
PLURALSIGHT

Data Engineering

THE HARTFORD

Tech Catalyst Bootcamp

Tarek Atwan
Instructor, Pluralsight

PLURALSIGHT

STANDUP

What are some of the challenges you faced last week?

How did you resolve it?
What did you learn from it?

What was something “new” or interesting things you learned last week?

Python for Data Engineering

Concepts Covered

- Basic Terminal Commands
- GitHub SSH Setup
- Python Development Environments
- Python Virtual Environments
- Reading Data with Python (Basics)
- Intro to Pandas (DataFrame and Series)
- I/O Operations in Pandas
- Data Transformations in Pandas
  - Missing Data
  - Date/Time/UTC/POSIX
  - Merging Data
  - Other Data Enrichment

INTRO

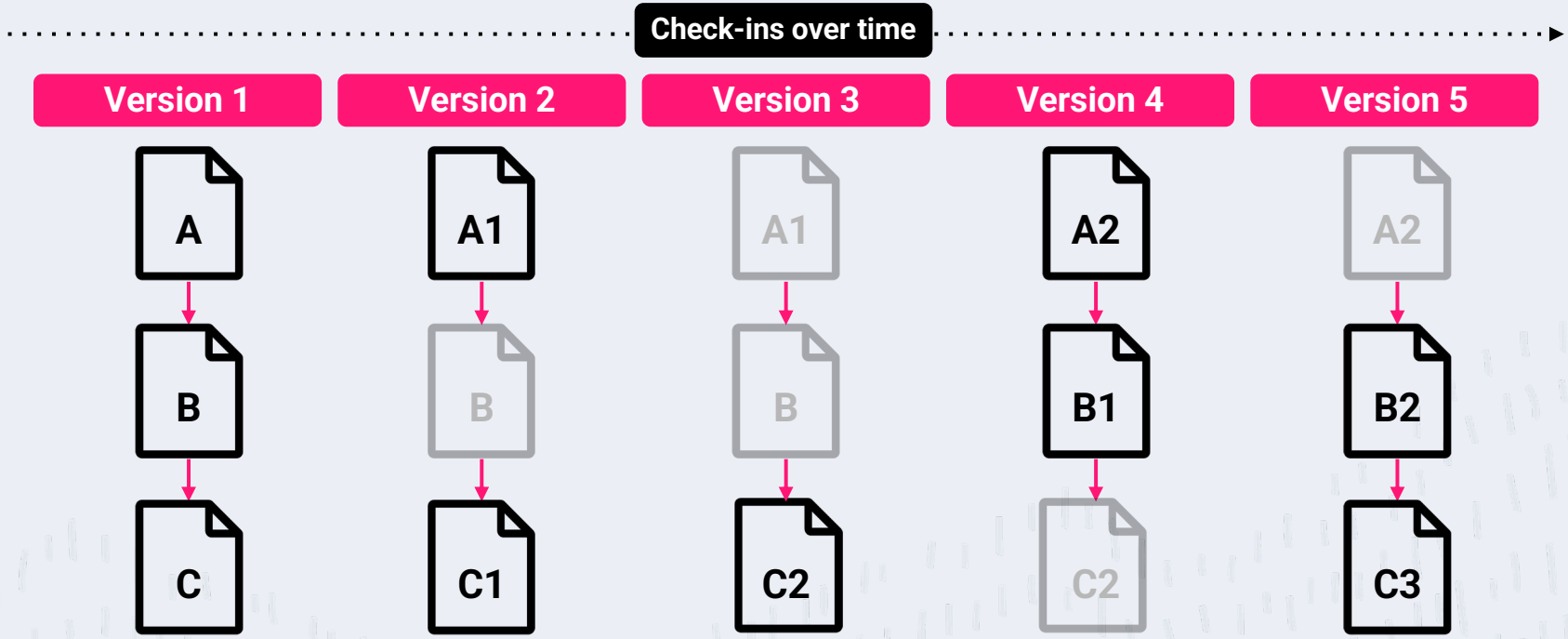
Terminal, Git, and GitHub Basics

Basic Linux Commands

`cd`	changes the directory.
`cd ~`	changes to the home directory.
`cd ..`	moves up one directory.
`ls`	lists files in the folder.
`pwd`	shows the current directory.
`Mkdir <FOLDERNAME>`	creates a new directory with the FOLDERNAME
`touch <FILENAME>`	creates a new file with the FILENAME.
`rm <FILENAME>`	deletes a file.
`rm -r <FOLDERNAME>`	deletes a folder; note the -r.
`open .`	opens the current folder on Macs.
`explorer .`	opens the current folder on Windows.
`open <FILENAME>`	opens a specific file on Macs.
`explorer <FILENAME>`	opens a specific file on Windows.

