and Anomalies, Encoding Categorical Variables (Label, One-hot), Data Transformation: Scaling, Normalization, Binning, Data Types Conversion and Data Type Casting

Handling Missing Data (mean, median, drop, interpolation)

Handling missing data is a crucial step in data preprocessing, as missing values can negatively impact the performance of machine learning models. To ensure the quality of analysis and model performance, we handle missing data using different strategies:

1. **Drop the data** (if the impact is minimal)
2. **Replace with Mean**
3. **Replace with Median**
4. **Interpolate** (predict missing value using surrounding data)

Sample DataFrame

We'll create a simple pandas DataFrame with missing values.

```
import pandas as pd
import numpy as np
# Sample data
data = {
    'Name': ['Aarav', 'Bhavya', 'Chirag', 'Divya', 'Esha'],
    'Age': [25, np.nan, 34, np.nan, 29],
    'Monthly_Income': [45000, 52000, np.nan, 48000, np.nan]
}

df = pd.DataFrame(data)
print("Original Data:\n", df)
```

#Output

```
   Name  Age  Monthly_Income
0 Aarav  25.0        45000.0
I Bhavya  NaN        52000.0
2 Chirag  34.0          NaN
3  Divya  NaN        48000.0
4   Esha  29.0          NaN
```

1. Dropping Missing Data

Drop Rows with Any Missing Value: Removes rows where any column has a missing value.

```
df_dropped_rows = df.dropna()
print("After Dropping Rows:\n", df_dropped_rows)
```

#Output

After Dropping Rows:

	Name	Age	Monthly_Income
0	Aarav	25.0	45000.0

Use When:

- Dataset is large
- Missing data is very minimal

2. Fill Missing Data with Mean

```
df_mean = df.copy()
df_mean['Age'] = df_mean['Age'].fillna(df_mean['Age'].mean())
df_mean['Monthly_Income'] =
df_mean['Monthly_Income'].fillna(df_mean['Monthly_Income'].mean())
print("Filled with Mean:\n", df_mean)
```

#Output

Filled with Mean:

	Name	Age	Monthly_Income
0	Aarav	25.000000	45000.000000
1	Bhavya	29.333333	52000.000000
2	Chirag	34.000000	48333.333333
3	Divya	29.333333	48000.000000
4	Esha	29.000000	48333.333333

Mean Age = (25 + 34 + 29) / 3 = 29.33
Mean Monthly_Income = (45000 + 52000 + 48000) / 3 = 48333.33

Use When:

- Data is **normally distributed** (symmetrical)
- No extreme outliers

3. Fill Missing Data with Median

```
df_median = df.copy()
# Replace missing values with the median (no inplace=True)
df_median['Age'] = df_median['Age'].fillna(df_median['Age'].median())
df_median['Monthly_Income'] =
df_median['Monthly_Income'].fillna(df_median['Monthly_Income'].median())
print("Filled with Median:\n", df_median)
```

#Output

Filled with Median:

	Name	Age	Monthly_Income
0	Aarav	25.0	45000.0
1	Bhavya	29.0	52000.0
2	Chirag	34.0	48000.0
3	Divya	29.0	48000.0
4	Esha	29.0	48000.0

Explanation:

- Median Age = 29
- Median Income = 48000

How to calculate Median

The median is the middle point in a dataset—half of the data points are smaller than the median and half of the data points are larger.

To find the median:

- Arrange the data points from smallest to largest.
- If the number of data points is odd, the median is the middle data point in the list.
- If the number of data points is even, the median is the average of the two middle data points in the list.

Use When Data is **skewed and y**ou want to avoid outlier influence.

4. Fill Missing Data using Interpolation

```
df_interp = df.copy()
# Interpolate missing values (linear by default)
df_interp['Age'] = df_interp['Age'].interpolate()
df_interp['Monthly_Income'] = df_interp['Monthly_Income'].interpolate()
print("Filled with Interpolation:\n", df_interp)
```

#Output
Filled with Interpolation:

	Name	Age	Monthly_Income
0	Aarav	25.0	45000.0
1	Bhavya	29.5	52000.0
2	Chirag	34.0	50000.0
3	Divya	31.5	48000.0
4	Esha	29.0	48000.0

Explanation:

Interpolates values **linearly** using available values above and below the missing data.

Use When:

- Data has a natural order (like time-series, sequences)
- Values are continuous

Step-by-Step Interpolation Math

We use the **linear interpolation formula**:

$$y = y_1 + \frac{(x - x_1)}{(x_2 - x_1)} \cdot (y_2 - y_1)$$

Dealing with Duplicates

In data analysis, duplicate data can skew your results. Pandas provides useful functions like duplicated() and drop_duplicates() to handle such issues.

Pandas functions for handling duplicates

Purpose	Code Example
Detect duplicate rows	df.duplicated()
Remove duplicate rows	df.drop_duplicates()
Remove duplicates from specific col	df.drop_duplicates(subset='Name')
Count duplicate values	df['Name'J.value_counts()
Show all duplicate rows	df[df.duplicated('Name', keep=False)J
Mark duplicates in column	df['is_duplicate'J = df.duplicated(...)

Creating a Sample DataFrame with Duplicates

```
import pandas as pd
data = {
    'EmpID': [201, 202, 203, 201, 204, 205, 202],
    'Name': ['Amit', 'Priya', 'Ravi', 'Amit', 'Neha', 'Kiran', 'Priya'],
    'Department': ['HR', 'Finance', 'IT', 'HR', 'Finance', 'IT', 'Finance']
}
df = pd.DataFrame(data)
print("Original DataFrame:")
print(df)
```

#Output

	EmpID	Name	Department
0	20I	Amit	HR
I	202	Priya	Finance
2	203	Ravi	IT
3	20I	Amit	HR
4	204	Neha	Finance
5	205	Kiran	IT
6	202	Priya	Finance

1. Detecting Duplicate Rows

Syntax : `df.duplicated()`

Example

```
print(df.duplicated())
```

#Output:
0 False

I False

2 False

3 True

4 False

5 False

6 True

Rows 3 and 6 are duplicates of 0 and I respectively.

