

# Data Analysis Using Python

Samatrix Consulting Pvt Ltd

# Python Installation

# Python Installation

- Since everyone uses Python for different applications, there is no single solution for setting up Python and required add-on packages.
- We have provided detailed instructions to get set up.
- We recommend using the free Anaconda distribution.

# What is Anaconda?

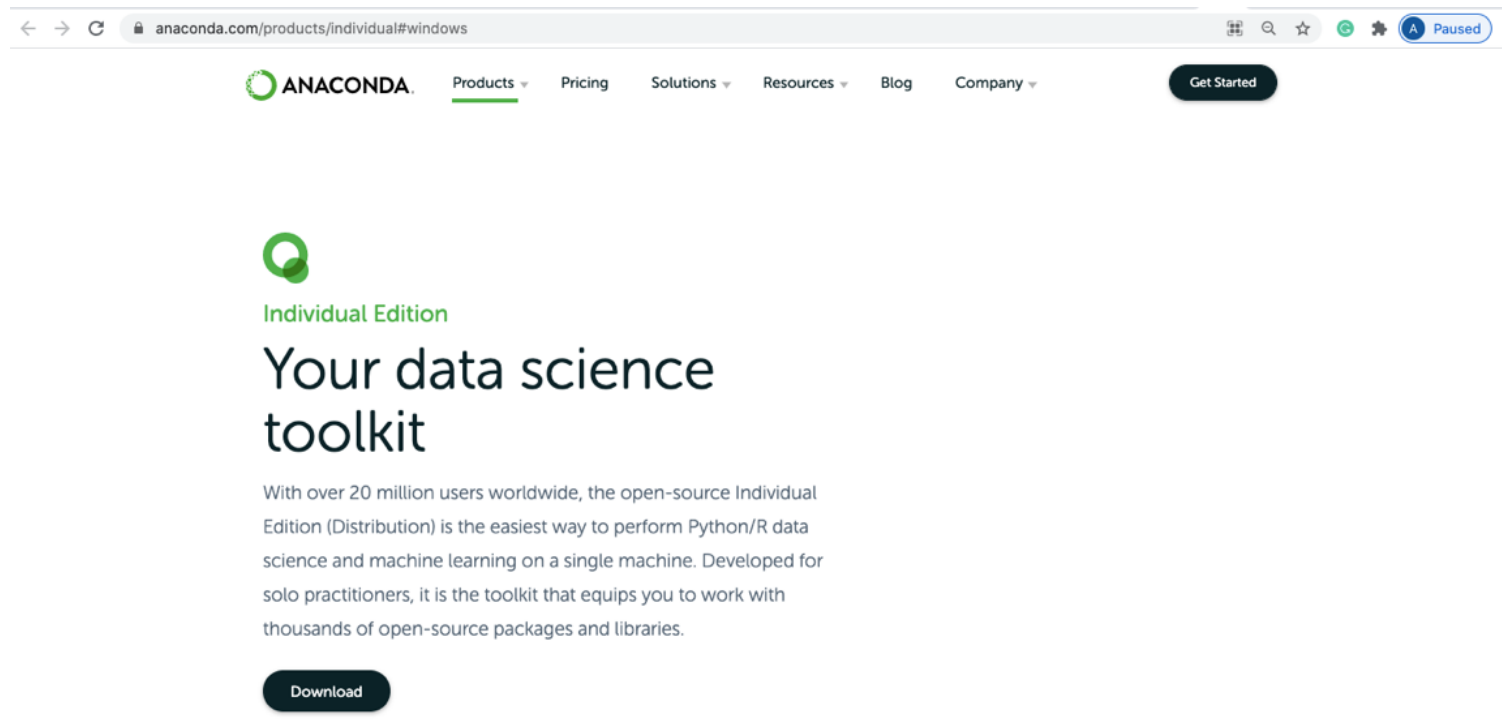
- Anaconda is a free and open-source distribution of Python and R programming languages for data science and machine learning.
- It is an easy-to-install collection of high-performance Python libraries along with Conda
- Anaconda helps you simplify your Python deployment and later on your package management.
- Anaconda comes with 1500 packages (including the package management system conda) and a GUI named Anaconda Navigator.
- The Anaconda Navigator also install some applications by default such as Jupyter Notebook, Spyder IDE and Rstudio (for R).

# Why do we need Anaconda Installation?

- Several scientific packages need a specific version of Python.
- It is challenging to keep them up-to-date and prevent from breaking.
- The Anaconda Distribution helps in management of multiple Python versions on one computer and provides a large collection of highly optimized, commonly used data science libraries to get you started faster.

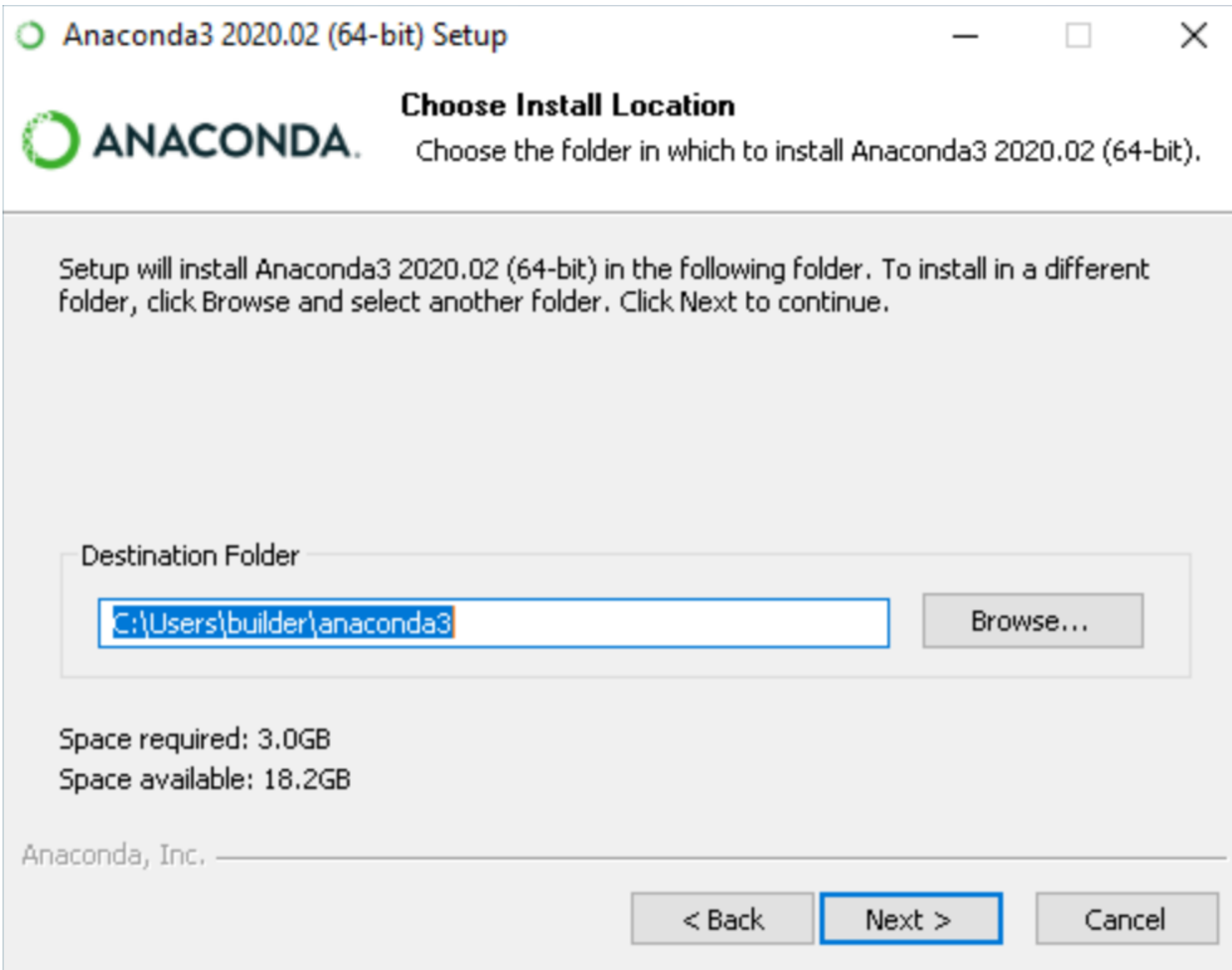
# Installing Anaconda on Windows

- Download the Anaconda installer from the following link  
<https://www.anaconda.com/products/individual#windows>



# Installing Anaconda on Windows

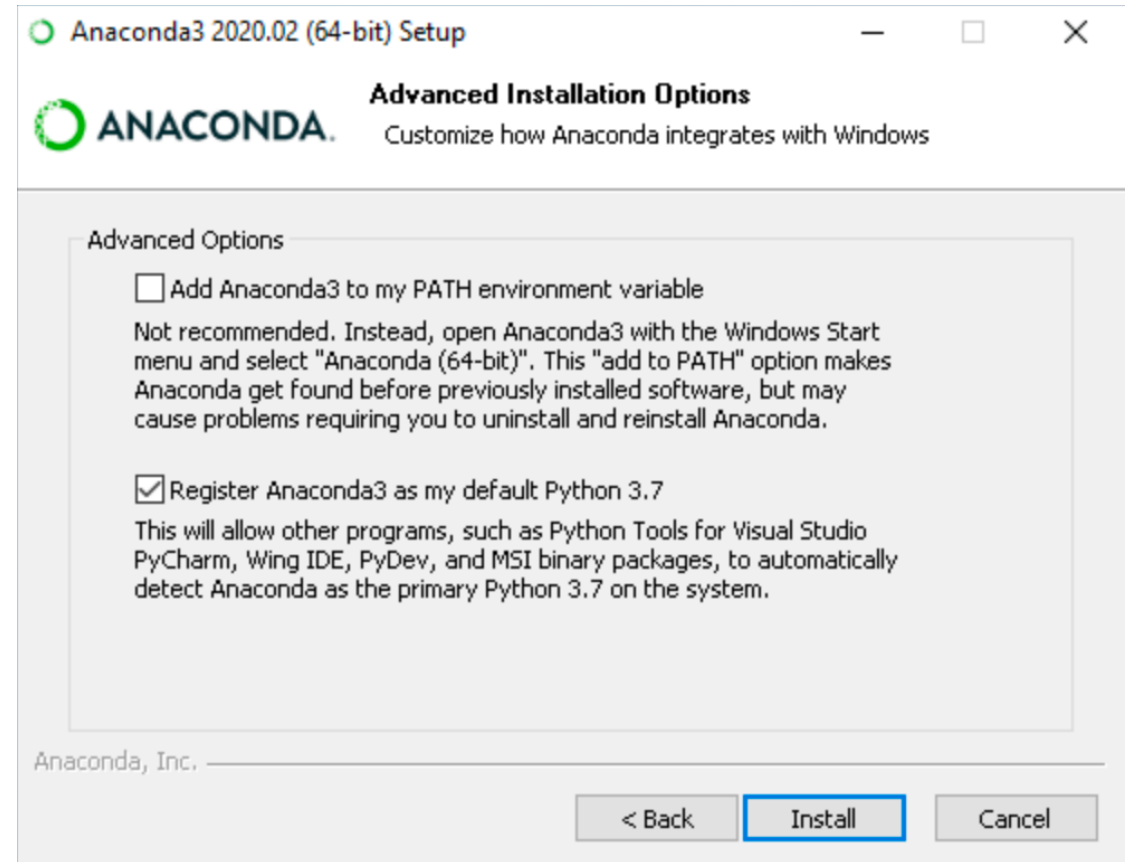
- Double click the installer to launch.
- Note If you encounter issues during installation, temporarily disable your anti-virus software during install, then re-enable it after the installation concludes. If you installed for all users, uninstall Anaconda and re-install it for your user only and try again.
- Click Next.
- Read the licensing terms and click “I Agree”.
- Select an install for “Just Me” unless you’re installing for all users (which requires Windows Administrator privileges) and click Next.
- Select a destination folder to install Anaconda and click the Next button.