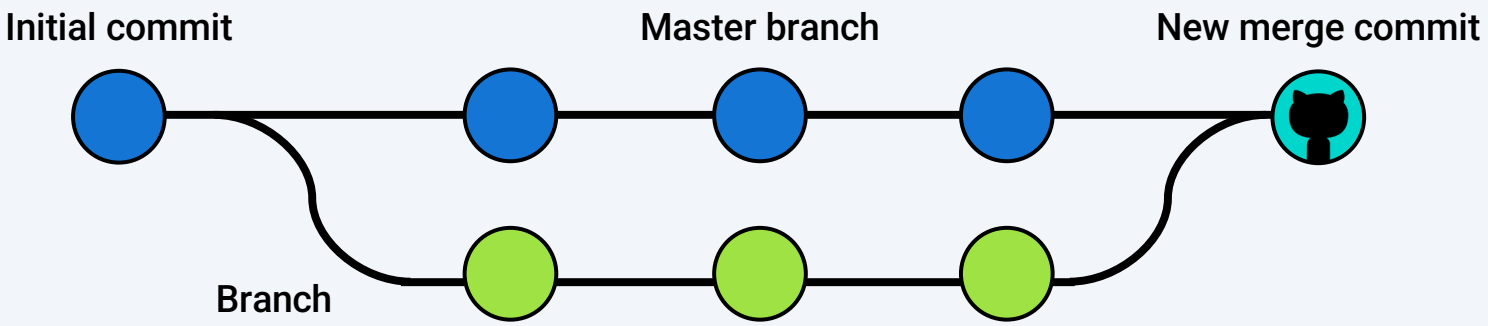
Quick Intro to Git

Git is essentially a way for us to keep track of our work over time. Whenever we get another piece of a project working, we can save the change with Git.



Git Commit

A Git “save” is called a **commit**. It represents a checkpoint for our project where we save and describe our work.

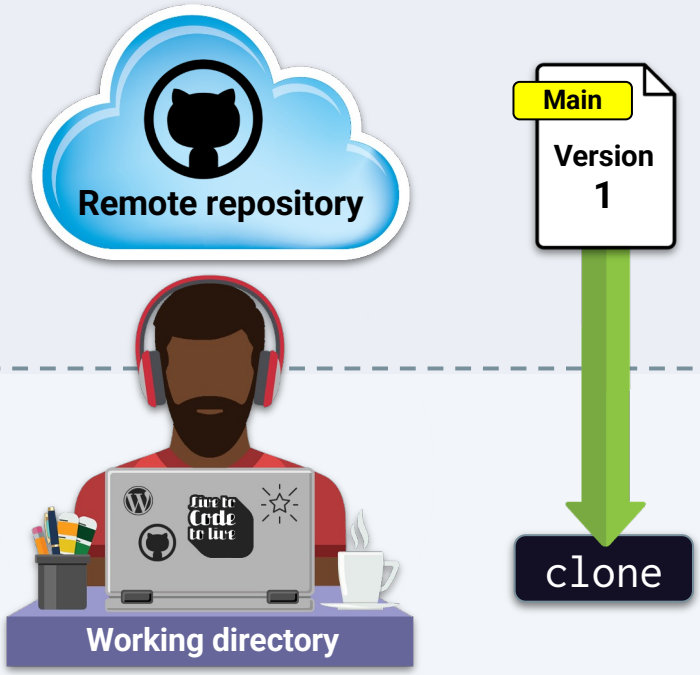


If we break something while working on our code, this system allows us to restore working code from earlier. Since Git remembers these checkpoints,
we can work on several different concerns all at once.