2. Removing Duplicate Rows

Syntax: `df.drop_duplicates()`

Example

```
df_unique = df.drop_duplicates()

print(df_unique)
```

#Ouput

	EmpID	Name	Department
0	20I	Amit	HR
I	202	Priya	Finance
2	203	Ravi	IT
4	204	Neha	Finance
5	205	Kiran	IT

3. Removing Duplicates Based on Specific Column

Syntax: `df.drop_duplicates(subset='Column Name')`

Example: Drop duplicate Names

```
df_unique_names = df.drop_duplicates(subset='Name')

print(df_unique_names)
```

#Output

```
  EmpID  Name Department
0  20I   Amit     HR
I  202  Priya   Finance
2  203   Ravi     IT
4  204   Neha   Finance
5  205  Kiran     IT
```

4. Find All Duplicate Names (Keep=False)

```
duplicates = df[df.duplicated(subset='Name', keep=False)]

print(duplicates)
```

#Output

```
EmpID  Name Department
0  20I   Amit     HR
I  202  Priya   Finance
3  20I   Amit     HR
6  202  Priya   Finance
```

5. Count How Many Times Each Name Appears

```
print(df['Name'].value_counts())
```

#Outpu

Amit 2
Priya 2
Ravi 1
Neha 1
Kiran 1

6. Mark Duplicates in a New Column

```
df['is_duplicate'] = df.duplicated(subset=['EmpID', 'Name'], keep=False)
print(df)
```

#Output

	EmpID	Name	Department	is_duplicate
0	201	Amit	HR	True
1	202	Priya	Finance	True
2	203	Ravi	IT	False
3	201	Amit	HR	True
4	204	Neha	Finance	False
5	205	Kiran	IT	False
6	202	Priya	Finance	True

Problem Statement:

A company tracks employee visits to its branches. Some employees may have visited the same branch more than once. Here handle duplicates in pandas.

Dealing with Duplicates in Pandas

```
import pandas as pd
# Step 1:Create sample DataFrame with Indian employee visit data
data = {
    'EmpID': [301, 302, 303, 301, 304, 305, 302],
    'EmployeeName': ['Lakshmi', 'Manoj', 'Anjali', 'Lakshmi', 'Rohit', 'Divya',
  'Manoj'],
    'Branch': ['Hyderabad', 'Chennai', 'Bangalore', 'Hyderabad', 'Chennai', 'Mumbai',
  'Chennai']
}
df = pd.DataFrame(data)
print("Original DataFrame:\n", df)
# Step 2: Identify duplicate visits (same EmpID & Branch)
# Use duplicated() with subset on both EmpID and Branch
df['Is_Duplicate_Visit'] = df.duplicated(subset=['EmpID', 'Branch'], keep=False)
print("\n🔍 Step 2 - Duplicates Identified (EmpID + Branch):\n", df)
# Step 3: Remove duplicate visits — keep only the first occurrence
```

```
df_unique_visits = df.drop_duplicates(subset=['EmpID', 'Branch'], keep='first')
print("\n Step 3 - Unique Visits (Removed Duplicates):\n", df_unique_visits) #
Step 4:Count how many times each branch was visited
branch_counts = df['Branch'].value_counts()
print("\n Step 4 - Branch Visit Counts:\n", branch_counts)
# Step 5: Mark all duplicate employee entries regardless of branch
df['Is_Duplicate_Emp'] = df.duplicated(subset='EmpID', keep=False)
print("\ncn Step 5 - Duplicate Employee Records (Based on EmpID):\n", df)
```

Dealing with Outliers

Outliers are data points that differ significantly from other observations. They can:

- Indicate variability in data.
- Be due to measurement error or data entry issues.
- Skew the statistical analysis if not handled.

Tools from pandas & numpy for Detecting Outliers
We often combine pandas with numpy or scipy for detecting outliers using:

I. **Statistical methods** (IQR, Z-Score)
2. **Visual methods** (Boxplots, Histograms)

Detecting Outliers using IQR (Interquartile Range)

The IQR is the range between the 25th percentile (QI) and the 75th percentile (Q3).
Formula:

IQR = Q3 - QI
Lower bound = QI - I.5 * IQR
Upper bound = Q3 + I.5 * IQR

Here **Outliers** are any data point below the lower bound or above the upper bound.

```
#Program
import pandas as pd
data = {'score': [55, 60, 61, 62, 63, 64, 1000, 65, 66, 67]}
df = pd.DataFrame(data)
Q1 = df['score'].quantile(0.25)
Q3 = df['score'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
outliers = df[(df['score'] < lower_bound) I (df['score'] > upper_bound)]
print("Outliers:")
print(outlier
```

```
#Output
Outliers:
   score
6   1000
```

2. Using Z-Score (Standard Deviation Method)

Z-Score tells us how many standard deviations a value is from the mean.

Formula:

Z = (X - mean) / standard deviation

Threshold: Common cut-off(IZI) is ±3. Values beyond ±3 are considered outliers.