# Installing Anaconda on Windows

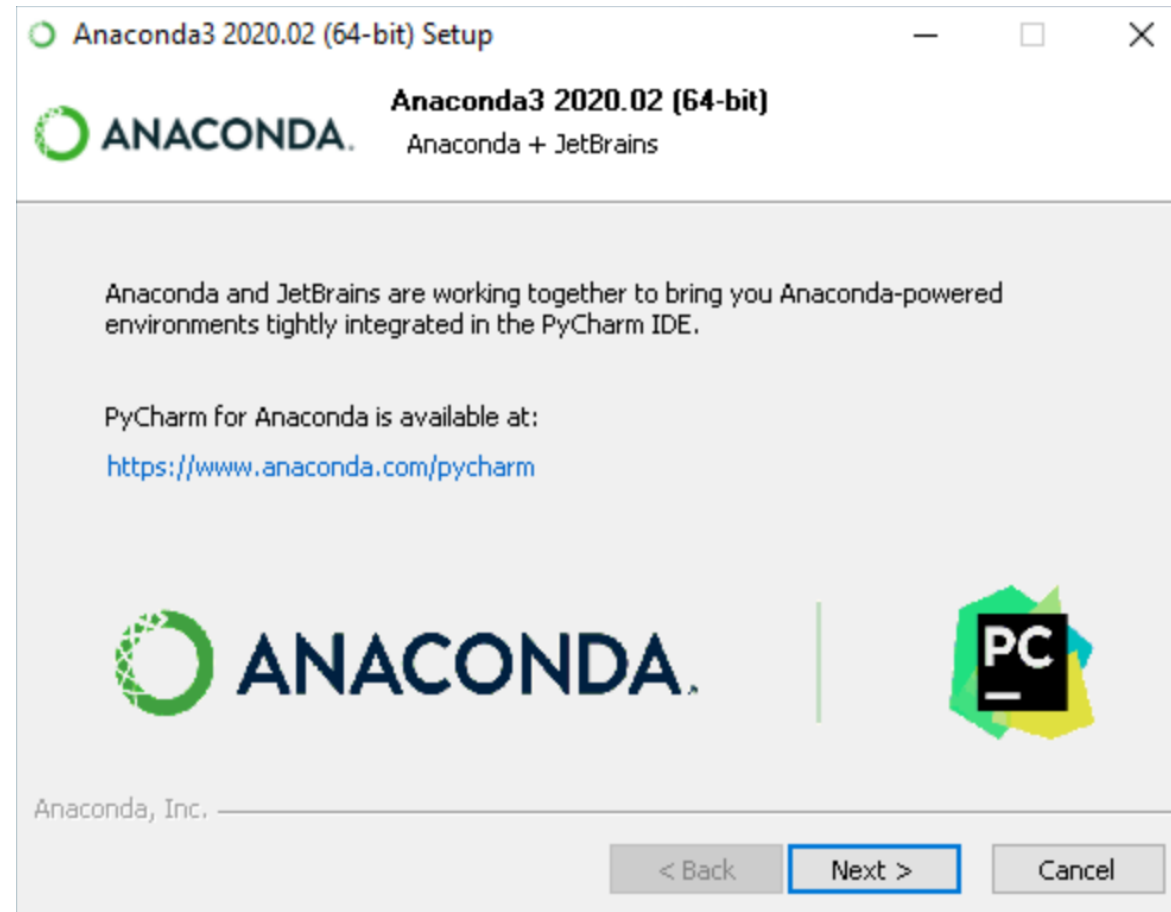
- Choose whether to add Anaconda to your PATH environment variable. We recommend not adding Anaconda to the PATH environment variable, since this can interfere with other software. Instead, use Anaconda software by opening Anaconda Navigator or the Anaconda Prompt from the Start Menu.



# Installing Anaconda on Windows

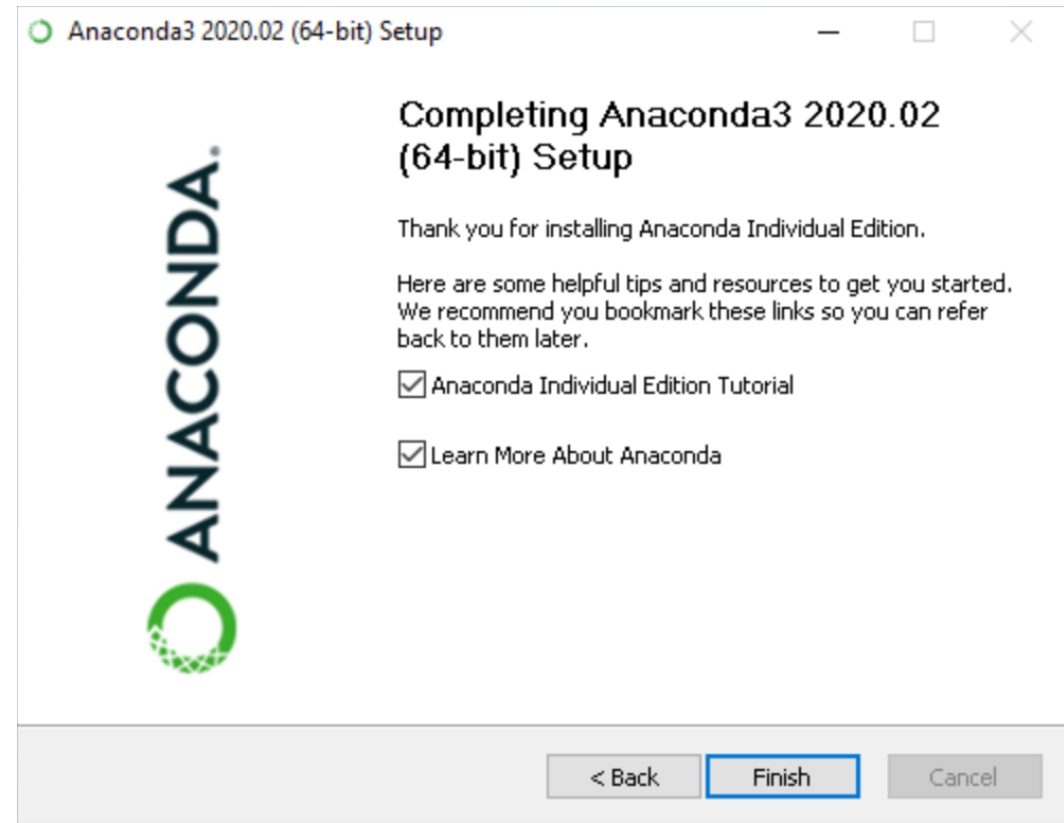
- Choose whether to register Anaconda as your default Python. Unless you plan on installing and running multiple versions of Anaconda or multiple versions of Python, accept the default and leave this box checked.
- Click the Install button. If you want to watch the packages Anaconda is installing, click Show Details.
- Click the Next button.
- Optional: To install PyCharm for Anaconda, click on the link to <https://www.anaconda.com/pycharm>.
- Or to install Anaconda without PyCharm, click the Next button.

# Installing Anaconda on Windows



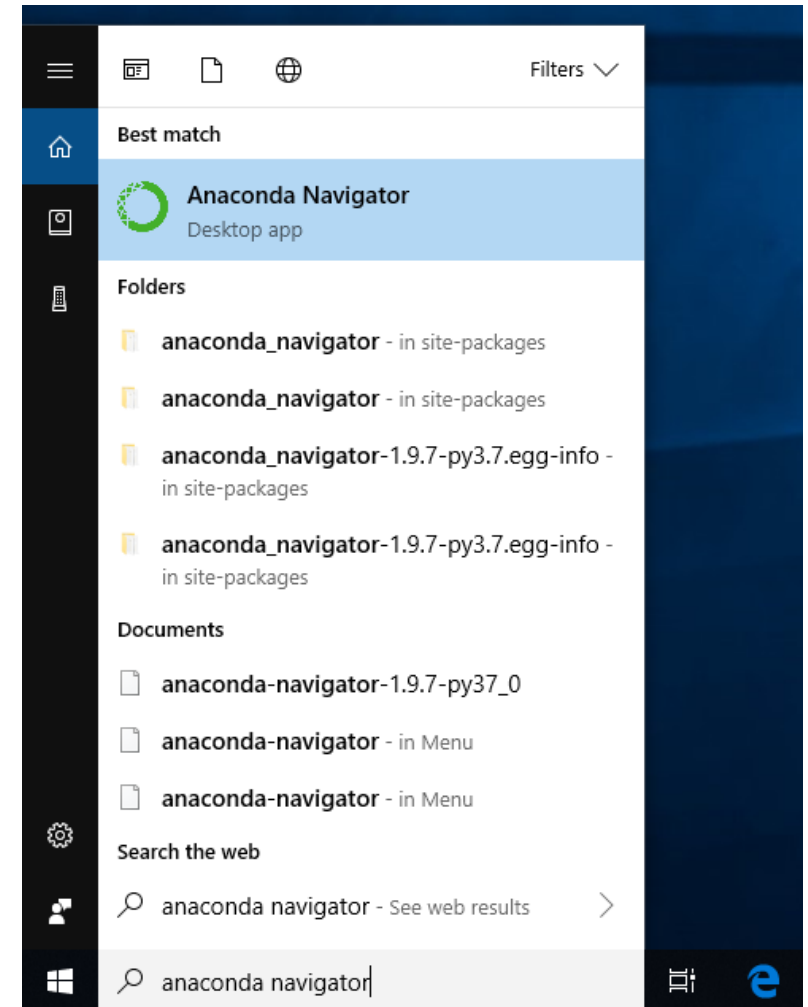
# Installing Anaconda on Windows

- After a successful installation you will see the “Thanks for installing Anaconda” dialog box:
- If you wish to read more about Anaconda.org and how to get started with Anaconda, check the boxes “Anaconda Individual Edition Tutorial” and “Learn more about Anaconda”.
- Click the Finish button.



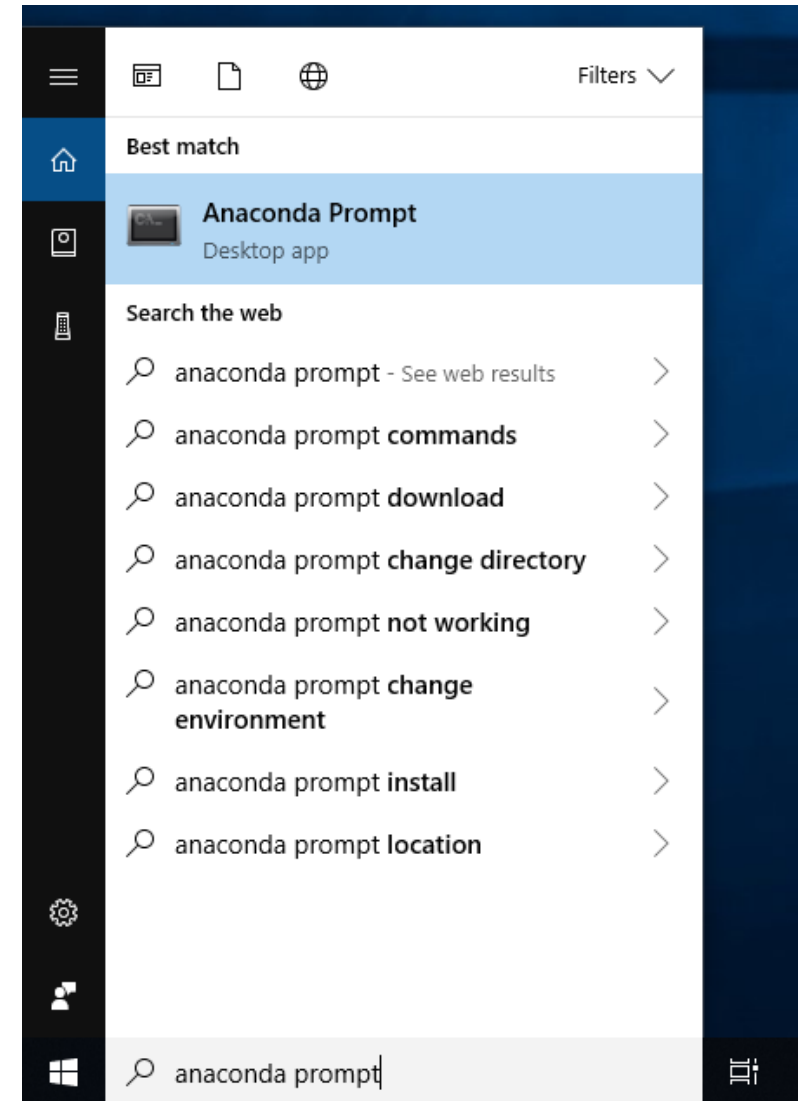
# Confirm Anaconda Installation

- You can confirm that Anaconda is installed and working with Anaconda Navigator or conda.
- Anaconda Navigator is a graphical user interface that is automatically installed with Anaconda.
- Navigator will open if the installation was successful.
- Windows: Click Start, search or select Anaconda Navigator from the menu.



