LIVE ONLINE TRAINING

Business Analytics With Python Bootcamp

Week 2

Agenda

1. **Recap & Intro**
   (15 minutes)

2. **Getting Started With Python (90 minutes)**
   - Exercise: Install Anaconda with Jupyter Notebooks and print “Hello World!”
   - Break

3. **Essential coding best practices (60 minutes)**
   - Exercise: Build your first repo
   - Break

4. **Running your code (60 minutes)**
   - Exercise: Work with VS Code and the command line
   - Outlook for next week

Recap

Bootcamp overview

Learning goals:

- Derive actionable insights from data
- Perform exploratory data analysis and create meaningful visualizations
- Use value-based analysis techniques and create association rules for effective decision support
- Apply clustering techniques to discover segments in your data, e.g. different customer groups
- Build predictive models for regression and classification tasks
- Understand the key criteria for evaluating the performance of a predictive model
- Suggest specific business actions that will lead to better results

Bootcamp overview

Last week

Week 1 Jan 31	Week 2 Feb 7	Week 3 Feb 14	Week 4 Feb 21	Week 5 Feb 28	Week 6 Mar 7
Business Analytics Fundamentals	**Data Analysis With Python Fundamentals**	**Descriptive Business Analytics With Python**	**Diagnostic Business Analytics With Python**	**Predictive Business Analytics With Python**	**Prescriptive Business Analytics With Python**
Goal: Get the theoretical foundations for doing business analytics	**Goal:** Get the practical foundations for using Python for data analysis	**Goal:** Learn how to do explore & describe your data (What happened?)	**Goal**: Learn how to get insights from your data (Why did it happen?)	**Goal:** Learn how to predict the future, given historical data (What will happen?)	**Goal:** Suggest better actions based on data (What to do?)

NOTE: With today's registration, you'll be signed up for all six sessions. Although you can attend any of the sessions individually, it's recommended participating in all six weeks.

Quiz time!

Which of these is NOT a primary goal for business analytics?

A) Solve business problems by analyzing data

B) Make better business decisions

C) Make business processes more efficient

D) Organizing data in a better way

What are the three major types of business processes?

A) Planning, Designing, and Executing

B) Management, Sales, and Marketing

C) Management, Operational, and Supporting

D) Customer Service, R&D, and Finance

What does MECE stand for in the context of Issue Trees?

A) Mutually Exclusive, Continuously Evolving

B) Mutually Exclusive, Collectively Exhaustive

C) Marginally Exhaustive, Continuously Evolving

D) Marginally Exhaustive, Completely Exclusive

What does Mutually Exclusive mean?

A) There are no overlaps

B) There are no gaps

C) There’s a complete overlap

D) There are no gaps and no overlaps

What’s the best way to start a data analysis project?

A) Start with the data

B) Start with a problem

C) Hire a consultant

D) Grab a coffee

Bootcamp overview

Week 1 Jan 31	Week 2 Feb 7	Week 3 Feb 14	Week 4 Feb 21	Week 5 Feb 28	Week 6 Mar 7
Business Analytics Fundamentals	**Data Analysis With Python Fundamentals**	**Descriptive Business Analytics With Python**	**Diagnostic Business Analytics With Python**	**Predictive Business Analytics With Python**	**Prescriptive Business Analytics With Python**
Goal: Get the theoretical foundations for doing business analytics	**Goal:** Get the practical foundations for using Python for data analysis	**Goal:** Learn how to do explore & describe your data (What happened?)	**Goal**: Learn how to get insights from your data (Why did it happen?)	**Goal:** Learn how to predict the future, given historical data (What will happen?)	**Goal:** Suggest better actions based on data (What to do?)

NOTE: With today's registration, you'll be signed up for all six sessions. Although you can attend any of the sessions individually, it's recommended participating in all six weeks.

Bootcamp overview

Learning goals Week 2:

- Getting started with Python – what is it, why is it so popular?
- Understanding the Python Ecosystem
- Knowing the most popular packages for data analysis
- Differentiating between Notebooks vs. script files
- Install Anaconda & Jupyter Notebook on your computer
- Learn essential coding best practices (versioning, clean code, functions, …)
- Build your first code repository
- Organizing your code files
- Running scripts from the command line

Getting Started With Python

What is Python?

- Python is a simple and easy to read high-level programming language
- Created in the late 1980s by Guido van Rossum as a more user-friendly alternative to low-level programming languages such as C and C++
- Open source and maintained by the Python Software Foundation
- Name: Inspired by “Monty Python’s Flying Circus”

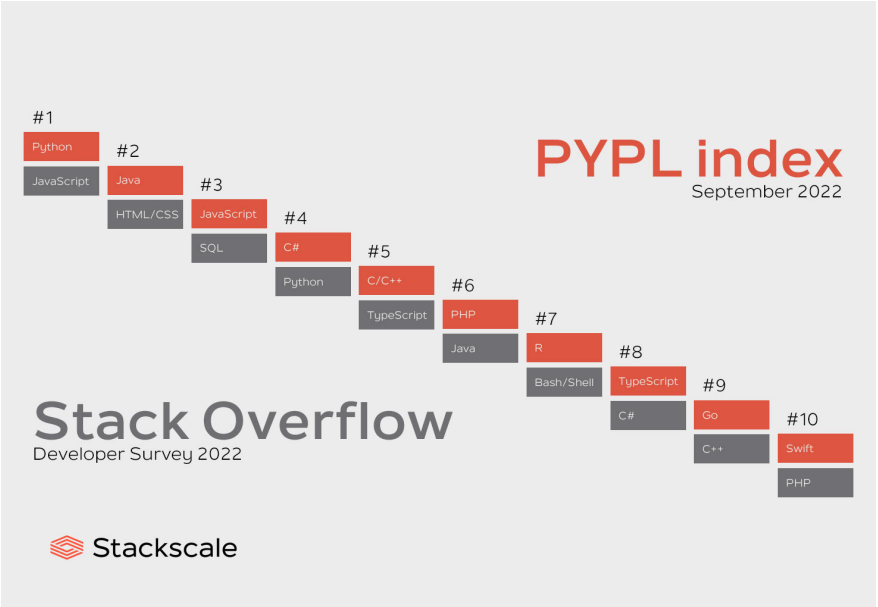
Python success stories

- **Google**: YouTube is a big user of Python
- **Instagram**: Started being built on Django
- **Netflix**: Python for user recommendations
- **Facebook**: 21% of the internal code base is Python
- **Spotify**: Heavy use of Python for data analysis
- **Dropbox**: Hired Guido van Rossum away from Google in 2012 – Python stack
- **Pinterest**: Python + Django
- **NASA**: Python used across different processes
- **Uber**: Python powers most services running Uber

- Source: https://www.linkedin.com/pulse/power-python-yogesh-raghupati/?articleId=6659675925707399168

What is Python?