If |Z| > 3 → Outlier (can also use threshold 2.5 or 2)

```
import pandas as pd
import numpy as np
# Create a sample dataset
data = {'score': [55, 60, 61, 62, 63, 64, 1000, 65, 66, 67]}
df = pd.DataFrame(data)
print(df)
#Calculate Z-Scores
mean = df['score'].mean()
std = df['score'].std()
df['z_score'] = (df['score'] - mean) / std
print(df)
#Filter Outliers (IZI > 2.5)
z_outliers = df[df['z_score'].abs() > 2.5]
print(z_outliers)
```

```
#Output
    score
0     55
1     60
2     61
3     62
4     63
5     64
6   1000
7     65
8     66
9     67
```

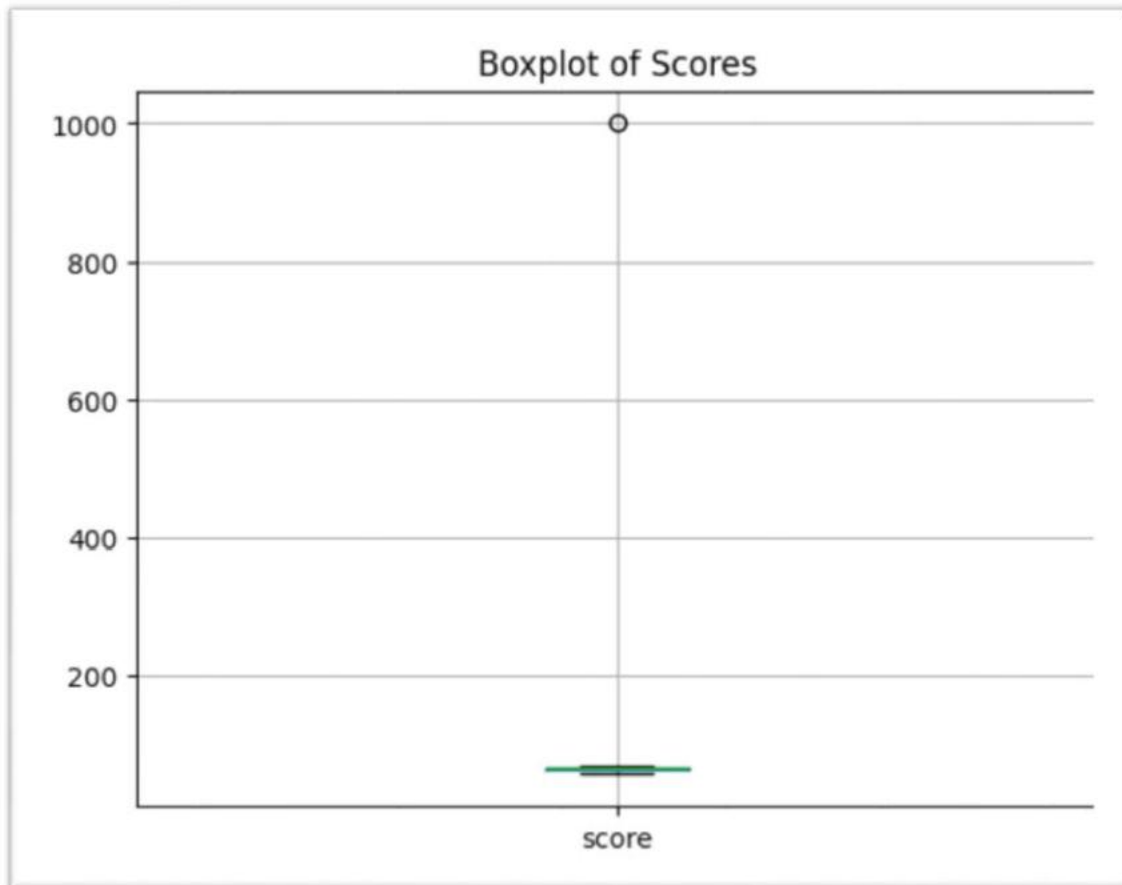
```
    score   z_score
0      55 -0.341692
1      60 -0.324827
2      61 -0.321454
3      62 -0.318080
4      63 -0.314707
5      64 -0.311334
6    1000  2.845859
7      65 -0.307961
8      66 -0.304588
9      67 -0.301215
   score   z_score
6   1000  2.845859
```

3. Using Boxplots (Visualization)

Boxplots show outliers as dots outside the whiskers.

```
import matplotlib.pyplot as plt
# Create a sample dataset
data = {'score': [55, 60, 61, 62, 63, 64, 1000, 65, 66, 67]}
df = pd.DataFrame(data)
print(df)
df.boxplot(column='score')
plt.title("Boxplot of Scores")
plt.show()

#Output
    score
0      55
1      60
2      61
3      62
4      63
5      64
6    1000
7      65
8      66
9      67
```



Dealing with Outliers

Once detected, you can choose to either **remove**, **cap**, **transform**, or **replace** the outliers.

1. Removing Outliers (Using IQR)

If outliers are likely due to error or noise, just remove them.

```
#Program
import pandas as pd
data = {'score': [55, 60, 61, 62, 63, 64, 1000, 65, 66, 67]}
df = pd.DataFrame(data)
Q1 = df['score'].quantile(0.25)
Q3 = df['score'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
df_no_outliers = df[(df['score'] >= lower_bound) & (df['score'] <=
upper_bound)]
print(df_no_outliers)

#Output
   score
```

```
0      55
1      60
2      61
3      62
4      63
5      64
7      65
8      66
9      67
```

2: **Capping Outliers (Winsorizing)**

Capping, or **winsorizing**, is a technique used to **limit extreme values** in a dataset by setting them to a specified percentile value rather than removing them.

Purpose:

- Keep **all rows** in the dataset (unlike removing outliers).
- Reduce the **influence** of extreme values on statistical models.
- Maintain the **data structure** and **row count**.

How Does It Work?

Instead of removing outliers beyond a threshold, we **cap** them at a chosen **percentile limit**, like the 5th and 95th percentiles. This retains all data points but reduces the influence of outliers.

