Published in The Startup

Zaheeda Tshankie Follow
Jun 30, 2020 · 6 min read · Listen

How to set up a Data Science Project

credit

More often than not, those of us who are new to the Data Science field wish to transform newly acquired Data Science skills into something tangible. This is often done by means of embarking on a project.

Whether you're a beginner, a graduate, looking to build a Portfolio for your CV or to prepare for projects in your future workplace — this is the post for you!

"Where do I start? How do I structure my project? What do I do with project results once I'm done?" Just some questions I know I used to have in the early stages of my Data Science journey/career.

I want to provide a protocol: basic steps to beginning

The topics I'll cover are:

1. Creating a Git Repository
2. Building a Project folder structure
3. Creating a Virtual Environment
4. Notebooks
5. Module/Script Writing

Create a Git Repository

> *Git is a version-control system for tracking changes in source code during software development.*

Git basically let's you host your project online which allows your work to be shareable. On it, you can update and track any changes you make to your project. The Git Repository is what collect these files of various different versions of a Project.

I see it more as a platform to manage and structure my work. It's the one thing in my life I keep neat and tidy :")

Steps:

1. Head on over to a site like GitHub or GitLab and create a free account.
2. Create a Repository for your project. It is where you will push all the work and progress of a project that you work on.

Confused? I recommend a git short course with videos (here), or a tutorial with coding (git

> *The code for a project, app or software component is typically organized in a folder structure or "file tree".*

Once you've created a Repository, you will need to decide how to structure your project. The typical "Tree" structure I use for most of my Machine Learning projects is as follows:

```
|-- my_project
|   |-- data
|   |   |-- example_data.csv
|   |-- notebooks
|   |   | -- example_notebook.ipynb
|   |--modules/scripts (optional)
|   |   | -- __init__.py
|   |   | -- example_script.py
|   |-- setup
|   |   |-- setup.sh
|   |   |-- requirements.txt
|-- README.md

4 directories, 5 files
```

In plain English:

In the project (i.e. the Repository) you just created, create the folders: `data` , `notebooks` and `setup` . Put the necessary files into their respective folders.

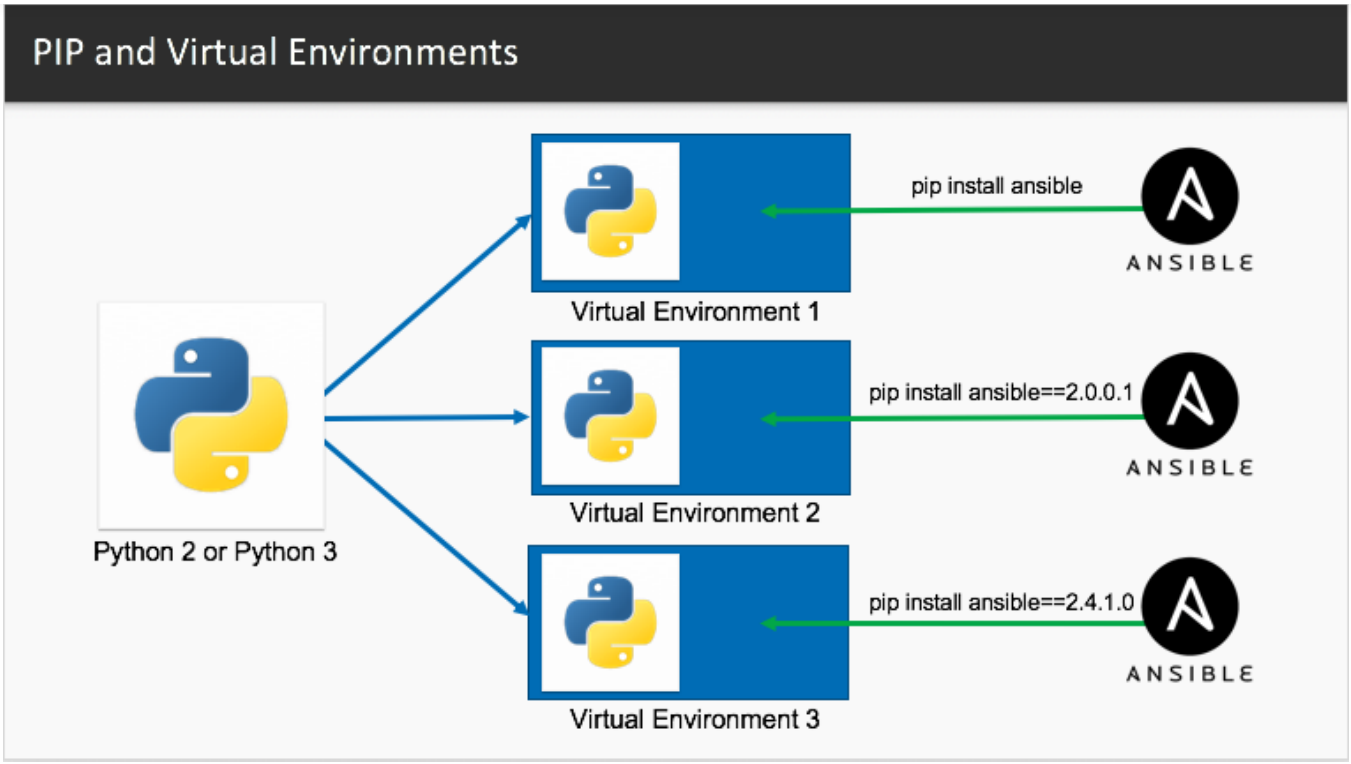
The `README.md` file is a file that is automatically created when you create your Git Repository, in it you outline a summary of your project i.e. the Problem Statement.

