Trainer

Andrés Felipe Rojas Parra

- Electronics Engineer with over 23 years of experience
- PGP Artificial Intelligence & Machine Learning at Texas University – Austin, TX
- Texas Executive Education Program – McCombs School of Business – Decision Science and AI Immersion Program
- Masters in Artificial Intelligence and Big Data – IEBSchool
- PGP Student in Cloud Computing at Texas University – Austin, TX
- PGP Student in MLOps at Duke University

@arojaspa

https://www.linkedin.com/in/arojaspa/

+573005906373 / +573052038020

Agenda

1. Introduction to Jupyter Notebook
2. Navigation
3. Working with Cells
4. Basic Python in Jupyter Notebook
5. Markdown and Rich Text
6. Data Science with Jupyter Notebook
7. Advanced Features
8. Best Practices
9. Practical Exercises - Students
10. Questions

Introduction to Jupyter Notebook

Introduction to Jupyter Notebook

Jupyter Notebook is an interactive web application that allows you to create and share documents containing live code, equations, visualizations, and narrative text. It is a powerful and flexible tool widely used in data science, education, and research. Below is an introduction to its main features and usage.

What is the Jupyter Notebook?

Jupyter Notebook is part of the Jupyter Project, which includes a variety of tools for creating and presenting interactive code and data analysis. Jupyter Notebooks enable seamless integration of code with results, visualizations, and explanatory text, facilitating the research process and presenting results effectively.

Introduction to Jupyter Notebook

Installation

To install Jupyter Notebook, you first need to install Python and pip on your system. You can install Jupyter using the following command:

pip install notebook Jupyter jupyterlab

Once installed, you can launch Jupyter Notebook with the following command:

jupyter notebook

This will open a new tab in your web browser where you can create and manage your notebooks.

Introduction to Jupyter Notebook

Jupyter Notebook Interface

The Jupyter Notebook interface is divided into several key parts:

- Menu: Provides access to core functions, such as creating new notebooks, opening existing files, running cells, and exporting notebooks.
- Toolbar: Contains buttons for running cells, stopping execution, restarting the kernel, and other quick actions.
- Cells: Notebooks are composed of cells. There are two main types of cells:
- Code cells: Where code is written and executed.
- Text cells (Markdown): Where you can write formatted text using Markdown, a lightweight markup language.

Introduction to Jupyter Notebook

Advantages and Common Uses

- *Research and Development:* Ideal for data scientists and developers who need to perform exploratory data analysis and share their results.
- *Education:* A valuable tool for teaching programming and data analysis concepts, allowing students to interact with the code and view results in real-time.
- *Documentation and Presentation:* This feature enables you to create comprehensive documents that incorporate code, results, and explanatory text, facilitating the effective presentation of reports and projects.