Git Version Control

Scenario: Your group has been working with Uber's rider data, and you've decided to analyze the average age of the riders:

The root code for the project is called `main`.



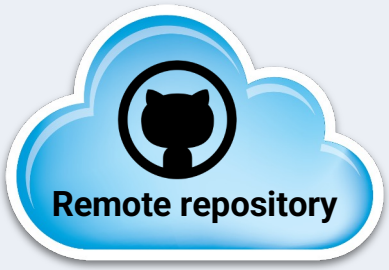
The **staging area** is where you edit the files that will be part of the next commit.

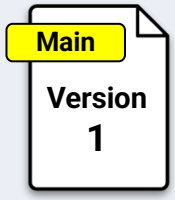
Staging area

Takes an existing GitHub repository and downloads it to the local computer, and links it to GitHub.

Git Version Control

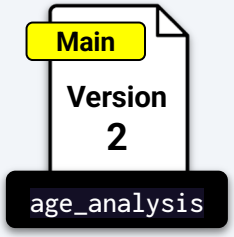
Git essentially allows us to write this code and save it with the name `age_analysis`.





The **staging area** is the where you edit the files that will be part of the next commit.

Staging area

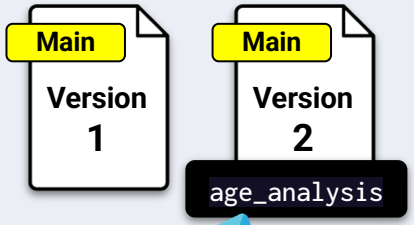


`git commit`

Your staged changes are saved once you commit.

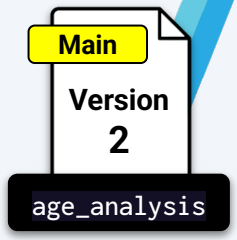
Git Version Control

`age_analysis` is a branch that originates from the main branch. It contains updates that will be added to the main branch when it's ready to `merge`.



The **staging area** is the where you edit the files that will be part of the next commit.

Staging area



Popular Git CLI Commands

`git clone`	Clones a git repository onto the local file system.
`git add`	Adds changed files to the queue of tracked files ready to be committed.
`git commit`	Adds tracked files as a bulk checkpoint ready to be pushed to the remote git repository.
`git push`	Uploads changed files from the local git repository to the remote git repository and updates the remote files.
`git pull`	Downloads changed files from the remote git repository to the local git repository and updates the local files.

A commit in GitHub is like a snapshot of what your project or file looks like at a particular moment in time. If a file doesn't contain any changes, the file is not stored again; instead,
Git provides a link to the identical file that it previously stored.

Git and GitHub (In Class Activity)