- *Note for the data folder: Rather provide external links where the data can be downloaded. It's unsafe to keep (confidential) data on a Repo.*

Create a Virtual Environment

> *The main purpose of virtual environments is to create an isolated environment for projects. This means that each project can have its own dependencies (installed packages), regardless of what dependencies every other project has.*



In short, for each project you begin you will install and use different packages OR

you may use the same packages for projects — only difference being the version of the package. The version control depends on which Python Version your project requires. (Image on left).

An example of the prior: You may need Scikit-Learn in a simple Machine Learning project you begin and Tensorflow in a different Deep Learning project you begin. So for each project you create an environment with **only the packages you need.**

This procedure also prevents import errors you may encounter, because you will install all required packages beforehand using the “setup” folder we created in our Tree Structure.

Steps using the command-line:

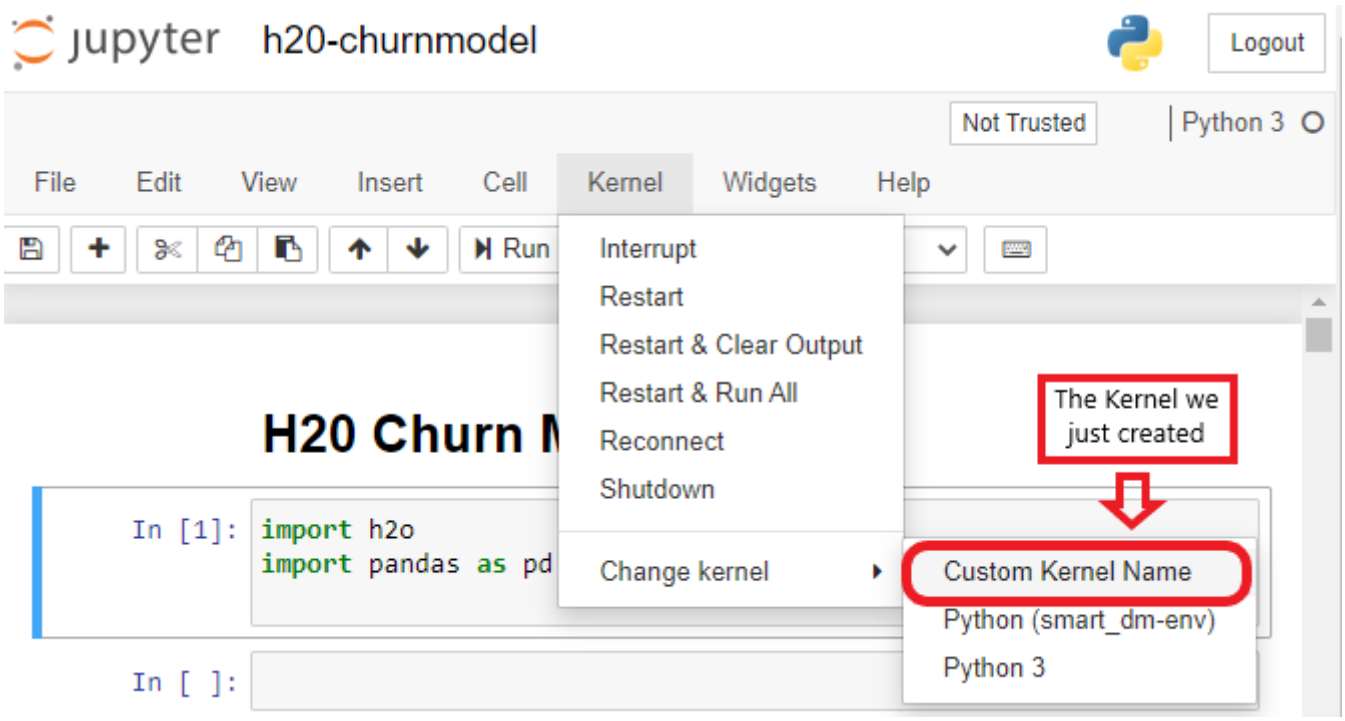
In the `setup` folder , create the `setup.sh` bash file, when you run this file in the terminal using the command: `$ source setup.sh` , it automatically runs the code within it which installs all needed packages:

```
# create the virutual environment in the project root
pip3 install virtualenv
virtualenv -p python3 project_name_env

# activate the virtual environment
source project_name_env/bin/activate

# install packages you will need
pip3 install -r setup/requirements.txt
```

Bonus: Create a Custom Kernel. It applies the Virtual Environment you just created and "activates" it in your notebook to make sure you use only the packages in the environment.

```
python3 -m ipykernel install – user – name project_name_env –