Syntax: DataFrame.clip(lower=lower_cap, upper=**upper_cap**)
Where lower_cap=5th percentile and upper_cap= 95th percentile.

```
import pandas as pd
data = {'score': [55, 60, 61, 62, 63, 64, 1000, 65, 66, 67]}
df = pd.DataFrame(data)
lower_cap = df['score'].quantile(0.05)
upper_cap = df['score'].quantile(0.95)
df['score_capped'] = df['score'].clip(lower=lower_cap, upper=upper_cap)
print(df[['score', 'score_capped']])


#Output
    score   score_capped
0     55          57.25
1     60          60.00
2     61          61.00
```

```
3      62           62.00
4      63           63.00
5      64           64.00
6    1000          580.15
7      65           65.00
8      66           66.00
9      67           67.00
```

3: **Replacing Outlier with Median**

Instead of **removing** or **capping**, we **replace outlier values** (detected using Z-Score, IQR, etc.) with the **median** of the data. The median is **resistant to outliers**, making it a good choice for substitution.

```
#Program
import pandas as pd
import numpy as np
#Create a sample dataset
data = {'score': [55, 60, 61, 62, 63, 64, 1000, 65, 66, 67]}
df = pd.DataFrame(data)
print(df)
#Calculate Z-Scores
mean = df['score'].mean()
std = df['score'].std()
df['z_score'] = (df['score'] - mean) / std
print(df)
#Find median
median_val = df['score'].median()
#Replace outliers (Z <= 2.5 or Z < -2.5) with median
df['score_replaced'] = df['score'].where(df['z_score'].abs() < =2.5, median_val)
print(df[['score', 'z_score', 'score_replaced']])
```

```
#Output
   score
```

```
0      55
1      60
2      61
3      62
4      63
5      64
6    1000
7      65
8      66
9      67
   score   z_score
0     55 -0.341692
1     60 -0.324827
2     61 -0.321454
3     62 -0.318080
4     63 -0.314707
5     64 -0.311334
6   1000  2.845859
7     65 -0.307961
8     66 -0.304588
9     67 -0.301215
   score   z_score  score_replaced
0     55 -0.341692            55.0
1     60 -0.324827            60.0
2     61 -0.321454            61.0
3     62 -0.318080            62.0
4     63 -0.314707            63.0
5     64 -0.311334            64.0
6   1000  2.845859            63.5
7     65 -0.307961            65.0
8     66 -0.304588            66.0
9     67 -0.301215            67.0
```

4. Replacing Outlier with Log Transformation

Log transformation compresses **large values** and expands **small ones**, helping reduce the **impact of outliers** and **skewness**.

Formula: $\log(x)$

Note: Apply only to **positive** values (non-zero, non-negative).

```
#Program
import pandas as pd
import numpy as np
```

```
# Create a sample dataset
data = {'score': [55, 60, 61, 62, 63, 64, 1000, 65, 66, 67]}
df = pd.DataFrame(data)
print(df)
# Apply log transformation
df['log_score'] = np.log(df['score'])
print(df[['score', 'log_score']])
```

```
#Output
   score
0     55
1     60
2     61
3     62
4     63
5     64
6   1000
7     65
8     66
9     67
   score  log_score
0     55   4.007333
1     60   4.094345
2     61   4.110874
3     62   4.127134
4     63   4.143135
5     64   4.158883
6   1000   6.907755
7     65   4.174387
8     66   4.189655
9     67   4.204693
```

Dealing with Anomalies

Anomalies (also known as outliers or unusual data points) are data values that deviate significantly from other observations. These can occur due to measurement errors, data entry issues, or natural variability. Handling them correctly is important for robust data analysis.

Types of Anomalies

I. Global outliers – Entirely different from the rest of the dataset.
2. Contextual Anomalies: Unusual in specific context (e.g., temperature high in winter).
3. Collective Anomalies: A group of related data points is anomalous.

Causes of Anomalies

- Data entry or measurement errors.
- Sensor malfunctions or logging issues.
- Natural rare events (e.g., fraud transactions).
- Sudden changes in patterns or trends.

Techniques to Detect Anomalies

1. Statistical Methods

- Z-score: Values beyond ±3 standard deviations are considered anomalies.
- IQR (Interquartile Range): Data outside QI−I.5xIQR and Q3+I.5xIQR are flagged as outliers.

2. Visualization

- Box Plots: Highlight extreme values.
- Scatter Plots: Show unusual data points.
- Histograms: Reveal unusual distributions.

3. Machine Learning Approaches

- Clustering-based (e.g., DBSCAN, K-Means): Points far from clusters are anomalies.
- Isolation Forests: Isolate anomalies by random partitioning.
- Auto encoders: Use reconstruction error to detect anomalies in high-dimensional data.

Dealing with Anomalies

1. Remove Anomalies

- Suitable when anomalies are due to data entry errors or irrelevant noise.
- Example: Dropping values with Z-score > 3.

2. Cap/Floor Values (Winsorizing)

- Replace extreme values with nearest acceptable threshold.

3. Transformation

- Apply log, square root, or Box-Cox transformations to reduce the effect of outliers.

4. Imputation

- Replace anomalies with mean, median, or model-based predictions.

5. Keep Anomalies

- If anomalies are valid rare events (like fraud detection), retain them for analysis.

Encoding Categorical Variables

Encoding categorical variables is a process of converting non-numeric (categorical) data into numerical format so that machine learning algorithms can interpret and process them effectively. Many algorithms (e.g., linear regression, neural networks, SVM) work only with numerical values, hence categorical variables must be encoded.

Types of Categorical Variables

I. **Nominal Variables**: Categories without any order (e.g., color: red, blue, green).

2. **Ordinal Variables**: Categories with a defined order or ranking (e.g., low, medium, high).