1. Generating a new SSH key and adding it to the ssh-agent
2. Adding a new SSH key to your GitHub account

Managing

Python Virtual Environments

Typical Python Development Environment Options

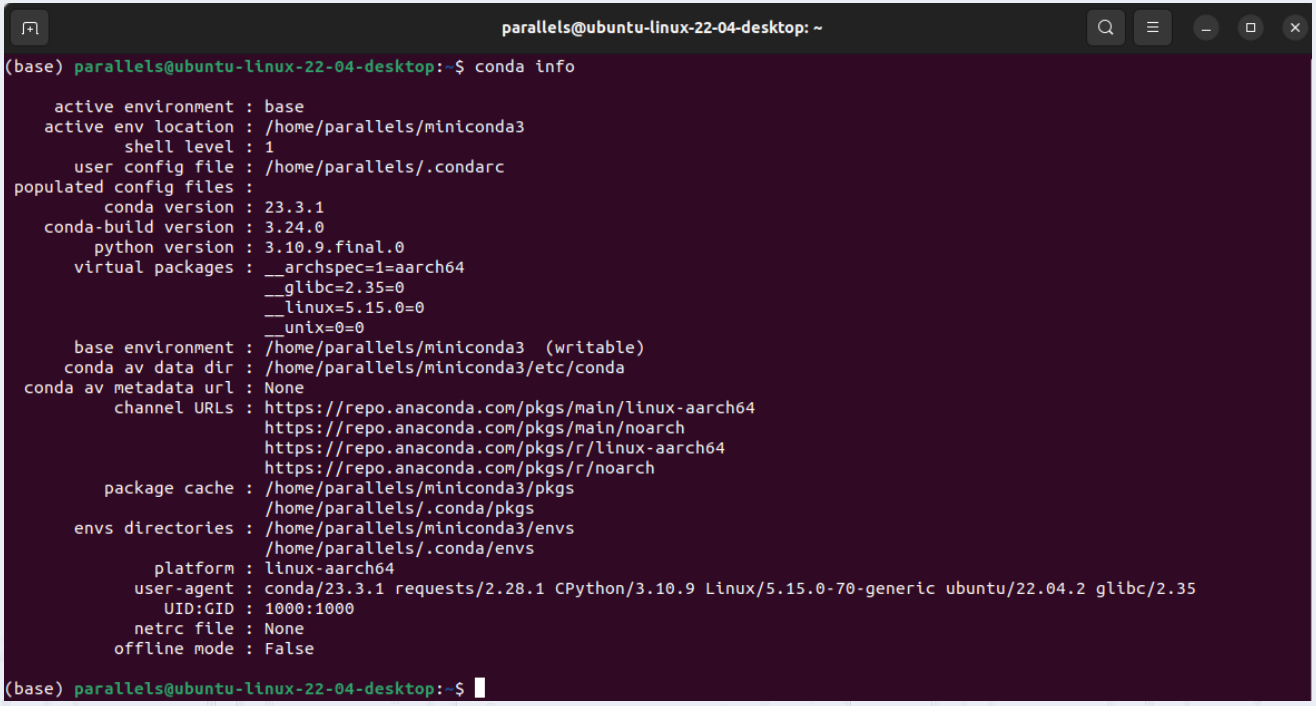
- Install through a Python distribution such as **Anaconda** *(https://www.anaconda.com/products/distribution)*, which comes preloaded with all the essential packages and supports **Windows**, **Linux**, and **macOS** *(including M1 support as of version 2022.05)*. Alternatively, you can install **Miniconda** *(https://docs.conda.io/en/latest/miniconda.html)* or **Miniforge** *(https://github.com/conda-forge/miniforge)*.
- Download an installer directly from the official **Python** site *https://www.python.org/downloads/*.
- If you are familiar with **Docker**, you can download the official Python image. You can visit Docker Hub to determine the desired image to pull *https://hub.docker.com/_/python*.
- Similarly, **Anaconda** and **Miniconda** can be used with Docker by following the official instructions here: *https://docs.anaconda.com/anaconda/user-guide/tasks/docker/*

Alternative Options

- Google Colab https://colab.research.google.com
- Kaggle https://www.kaggle.com
- AWS SageMaker Studio Lab https://studiolab.sagemaker.aws
- Replit https://replit.com
- VSCode (online) https://vscode.dev