# Confirm Anaconda Installation

- If you prefer using a command line interface (CLI), you can use conda to verify the installation using Anaconda Prompt on Windows or terminal on Linux and macOS.
- To open Anaconda Prompt: Windows: Click Start, search, or select Anaconda Prompt from the menu.



# Confirm Anaconda Installation

- After opening Anaconda Prompt or the terminal, choose any of the following methods to verify:
  - Enter `conda list`. If Anaconda is installed and working, this will display a list of installed packages and their versions.
  - Enter the command `python`. This command runs the Python shell. If Anaconda is installed and working, the version information it displays when it starts up will include “Anaconda”. To exit the Python shell, enter the command `quit()`.
  - Open Anaconda Navigator with the command `anaconda-navigator`. If Anaconda is installed properly, Anaconda Navigator will open.

# Packages included in Anaconda

Packages included in Anaconda 4.4+, or install with "conda install PACKAGENAME"

- NumPy
  - [numpy.org](http://numpy.org)
  - N-dimensional array for numerical computation
- SciPy
  - [scipy.org](http://scipy.org)
  - Scientific computing library for Python
- Matplotlib
  - [matplotlib.org](http://matplotlib.org)
  - 2D Plotting library for Python
- Pandas
  - [pandas.pydata.org](http://pandas.pydata.org)
  - Powerful Python data structures and data analysis toolkit



# Packages included in Anaconda

- Seaborn
  - [seaborn.pydata.org/](https://seaborn.pydata.org/)
  - Statistical graphics library for Python
- Bokeh
  - [bokeh.pydata.org](https://bokeh.pydata.org)
  - Interactive web visualization library
- Scikit-Learn
  - [scikit-learn.org/stable](https://scikit-learn.org/stable)
  - Python modules for machine learning and data mining
- NLTK
  - [nltk.org](https://nltk.org)
  - Natural language toolkit

# Packages included in Anaconda

- Jupyter Notebook
  - [jupyter.org](https://jupyter.org)
  - Web app that allows you to create and share documents that contain live code, equations, visualizations and explanatory text
- R essentials
  - <https://docs.anaconda.com/anaconda/user-guide/tasks/use-r-language>
  - 80+ of the most used R packages for data science can be installed with “conda install r-essentials”
  - R package list
  - <https://docs.anaconda.com/anaconda/packages/r-language-pkg-docs>

# What is Miniconda?

- Miniconda is a free minimal installer for conda.
- It is a small, bootstrap version of Anaconda that includes only conda, Python, the packages they depend on, and a small number of other useful packages, including pip, zlib and a few others.
- Use the conda install command to install 720+ additional conda packages from the Anaconda repository.

# What is Conda?

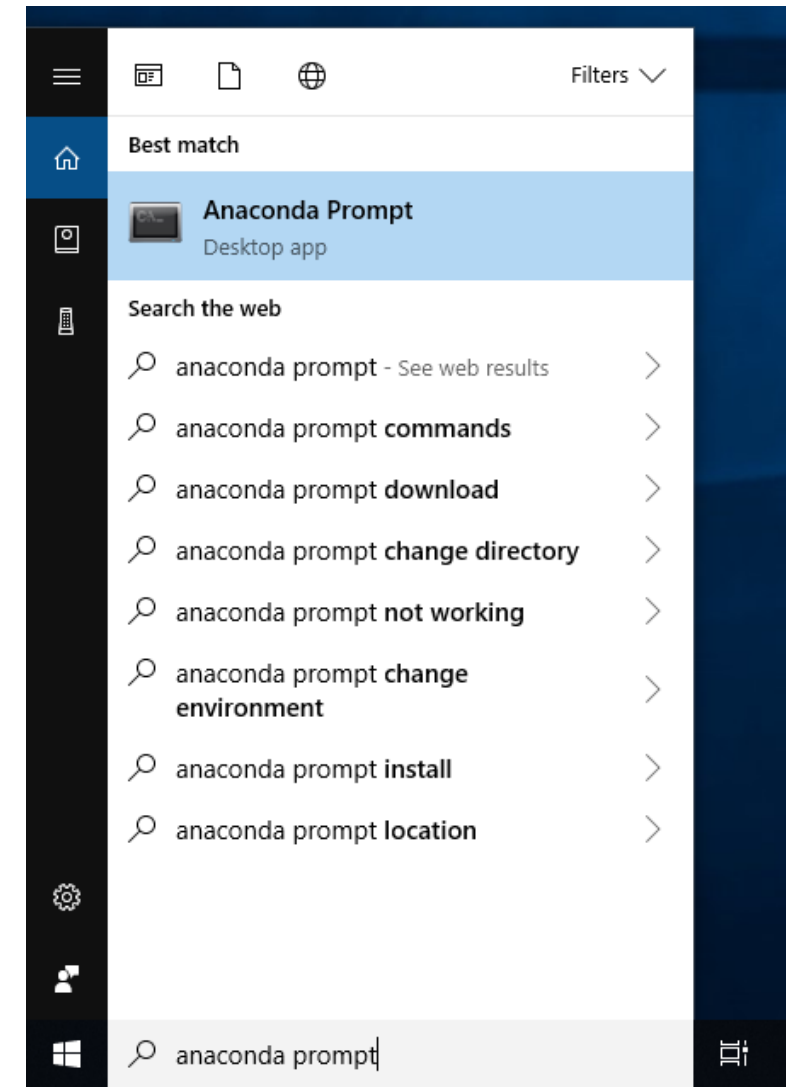
- Conda is an open-source package management system and environment management system that runs on Windows, macOS and Linux.
- Conda quickly installs, runs and updates packages and their dependencies.
- Conda easily creates, saves, loads and switches between environments on your local computer.
- It was created for Python programs, but it can package and distribute software for any language.

# What is Conda?

- Conda as a package manager helps you find and install packages.
- If you need a package that requires a different version of Python, you do not need to switch to a different environment manager, because conda is also an environment manager.
- With just a few commands, you can set up a totally separate environment to run that different version of Python, while continuing to run your usual version of Python in your normal environment.

# Starting Conda

- **Starting Conda using Windows**
- From the Start menu, search for and open "Anaconda Prompt."
- On Windows, all commands are typed into the Anaconda Prompt window.
- **Starting Conda using MacOS**
- Open Launchpad, then click the terminal icon. On macOS, all commands are typed into the terminal window.
- **Starting Consa using Linux**
- Open a terminal window. On Linux, all commands are typed into the terminal window.



# Managing Conda

- Verify that conda is installed and running on your system by typing:

```
conda --version
```

- Conda displays the number of the version that you have installed. You do not need to navigate to the Anaconda directory.

```
conda 4.8.3
```

- Update conda to the current version. Type the following:

```
conda update conda
```

- Conda compares versions and then displays what is available to install. If a newer version of conda is available, type y to update:

```
Proceed ([y]/n)? y
```

- It is recommended that you always keep conda updated to the latest version.

# Managing Packages

- You check which packages you have installed, check which are available and look for a specific package and install it.
- Check to see if a package you have not installed named `"beautifulsoup4"` is available from the Anaconda repository (must be connected to the Internet):

```
conda search beautifulsoup4
```

- Conda displays a list of all packages with that name on the Anaconda repository, so we know it is available.
- Install this package into the current environment:

```
conda install beautifulsoup4
```

- Check to see if the newly installed program is in this environment:

```
conda list
```

- To uninstall the package into the current environment

```
conda uninstall beautifulsoup4
```



# What is pip (Package Manager)

- pip is the package installer for Python.
- You can use pip to install packages from the Python Package Index and other indexes.
- Most distributions of Python come with pip preinstalled. Python 2.7.9 and later (on the python2 series), and Python 3.4 and later include pip (pip3 for Python 3) by default.
- To install a package using pip, you can execute the following command

```
pip install some-package-name
```

- You can uninstall the package using the following command

```
pip uninstall some-package-name
```

# Conda versus pip

- Conda and pip are often considered as being nearly identical. Although some of the functionality of these two tools overlap, they were designed and should be used for different purposes.
- **Pip** installs Python software packaged as wheels or source distributions. The latter may require that the system have compatible compilers, and possibly libraries, installed before invoking pip to succeed.
- **Conda** is a cross platform package and environment manager that installs and manages conda packages from the Anaconda repository as well as from the Anaconda Cloud.
- Conda packages are binaries.
- There is never a need to have compilers available to install them.
- Additionally, conda packages are not limited to Python software. They may also contain C or C++ libraries, R packages or any other software.

# Conda versus pip

- Pip installs Python packages whereas conda installs packages which may contain software written in any language.
- For example, before using pip, a Python interpreter must be installed via a system package manager or by downloading and running an installer.
- Conda on the other hand can install Python packages as well as the Python interpreter directly.
- Conda has the ability to create isolated environments that can contain different versions of Python and/or the packages installed in them.
- This can be extremely useful when working with data science tools as different tools may contain conflicting requirements which could prevent them all being installed into a single environment.
- Pip has no built in support for environments but rather depends on other tools like virtualenv or venv to create isolated environments.

# Conda versus pip

- When installing packages, pip installs dependencies in a recursive, serial loop.
- No effort is made to ensure that the dependencies of all packages are fulfilled simultaneously.
- This can lead to environments that are broken in subtle ways, if packages installed earlier in the order have incompatible dependency versions relative to packages installed later in the order.
- In contrast, conda uses a satisfiability (SAT) solver to verify that all requirements of all packages installed in an environment are met.
- This check can take extra time but helps prevent the creation of broken environments.
- As long as package metadata about dependencies is correct, conda will predictably produce working environments.

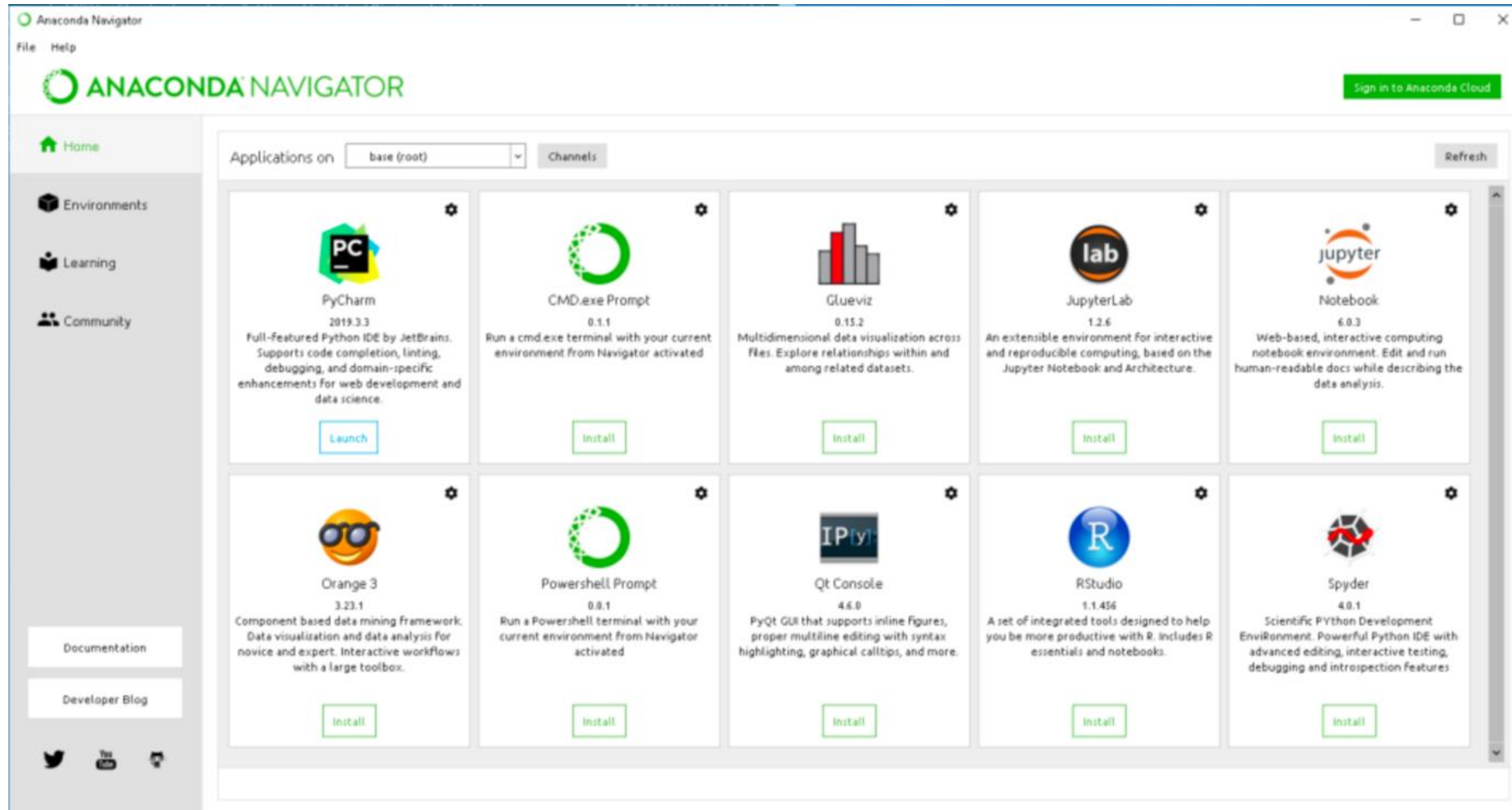
# Conda versus pip

	Conda	Pip
<b>Manages</b>	binaries	wheel or source
<b>Require Compiler</b>	No	Yes
<b>Package type</b>	Any	Python only
<b>Create environment</b>	Yes-built in	No require virtualenv or venv
<b>Check dependencies</b>	Yes	No
<b>Package Source</b>	Anaconda repo and cloud	PyPI

# Anaconda Navigator

- Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda distribution that allows you to launch applications and easily manage conda packages, environments, and channels without using command-line commands.
- Navigator can search for packages on Anaconda.org or in a local Anaconda Repository.
- It is available for Windows, macOS, and Linux.