Common Encoding Techniques

1. Label Encoding

- Converts each category into a unique integer.
- Suitable for ordinal variables.

```
#Program
from sklearn.preprocessing import LabelEncoder
import pandas as pd
data = pd.DataFrame({'Color': ['Red', 'Blue', 'Green', 'Red']})
encoder = LabelEncoder()
```

```
data['Color_encoded'] = encoder.fit_transform(data['Color'])
print(data)
```

```
#Output
   Color  Color_encoded
0    Red              2
1   Blue              0
2  Green              1
3    Red              2
```

2. One-Hot Encoding

- Creates a new binary column for each category.
- Suitable for nominal variables.

#Program

```
import pandas as pd
data = pd.DataFrame({'Color': ['Red', 'Blue', 'Green', 'Red']})
encoded = pd.get_dummies(data, columns=['Color'])
print(encoded)
```

```
#Output
   Color_Blue  Color_Green  Color_Red
0           0            0          1
1           1            0          0
2           0            1          0
3           0            0          1
```

3. Ordinal Encoding

- Assigns ordered integers to categories based on their rank.
- You define the order manually.

#Program

```
import pandas as pd
from sklearn.preprocessing import OrdinalEncoder
data = pd.DataFrame({'Size': ['Small', 'Medium', 'Large', 'Medium']})
encoder = OrdinalEncoder(categories=[['Small', 'Medium', 'Large']])
data['Size_encoded'] = encoder.fit_transform(data[['Size']])
print(data)
```

```
#Output
     Size  Size_encoded
0   Small           0.0
1  Medium           1.0
2   Large           2.0
3  Medium           1.0
```

4. Binary Encoding

- Combines hashing and one-hot encoding to reduce dimensionality.
- Converts categories into binary code.
- Useful when you have many categories.

```
#Program
import category_encoders as ce
import pandas as pd
data = pd.DataFrame({'City': ['Paris', 'London', 'Berlin', 'Paris']})
encoder = ce.BinaryEncoder(cols=['City'])
encoded = encoder.fit_transform(data)
print(encoded)
```

5. Target / Mean Encoding

- Replaces each category with the mean of the target variable for that category.
- Useful in supervised learning.

```
#Program
import pandas as pd
data = pd.DataFrame({
                'City': ['Paris', 'London', 'Berlin', 'Paris'],
                'Sales': [100, 200, 150, 120] })
target_mean = data.groupby('City')['Sales'].mean()
data['City_encoded'] = data['City'].map(target_mean)
print(data)

#Output
     City  Sales  City_encoded
0   Paris    100         110.0
1  London    200         200.0
2  Berlin    150         150.0
3   Paris    120         110.0
```

Choosing the Right Technique

- **Nominal variables: One-hot encoding or binary encoding.**
- **Ordinal variables: Label or ordinal encoding.**
- **High-cardinality categorical variables: Binary or target encoding.**

Data Transformations

Data transformations are essential steps in data preprocessing, helping to make datasets suitable for analysis or machine learning models.

Three common techniques are Scaling, Binning, and Normalization.

1. Scaling

Scaling adjusts the range of features so they can be compared or used in algorithms that are sensitive to feature magnitude (e.g., k-NN, gradient descent, SVM).

Why use Scaling?

- Features with large ranges can dominate those with small ranges.
- Speeds up convergence in optimization algorithms.
- Ensures uniformity for distance-based algorithms.

Common Types:

- **Min-Max Scaling (Rescaling)**
  Brings data into a fixed range, usually [0,IJ.

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

- **Standardization (Z-score scaling)**
  Centers data around mean 0 with standard deviation I.

$$x_{scaled} = \frac{x - \mu}{\sigma}$$

```
#Program
from sklearn.preprocessing import MinMaxScaler, StandardScaler
import numpy as np
data = np.array([[10], [20], [30], [40], [50]])
# Min-Max Scaling
minmax = MinMaxScaler()
print(minmax.fit_transform(data))
# Standardization
standard = StandardScaler()
print(standard.fit_transform(data))
```

```
#Output
[[0.  ]
 [0.25]
 [0.5 ]
 [0.75]
 [1.  ]]
[[-1.41421356]
 [-0.70710678]
 [ 0.        ]
 [ 0.70710678]
  [ 1.41421356]]
```

2. Binning (Discretization)

Binning converts continuous values into categorical bins or intervals.

Why use Binning?

- Reduces noise and variability.
- Makes models easier to interpret.
- Useful in histogram creation and feature engineering.

Types:

- Equal-width Binning: Divides range into equal-sized intervals.
- Equal-frequency Binning: Each bin has approximately the same number of samples.
- Custom Binning: Based on domain knowledge.

```
#Program
import pandas as pd
data = [5, 7, 12, 18, 24, 30, 35, 42]
bins = [0, 10, 20, 30, 50]
labels = ['Low', 'Medium', 'High', 'Very High']
binned_data = pd.cut(data, bins=bins, labels=labels)
print(binned_data)
```

```
#Output
['Low', 'Low', 'Medium', 'Medium', 'High', 'High', 'Very High', 'Very High']
Categories (4, object): ['Low' < 'Medium' < 'High' < 'Very High']
```

3. Normalization

Normalization scales individual samples so that the entire vector or row has a unit norm. It's often used for text data, clustering, and algorithms where direction of data matters more than magnitude.

Why use Normalization?

- Useful in distance-based models (k-NN, cosine similarity).
- Ensures fair contribution of each feature.