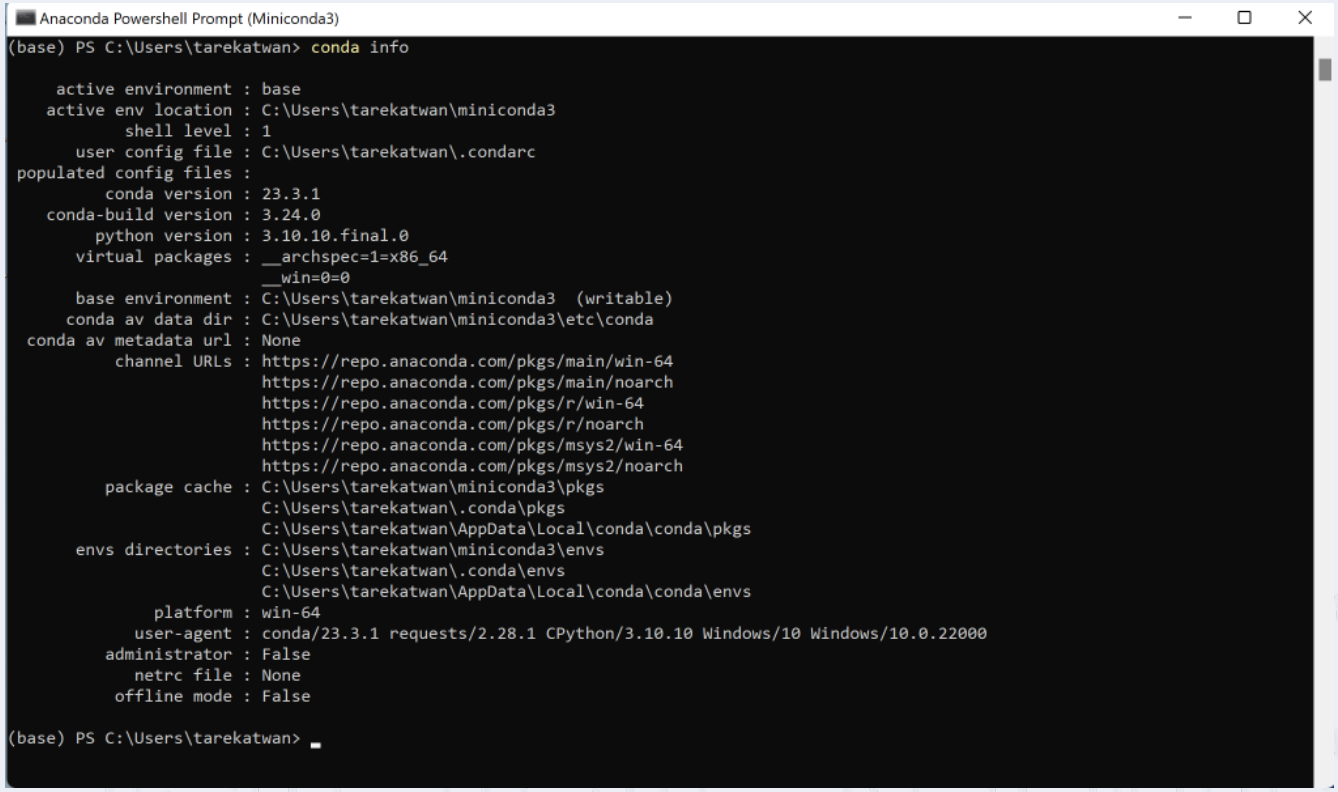
Conda on Linux/MacOS

```
$ conda info
```



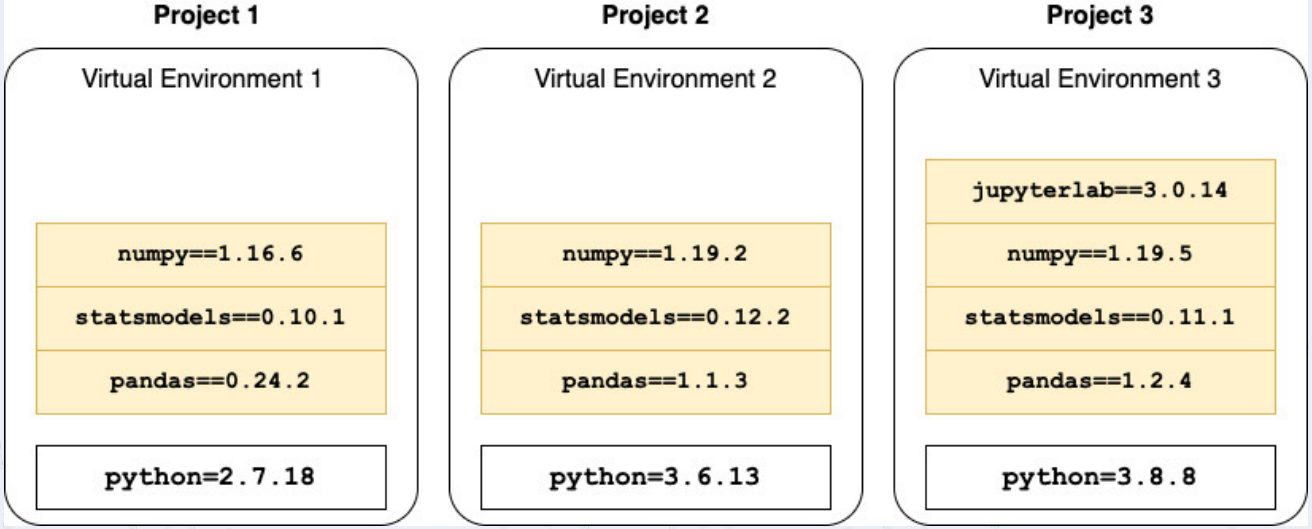
Conda on Windows

If you installed Anaconda on a Windows OS, you need to use the **Anaconda Prompt**



What Are Python Virtual Environments?