# Anaconda Navigator



# Jupyter Notebook



# Jupyter Notebook

- The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text.
- Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.

# Features of Jupyter Notebook

- **Language of choice:** Jupyter supports over 40 programming languages, including Python, R, Julia, and Scala.
- **Share notebooks:** Notebooks can be shared with others using email, Dropbox, GitHub and the Jupyter Notebook Viewer.
- **Interactive output:** Your code can produce rich, interactive output: HTML, images, videos, LaTeX, and custom MIME types.
- **Big data integration:** Leverage big data tools, such as Apache Spark, from Python, R and Scala. Explore that same data with pandas, scikit-learn, ggplot2, TensorFlow.

# Features of Jupyter Web Application

- In-browser editing for code, with automatic syntax highlighting, indentation, and tab completion/introspection.
- The ability to execute code from the browser, with the results of computations attached to the code which generated them.
- Displaying the result of computation using rich media representations, such as HTML, LaTeX, PNG, SVG, etc. For example, publication-quality figures rendered by the matplotlib library, can be included inline.
- In-browser editing for rich text using the Markdown markup language, which can provide commentary for the code, is not limited to plain text.
- The ability to easily include mathematical notation within markdown cells using LaTeX, and rendered natively by MathJax.

# Notebook Document

- Notebook documents contains the inputs and outputs of an interactive session as well as additional text that accompanies the code but is not meant for execution.
- These documents are internally JSON files and are saved with the `.ipynb` extension.
- Since JSON is a plain text format, they can be version-controlled and shared with colleagues.
- Notebooks may be exported to a range of static formats, including HTML (for example, for blog posts), reStructuredText, LaTeX, PDF, and slide shows, via the `nbconvert` command.

# Starting Notebook Server

- You can start running a notebook server from the command line using the following command:

```
jupyter notebook
```

- This will print some information about the notebook server in your console, and open a web browser to the URL of the web application (by default, <http://127.0.0.1:8888>).
- The landing page of the Jupyter notebook web application, the dashboard, shows the notebooks currently available in the notebook directory (by default, the directory from which the notebook server was started).
- You can create new notebooks from the dashboard with the New Notebook button, or open existing ones by clicking on their name.

# Starting Notebook Server

- When starting a notebook server from the command line, you can also open a particular notebook directly, bypassing the dashboard, with `jupyter notebook my_notebook.ipynb`. The `.ipynb` extension is assumed if no extension is given.
- When you are inside an open notebook, the File | Open... menu option will open the dashboard in a new browser tab, to allow you to open another notebook from the notebook directory or to create a new notebook.
- You can start more than one notebook server at the same time, if you want to work on notebooks in different directories.
- By default the first notebook server starts on port `8888`, and later notebook servers search for ports near that one. You can also manually specify the port with the `--port` option.

# Create New Notebook Document

- A new notebook may be created at any time, either from the dashboard, or using the File ► New menu option from within an active notebook.
- The new notebook is created within the same directory and will open in a new browser tab.
- It will also be reflected as a new entry in the notebook list on the dashboard.

# Create New Notebook Document



Files

Running

Clusters

Select items to perform actions on them.

Upload

New ▾



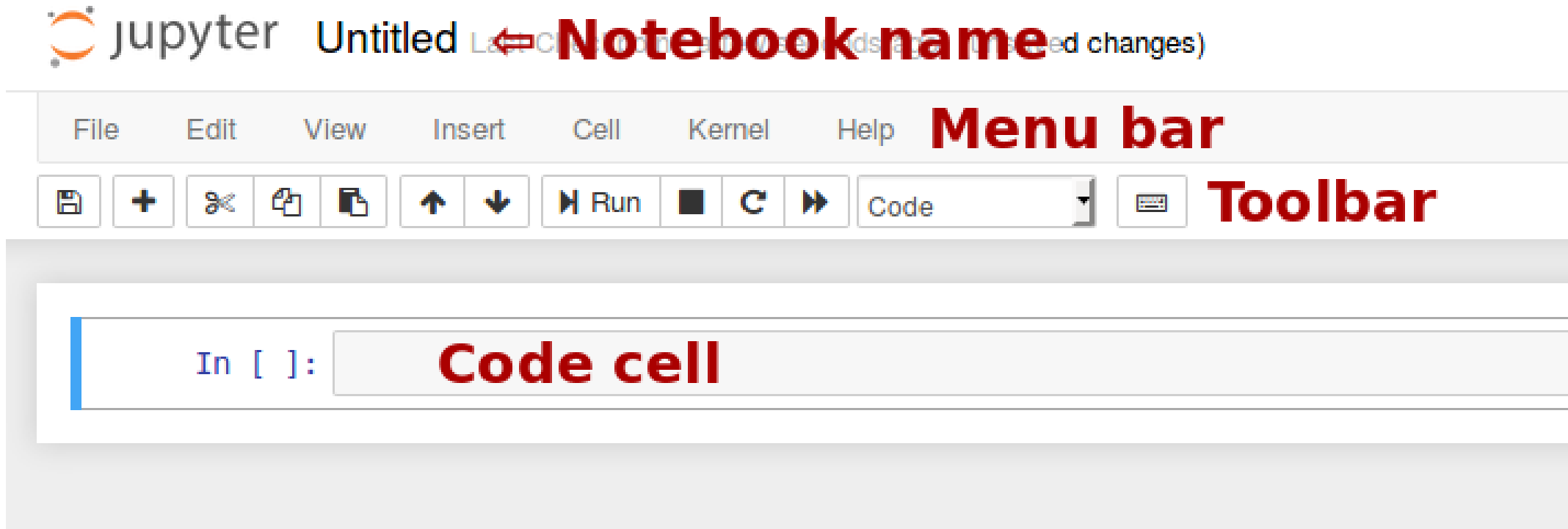
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
<input type="checkbox"/>		basic	
<input type="checkbox"/>		citations	
<input type="checkbox"/>		custom_filter	
<input type="checkbox"/>		custom_latex_cell_style	
<input type="checkbox"/>		custom_preprocessor	
<input type="checkbox"/>		custom_template	
<input type="checkbox"/>		hr_cell_style	
<input type="checkbox"/>		images	
<input type="checkbox"/>		latex_cell_style	
<input type="checkbox"/>		notebook_cell_style	
<input type="checkbox"/>		ntf	



# Notebook User Interface

- When you create a new notebook document, you will be presented with the notebook name, a menu bar, a toolbar and an empty code cell.
- **Notebook name:** The name displayed at the top of the page, next to the Jupyter logo, reflects the name of the .ipynb file.
  - Clicking on the notebook name brings up a dialog which allows you to rename it. Thus, renaming a notebook from “Untitled0” to “My first notebook” in the browser, renames the Untitled0.ipynb file to My first notebook.ipynb.
- **Menu bar:** The menu bar presents different options that may be used to manipulate the way the notebook functions.
- **Toolbar:** The tool bar gives a quick way of performing the most-used operations within the notebook, by clicking on an icon.
- **Code cell:** the default type of cell; read on for an explanation of cells.

# Notebook User Interface



# Structure of Notebook Document

- The notebook consists of a sequence of cells.
- A cell is a multiline text input field, and its contents can be executed by using Shift-Enter, or by clicking either the “Play” button the toolbar, or Cell, Run in the menu bar.
- The execution behavior of a cell is determined by the cell’s type. There are three types of cells: code cells, markdown cells, and raw cells.
- Every cell starts off being a code cell, but its type can be changed by using a drop-down on the toolbar (which will be “Code”, initially), or via keyboard shortcuts.

# Code Cell

- A code cell allows you to edit and write new code, with full syntax highlighting and tab completion.
- The programming language you use depends on the kernel, and the default kernel (IPython) runs Python code.
- When a code cell is executed, code that it contains is sent to the kernel associated with the notebook.
- The results that are returned from this computation are then displayed in the notebook as the cell's output.
- The output is not limited to text, with many other possible forms of output are also possible, including matplotlib figures and HTML tables (as used, for example, in the pandas data analysis package).
- This is known as IPython's rich display capability.

# Markdown Cell

- You can document the computational process in a literate way, alternating descriptive text with code, using rich text.
- In IPython this is accomplished by marking up text with the Markdown language.
- The corresponding cells are called Markdown cells. The Markdown language provides a simple way to perform this text markup, that is, to specify which parts of the text should be emphasized (italics), bold, form lists, etc.
- If you want to provide structure for your document, you can use markdown headings. Markdown headings consist of 1 to 6 hash # signs # followed by a space and the title of your section.
- The markdown heading will be converted to a clickable link for a section of the notebook. It is also used as a hint when exporting to other document formats, like PDF.
- When a Markdown cell is executed, the Markdown code is converted into the corresponding formatted rich text. Markdown allows arbitrary HTML code for formatting.

# Raw Cell

- Raw cells provide a place in which you can write output directly. Raw cells are not evaluated by the notebook.

# Spyder IDE

# Spyder IDE

- Spyder is a free and open-source scientific environment written in Python, for Python, and designed by and for scientists, engineers and data analysts.
- It features a unique combination of the advanced editing, analysis, debugging, and profiling functionality of a comprehensive development tool with the data exploration, interactive execution, deep inspection, and beautiful visualization capabilities of a scientific package.



# Spyder

The screenshot displays the Spyder Python IDE interface. The main window is divided into several panels:

- Left Panel (Project Explorer):** Shows a file tree with folders like 'App.py', 'template.py', 'plot\_example.py', and 'plugin.py'. The 'plugin.py' folder is expanded, showing various methods like `_init_`, `update_font`, `apply_plugin_settin.`, `toggle_view`, `get_plugin_title`, `get_plugin_icon`, `get_focus_widget`, `closing_plugin`, `refresh_plugin`, `get_plugin_actions`, `register_plugin`, `get_clients`, `get_focus_client`, `get_current_client`, `get_current_shellwi.`, `run_script`, `run_cell`, `debug_cell`, `set_current_client_.`, `set_working_direct.`, `update_working_dir`, and `update_path`.
- Top Panel (Code Editor):** Displays the code for `plot_example.py`. The code includes imports for `numpy`, `matplotlib.pyplot`, `matplotlib.cm`, `matplotlib.colors`, and `mpl_toolkits.mplot3d`. It defines two functions: `generate_polar_plot()` and `generate_dem_plot()`. The `generate_polar_plot()` function generates a polar plot with a specified number of slices and radii. The `generate_dem_plot()` function generates a 3D representation of a terrain DEM.
- Right Panel (Variable Explorer):** Shows a table of variables in the current scope. The table has columns for Name, Type, Size, and Value.
- Bottom Panel (Plots and IPython console):** Displays two plots: a 3D surface plot of a terrain DEM and a polar plot. The IPython console is also visible.