Common Method:

- L2 Normalization:

$$x_{norm} = \frac{x}{||x||_2}$$

- **L1 Normalization:**

$$x_{norm} = \frac{x}{||x||_1}$$

```
#Program
from sklearn.preprocessing import Normalizer
import numpy as np
data = np.array([[3, 4], [1, 2], [2, 2]])
norm = Normalizer(norm='l2')
print(norm.fit_transform(data))
```

#Output

```
[[0.6        0.8       ]
 [0.4472136  0.89442719]
 [0.70710678 0.70710678]]
```

Data type conversions and Data type casting

In pandas, data type conversions and casting are crucial for preparing data, ensuring consistency, and optimizing memory. Conversion changes a Series or DataFrame column to types like int, float, string, datetime, or categorical, often needed when importing data stored as text. Casting further optimizes types, such as downcasting float64 to float32 or converting integers to categories to save memory.

Method	Purpose	Example
astype(dtype)	**Convert Series/DataFrame column to a specified type**	**df['A'J = df['A'J.astype(float)**
pd.to_numeric()	**Convert values to numeric, with error handling**	**pd.to_numeric(df['col'J, errors='coerce')**
pd.to_datetime()	**Convert values to datetime format**	**df['date'J = pd.to_datetime(df['date'J)**
pd.to_timedelta()	**Convert values to timedelta objects**	**df['duration'J = pd.to_timedelta(df['duration'J)**
Converting to categorical	**Optimize repeated values for memory and speed**	**df['category'J = df['category'J.astype('category')**

```
#Program
import pandas as pd
# Sample DataFrame with mixed types
data = {
    'id': ['1', '2', '3'],
    'amount': ['100.5', '200.0', '300.25'],
    'date': ['2025-08-01', '2025-08-02', '2025-08-03'],
    'status': ['paid', 'pending', 'paid']
}
df = pd.DataFrame(data)
print("Before Conversion:")
```

```
print(df.dtypes)
print(df)
# 1. Convert 'id' to integer
df['id'] = df['id'].astype(int)
# 2. Convert 'amount' to float and downcast to save memory
df['amount'] = pd.to_numeric(df['amount'], downcast='float')
# 3. Convert 'date' to datetime
df['date'] = pd.to_datetime(df['date'])
# 4. Convert 'status' to categorical
df['status'] = df['status'].astype('category')
print("\nAfter Conversion and Casting:")
print(df.dtypes)
print(df)
```

#Output

```
Before Conversion:
id          object
amount      object
date        object
status      object
dtype: object
  id  amount        date   status
0  1  100.5  2025-08-01     paid
1  2  200.0  2025-08-02  pending
2  3  300.25 2025-08-03     paid

After Conversion and Casting:
id             int64
amount       float32
date         datetime64[ns]
status    category
dtype: object
   id  amount        date   status
0   1   100.5 2025-08-01     paid
1   2   200.0 2025-08-02  pending
2   3   300.2 2025-08-03     paid
```

UNIT III – Univariate and Bivariate Analysis

Measures of Central Tendency and Dispersion, Distribution Plots: Histograms, Boxplots, KDE, Bar Charts, Count Plots, Pie Charts, Bivariate Analysis: Scatter Plots, Pair Plots, Heatmaps, Correlation and Covariance Analysis

1. Mean (Arithmetic Mean):-Average of data.

Formula (Ungrouped data)

$$\bar{x} = \frac{\sum x}{n}$$

Formula (Grouped data)

$$\bar{x} = \frac{\sum fx}{\sum f}$$

Where

x = value, f = frequency, n = number of observations

Properties

I.Uses all data values

II.Sensitive to outliers

III.Most widely used

2. Median:-Middle value after arranging data in ascending order.

Formula

If n is odd:

$$Median = x_{(n+1)/2}$$

If n is even:

$$Median = \frac{x_{n/2} + x_{(n/2+1)}}{2}$$

For grouped data

$$Median = L + \left(\frac{\frac{N}{2} - cf}{f}\right) h$$

Where

L = lower class boundary

N = total frequency

cf = cumulative frequency before median class

f = frequency of median class

h = class width

Properties

I.Not affected by outliers

II.Used in skewed data

3. Mode

- Most frequently occurring value.
- Grouped Data Formula

$$Mode = L + \left(\frac{f_1 - f_0}{2f_1 - f_0 - f_2}\right) h$$

- Where
  f_1 = frequency of modal class
  f_0 = frequency of previous class
  f_2 = frequency of next class
- Empirical Relation

$$Mode \approx 3Median - 2Mean$$

Relationship in Skewed Data

Distribution	Relation
Symmetrical	Mean = Median = Mode
Positively skewed	Mean > Median > Mode
Negatively skewed	Mean < Median < Mode

Measures of Dispersion
Shows spread/variability of data.
Range

- Range=Max−Min

Coefficient of range:

$$\frac{Max - Min}{Max + Min}$$

Quartile Deviation (IQR)

$$QD = \frac{Q_3 - Q_1}{2}$$

Coefficient:

$$\frac{Q_3 - Q_1}{Q_3 + Q_1}$$

Variance
Population:

$$\sigma^2 = \frac{\sum(x - \mu)^2}{N}$$

- Sample:

$$s^2 = \frac{\sum(x - \bar{x})^2}{n - 1}$$

Mean Deviation

$$MD = \frac{\sum |x - \bar{x}|}{n}$$

Standard Deviation

$$\sigma = \sqrt{\frac{\sum(x - \mu)^2}{N}}$$

Shortcut formula:

$$\sigma = \sqrt{\frac{\sum x^2}{n} - (\bar{x})^2}$$

Coefficient of Variation (CV)

$$CV = \frac{\sigma}{\bar{x}} \times 100$$

Used to compare consistency.

Distribution Plots (Univariate Analysis)

Histogram
Shows frequency distribution of continuous data.

Key Points

- X-axis → bins
- Y-axis → frequency
- Shows shape of distribution

Box Plot

Shows **5-number summary**

$$Min,\ Q1,\ Median,\ Q3,\ Max$$

Outlier Rule

$$Outlier > Q3 + 1.5(IQR)$$
$$Outlier < Q1 - 1.5(IQR)$$

KDE Plot (Kernel Density Estimation)
Smooth curve of probability density.
Used to understand data distribution shape.