display-name "Custom Kernel Name"
```

Result:



Notebooks

Ahhh Thee Notebook, what a marvelous creation it is :"D

Clicking *Shift + Enter* is addictive (for me at least 😅), it's just so simple. Notebooks give added motivation to just hit the ground running on a project you've been itching to smash!

The objective of this entire article, however, is to outline a way to avoid making a Notebook the only method you have to represent /"host" a project.

In practice, in the workplace for example, Notebooks are only required as a way to show examples of how the code you've written or the model you've built work when implemented.

Module/Script Writing

What to do with the code you were meant to put in a Notebook? After creating the project structure as outlined in this Article, you could wish to work backwards when writing script. You could create and work on a Notebook, take the lines of code that are related to one-another and make them functions (`def function()` vibes).

After compiling the functions, place them into python `.py` files. Those files are your modules. For more structure, you could put related `.py` files into separate folders with

To test your scripts, you'd import them into your Notebook that shows examples of how your code/model works.

Confused? I found a short tutorial here.

— —

At the end of all these steps, you'll have a well packaged project which you can refer back to when required or be able to share Git links to Recruiters as a Portfolio of your Resume.

If you ***commit*** (excuse the pun) to starting all your projects this way, it will become a part of your routine. Your Git coding and collaboration skills will get stronger and you'll bag some new skills to put onto your CV.

Let's make a pledge:

That from the day you read this Article you will put some thought behind structuring your ML projects. Pledge that you will fight the urge of relying overly on Notebooks.