Name	Type	Size	Value
data	Array of str128	(3, 3)	ndarray object of numpy module
df	DataFrame	(2, 2)	Column names: Col1, Col2
filename	str	1	/Users/Documents/spyder/spyder/tests/test_dont_use.py
i	Array of uint32	(10, 10)	[[0 0 0 ... 0 0 0] [0 0 0 ... 0 0 0]
li	list	5	['abcd', 745, 2.23, 'efgh', 70.2]
r	float	1	6.46567886443
t	tuple	5	('abcd', 745, 2.23, 'efgh', 70.2)
tinylst	list	2	[123, 'efgh']

Variable explorer Help Files Online help Profiler Code Analysis

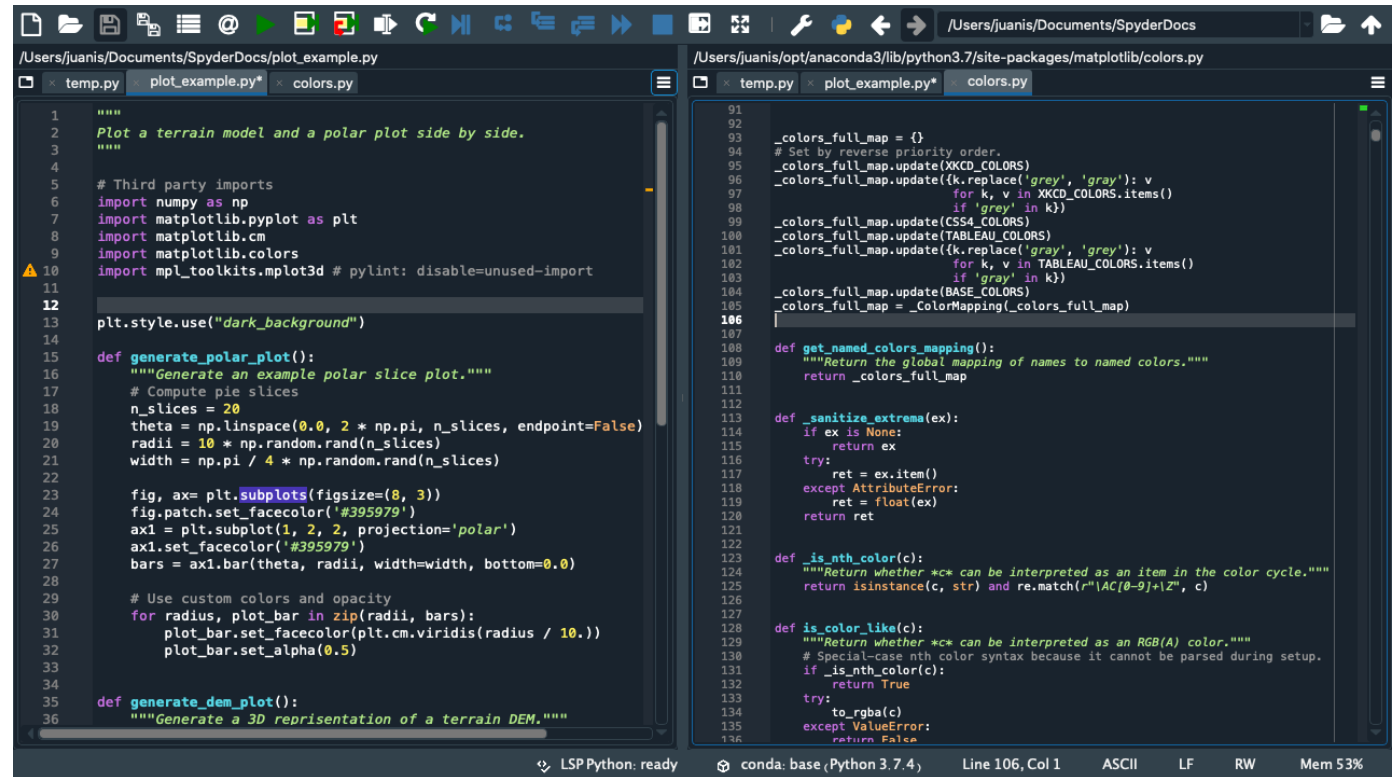
58 %

Plots IPython console

LSP Python: restarting... conda: base (Python 3.7.4) Line 1, Col 1 ASCII LF RW Mem 50%

# Spyder Editor

- Spyder's multi-language Editor integrates a number of powerful tools right out of the box for an easy to use, efficient editing experience.
- The Editor's key features include syntax highlighting; real-time code and style analysis; on-demand completion, calltips and go-to-definition features; a function/class browser, horizontal and vertical splitting, and much more.



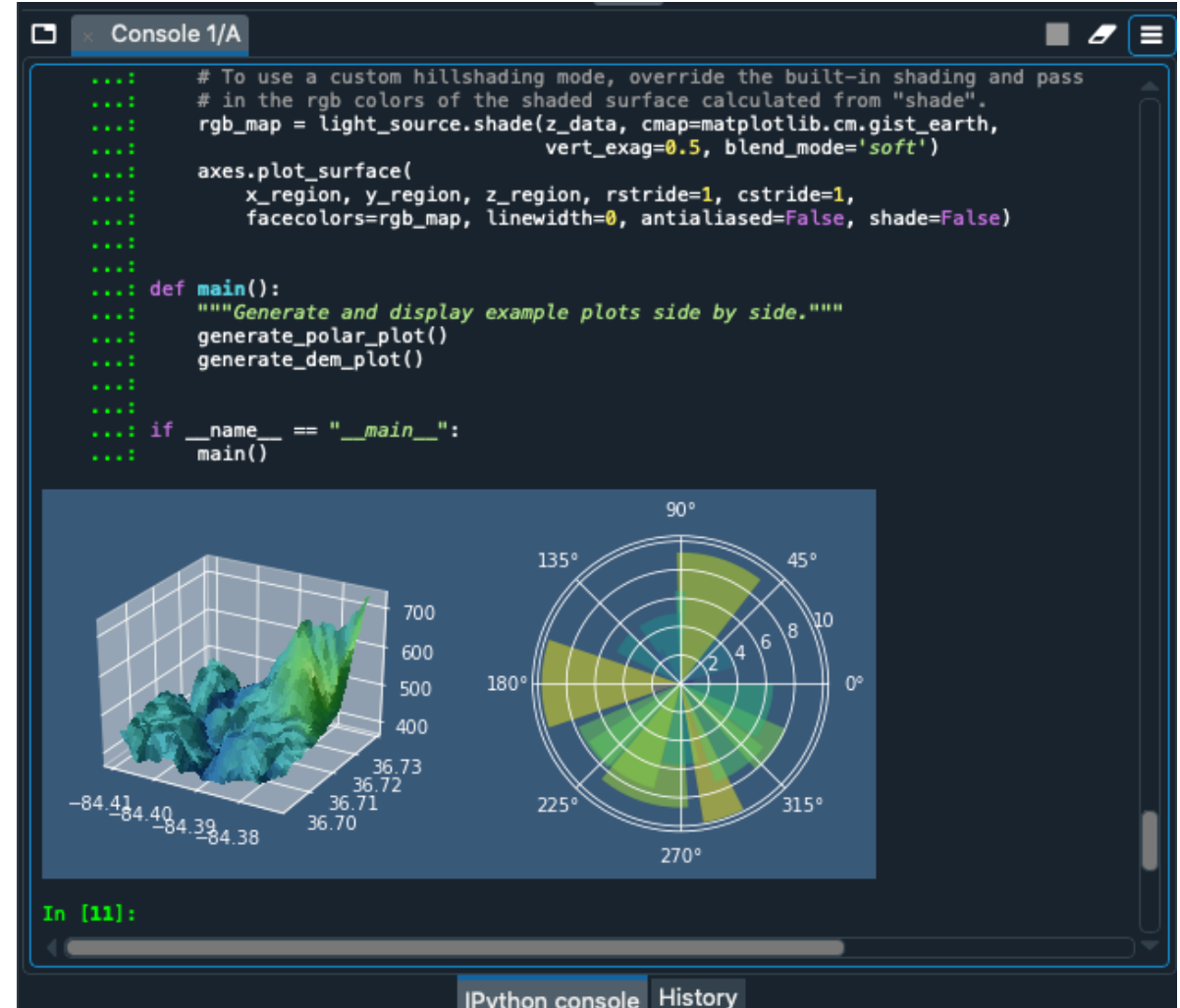
The screenshot displays the Spyder Editor interface with two Python files open side-by-side. The left pane shows `plot_example.py` with line numbers 1 through 36. It contains a docstring, imports for `numpy`, `matplotlib.pyplot`, `matplotlib.cm`, and `mpl_toolkits.mplot3d`, followed by a `plt.style.use("dark_background")` call and two function definitions: `generate_polar_plot()` and `generate_dem_plot()`. The right pane shows `colors.py` with line numbers 91 through 146. It defines a `_colors_full_map`, a `get_named_colors_mapping()` function, a `sanitize_extrema()` function, and two utility functions `is_nth_color()` and `is_color_like()`. The status bar at the bottom indicates "LSP Python: ready", "conda: base (Python 3.7.4)", "Line 106, Col 1", "ASCII", "LF", "RW", and "Mem 53%".

```
1 """
2 Plot a terrain model and a polar plot side by side.
3 """
4
5 # Third party imports
6 import numpy as np
7 import matplotlib.pyplot as plt
8 import matplotlib.cm
9 import matplotlib.colors
10 import mpl_toolkits.mplot3d # pylint: disable=unused-import
11
12 plt.style.use("dark_background")
13
14 def generate_polar_plot():
15     """Generate an example polar slice plot."""
16     # Compute pie slices
17     n_slices = 20
18     theta = np.linspace(0.0, 2 * np.pi, n_slices, endpoint=False)
19     radii = 10 * np.random.rand(n_slices)
20     width = np.pi / 4 * np.random.rand(n_slices)
21
22     fig, ax = plt.subplots(figsize=(8, 3))
23     fig.patch.set_facecolor('#395979')
24     ax1 = plt.subplot(1, 2, 2, projection='polar')
25     ax1.set_facecolor('#395979')
26     bars = ax1.bar(theta, radii, width=width, bottom=0.0)
27
28     # Use custom colors and opacity
29     for radius, plot_bar in zip(radii, bars):
30         plot_bar.set_facecolor(plt.cm.viridis(radius / 10.))
31         plot_bar.set_alpha(0.5)
32
33
34 def generate_dem_plot():
35     """Generate a 3D representation of a terrain DEM."""
36
```

```
91
92
93 _colors_full_map = {}
94 # Set by reverse priority order.
95 _colors_full_map.update(XKCD_COLORS)
96 _colors_full_map.update({k.replace('gray', 'grey'): v
97                        for k, v in XKCD_COLORS.items()
98                        if 'gray' in k})
99 _colors_full_map.update(CSS4_COLORS)
100 _colors_full_map.update(TABLEAU_COLORS)
101 _colors_full_map.update({k.replace('gray', 'grey'): v
102                        for k, v in TABLEAU_COLORS.items()
103                        if 'gray' in k})
104 _colors_full_map.update(BASE_COLORS)
105 _colors_full_map = _ColorMapping(_colors_full_map)
106
107
108 def get_named_colors_mapping():
109     """Return the global mapping of names to named colors."""
110     return _colors_full_map
111
112
113 def sanitize_extrema(ex):
114     if ex is None:
115         return ex
116     try:
117         ret = ex.item()
118     except AttributeError:
119         ret = float(ex)
120     return ret
121
122
123 def is_nth_color(c):
124     """Return whether *c* can be interpreted as an item in the color cycle."""
125     return isinstance(c, str) and re.match(r"^\d+(?=[-+]?[a-zA-Z0-9_]+)$", c)
126
127
128 def is_color_like(c):
129     """Return whether *c* can be interpreted as an RGB(A) color."""
130     # Special-case nth color syntax because it cannot be parsed during setup.
131     if is_nth_color(c):
132         return True
133     try:
134         to_rgba(c)
135     except ValueError:
136         return False
```