Navigation

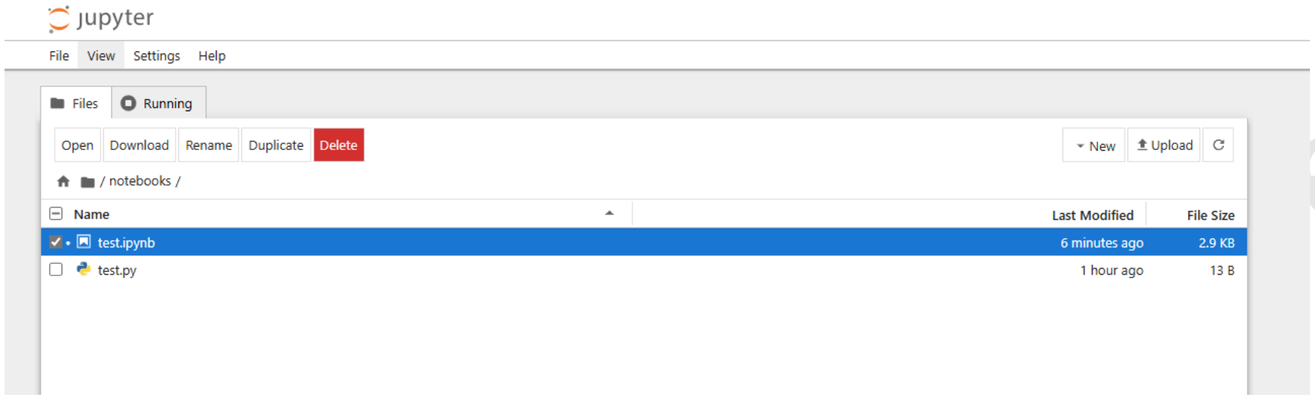
Navigation

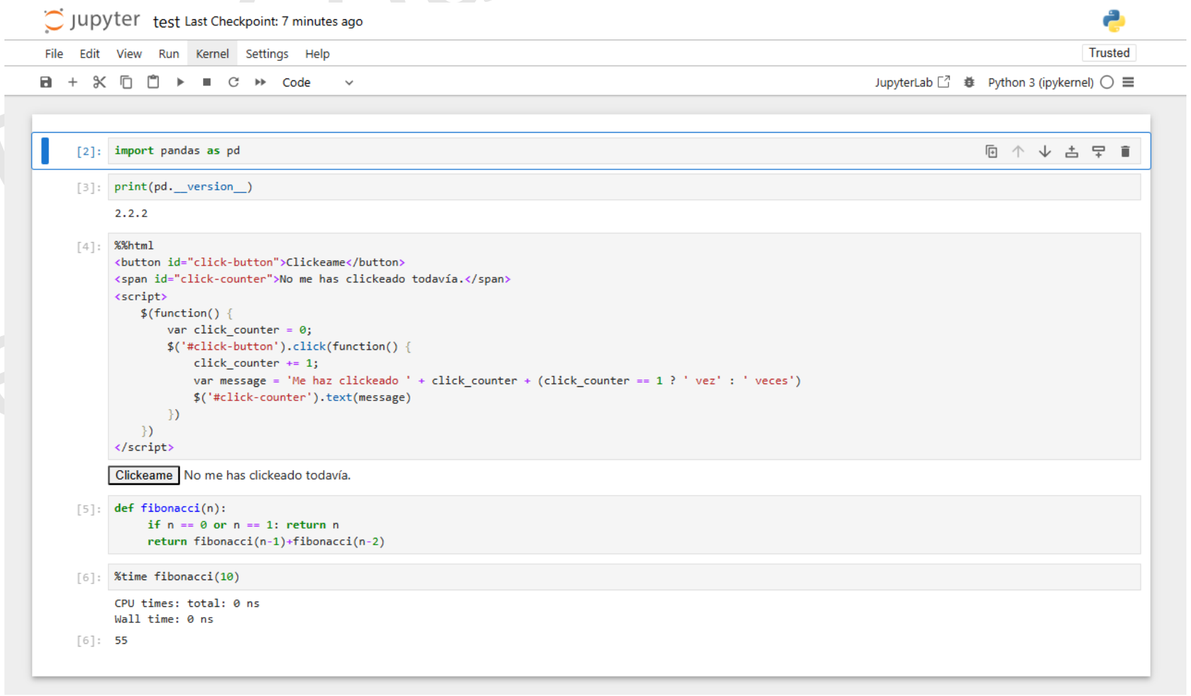
Start and stop Jupyter Notebook:

jupyter notebook --notebook-dir="working directory path"

Navigation

- Jupyter Notebook Interface
- Create and Save Notebooks





Working with Cells

Working with Cells

- Cell Types: Code, Markdown
- Adding, Deleting, and Moving Cells
- Running Cells

Basic Python in Jupyter Notebook

Basic Python in Jupyter Notebook

Using Jupyter Notebooks

- Notebooks have two types of cells:
  - Code
  - Text.
- Code cells are cells where you can run Python, R, and other code.
- Each cell can be run individually and as many times as needed.
- To run a cell, press Ctrl + Enter or use the Run->Run Selected Cell menu, or press the Play button.

Basic Python in Jupyter Notebook

- Variables in Python don't need to be declared; they are defined upon first use. They can be changed by redefining them.
- Python has the same basic data types as other languages: integers, floats, strings, and booleans.
- Lists are a predefined type in the language.
- Tuples are lists that cannot be modified. They are read-only lists.
- Loops and control structures (if, for, while)
- Dictionaries: used to store data values in (key : value) pairs
- Functions

Markdown and Rich Text

Markdown and Rich Text

- **Text in HTML format**

Heading 1
Heading 2
Normal text with **bold** and *italic*.
- List
- Of
- Items

Formulas: \[E = mc^2 \]

Markdown and Rich Text

- **Equations**

$$ \int_0^\infty \frac{x^3}{e^x-1}\,dx = \frac{\pi^4}{15}$$

\begin{gather*}a_1=b_1+c_1\\a_2=b_2+c_2-d_2+e_2\end{gather*}\begin{align}a_{11}& =b_{11}& a_{12}& =b_{12}\\a_{21}& =b_{21}& a_{22}& =b_{22}+c_{22}\end{align}

Data Science with Jupyter Notebook

Data Science with Jupyter Notebook

1. What is NumPy?

- NumPy stands for Numerical Python.
- It is a fundamental package for data science in Python.
- It provides support for large matrices and multidimensional arrays.