You can think of a virtual environment as isolated buckets or folders, each with a **Python interpreter** and **associated libraries**. The following diagram illustrates the concept behind isolated, self-contained virtual environments, each with a different Python interpreter and different versions of packages and libraries installed:



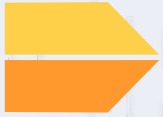
What Are Python Virtual Environments?

What is a virtual environment?

Virtual environments create an isolated environment for Python projects.

You may work on different projects that have different dependencies.

Different projects might also use different types and versions of libraries.



This virtual environment ensures you have the required dependencies for future class activities.

Conda vs Venv

Both **conda** and **venv** allow you to create multiple virtual environments for your Python projects that may require different Python interpreters (for example, 3.4, 3.8, or 3.9) or different Python packages

conda provides both package dependency management and environment management for Python (and supports many other languages).

venv is a built-in Python module which provides environment management and requires no additional installation.

Using Conda

```
$ conda create -n py310 python=3.10
```

Here, `-n` is a shortcut for `--name`

```
$ conda create -n py310 python=3.10 -y
```

To activate the newly created environment

```
$ conda activate py310
```

Using Conda

```
$ conda info --envs
```

```
❯ conda info --envs
# conda environments:
#
base                     /opt/anaconda3
dev                      /opt/anaconda3/envs/dev
py310                 *  /opt/anaconda3/envs/py310
```

```
$ conda install pandas
$ conda install pandas=2.0.1
```

Using Conda

```
$ conda config --add channels conda-forge
```

```
$ conda deactivate
```

```
$ conda env remove -n py310
```

Using Venv

```
$ python -m venv py310
```

```
$ source py310/bin/activate
```

On Windows using Anaconda PowerShell Prompt, there is no **`bin`** subfolder, so you will need to run the command using the following syntax, again assuming you are running the command from the **`Desktop`** directory:

```
$ .\py310\Scripts\activate
$ .\py310\Scripts\Activate.ps1
```

There are two activate files under the Scripts folders: `activate.bat` and `Activate.ps1` where the latter to be used with Anaconda PowerShell Prompt instead of the Anaconda Windows Command Prompt

Creating a Virtual Environment using YAML

You can create a virtual environment from a **YAML** file. This option gives greater control in defining many aspects of the environment, including all the packages that should be installed all in one step

```
$ conda env create -f env.yml
$ conda activate tscookbook
```

```yaml
# A YAML for creating a conda environment
# file: env.yml
# example creating an environment named tscookbook

name: tscookbook
channels:
  - conda-forge
  - defaults
dependencies:
  - python=3.10
  - pip
  # Data Analysis
  - statsmodels
  - scipy
  - pandas
  - numpy
  - tqdm

  # Plotting
  - matplotlib
  - seaborn

  # Machine learning
  - scikit-learn
  - jupyterlab
```

Bootsraping and Environment

You can also bootstrap your YAML file from an existing environment. This is very useful if you want to share your environment configurations with others or create a backup for later use

```
$ conda env export -n py310 > env.yml
$ conda env export -n py310 -f env.yml
$ conda env export -name py310 -file env.yml


This will generate the env.yml file for you in the current directory.
```

Cloning

You can also a Python virtual environment from another Python virtual environment

```
$ conda create --name py310_clone --clone py310
```

Installing Packages

Option 1: Create a new virtual environment and install specific packages

```
$ conda create --name ch1 --file requirements.txt
$ conda create -n ch1 --file requirements.txt
$ conda env create --name ch1 --file requirements.txt
$ conda env create -n ch1 -f requirements.txt
```

Installing Packages using Conda

Option 2: Install specific packages into an existing Python environment

```
$ conda activate timeseries
$ conda install --file requirements.txt
```

Installing Packages using Venv and Pip (MacOS/Linux)

