Chapter 5: Jupyter Notebooks

# Objectives

* Understand iPython development patterns
* Get an overview of Jupyter Notebooks
* Review where Jupyter Notebooks exist in the ArcGIS ecosystem
* Build a Notebook effectively showcasing the results of an analytical exercise

# Introduction

Writing a Python script can be a daunting task if you’re not entirely sure of all the steps you’re going to want to do. All the patterns we’ve discussed up to this point involve being very thoughtful about our process and our solution. There are other development patterns out there, though, that will let us write and test our code one step at a time.

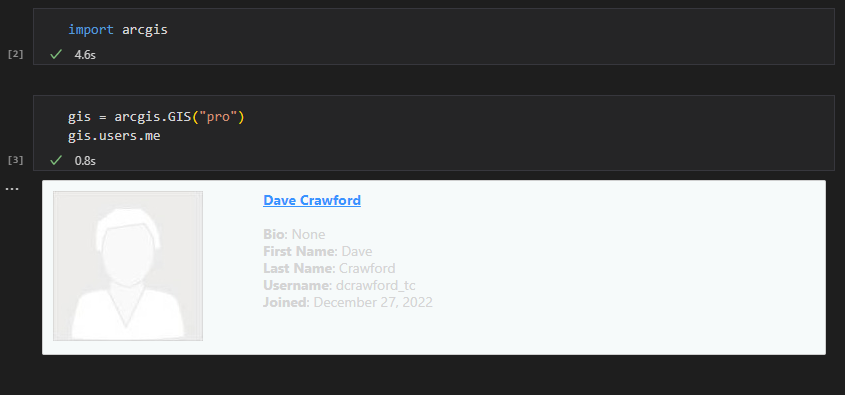
In this chapter, we’ll look at some interactive Python development patterns and how we can use them with specific regard to spatial data and automation. First, we’ll introduce some basic concepts with interactive terminals. Then we’ll dive into Jupyter Notebooks as a development environment and documentation tool. Finally, we’ll explore how Jupyter Notebooks fit into the ArcGIS ecosystem and write a polished notebook to showcase the results of our analysis.

# Interactive Terminals

Depending on which development environment you work in, you may see some variation of the phrase “Interactive Terminal” or “Interactive Window” in your options. Using an interactive terminal is a bit different than writing your script and running the whole thing. An interactive terminal allows you to write and execute smaller blocks of code and interrogate the results.

Many interactive development environments (IDEs) incoroporate an interactive iPython terminal. iPython is an open-source interactive shell that allows the user to have a more back-and-forth writing experience. In a practical sense, it allows you to run blocks of code and gives you back the results. The blocks of code in this case are called “cells”.

Interactive iPython terminals were designed with data scientists in mind and have some handy tools that we’ll explore in our exercise. One of the big features that comes with iPython is a concept called “magic”. iPython has plenty of built-in magics that can apply to either a single line or an entire cell. You can also write your own magic.



<insert 05-01>

<alt>An example of an interactive window in Visual Studio Code.</alt>

Figure 5-1. Visual Studio Code’s interactive window executing code and displaying results.

# Jupyter Notebooks

So now that we’ve got our heads wrapped around iPython, there’s one more thing we need to cover in this chapter before we get our hands dirty and start writing some code. We’re going to be using Jupyter Notebooks in our exercises. Jupyter Noteboks are special open-source development environment where a Python kernel is running behind the scenes and all the interaction takes place in a web browser. Because all the interactivity is built into a web browser, Jupyter Notebooks can have lots of great interactive widgets and in-line documentation. Jupyter Notebooks are excellent environments for early-stage script development or exploratory data analysis.

Similarly to iPython interactive terminals, Jupyer Notebooks consist of series of cells. Each cell consists of a user input and (depending on what you write) an output returned from the Python kernel.

# Tutorial 1-1: Open up a Jupyter Notebook

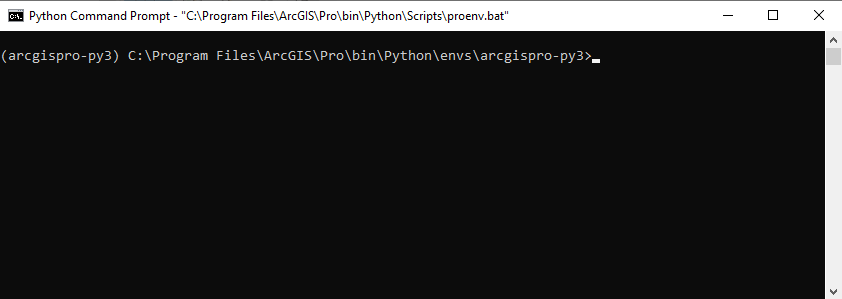
Now we’re almost ready to get writing. We’ll see later in the chapter that there are easier ways to open up a Jupyter Notebook, but it’s worth learning how to do it the hard way. We’ll learn a little bit about creating a Notebook from the command line.