2. Key Features of NumPy

- Efficient storage and manipulation of numerical data.
- Functions for mathematical operations on matrices.
- Support for linear algebra, random number generation, and more.

3. Basic NumPy Operations

- Creating matrices or arrays of data

python Copy code

```python
import numpy as np
a = np.array([1, 2, 3])
b = np.array([[1, 2, 3], [4, 5, 6]])
```

Data Science with Jupyter Notebook

3. Basic NumPy Operations

Operations between matrices or data arrays

```python
print(a + 1)        # Add 1 to each element
print(a * 2)        # Multiply each element by 2
print(a + b)        # Element-wise addition
```

- Manipulating arrays or matrices of data

```python
print(a[0:2])       # Access first two elements
print(b[:, 1])      # Access second column
```

Data Science with Jupyter Notebook

1. What is the pandas' library?

- Pandas is an open-source data analysis and manipulation library for Python.
- It provides data structures such as DataFrames and Series.

2. Creating DataFrames

- Creating a DataFrame from a dictionary:

python Copy code

```python
import pandas as pd
data = {'Name': ['Alice', 'Bob', 'Charlie'], 'Age': [25, 30, 35]}
df = pd.DataFrame(data)
```

- Reading data from a CSV file:

python Copy code

```python
df = pd.read_csv('data.csv')
```

Data Science with Jupyter Notebook

3. Operations between Data Frames

- Data Visualization:

```python
print(df.head())     # View first 5 rows
print(df.describe()) # Summary statistics
```

- Data Filtering:

```python
df_filtered = df[df['Age'] > 30]
```

- Adding/Deleting Columns:

```python
df['Salary'] = [50000, 60000, 70000]  # Adding a new column
df.drop('Salary', axis=1, inplace=True)  # Removing a column
```

Data Science with Jupyter Notebook

1. What is Matplotlib?

- Matplotlib is a plotting library for the Python programming language.
- It provides an object-oriented API for embedding graphics in applications.

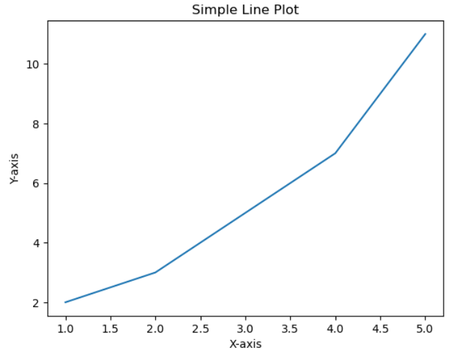
2. Basic Plotting

- Importing Matplotlib:

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

- Creating a line (simple line plot):

```python
x = [1, 2, 3, 4, 5]
y = [2, 3, 5, 7, 11]
plt.plot(x, y)
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Simple Line Plot')
plt.show()
```

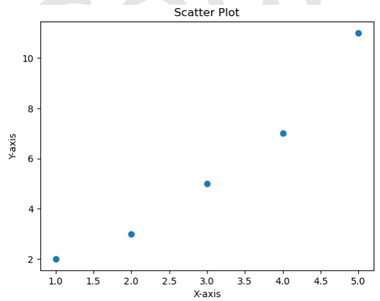


Data Science with Jupyter Notebook

3. Different types of plots

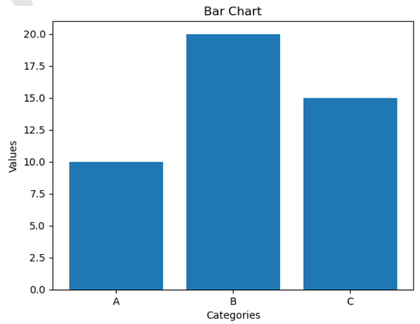
- Scatter Plot:

```python
plt.scatter(x, y)
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Scatter Plot')
plt.show()
```