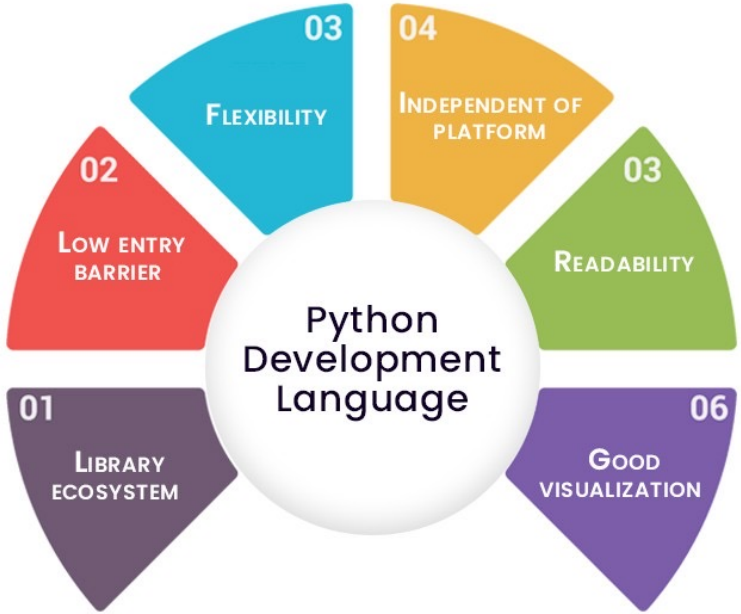
- Python is one of the most popular programming languages worldwide!
- Key concepts in Python include:
  - **Variables and data types**: Store and manipulate data in a program
  - **Control flow**: if/else statements and loops to control execution order
  - **Functions**: reusable code blocks that can be called from other parts
  - **Modules**: Libraries containing Python code that can be imported
  - **Object-Oriented Programming**: using classes and objects to model real-world concepts and encapsulate data and behavior.



- https://www.linkedin.com/pulse/power-python-yogesh-raghupati/?articleId=6659675925707399168

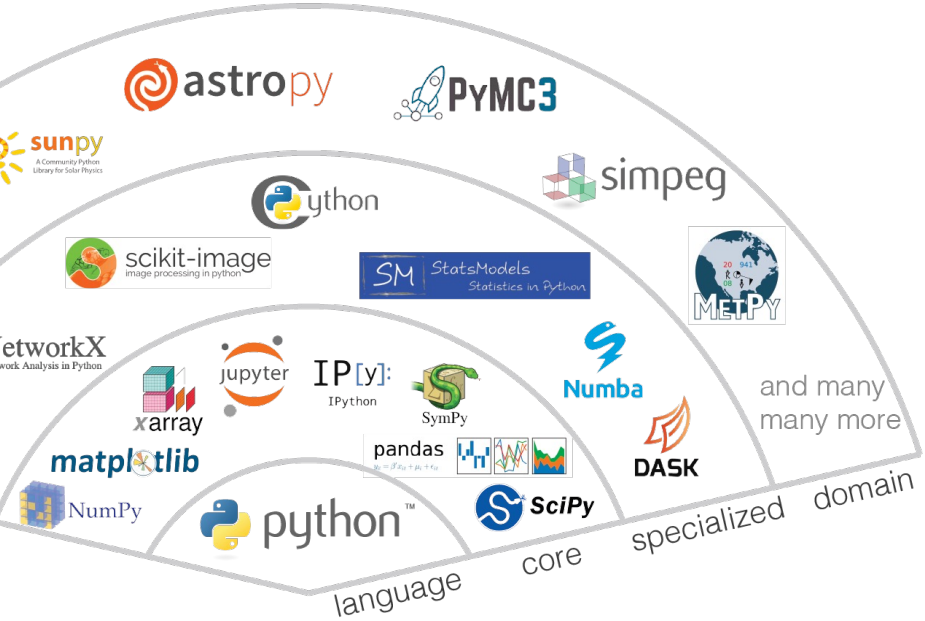
Why is Python so popular?

- Python is widely used for various applications:
  - Web Development,
  - Data Analysis,
  - Artificial Intelligence
  - Scientific computing
  - Gaming
  - Automation
  - …
- Large and active community, wealth of libraries and tools available to help with almost any task.
- Easy to learn with a relatively short learning curve



- Source: https://www.stackscale.com/blog/most-popular-programming-languages/

Python ecosystem



- Source: jupytearth.org

- Python offers a **large ecosystem** of packages, frameworks & tutorials
- Python is general purpose, that's why multiple "add-ons" for various domains have been developed by the Python community

Example Data Analysis:

- Python is not a native programming language for data analysis (unlike R!)
- Example: Does not know tables as a data type
- The increasing popularity of Python for data analysis was driven by modules that made doing data analysis with Python a lot easier

Popular Packages for data analysis

Numpy	Numerical operations on multidimensional arrays / vectorization
Pandas	Introduces tables (DataFrame) as a new data type
Matplotlib	A plotting library for creating data visualizations
Seaborn	Based on Matplotlib, provides advanced plotting interface
Scikit-learn	Machine learning library offering plenty of ML algorithms

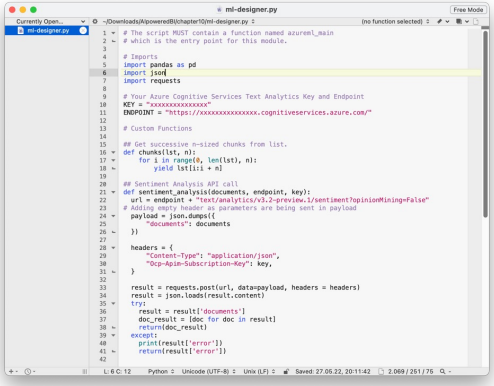
Typical Python Workflow

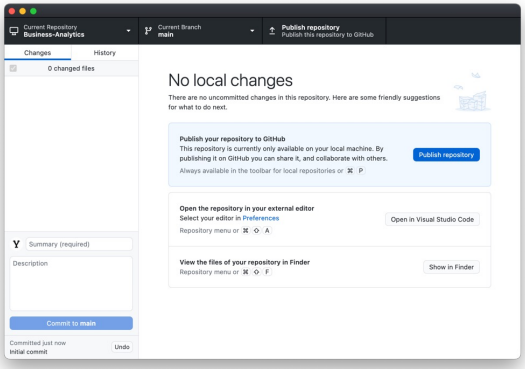
Write code → **Store code** → **Run code**

Developing Python code, testing and debugging.