:Create and then activate the venv environment before you install the packages. The following code assumes the requirements.txt file is in the Downloads folder:

```
$ python -m venv ~/Desktop/timeseries
$ source Desktop/timeseries/bin/activate
$ pip install -r ~/Downloads/requirements.txt
```

Installing Packages using Venv and Pip (Windows)

Create and activate the venv environment and then install the packages.

```
$ python -m venv .\Desktop\timeseries
$ .\Desktop\timeseries\Scripts\activate
$ pip install -r .\Downloads\requirements.txt
```

Pip Freeze

pip freeze allows you to export all pip-installed libraries in your environment

```
$ source ~/Desktop/ch1/bin/activate
$ pip freeze > ~/Downloads/requirements.txt
$ cat ~/Downloads/requirements.txt
```

Conda list into

You can also export the list of packages using Conda

```
$ conda activate ch1
$ conda list -e > ~/Downloads/requirements.txt
$ cat ~/Downloads/requirements.txt
```

Conda from-history

. If you want to export only the packages that you explicitly installed (without the additional packages that conda added), then you can use conda env export command with the `--from-history` flag

```
$ conda activate ch1
$ conda env export --from-history > ~/Downloads/environment.yml
$ cat ~/Downloads/environment.yml
```

Naming your environment for Jupyter

```
python -m ipykernel install --user  --name timeseries --display-name "Time Series"
```

Using nb_conda_kernels

This extension enables a Jupyter Notebook or JupyterLab application in one conda environment to access kernels for Python, R, and other languages found in other environments

```
https://github.com/anaconda/nb_conda_kernels
```

Data Analytics

Intro to Python (Code Along)

Reading Files in Python

`with` is a special syntax block that allows us to perform operations that require a safety clean-up after the code block is completed.

`open<File Path>, <Read/Write>` is the function that Python uses to open a file. By specifying either `'r'`, `'w'`, or `'rw'`, we can read from a text file, write to a text file, or perform both operations.

`text.read()` reads the entire file and converts it to a string type.

```
# Open the file in "read" mode ('r') and store the contents in the variable "text"
with open(file, 'r') as text:

      # Store all of the text inside a variable called "lines"
      lines = text.read()

      # Print the contents of the text file
      print(lines)
```

Reading CSV files in Python

Comma Separated Values

- **CSV** stands for **comma separated values**. This file type is essentially a table that has been converted into text format with each row and column separated by specified symbols.
- More often than not, each row is located on a new line, and each column is separated by a comma, as indicated in the name “CSV.”

```
First Name,Last Name,Phone
Janetta,Bolduc,499-820-0212
Bent,Hanburry,125-890-6291
Kath,Beeres,807-511-9864
Clarisse,Surgeon,499-224-5982
Hae-Won,Park,727-224-1623
Rodd,Camier,199-541-8033
Javier,Martinez,543-206-4422
Patty,De'Ath,950-579-4341
Charlie,Clewlow,874-246-8418
Izel,Xiu,362-965-1637
Zechariah,Spikings,570-486-2219
Stephie,Tootal,326-912-0003
```

Python has a module called `csv` that pulls in data from external CSV files and allows users to perform operations on the data.

Reading CSV files in Python

`os` module

This module allows Python programmers to easily create dynamic paths to external files that will function across different operating systems.

```
# First we'll import us module
# This will allow us to create file path across operating systems
import os

# Module for reading CSV files
import csv

csvpath = os.path.join('..', 'Resources', 'contacts.csv')
```

Reading CSV files in Python

`csv` reader

Instead of `text.read()`, this new code instead utilizes `csv.reader()` to translate the object being opened by Python. Note the `delimiter=','` parameter, which tells Python that each comma within the CSV should be seen as moving into a new column for a row.

```
with open(csvpath) as csvfile:

    # CSV reader specifies delimiter and variable that holds contents
    csvreader = csv.reader(csvfile, delimiter=',')

    print(csvreader)

    # Read the header row first (skip this step if there is now header)
    csv_header = next(csvreader)
    print(f"CSV Header: {csv_header}")

    # Read each row of data after the header
    for row in csvreader:
        print(row)
```

Data Analytics

Intro to Pandas

Thank you!

If you have any additional questions, please reach out to me at: (email address).