## Start a Jupyter Notebook from the Command Prompt

The Python Command Prompt is a tool that comes with ArcGIS Pro, but you could do this same thing with any command prompt that can access an Anaconda environment that has Jupyter Notebooks installed.

1. Open the “Python Command Prompt” from your Start Menu

When you open the Python Command Prompt, you’ll see a command prompt with two components. On the left, in parentheses, is your current Anaconda environment. In the case of the screenshot below, the environment is ArcGIS Pro’s default environment “arcgispro-py3”. To the right of the environment is a folder path. This folder path is a location on your computer, probably the default location of the ArcGIS Pro environment.



<insert 05-02>

<alt>The Python Command Prompt showing the environment and local path</alt>

1. Change the directory of the command prompt

This is something people often forget to do before they start Jupyter Notebooks. It’s a good idea to change our directory here because the file navigation within the Jupyter Notebook interface is limited. Let’s navigate to the path for Chapter 5 in the sample data you downloaded. In my case, the path is C:\Top20Python\Chapter05. We’ll use a handy command called “cd” which stands for “Change Directory”. Run the following code in the command prompt (using your folder path).

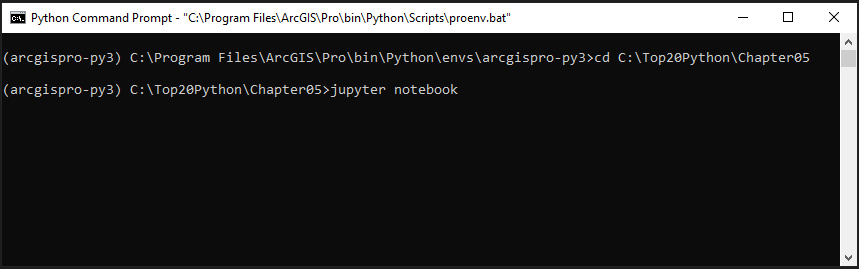
cd C:\Top20Python\Chapter05

You should notice now that the directory or path of the command prompt has changed. This is beneficial going forward because now we can open Jupyter Notebooks in this directory.

1. Open Jupyter Notebooks

This is going to be easy. Once you’ve changed the directory of your prompt to the folder you want to open Jupyter Notebooks in, you can just use the following line of code to open Jupyter Notebooks.

jupyter notebook



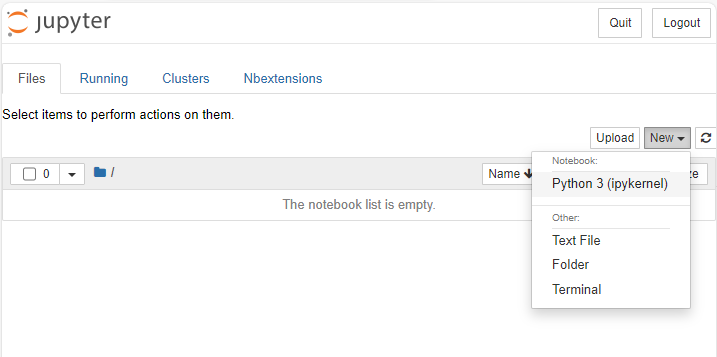
<insert 05-03>

<alt>Python Command Prompt showing the first three steps opening Jupyter Notebooks</alt>

After you run the last line of code, Jupyter Notebooks should start up and open a browser window. If not, check the text that gets displayed in the command prompt. Jupyter should display a URL that you can enter to access the Notebooks interface.

1. Create a new Notebook

In the web browser Jupyter Notebooks interface, find the drop down that says New then choose Python 3 (ipykernel).



<insert 05-04>

<alt>Jupyter Notebooks interface highlighting the new notebook dialogue.</alt>

A new tab will open in your browser that has a Jupyter Notebook.

### Explore some properties of Notebooks

Before we get into some more in-depth code, let’s explore some of the special properties of Notebooks. These little things can be helpful in your future development.