Storing, organizing and sharing code

Executing code when it's needed

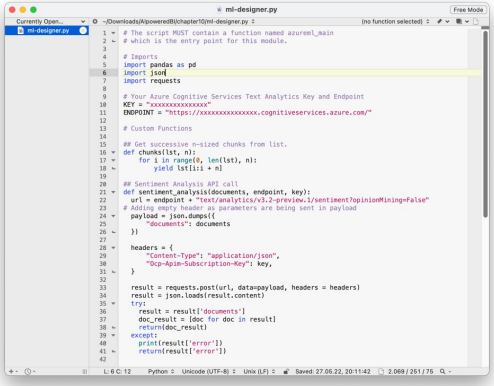




Typical Python Workflow

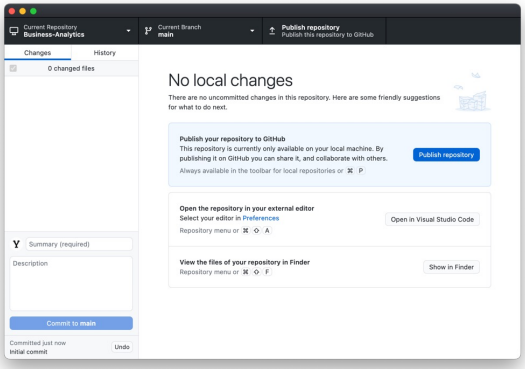
Write code

Developing Python code, testing and debugging.



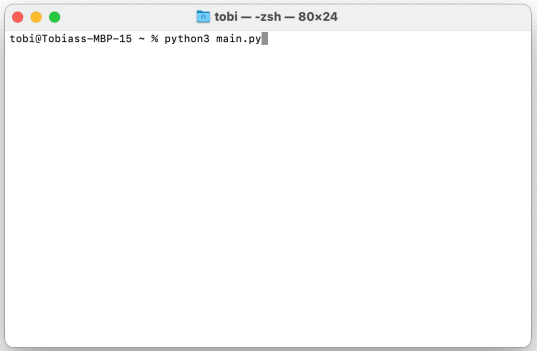
Store code

Storing, organizing and sharing code



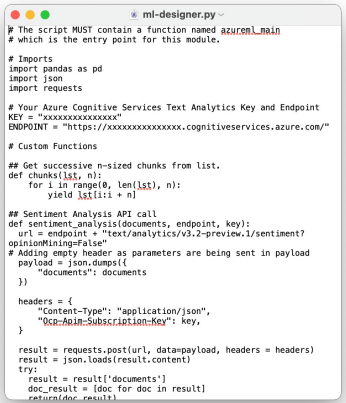
Run code

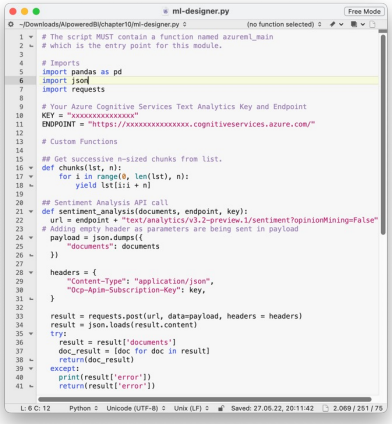
Executing code when it's needed

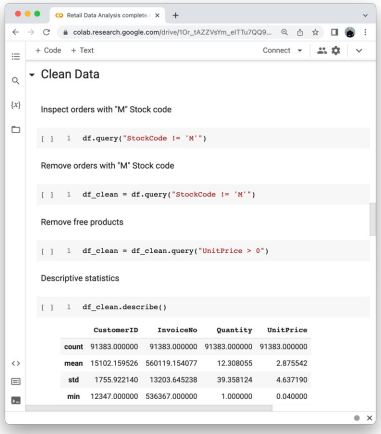


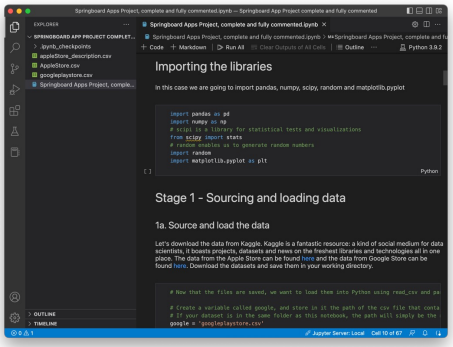
Options for Writing Code

Text editor	Code editor	Notebooks	IDE
Notepad, TextEdit, …	Notepad++, Sublime Text, Vim, BBedit, …	Jupyter, Colab, Azure Notebooks, …	VS Code, PyCharm, Spyder, IDLE, Jupyter Lab, …









Complexity of software project

Options for Writing Code

Text Editor

- Most lightweight option
- Typically pre-installed
- No features (just a text file)

Use for:

- Small edits / changes
- When you don't have anything else

ml-designer.py

```
# The script MUST contain a function named azureml_main
# which is the entry point for this module.

# Imports
import pandas as pd
import json
import requests

# Your Azure Cognitive Services Text Analytics Key and Endpoint
KEY = "xxxxxxxxxxxxxx"
ENDPOINT = "https://xxxxxxxxxxxxxx.cognitiveservices.azure.com/"

# Custom Functions

## Get successive n-sized chunks from list.
def chunks(lst, n):
    for i in range(0, len(lst), n):
        yield lst[i:i + n]

## Sentiment Analysis API call
def sentiment_analysis(documents, endpoint, key):
  url = endpoint + "text/analytics/v3.2-preview.1/sentiment?
opinionMining=False"
# Adding empty header as parameters are being sent in payload
  payload = json.dumps({
      "documents": documents
  })

  headers = {
      "Content-Type": "application/json",
      "Ocp-Apim-Subscription-Key": key,
  }

  result = requests.post(url, data=payload, headers = headers)
  result = json.loads(result.content)
  try:
    result = result['documents']
    doc_result = [doc for doc in result]
    return(doc_result)
```

