Sagol School of Neuroscience
Tel Aviv University

Creating FAIR Python Code in Jupyter Notebook

Introduction: What is FAIR?

FAIR is an acronym that defines best practices for organizing and sharing data, code, and analyses:

- **Findable**: Clearly labeled and documented.
- **Accessible**: Easy to retrieve and use by others.
- **Interoperable**: Compatible across different platforms and tools.
- **Reusable**: Clear and well-structured for future use and adaptation.

Using Jupyter Notebook to Be FAIR

We will use **Jupyter Notebook**, a widely-used environment that allows you to combine code, output, and text in a single, readable document. It supports literate programming and makes your work easier to follow, revise, and reproduce. This data file was constructed using jupyter notebook.

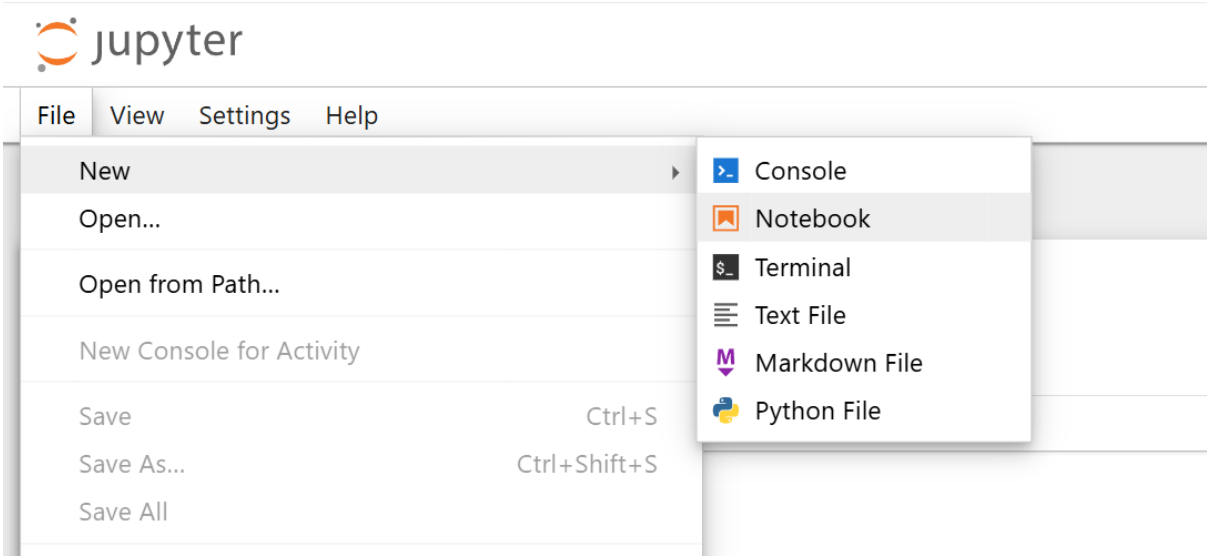
---

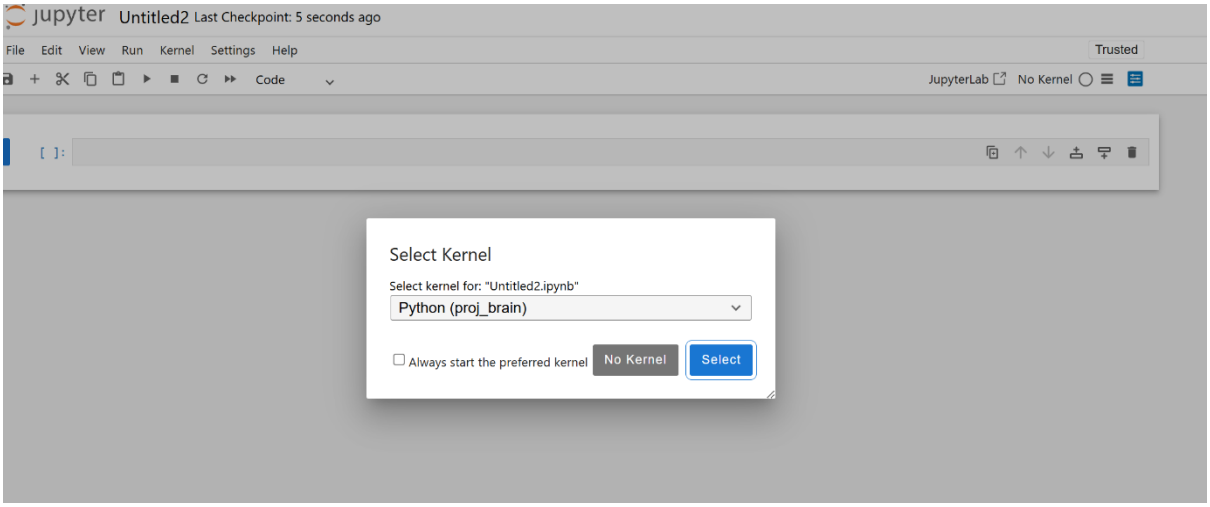
1. Set Up Your Jupyter Notebook

This guide assumes you have a python platform installed. If so, make sure to pip install Jupyter.