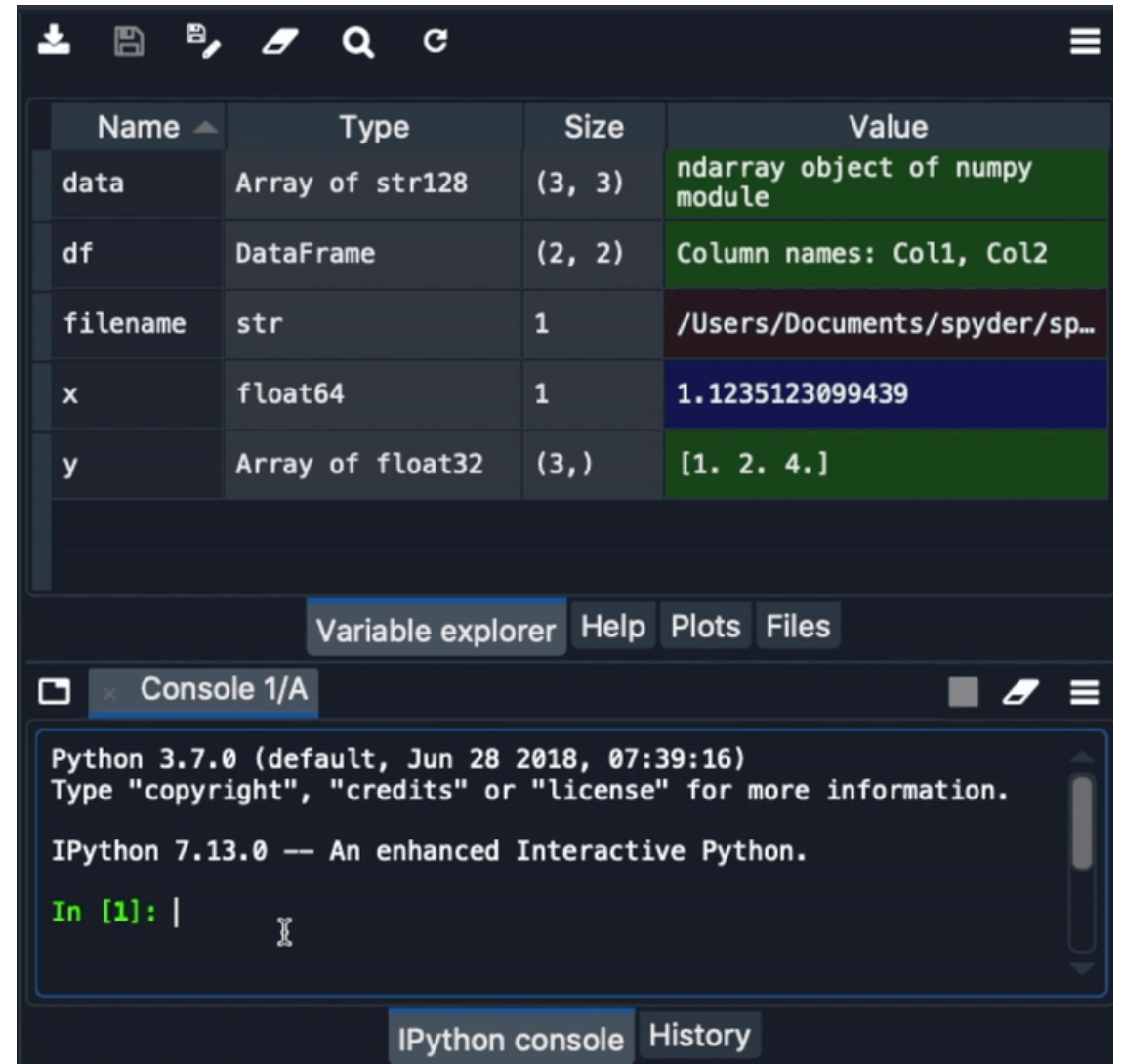
# IPython Console

- The IPython Console allows you to execute commands and enter, interact with and visualize data inside any number of fully featured IPython interpreters.



# Spyder Variable Explorer

- The Variable Explorer allows you to interactively browse and manage the objects generated running your code.



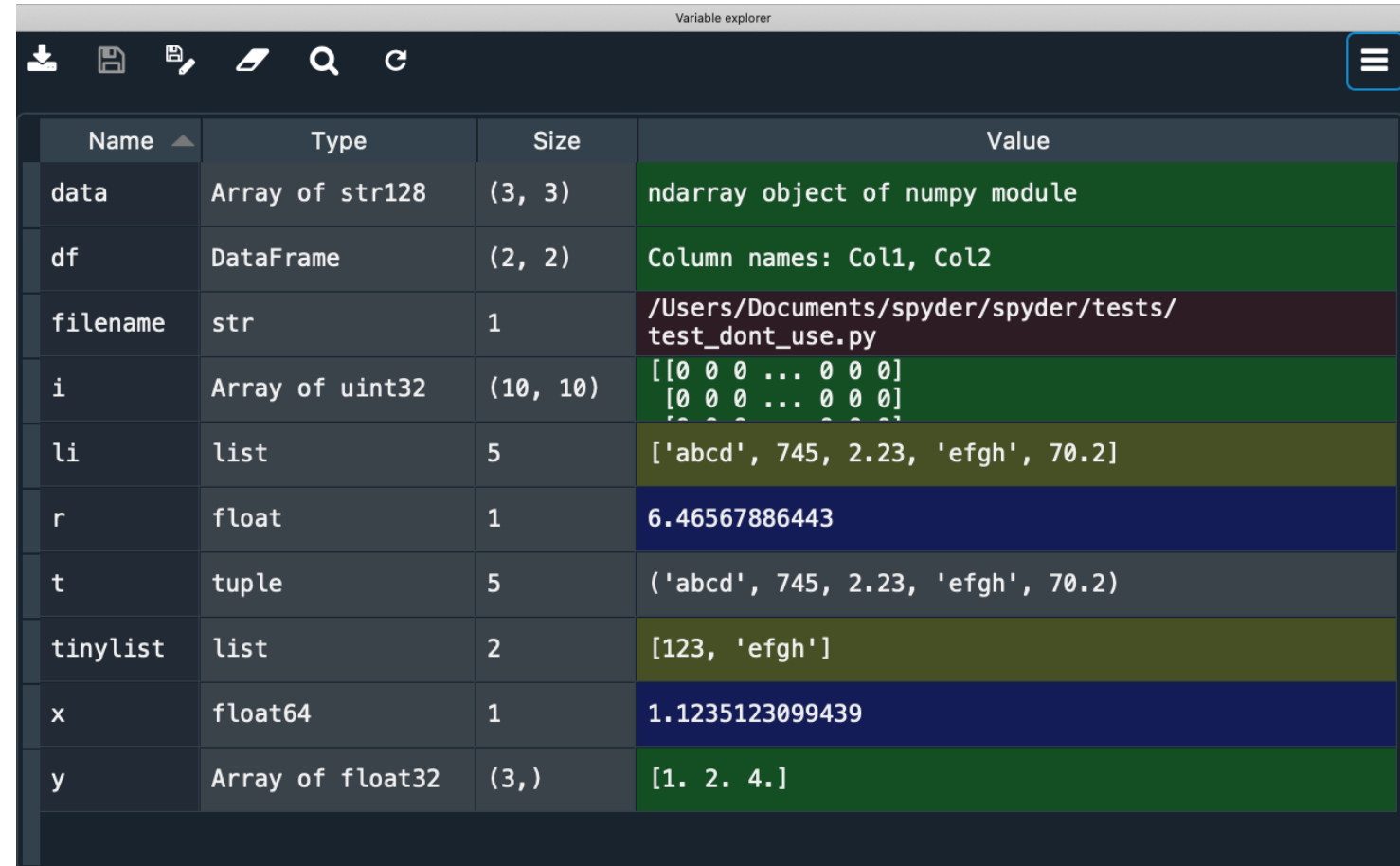
The screenshot displays the Spyder Variable Explorer window, which lists variables in a table. Below the table are tabs for 'Variable explorer', 'Help', 'Plots', and 'Files'. At the bottom, there is an 'IPython console' window with tabs for 'Console 1/A', 'IPython console', and 'History'.

Name	Type	Size	Value
data	Array of str128	(3, 3)	ndarray object of numpy module
df	DataFrame	(2, 2)	Column names: Col1, Col2
filename	str	1	/Users/Documents/spyder/sp...
x	float64	1	1.1235123099439
y	Array of float32	(3,)	[1. 2. 4.]

Python 3.7.0 (default, Jun 28 2018, 07:39:16)  
Type "copyright", "credits" or "license" for more information.  
IPython 7.13.0 -- An enhanced Interactive Python.  
In [1]: |

# Spyder Variable Explorer

- It shows the namespace contents (including all global objects, variables, class instances and more) of the currently selected IPython Console session, and allows you to add, remove, and edit their values through a variety of GUI-based editors.

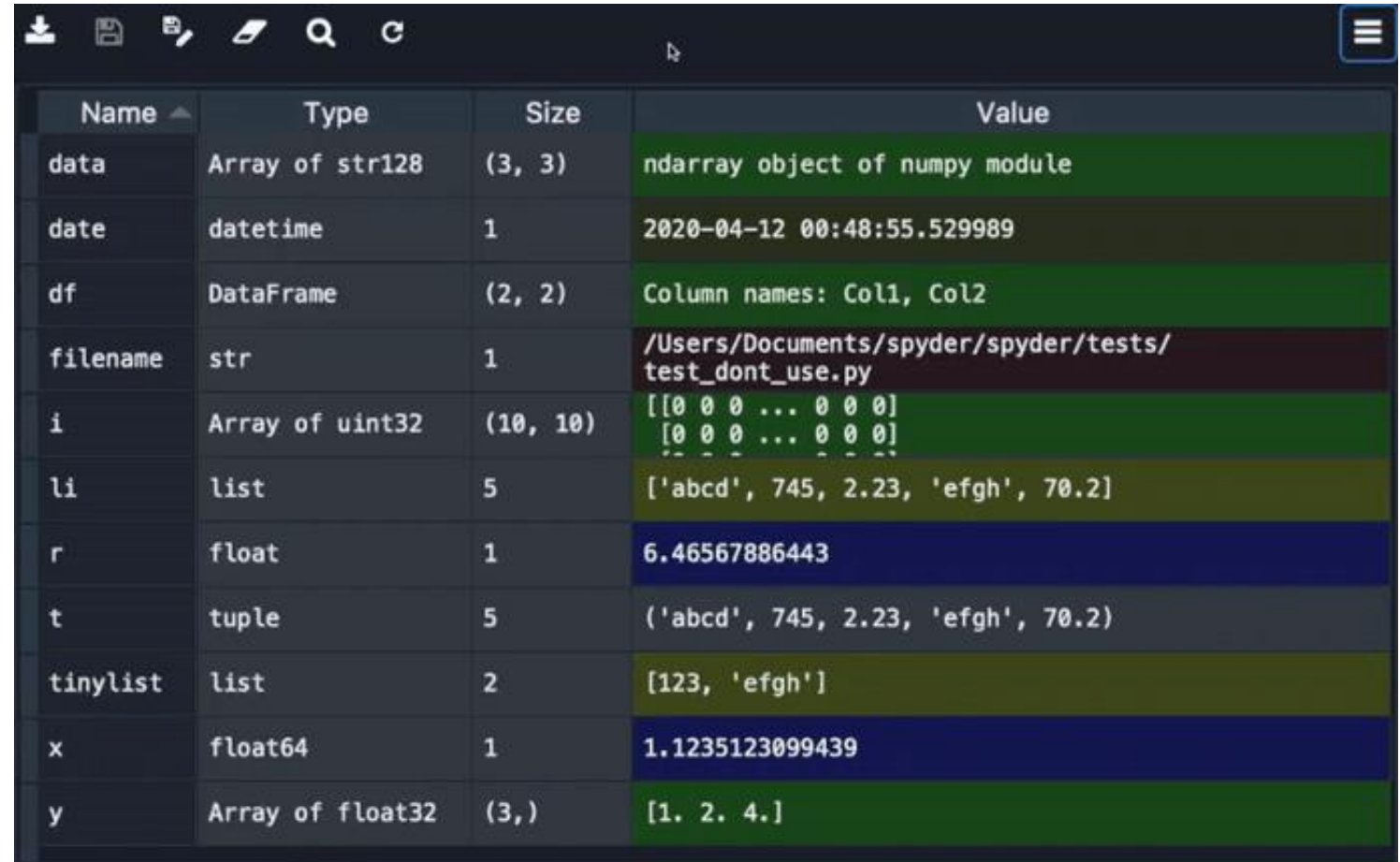


The screenshot shows the 'Variable explorer' window in Spyder. It contains a table with four columns: Name, Type, Size, and Value. The table lists several variables: 'data' (ndarray), 'df' (DataFrame), 'filename' (str), 'i' (ndarray), 'li' (list), 'r' (float), 't' (tuple), 'tinylst' (list), 'x' (float64), and 'y' (ndarray). Each row is color-coded: green for arrays, dark blue for floats, olive for lists, and grey for tuples and DataFrames.