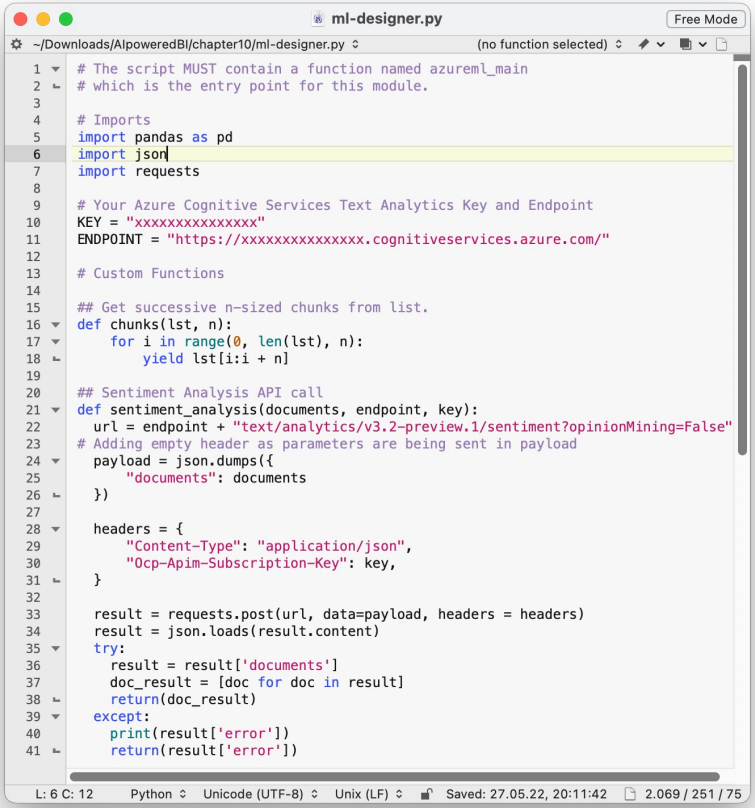
Options for Writing Code

Code Editor

- Still very lightweight
- ”Portable apps”
- Basic functionality like syntax highlighting
- Can also be very powerful (extensions)

Use for:

- Few files / small projects
- Get started



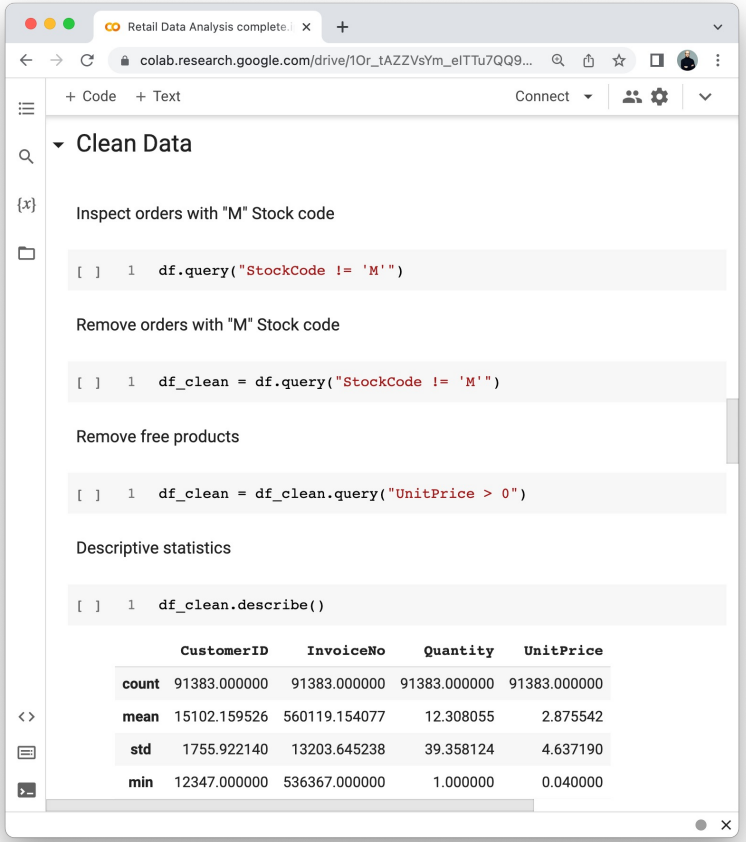
Options for Writing Code

Notebooks

- Combine code + markup
- Combine writing + running code
- High interactivity
- Visual outputs

Use for:

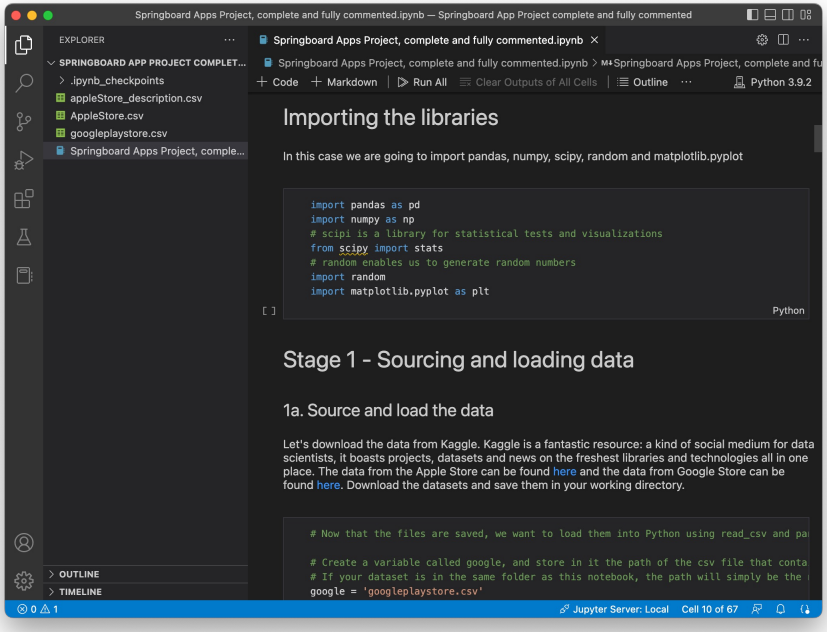
- Data analysis
- “Input – output” workflows
- Few files