- Bar Chart:

```python
categories = ['A', 'B', 'C']
values = [10, 20, 15]
plt.bar(categories, values)
plt.xlabel('Categories')
plt.ylabel('Values')
plt.title('Bar Chart')
plt.show()
```

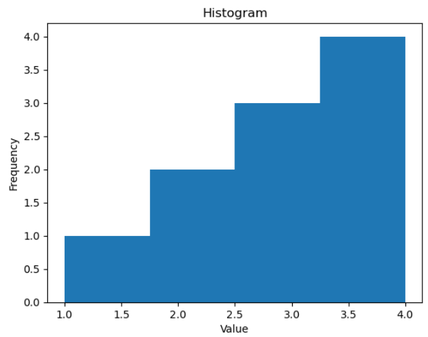


Data Science with Jupyter Notebook

3. Different types of plots

- Histogram:

```python
data = [1, 2, 2, 3, 3, 3, 4, 4, 4, 4]
plt.hist(data, bins=4)
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.title('Histogram')
plt.show()
```



Advanced Features

Advanced Features: Magic

- **Magic**

https://ipython.readthedocs.io/en/stable/interactive/magics.html

```
%lsmagic

%time fibonacci(10)

def fibonacci(n):
    if n == 0 or n == 1: return n
    return fibonacci(n-1)+fibonacci(n-2)
```

Advanced Features: Magic

1. What are magic commands?

- Special commands in Jupyter Notebooks to make various tasks more manageable.
- There are two types: line magic (prefixed with %) and cell magic (prefixed with %%).

2. Single-line magic commands

- %time: Execution time of a single statement: %time sum(range(1000000))
- %matplotlib inline: Display Matplotlib plots inline: %matplotlib inline
- %ls: List files in the current directory: %ls

3. Multi-line magic commands

- %%timeit: Execute a cell multiple times:
- %%timeit
- sum(range(1000000))
- %%writefile: Write the cell contents to a file:
- %%writefile
- hello.py print("Hello, World!")

Best Practices

Best Practices

Organizing notebooks helps maintain clarity and structure, making it easier to navigate and understand the code. Here's a sample structure:

- data/: Contains raw and processed data.
- notebooks/: Contains Jupyter notebooks for different stages of analysis.
- scripts/: Python scripts for data processing and modeling.
- README.md: Project overview, setup, and usage instructions.
- Requirements.txt: File containing all the libraries installed in the virtual environment with their respective versions.

kotlin Copy code

```
project/
├── data/
│   ├── raw/
│   ├── processed/
├── notebooks/
│   ├── exploratory_analysis.ipynb
│   ├── data_preprocessing.ipynb
│   ├── model_training.ipynb
├── scripts/
│   ├── data_processing.py
│   ├── model.py
├── README.md
└── requirements.txt
```

Best Practices

Inside a Notebook (Header):

Specify the project name, author, and date for easy identification and attribution.

yaml Copy code

```yaml
Project Name: Machine Learning Pipeline
Author: Your Name
Date: July 11, 2024
```

Sections and Markdown Cells:

- Use Markdown cells for section headings (## Exploratory Data Analysis) and to describe the purpose and findings of each section.
- Include a table of contents for quick navigation ([TOC]).

Best Practices

Documenting your code

Documenting your code within Jupyter Notebooks improves readability and comprehension. Use Markdown cells effectively:

markdown Copy code

```markdown
## Data Preprocessing

This section preprocesses the raw data before modeling.

### Steps:

1. **Load Data**: Read data from CSV files.
2. **Clean Data**: Handle missing values and outliers.
3. **Normalize Data**: Scale numerical features.

### Parameters:

- `data_path`: Path to the raw data folder.
- `threshold`: Threshold value for outlier detection.
```

Best Practices

Comments in the execution cell:

python Copy code

```python
# Load data
data = pd.read_csv(data_path)

# Clean data
data.dropna(inplace=True)

# Normalize data
data[num_cols] = (data[num_cols] - data[num_cols].mean()) / data[num_cols].std()
```

Best Practices

Version Control with Git

Using version control helps you track changes and collaborate efficiently:

- Initializing the repository: get init
- Committing Notebooks: git add notebooks/data_preprocessing.ipynb, git commit -m "Added data preprocessing notebook."
- Collaborating and Branching: git checkout -b feature/new-feature
- Pushing Changes: git push origin feature/new-feature
- Merging Changes: git checkout main, git merge feature/new-feature

Andrés Felipe Rojas Parra
AI
arojaspa@outlook.com

Thank you