1. Execute the code in a cell

This may be a little different than the development you’re used to if you haven’t used an interactive terminal before. Now that you have a notebook open, you can start writing code. Let’s start by importing a package in the first cell. Type the following code in the first cell

import arcgis

Now you can execute that code by either finding the **Run** button or clicking **Shift+Enter**. While the code is executing, you’ll see an asterix to the left of the cell. Afterward, there will be a number indicating the order in which you executed the code. At this point, there should be some text to the left of your first cell that says “In [1]” indicating that you executed that cell first.

You may have noticed that there is no corresponding “Out” to our input. This is because we imported a package and didn’t return anything. Let’s write some code that actually returns something now. First we’ll create a GIS object. Then we’ll return that object so that we get a display.

gis = arcgis.GIS(“pro”)

gis.users.me

This should return a nicely formatted description of the account you’re currently using in ArcGIS Pro (similar to Figure 05-01).

1. Explore the doc-strings and source code for a function

This is one of the handiest things that Jupyter Notebooks allows you to do. If you’ve worked with Python packages before, you may have had to do a lot of consulting of package documentation. In a Notebook, you can directly reference the documentation in the code of a package as you’re writing. Let’s explore by writing the code

arcgis.GIS?

When you execute this code, Jupyter will bring up the documentation string (doc-string) for the GIS class in a window in the bottom of your browser tab.

On your own.

If you run the code from Step 6 but add an additional question mark, Jupyter will return all the source code for the GIS object.

# Notebooks in the ArcGIS Ecosystem

There are several places where you can use Notebooks in Esri software. In this section, we’ll explore a few and highlight the benefits of each.

## Notebooks in ArcGIS Pro

Notebooks were incorporated into ArcGIS Pro at version 2.5. Since then, they’ve integrated with a lot of the familiar components you may already be working with. You can create a new notebook in ArcGIS Pro by clicking the **New Notebook** button on the **Insert** toolbar of the ribbon at the top of the user interface. You can also create and them through the **Catalog View or Pane**.

The Notebook environment in ArcGIS Pro has some special features. Your Notebooks can interact with maps and reference layers by name. This environment also has access to the **arcpy** package, making it ideal for developing geoprocessing workflows.

## Notebooks in ArcGIS Online

This is one of the most interesting places to use Notebooks in ArcGIS. Because all the infrastructure, including the Python kernels that power ArcGIS Online Notebooks are cloud-hosted, you don’t have to worry about setting up an environment on your machine. As a Python developer, it’s much easier. You click a button and it starts a Notebook.

# Tutorial 5-2: Creating a Map in a Notebook

Let’s go back to our GeoNinjas Pythonanalytics job. Today, we’ve been tasked with some spatial analysis regarding public works data in San Francisco. We’ve been given a CSV file of all the 311 calls for the month of November 2023. A data scientist on the team noticed that this data has Latitude and Longitude columns and is wondering if we can display the data spatially for them.

Let’s start by importing the data into our Notebook.

1. Create a new Notebook

Use the steps from our first exercise to create a new notebook in the exercise folder for this chapter. Open the Python Command Prompt, navigate to the folder for this chapter, and create a new Notebook.

1. Import the 311 data from the included csv

We’re going to use a package called **pandas** to read our csv. If you’re not familiar with pandas or DataFrames in general, don’t worry too much. We’ll go into more detail on DataFrames in another chapter. In our case, we’re going to use pandas to read our csv with the intent to create a spatially enabled DataFrame.

Let’s start by importing the pandas package.

import pandas

Now we can use the pandas package to read our csv. If your Notebook isn’t in the same folder as your csv, you may want to use the full path to your csv.

df\_311 = pandas.read\_csv(‘./311\_Cases.csv’)

The dot notation that you see in my file path is a **relative path**. This is a handy way to use file paths that aren’t specific to any one machine. The dot at the beginning of the path means that the path starts in the folder that my Notebook is in. If the csv is in the same folder as the Notebook you created, this format should work just fine.

Now let’s take a look at the DataFrame we just read. There are three things we can do to get a good idea of what our data looks like. The first thing is we can find out how many records and columns there are by using this line of code.

df\_311.shape

Next, we can look at a description of the columns and what kinds of data are in our DataFrame using the following code.

df\_311.dtypes

Finally, we can take a look at the first five rows of the DataFrame to get an idea of what our data really looks like using the following code

df\_311.head()

Now we’re ready to create a spatially enabled DataFrame.

1. Create a spatially enabled DataFrame

The ArcGIS API for Python allows us to create a spatially enabled DataFrame that we can use for analysis and add to maps.

First we’ll need to import the ArcGIS API for Python

import arcgis

Next we’ll use the spatial accessor to turn our DataFrame into a spatially enabled DataFrame.