Options for Writing Code

IDEs

- File management / version control built-in
- Syntax highlighting + code completion
- Support both script files and notebooks
- Not so lightweight



Use for:

- Complex software projects
- Many files
- Many collaborators

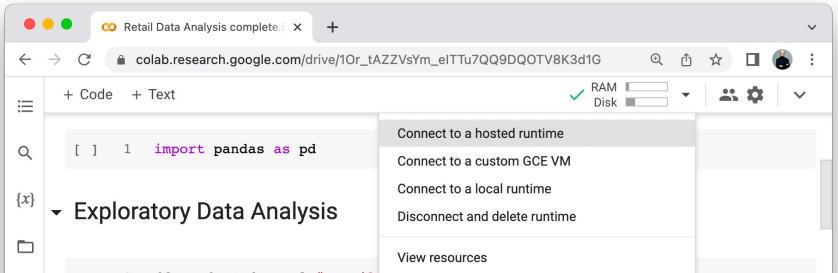
Notebooks vs. Scripts

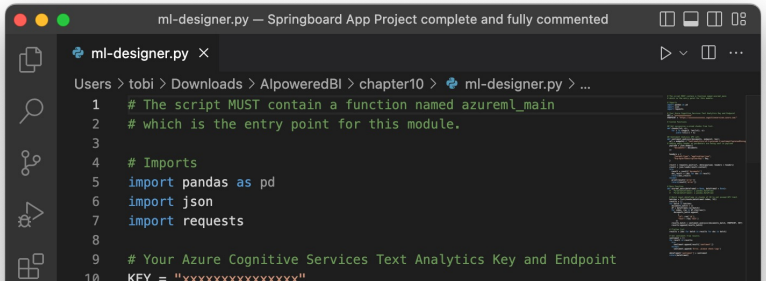
Use Notebooks:

- For exploratory data analysis and prototyping, quickly iterate and visualize your results.
- Need to present your work in a less-technical readable format
- Collaborate with others and share your work

Use Script files:

- Production-level code with many modular files and libraries
- Need for automated scheduling or integrate it into a larger workflow.
- Reuse code across projects (functions and classes defined in script files can be imported into other scripts and notebooks)

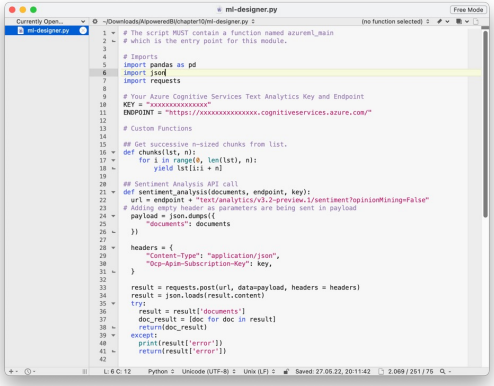




Typical Python Workflow

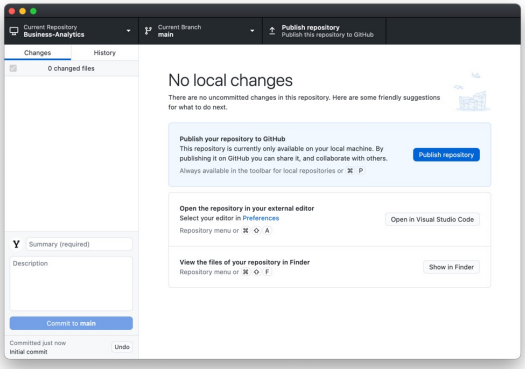
Write code

Developing Python code, testing and debugging.



Store code

Storing, organizing and sharing code



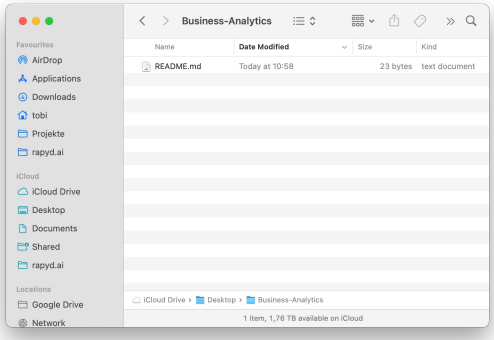
Run code

Executing code when it's needed

Options for Storing Code

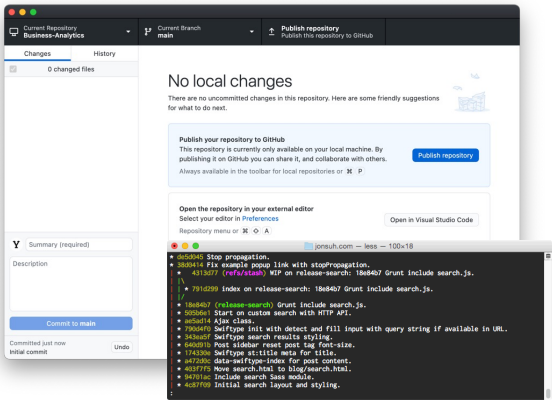
As a file on a disk

Explorer, Finder, GDrive, …



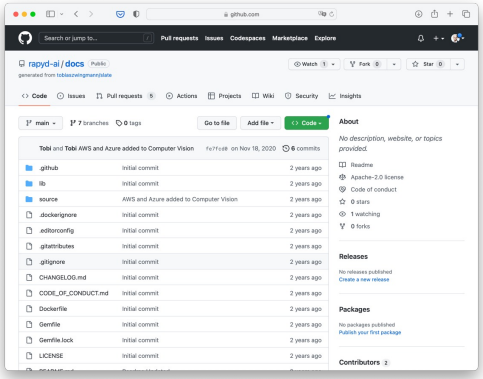
In a local repository

Git, SVN, …



In a remote repository

Github, Bitbucket, Gitlab, …

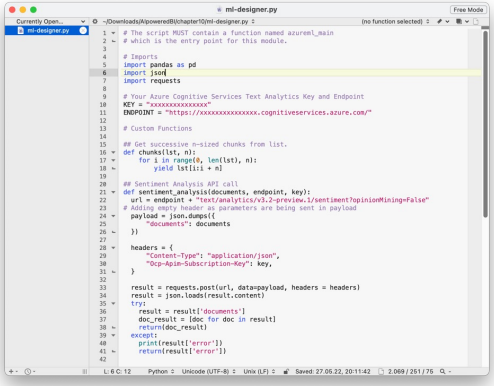


Number of collaborators

Typical Python Workflow

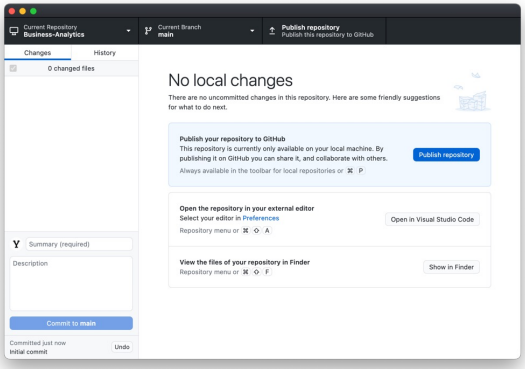
Write code

Developing Python code, testing and debugging.