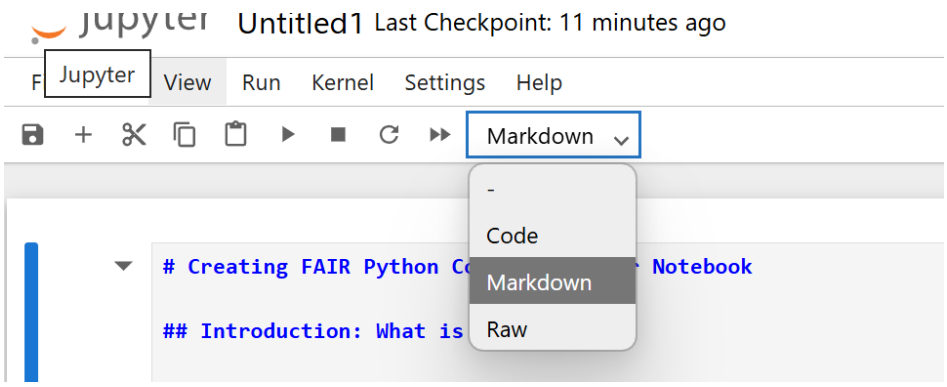
To begin:

1. Open your preferred platform (e.g., JupyterLab, Anaconda Navigator, or Jupyter in VS Code).
2. Create a new notebook and select **Python** as the kernel.





3. Rename your notebook file (by clicking on the title) to reflect the content and purpose of your analysis.
4. At the top of the notebook, insert a **Markdown cell** that includes:
    - A descriptive **title**
    - The **author(s)** of the notebook
    - The **date** and a brief description of the purpose of the notebook
5. **Address package and environment versions.** We recommend adding a short "Environment & Dependencies" section at the top or bottom of the notebook, including a **requirements.txt** or **environment.yml** file, and/or a code cell that prints the versions of core libraries used (e.g. pandas, numpy, matplotlib, seaborn, scikit learn).



2. Notebook Structure: Markdown and Code Cells

A Jupyter Notebook is composed of two primary types of cells:

Markdown Cells (Text/White Cells)

These cells are used to explain your code, structure your analysis, and provide context. Use Markdown syntax to format your text:

Syntax	**Output Description**
#	Main title (Level 1 header)
##	Subtitle (Level 2 header)
###	Sub-subtitle (Level 3 header)
italic	*italic*
bold	**bold**
- or *	Bullet points
1.	Numbered lists
`code`	Inline code

Use these cells generously to clarify the purpose of each code block, summarize results, and guide the reader.

Code Cells (Gray Cells)

These are used to execute Python code. Each code cell is independent, and you can run them individually or sequentially. Below are common conventions:

- Begin by importing necessary libraries:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Use comments (#) to describe each step within a code cell.

Keep code clean, well-indented, and consistently formatted.

3. Start Writing Code

Before beginning your analysis, follow these best practices:

- **Comment your code**: Use inline comments to explain key steps.
- **Use descriptive names**: For variables, datasets, and functions (for example, use anxiety_df over df2 for data frame that include anxiety questionnaire results).
- **Keep formatting consistent**: Ensure uniform indentation and spacing.

Example:

```
# Load Excel file
df = pd.read_excel('data.xlsx')

# Display the first five rows
df.head()
```

Explain the above with a Markdown cell, describing what the code does and why.

4. Stats and graphs

Use statistical summaries and visualizations to explore and present your data. It is best to use commonly used libraries such as seaborn and matplotlib:

```
# Descriptive statistics
df.describe()

# Histogram of a variable
sns.histplot(df['column_name'])
plt.title('Distribution of Column Name')
plt.show()
```

Immediately follow visualizations with Markdown cells that interpret the results.

5. Create output file

Go to File → save and export notebook as:

HTML (.html)

PDF (.pdf) — requires LaTeX

Notebook (.ipynb) — ideal for reproducibility