Bar Chart:-Used for categorical data.
X-axis → categories
Y-axis → values
Count Plot:-Shows count of observations in each category.
Pie Chart:-Shows percentage distribution.

$$Percentage = \frac{Value}{Total} \times 100$$

Bivariate Analysis:-Studies relationship between two variables.
Scatter Plot:-Shows relationship between X and Y.

Interpretation

Pattern	Meaning
Upward trend	Positive correlation
Downward trend	Negative correlation
Random	No correlation

Pair Plot
Shows scatter plots for all variable pairs.

Heatmap
Color representation of matrix values.
Mostly used for correlation matrix.
Correlation Analysis
Measures strength of relationship.
Pearson Correlation

$$r = \frac{\sum(x-\bar{x})(y-\bar{y})}{\sqrt{\sum(x-\bar{x})^2 \sum(y-\bar{y})^2}}$$

r value	Meaning
+1	Perfect positive
0	No relation
-1	Perfect negative

Spearman Rank Correlation

$$\rho = 1 - \frac{6\sum d^2}{n(n^2-1)}$$

Used for ranked data.
Covariance
Measures direction of relationship**.**

$$Cov(X,Y) = \frac{\sum(x-\bar{x})(y-\bar{y})}{n}$$

Interpretation

Value	Meaning
Positive	Move together
Negative	Move opposite

Relation with correlation:

$$r = \frac{Cov(X,Y)}{\sigma_x \sigma_y}$$

UNIT IV – DATA VISUALIZATION TECHNIQUES

1. INTRODUCTION TO DATA VISUALIZATION

What is Data Visualization?

Data visualization is the graphical representation of data using plots, charts, and maps to make patterns, trends, and insights easier to understand.

Why Visualization?

- Converts raw data → meaningful information
- Helps in decision making
- Identifies trends, outliers, correlations
- Communicates results effectively

Types of Visualization

Type	Purpose
Univariate	Single variable
Bivariate	Two variables
Multivariate	More than two variables

Visualization with Matplotlib and Seaborn

In the Python ecosystem, two libraries reign supreme for static plotting

Matplotlib: The "grandfather" of Python visualization. It is a low-level library that offers maximum control over every element of a figure.

Seaborn: Built on top of Matplotlib, it provides a high-level interface for drawing attractive and informative statistical graphics. It integrates deeply with Pandas Data Frames.

Basic Syntax Example:

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
# Sample Data
data = {'Year': [2020, 2021, 2022, 2023], 'Sales': [250, 400, 350, 600]}
df = pd.DataFrame(data)
# Matplotlib Line Plot
plt.plot(df['Year'], df['Sales'])
plt.show()
# Seaborn Line Plot
sns.lineplot(x='Year', y='Sales', data=df)
plt.show()
```

Customizing Plots: Titles, Legends, Labels, and Themes

A plot without labels is just a bunch of lines. Customization makes data readable and "presentation-ready."

Feature	Matplotlib Command	Seaborn Equivalent/Notes
Title	plt.title("Name")	Same
Axis Labels	plt.xlabel(), plt.ylabel()	Automatically inferred from DataFrame columns
Legends	plt.legend()	Controlled via the hue parameter
Themes	plt.style.use('ggplot')	sns.set_theme(style="whitegrid")

Example with Customization:

```
sns.set_theme(style="darkgrid") # Setting a theme
plt.figure(figsize=(8, 5))
plot = sns.barplot(x='Year', y='Sales', data=df, palette='viridis')
plt.title("Annual Sales Growth", fontsize=15)
plt.xlabel("Financial Year", fontsize=12)
plt.ylabel("Revenue (in USD)", fontsize=12)
plt.legend(["Sales Line"])
plt.show()
```

Advanced Visuals: Violin, Strip, and Swarm Plots

When dealing with categorical data and distributions, standard bar charts often hide the "truth" of the data (like variance and outliers).

A. Violin Plots

A combination of a Box Plot and a Kernel Density Plot. It shows the peak, the median, and the distribution of the data.

- **Use Case:** Comparing the distribution of a variable across different categories.
- **Code:** sns.violinplot(x='day', y='total_bill', data=tips)

B. Strip Plots

A scatter plot where one variable is categorical.

- **Issue:** Points often overlap, making it hard to see density.
- **Code:** sns.stripplot(x='day', y='total_bill', data=tips)

C. Swarm Plots

Similar to a strip plot, but the points are adjusted (non-overlapping) so that they give a better representation of the distribution of values.

- **Pro-tip:** It looks like a "bee swarm."
- **Code:** sns.swarmplot(x='day', y='total_bill', data=tips)

Multivariate Visualization and Subplots

Multivariate Visualization

This involves looking at three or more variables simultaneously. In Seaborn, this is usually achieved using the hue, size, or style parameters.

Hue: Adds a color dimension (e.g., distinguishing Male vs Female in a scatter plot).

Relplot/Pairplot: Used to see relationships across the entire dataset.

Subplots

Subplots allow you to place multiple charts in a single figure (grid layout).

Example:

```
# Creating a 1x2 grid of plots
fig, axes = plt.subplots(1, 2, figsize=(12, 5))
# Plot 1: Left
sns.histplot(df['Sales'], ax=axes[0])
axes[0].set_title("Distribution")
# Plot 2: Right
sns.boxplot(y=df['Sales'], ax=axes[1])
axes[1].set_title("Outliers Check")
plt.tight_layout()
plt.show()
```

Plotly and Interactive Visualizations (Basic Overview)

While Matplotlib produces static images (PNG/PDF), **Plotly** creates interactive, web-based visualizations.

- **Interactivity:** You can hover over points to see values, zoom in on specific areas, and toggle categories on/off in the legend.
- **Plotly Express:** The high-level interface (similar to Seaborn).

Basic Plotly Example:

```
import plotly.express as px
# Loading built-in dataset
df_iris = px.data.iris()
# Interactive Scatter Plot
fig=px.scatter(df_iris, x="sepal_width", y="sepal_length", color="species", title="Interactive Iris Dataset")
fig.show()
```

UNIT V – EDA Case Studies and Real-Time Datasets

Step-by-step EDA on Sample Datasets (Titanic, Iris, Sales, etc.), Outlier Detection Techniques, Feature Engineering Techniques in EDA, EDA Report Generation using Python Notebooks, Preparing Data for Machine Learning Models

1. Step-by-Step EDA on Sample Datasets

While every dataset is unique, a standard EDA workflow involves the following phases:

A. Data Inspection

- **Loading:** Using pd.read_csv() or pd.read_excel().
- **Structure:** df.info() to check data types and df.shape for dimensions.
- **Summary Statistics:** df.describe() for central tendency and dispersion.

B. Case Study: Titanic Dataset (Classification Prep)

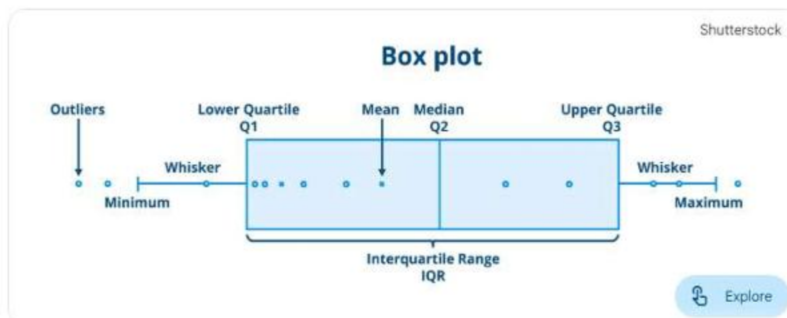
1. **Objective:** Predict survival.
2. **Missing Values:** Identify nulls in 'Age', 'Cabin', and 'Embarked'.
3. **Visualization:** Use sns.countplot(x='Survived', hue='Sex', data=df) to see survival patterns by gender.
4. **Correlation:** Use a Heatmap to see how 'Fare' or 'Pclass' relates to 'Survived'.

C. Case Study: Iris Dataset (Multivariate Analysis)

1. **Objective:** Categorize flowers into three species.
2. **Pairplots:** Use sns.pairplot(df, hue='species') to visualize clusters and feature separations instantly.

Outlier Detection Techniques

Outliers are data points that differ significantly from other observations. They can skew statistical results and degrade ML model performance.



A. Interquartile Range (IQR) Method

This is the most common method for skewed data.

- **IQR Calculation:** $IQR = Q3 - Q1$
- **Upper Bound:** $Q3 + (1.5 \times IQR)$
- **Lower Bound:** $Q1 - (1.5 \times IQR)$
- Any point outside these bounds is an outlier.

B. Z-Score Method

Used for data that follows a Normal (Gaussian) Distribution.

- **Formula:** $Z = \frac{(x - \mu)}{\sigma}$ (where μ is mean and σ is standard deviation).
- **Threshold:** Typically, a Z-score > 3 or < -3 is considered an outlier.

C. Visualization

- **Box Plots:** Best for seeing whiskers and points beyond them.
- **Scatter Plots:** Best for seeing multivariate outliers.

3. Feature Engineering Techniques in EDA

- Feature engineering is the process of using domain knowledge to extract features from raw data that make machine learning algorithms work.

Technique	**Description**	**Example**
Handling Missing Values	Imputation (Mean/Median/Mode) or Dropping.	Filling missing 'Age' with Median.
Encoding	Converting Categorical to Numerical.	One-Hot Encoding (Red, Green $\rightarrow$ 1,0,0) or Label Encoding.
Scaling/Normalization	Bringing all features to a similar range.	Min-Max Scaling (0 to 1) or Standardization (Mean=0, Std=1).
Binning	Converting continuous variables into groups.	Age $\rightarrow$ Child, Adult, Senior.
Feature Creation	Combining existing features.	Total_Price = Quantity $\times$ Unit_Price.

EDA Report Generation using Python Notebooks

Manual EDA is time-consuming. Automated tools can generate comprehensive HTML reports instantly.

Pandas Profiling (ydata-profiling): Generates reports including correlations, missing values, and distribution plots.

```
from ydata_profiling import ProfileReport
profile = ProfileReport(df, title="Sales Data EDA Report")
profile.to_file("report.html")
```

Sweetviz: Focuses on visualizing target variables and comparing datasets (e.g., Train vs Test).

Preparing Data for Machine Learning Models:-This is the final stage of EDA, ensuring the data is "model-ready."

1. **Feature Selection:** Dropping irrelevant columns (e.g., PassengerID, Names) using df.drop().
2. **Handling Categorical Data:** Applying pd.get_dummies() or LabelEncoder.
3. **Data Splitting:** Using Scikit-Learn to separate features (X) and target (y), followed by a Train-Test split.

```
from sklearn.model_selection import train_test_split
  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

4. **Addressing Class Imbalance:**

If one class (e.g., 'Fraud') is much smaller than the other, techniques like SMOTE (Oversampling) are used.

Lecturer Note:

When teaching this unit, emphasize that

Data Cleaning

And
EDA
usually take up
70-80%
of a Data Scientist's time. The quality of the Machine Learning model is directly proportional to the quality of the EDA performed.

Would you like me to provide a specific Python code template for the Titanic dataset EDA to include in your lecture materials?