Store code

Storing, organizing and sharing code



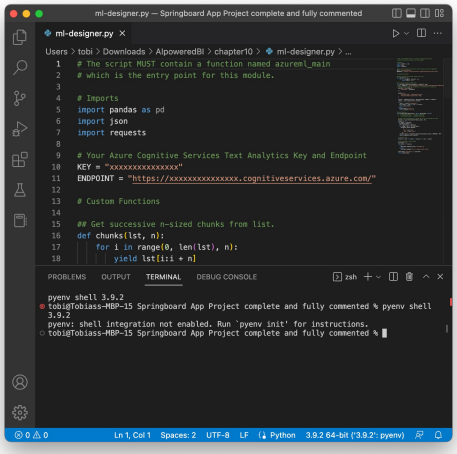
Run code

Executing code when it's needed

Options for Running Code

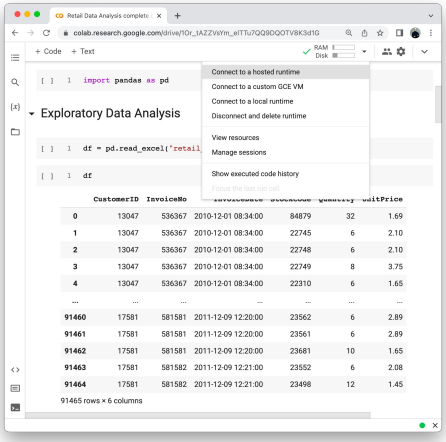
From an IDE

Jupyter Lab, VSCode, …



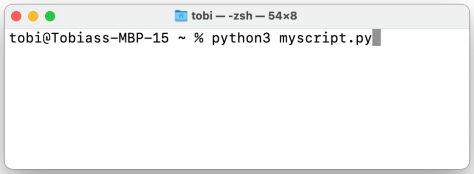
Within a notebook

Jupyter Notebooks, …



From the command line

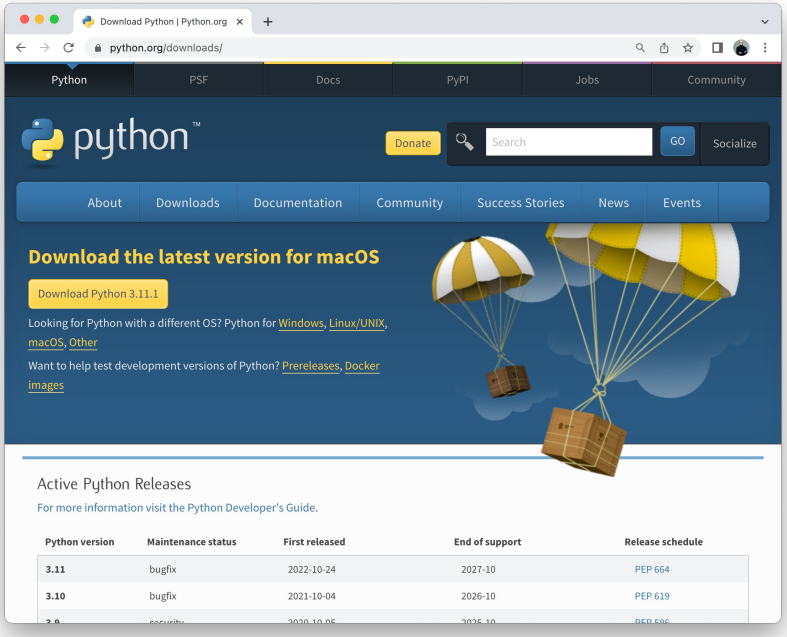
Terminal, PowerShell, …



Automation

How to Run Python Code

- To run Python code, you need to have a **Python interpreter** installed on your computer.
- The Python interpreter, also known as the Python kernel, acts as a bridge between the code you write and the computer.
- It takes the Python code and translates it into instructions that the computer can understand and execute.
- https://www.python.org/downloads/ (← What you get when you search “Python Download”!)
- Typically need admin rights for installing



Python Distributions

- Pre-packaged version of the Python interpreter that comes with a set of libraries and tools (Jupyter, Spyder, etc.)
- Popular Python distribution: Anaconda
- Includes a wide range of packages, package manager (conda), Anaconda Navigator GUI, …
- **Pros**: Installs everything with one click
- **Cons**: Installs many things you don't need…

Python Distributions

Miniconda

- Miniconda is a smaller version of Anaconda that includes the package- and environment manager conda, Python and a small number of other packages

Choose Anaconda if you:

- Prefer working with a GUI over command line
- Want Python with 1,500+ packages preinstalled
- Have the time and disk space (3 GB!).

Choose Miniconda if you:

- Don’t need the Anaconda Navigator GUI
- Want to install the packages you want to use individually.
- Do not have time or disk space to install over 1,500 packages at once.