# df\_311 = pandas.DataFrame.spatial.from\_xy(df\_311,

# x\_column = 'Longitude',

# y\_column = 'Latitude')

Now if you look at the DataFrame, you’ll notice a new column called “SHAPE”. This contains geometry that can be used for analysis and display.

1. Add the DataFrame to a map view in your Notebook

When we imported the ArcGIS API for Python in our notebook, we also imported some extra functionality into our Notebook. One of the extra pieces of functionality we imported was a “Map View” widget. This will let us create a map inside our notebook and add our data to it.

Let’s start by creating a map view. We’ll need to create a GIS object first. Then we can use that GIS object to create a map centered on our area of interest.

gis = arcgis.GIS()

map\_view = gis.map(“San Francisco, CA”)

Now that we’ve instantiated a map view, let’s add our layer. We’re going to use a Unique Value symbology type based on the “Category” field.

df\_311.spatial.plot(map\_widget = map\_view,

renderer\_type = 'u',

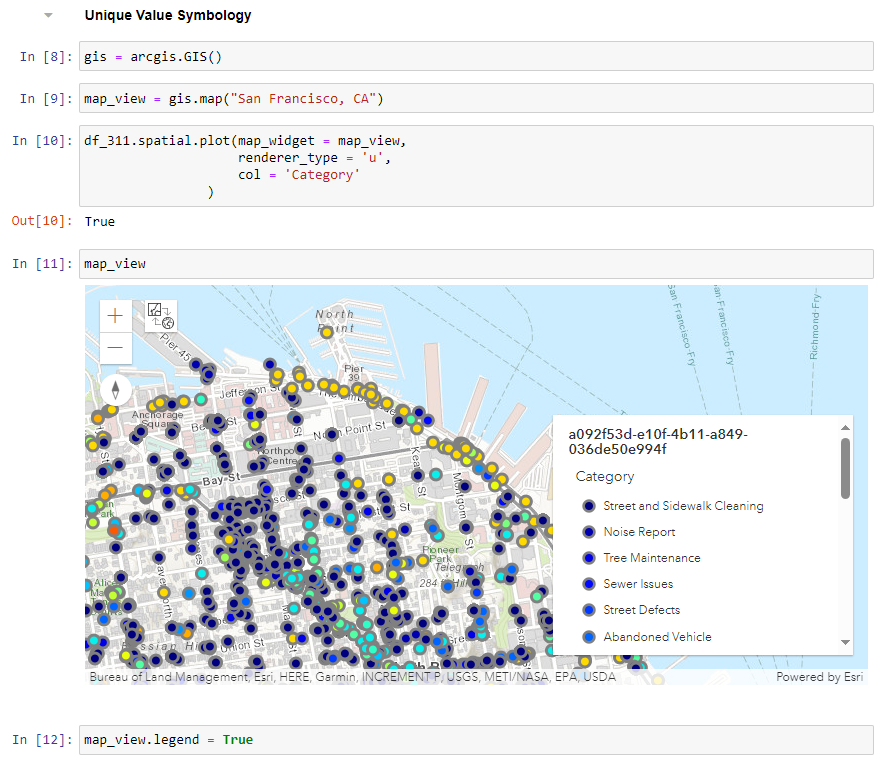
col = 'Category'

)

Now we’ve got our layer added to our map view, but we aren’t seeing a map. We’ll need to return the map view to see and interact with it. Executing the following code will return the map view.

map\_view

Now you can share your results with your coworkers outside the geospatial analytics team.



<insert 05-05>

<alt>ArcGIS Notebook showing code and a map view</alt>

On your own

Try some map configuration options. You can change the basemap or the scale. You can also try some additional symbology options. Setting the renderer\_type to “h”, for instance, will render a heat map.

# Summary

In this chapter, we learned about interactive Python development patterns. We learned how to use Jupyter Notebooks from the command line, write code in Notebooks, and create a map to visualize our data.

## Exercise workflow

1. Open the “Python Command Prompt”
2. Change the directory of the command prompt
3. Open Jupyter Notebooks
4. Create a New Notebook
5. Import 311 data from a csv
6. Create a spatially enabled DataFrame
7. Display the 311 data on a map view in a Notebook

**Information at your fingertips**

**Resources**

# User Story

References

Fernando Pérez, Brian E. Granger, *IPython: A System for Interactive Scientific Computing*, Computing in Science and Engineering, vol. 9, no. 3, pp. 21-29, May/June 2007, doi:10.1109/MCSE.2007.53. URL: https://ipython.org