Name	Type	Size	Value
data	Array of str128	(3, 3)	ndarray object of numpy module
df	DataFrame	(2, 2)	Column names: Col1, Col2
filename	str	1	/Users/Documents/spyder/spyder/tests/test_dont_use.py
i	Array of uint32	(10, 10)	[[0 0 0 ... 0 0 0] [0 0 0 ... 0 0 0] [0 0 0 ... 0 0 0]
li	list	5	['abcd', 745, 2.23, 'efgh', 70.2]
r	float	1	6.46567886443
t	tuple	5	('abcd', 745, 2.23, 'efgh', 70.2)
tinylst	list	2	[123, 'efgh']
x	float64	1	1.1235123099439
y	Array of float32	(3,)	[1. 2. 4.]

# Spyder Variable Explorer

- The Variable Explorer gives you information on the name, size, type and value of each object. To modify a scalar variable, like a number, string or boolean, simply double click it in the pane and type its new value.



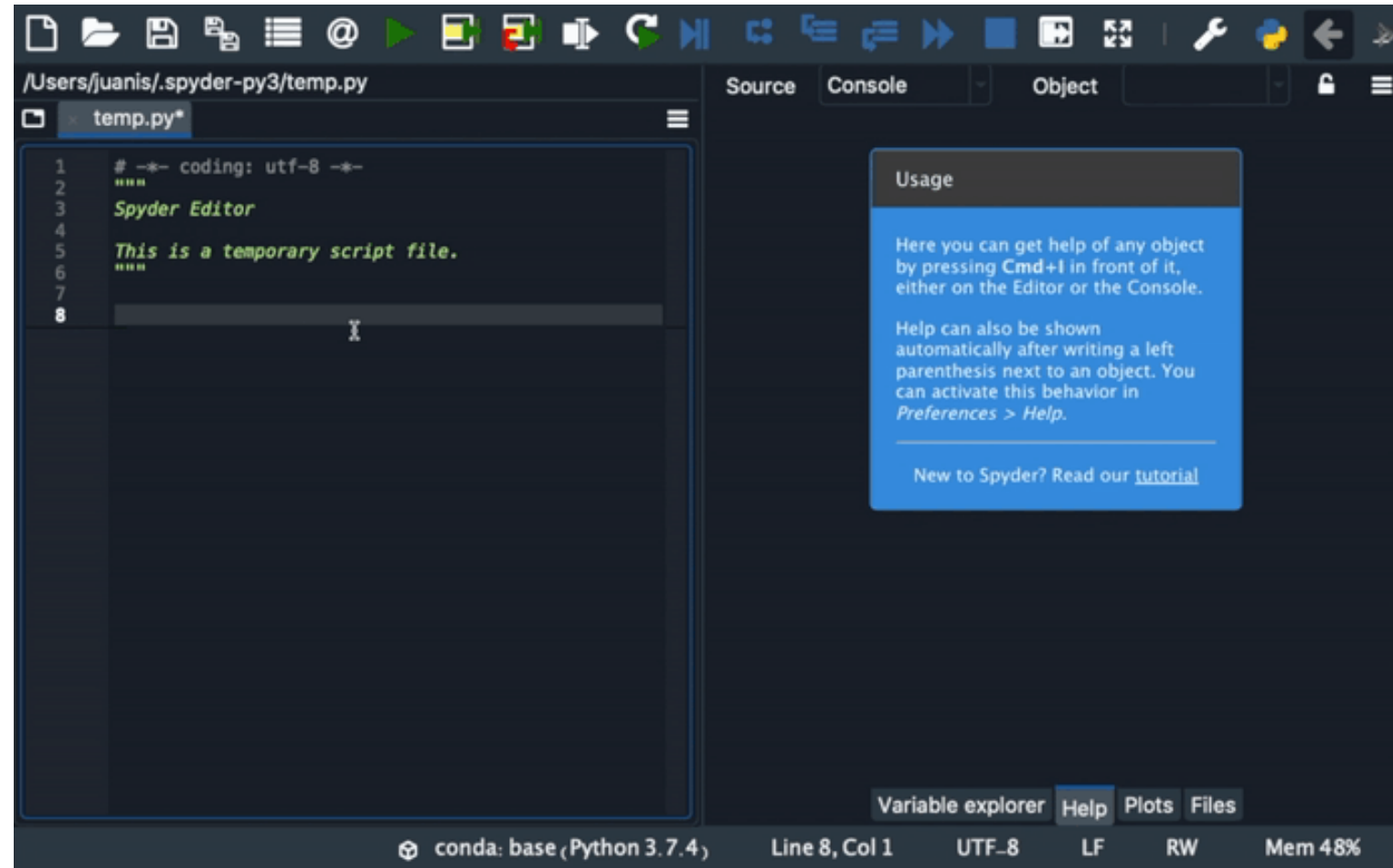
The screenshot shows the Spyder Variable Explorer window. It has a toolbar at the top with icons for download, save, delete, edit, search, and refresh. Below the toolbar is a table with four columns: Name, Type, Size, and Value. The table lists several variables: data, date, df, filename, i, li, r, t, tinylis, x, and y. Each row shows the variable's name, its data type, its size, and its current value. For example, 'data' is an 'Array of str128' with size '(3, 3)' and value 'ndarray object of numpy module'. 'date' is a 'datetime' with size '1' and value '2020-04-12 00:48:55.529989'. 'df' is a 'DataFrame' with size '(2, 2)' and value 'Column names: Col1, Col2'. 'filename' is a 'str' with size '1' and value '/Users/Documents/spyder/spyder/tests/test\_dont\_use.py'. 'i' is an 'Array of uint32' with size '(10, 10)' and value '[[0 0 0 ... 0 0 0] [0 0 0 ... 0 0 0]]'. 'li' is a 'list' with size '5' and value '['abcd', 745, 2.23, 'efgh', 70.2]'. 'r' is a 'float' with size '1' and value '6.46567886443'. 't' is a 'tuple' with size '5' and value>('abcd', 745, 2.23, 'efgh', 70.2). 'tinylis' is a 'list' with size '2' and value '[123, 'efgh']'. 'x' is a 'float64' with size '1' and value '1.1235123099439'. 'y' is an 'Array of float32' with size '(3,)' and value '[1. 2. 4.]'.

Name	Type	Size	Value
data	Array of str128	(3, 3)	ndarray object of numpy module
date	datetime	1	2020-04-12 00:48:55.529989
df	DataFrame	(2, 2)	Column names: Col1, Col2
filename	str	1	/Users/Documents/spyder/spyder/tests/test_dont_use.py
i	Array of uint32	(10, 10)	[[0 0 0 ... 0 0 0] [0 0 0 ... 0 0 0]]
li	list	5	['abcd', 745, 2.23, 'efgh', 70.2]
r	float	1	6.46567886443
t	tuple	5	('abcd', 745, 2.23, 'efgh', 70.2)
tinylis	list	2	[123, 'efgh']
x	float64	1	1.1235123099439
y	Array of float32	(3,)	[1. 2. 4.]



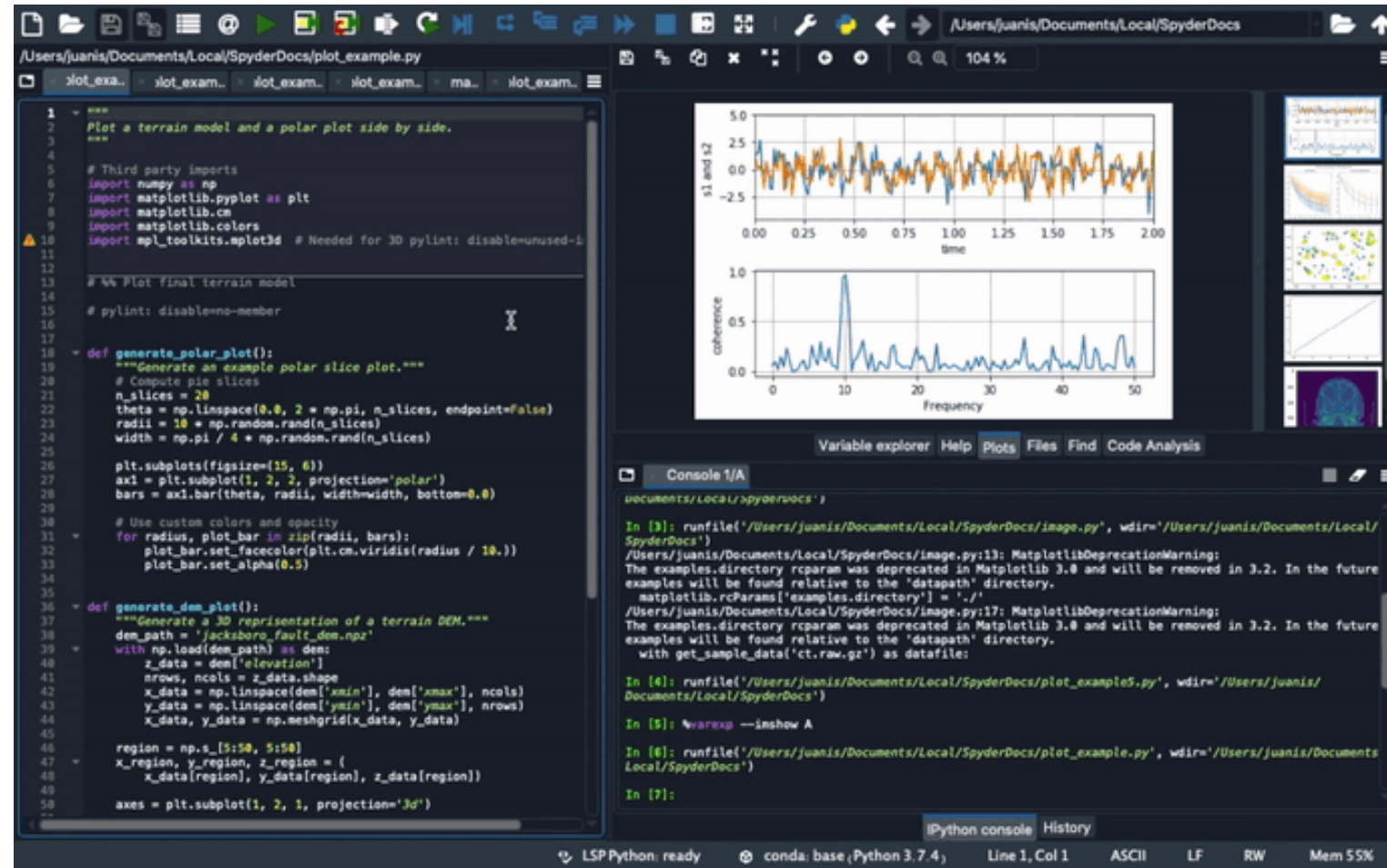
# Spyder Help Pane

- You can use the Help pane to find, render and display rich documentation for any object with a docstring, including modules, classes, functions and methods. This allows you to access documentation easily directly from Spyder, without having to interrupt your workflow.