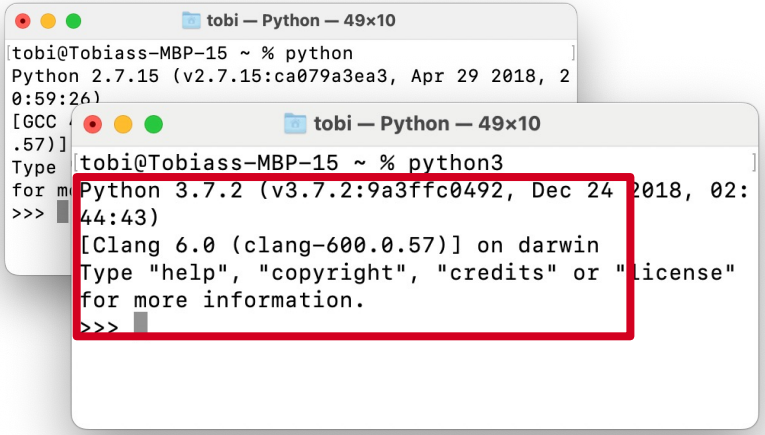
Installing Python

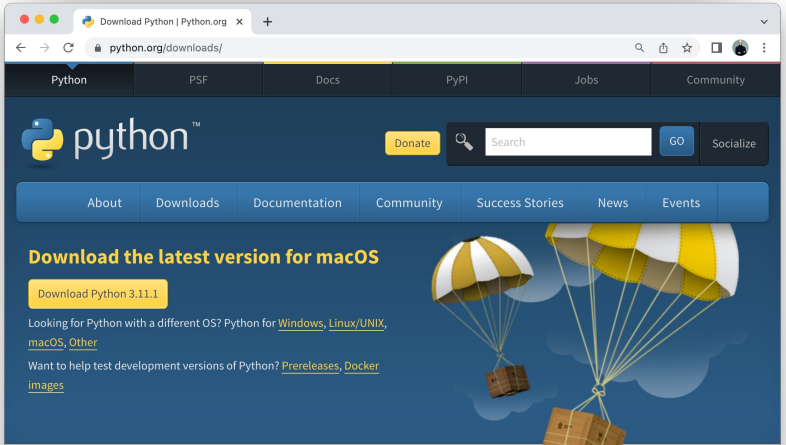
Step 1: Check if Python is pre-installed!

Python is pre-installed on many systems:

- Open Terminal / PowerShell and type “**python**”, ”**python3**” or “**py**”

- Note: There’s Python 2 and Python 3
  - Python 2 deprecated since 2020 (But don’t mess around with existing Python 2 installations!)
  - Use Python 3! (Python 3.11+) (Don’t upgrade unless you use environments – previous versions could be in use by other software)
  - Download Python here: https://www.python.org/downloads/





1. Install Anaconda &
Launch Jupyter Notebooks
2. Print ”Hello World”

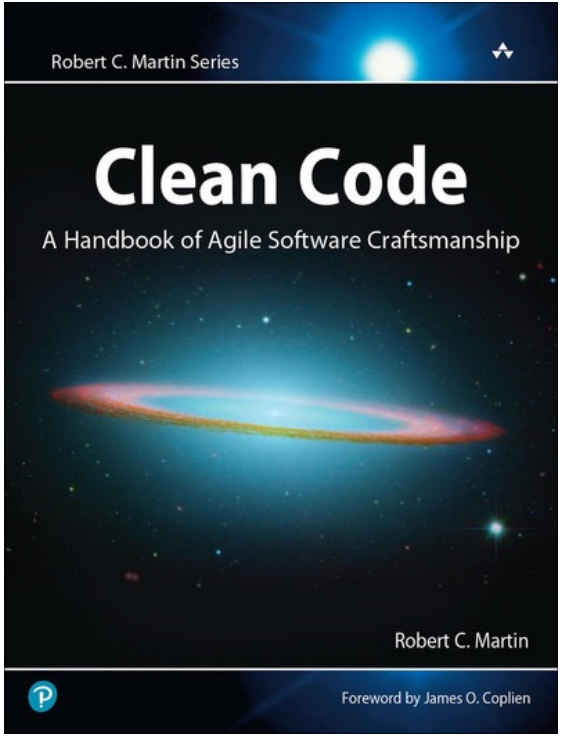
Q&A

Essential Coding Best Practices

Essential coding best practices

- Your goal is not to become a software engineer, **BUT**…
- …knowing the basics can still be still very helpful and make your life a lot easier!

Source: Clean Code: A Handbook of Agile Software Craftsmanship, R.C. Martin, Pearson 2008



Essential coding best practices

- There are good and bad ways to write code– even if both codes may return the same result

Good code is...

- easy to read
- simple to understand
- fast to execute
- free of redundancies

Bad code is...

- hard to read
- difficult to understand
- slow to execute
- full of redundancies

Essential coding best practices

→ Writing “clean” code is an art and skill in itself

Remember: You’re not trying to become a software engineer, BUT anticipating “clean” code as much as possible is hugely beneficial for:

- Your coworker
- **Your future self!**

Rule of thumb:

→ **Write the code for your future self!**

Source: tommasolizzul | iStock | Getty Images

Clean Code

- **What is clean code?**
- *I like my code to be elegant and efficient.*

  The logic should be straightforward to make it hard for bugs to hide, the dependencies minimal to ease maintenance, error handling complete according to an articulated strategy, and performance close to optimal so as not to tempt people to make the code messy with unprincipled optimizations.

  Clean code does one thing well.

Bjarne Stroustrup, inventor of C++ and author of The C++ Programming Language

- Source: Clean Code: A Handbook of Agile Software Craftsmanship, R.C. Martin, Pearson 2008

Clean Code: Naming things

- Names are everywhere: variables, functions, arguments, classes, packages, source files, directories etc.
- **Use Intention-Revealing Names:**
  - Poor: `d`
  - Better: `elapsed_time_in_days`
- **Avoid Disinformation**
  - Poor: `account_list` *(DataFrame)*
  - Better: `accounts`
- **Make Meaningful Distinctions**
  - Poor: `a1, a2`
  - Better: `source, destination`

- Source: Clean Code: A Handbook of Agile Software Craftsmanship, R.C. Martin, Pearson 2008

Clean Code: Naming things

- **Use Pronounceable Names:**
  - Poor: `genymdhms`
  - Better: `generationTimestamp`
- **Use Searchable Names**
  - Poor: `e`
  - Better: `sum`
- **Method / Function Names**
  - Poor: `csv_writer`
  - Better: `write_csv`
- **Don’t be funny**
  - Poor: `HolyHandGrenade`
  - Better: `DeleteItems`

- Source: Clean Code: A Handbook of Agile Software Craftsmanship, R.C. Martin, Pearson 2008

Clean Code: Writing Functions

- Functions are the first line of organization in any program.
- **Small:**
  - Rule of thumb: 2 - 20 lines of code
- **Do One Thing**
  - Example: Calculate a value, save a CSV file, sort a DataFrame, etc.
  - *”Do one thing. Do it well. Do it only”*
- **Arguments**
  - Ideal number of arguments: 0
  - Most common: 1 - 2
  - Danger zone: 3+ arguments
  - Don’t use flag arguments (True / False)

```python
def median(lst):
    lst = sorted(lst)
    n = len(lst)
    m = n // 2
    if n % 2 == 0:
        return (lst[m - 1] + lst[m]) / 2
    else:
        return lst[m]
```

Clean Code: Writing Functions

- **No side effects:**
  - Don’t make any unexpected changes to other variables
  - Do one thing!
- **Output arguments**
  - Don’t mix inputs and outputs
  - Example: **`appendFooter(s)`**
    → Does this function append s as the footer to something? Or does it append some footer to s? Is s an input or an output?
  - Better: **`report.appendFooter()`**
    → Output arguments should generally be avoided. If your function must change the state of something, have it change the state of its owning object.

