From Prototyping to Production: Converting Jupyter Notebooks to Python Scripts

A Guide for Efficient and Maintainable Code

Michael Borck

Table of contents

# Objectives

* Understand the strengths of Jupyter Notebooks for prototyping and analysis.
* Learn why and how to transition to Python scripts for production.
* Introduce a step-by-step process for conversion.
* Discuss a practical example to illustrate the process.

# Strengths of Jupyter Notebooks

* Interactive and user-friendly environment.
* Ideal for iterative development and rapid prototyping.
* Supports rich media outputs (graphs, charts, images).
* Easy to document code with markdown cells.

# Limitations of Jupyter Notebooks

* Difficult to manage large codebases.
* Limited support for version control and collaboration.
* Harder to modularize and reuse code.
* Performance issues with large-scale applications.

# Strengths of Python Scripts

* Better organization and modularization of code.
* Improved performance and efficiency.
* Easier integration with other systems and tools.
* Enhanced support for version control and collaboration.

# Python Workflows

* No one-size-fits-all solution; depends on project requirements, team size, data complexity, collaboration needs, and development speed.
* Use notebooks for exploratory data analysis and prototyping.
* Use Python scripts for long-term development and production code.
* Hybrid approaches combine benefits of both.

# Hybrid Approach: Notebook-Based Workflows

* Use notebooks for exploratory data analysis and prototyping.
* Refactor the code into Python scripts for production.
* Maintain the interactive and flexible nature of notebooks for initial development.
* Ensure the code is organised and efficient for long-term maintenance.

# Prototyping in Jupyter Notebooks

**Why Use Jupyter Notebooks?**

* Interactive coding environment
* Easy to test and debug code
* Great for data analysis and visualisation

# Refactoring Notebooks into Python Scripts

**Why Refactor?**

* Organise and structure your code
* Make it reusable and modular
* Prepare for deployment and sharing

# Example: Refactoring a Notebook

**Before: Jupyter Notebook**

# notebook.ipynb  
import pandas as pd  
  
data = pd.read\_csv('data.csv')  
result = data.describe()  
print(result)

**After: Python Script**

# analysis.py  
import pandas as pd  
  
def analyse\_data(file\_path):  
 data = pd.read\_csv(file\_path)  
 result = data.describe()  
 return result  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 result = analyse\_data('data.csv')  
 print(result)

# Using GitHub to Share Your Project

**Why Use GitHub?**

* Version control with Git
* Share code with the world
* Collaborate on projects

# Setting Up a GitHub Repository

1. Create a GitHub account
2. Create a new repository
3. Clone the repository to your local machine
4. Add your project files
5. Commit and push your changes

# Example: Git Commands

**Initialise and Push to GitHub**

# Initialise git in your project directory  
git init  
  
# Add your files to the repository  
git add .  
  
# Commit your changes  
git commit -m "Initial commit"  
  
# Add the remote repository URL  
git remote add origin https://github.com/yourusername/yourrepository.git  
  
# Push your changes to GitHub  
git push -u origin master

# Creating a README.md

**Why Include a README.md?**

* Provide an overview of your project
* Explain how to install and use it
* Highlight key features and dependencies

# Example: README.md

# Project Title  
  
 ## Overview  
 Brief description of your project.  
  
 ## Installation  
 ```bash  
 pip install your\_project  
 ```  
  
 ## Usage  
 ```python  
 from your\_project import your\_function  
 result = your\_function()  
 print(result)  
 ```  
  
 ## Features  
 - Feature 1  
 - Feature 2  
  
 ## License  
 MIT

# Summary

* Prototype in Jupyter Notebooks
* Refactor into Python scripts
* Share your project on GitHub