# Spyder Plots

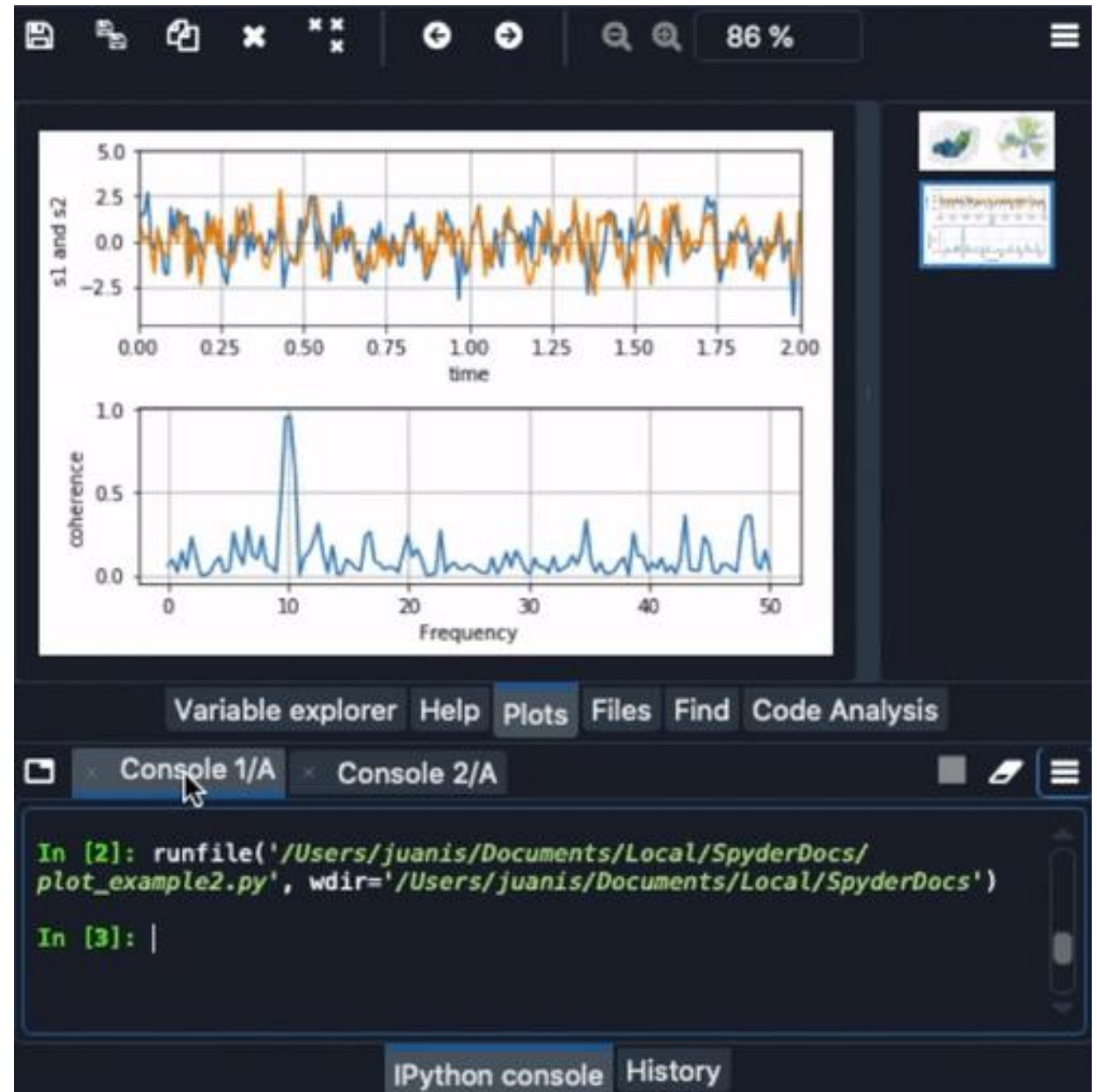
- The Plots pane shows the static figures and images created during your session.
- It will show you plots from the IPython Console, produced by your code in the Editor or generated by the Variable Explorer allowing you to interact with them in several ways.





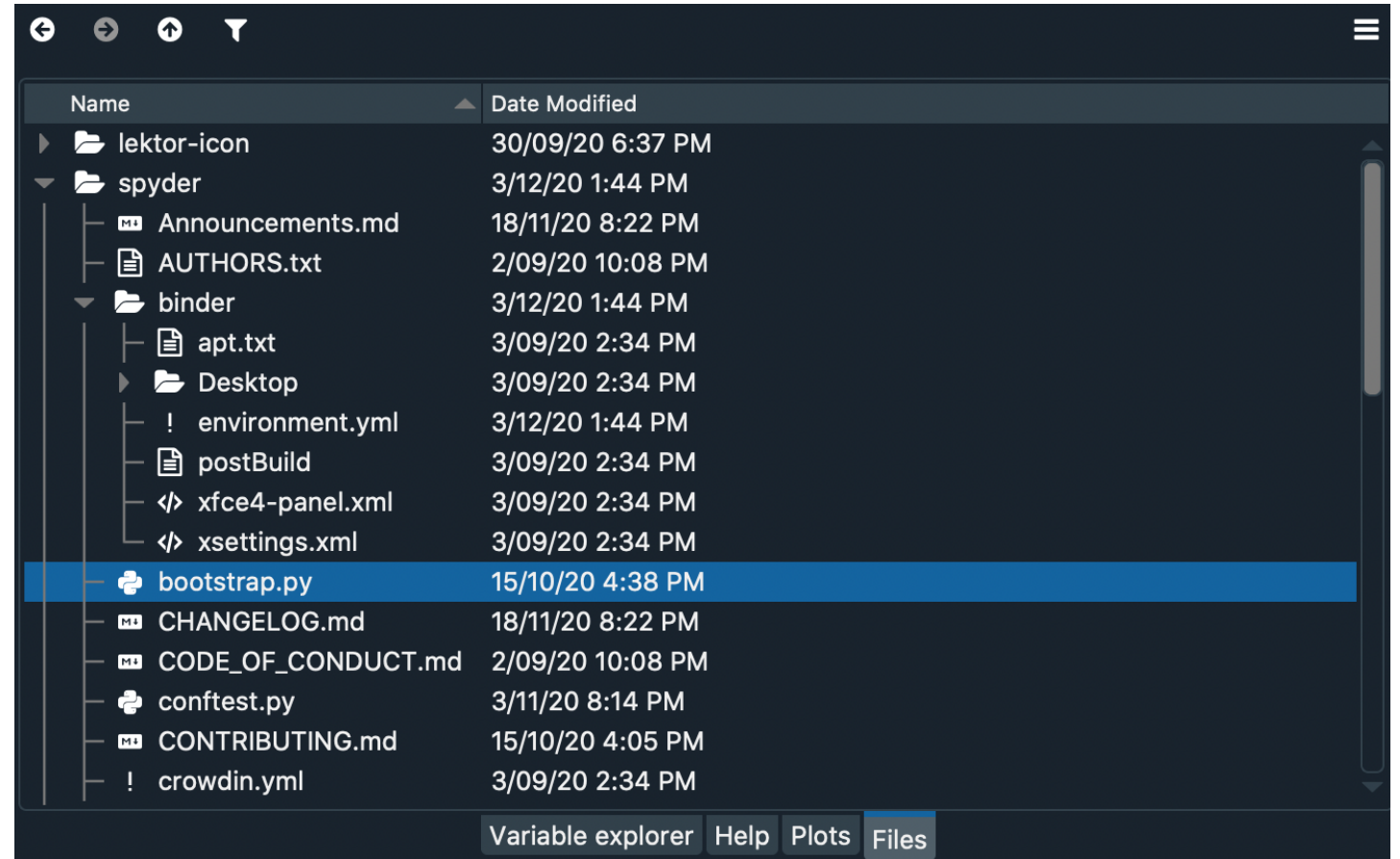
# Spyder Plots

- The figures shown in the Plots pane are those associated with the currently active Console tab;
- if you switch consoles, the list of plots displayed (or none at all, if a new console) will change accordingly.



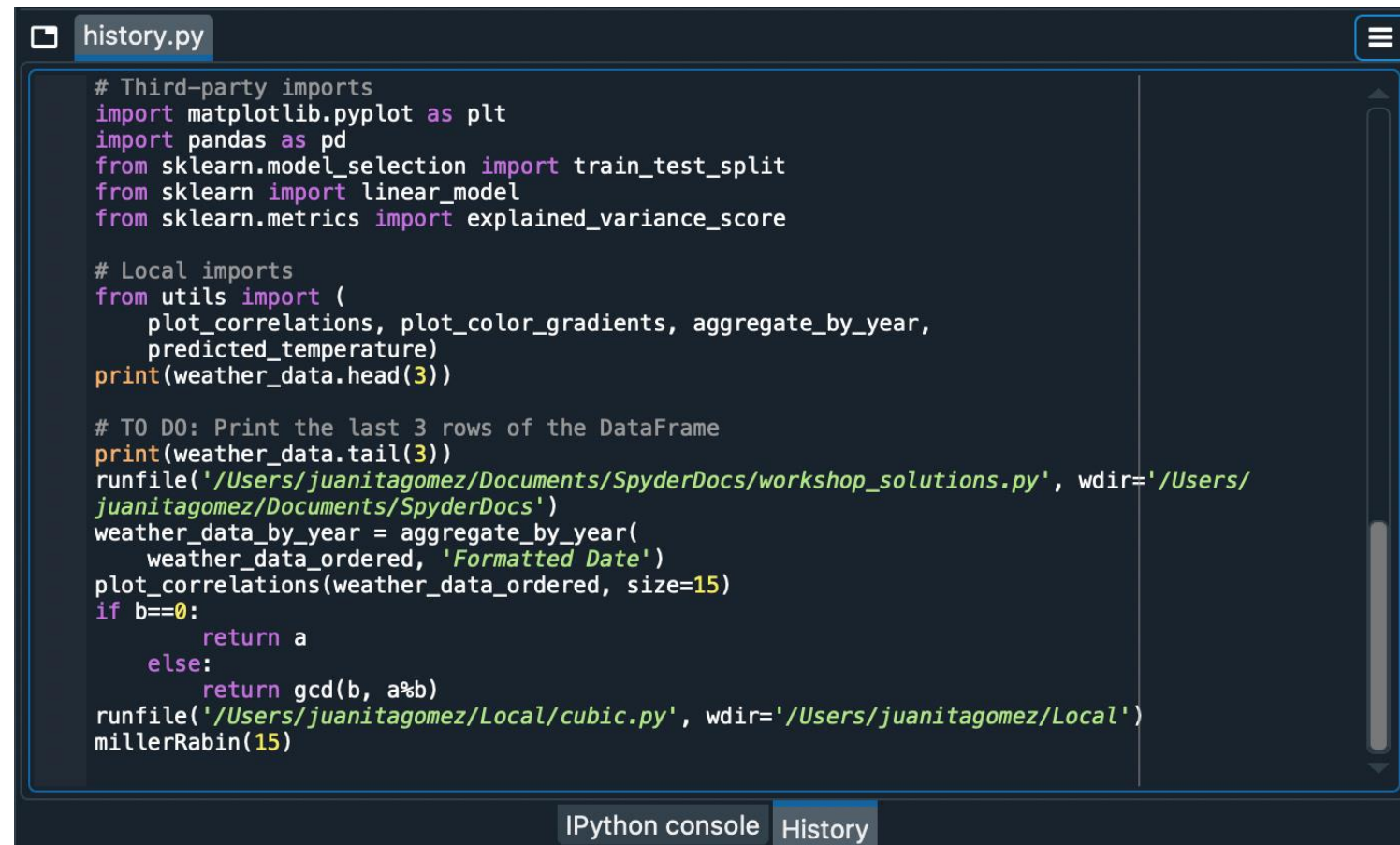
# Spyder Files

- The Files pane is a filesystem and directory browser built right into Spyder.
- You can view and filter files according to their type and extension, open them with the Editor or an external tool, and perform many common operations.



# Spyder History Pane

- With the History pane, you can view all the commands you've entered into any IPython Console, along with their timestamp.



The screenshot shows the Spyder IDE interface. The top pane displays a Python script named `history.py` with the following code:

```
# Third-party imports
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn import linear_model
from sklearn.metrics import explained_variance_score

# Local imports
from utils import (
    plot_correlations, plot_color_gradients, aggregate_by_year,
    predicted_temperature)
print(weather_data.head(3))

# TO DO: Print the last 3 rows of the DataFrame
print(weather_data.tail(3))
runfile('/Users/juanitagomez/Documents/SpyderDocs/workshop_solutions.py', wdir='/Users/juanitagomez/Documents/SpyderDocs')
weather_data_by_year = aggregate_by_year(
    weather_data_ordered, 'Formatted Date')
plot_correlations(weather_data_ordered, size=15)
if b==0:
    return a
else:
    return gcd(b, a%b)
runfile('/Users/juanitagomez/Local/cubic.py', wdir='/Users/juanitagomez/Local')
millerRabin(15)
```

The bottom pane shows the IPython console and the History pane. The History pane is active, displaying a list of executed commands and their timestamps:

- 11/11/2016 12:00:00 AM: `print(weather_data.head(3))`
- 11/11/2016 12:00:00 AM: `print(weather_data.tail(3))`
- 11/11/2016 12:00:00 AM: `runfile('/Users/juanitagomez/Documents/SpyderDocs/workshop_solutions.py', wdir='/Users/juanitagomez/Documents/SpyderDocs')`
- 11/11/2016 12:00:00 AM: `weather_data_by_year = aggregate_by_year(weather_data_ordered, 'Formatted Date')`
- 11/11/2016 12:00:00 AM: `plot_correlations(weather_data_ordered, size=15)`
- 11/11/2016 12:00:00 AM: `if b==0: return a`
- 11/11/2016 12:00:00 AM: `else: return gcd(b, a%b)`
- 11/11/2016 12:00:00 AM: `runfile('/Users/juanitagomez/Local/cubic.py', wdir='/Users/juanitagomez/Local')`
- 11/11/2016 12:00:00 AM: `millerRabin(15)`

# Thanks

Samatrix Consulting Pvt Ltd