Clean Code: Writing Functions

- **Error handling:**
  - Exceptions vs. Error codes
  - Poor:

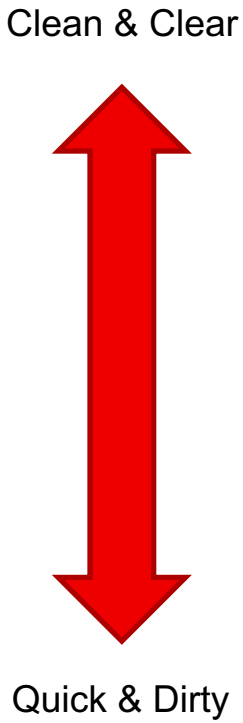
```
if deletePage(page) == E_OK:
    if registry.deleteReference(page.name) == E_OK:
        if configKeys.deleteKey(page.name.makeKey()) == E_OK:
            logger.log("page deleted")
        else:
            logger.log("configKey not deleted")
    else:
        logger.log("deleteReference from registry failed")
else:
    logger.log("delete failed")
    return E_ERROR
```

Better:

```
try:
    deletePage(page)
    registry.deleteReference(page.name)
    configKeys.deleteKey(page.name.makeKey())
except Exception as e:
    logger.log(str(e))
```

Clean Code: Writing Functions

- **Golden Rule of Writing Functions:**
  - Don’t repeat yourself!
- Avoid repetition in your code by writing functions, …
  - …but some repetition is still fine (2-3) – *don’t let you slow down too much!*
- **Rule of thumb:**
  - More than 3 repetitions: Write a function!
- **Don’t be too dogmatic!**
  (Balance the needs of your present and future self!)



Clean Code: Comments

Golden Rule of Writing Comments:

- Don’t commend bad code – rewrite it!
- Consider comments a necessary evil – if our programming were expressive enough, we would not need comments at all.
- Poor:

```
# Check to see if the employee is eligible for full benefits
if (employee.flags and HOURLY_FLAG) and (employee.age > 65):
```

- Better:

```
if employee.isEligibleForFullBenefits():
```

- Beginner tip: When in doubt, write a comment!

Clean Code: Comments

Types of comments:

- Legal comments

```
# Copyright (C) 2003,2004,2005 by Object Mentor, Inc. All rights reserved.
# Released under the terms of the GNU General Public License version 2 or later.
```

- Intent

```
# JSON to pd.DataFrame for easier handling
```

- Clarification

```
# Finds a string pattern starting with the letter M and ending with a digit
regex = "^M.*\d$"
```

- TODO

```
# TODO-MdM these are not needed
# We expect this to go away when we do the checkout model
```

Methods vs. Functions

- A method is a function **associated with an object**.
- It can access or alter the object

```
pd
.pivot_table(df_clean, index = ['Segment'], values = ['Revenue'], aggfunc = 'sum' )
.plot
.bar();
```

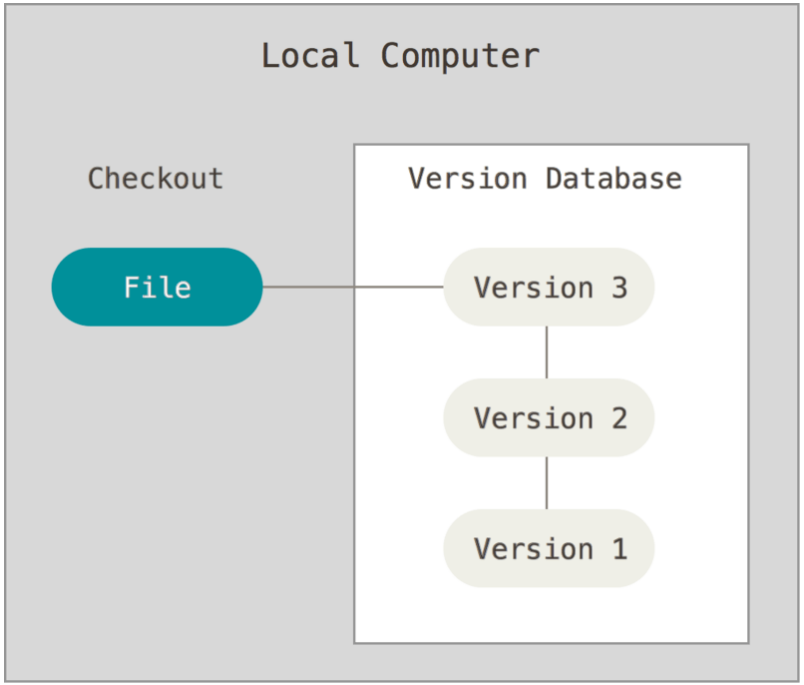
- A function is a **standalone block** of code for a particular task.
- Typically, no dependency to other objects

```
def flag_wholesale(quantity):
  quantity_avg = 12.3
  quantity_std = 39.36

  if quantity >= (quantity_avg + 3 * quantity_std):
    return "B2B"
  else:
    return "B2C"
```

Code Versioning

- Code versioning (also known as source control or revision control) is a system that **tracks changes made to source code** over time
- Allows multiple developers to work on a project simultaneously
- Makes it easier to revert to previous versions if necessary.
- **Git** is a popular code versioning tool.



- Source: Git

Code Versioning - Terminology

The main concepts of code versioning and Git are:

- **Repository**: Central location where all the code and its history are stored.
- **Commits**: Saved (group of) changes to the code in the repository. Each commit has a unique identifier and contains a message explaining what was changed.
- **Branches**: Branches allow you to work on different versions of the code in the same repository simultaneously. The default branch is usually called "master", but other branches can be created for features, bug fixes, or other purposes.

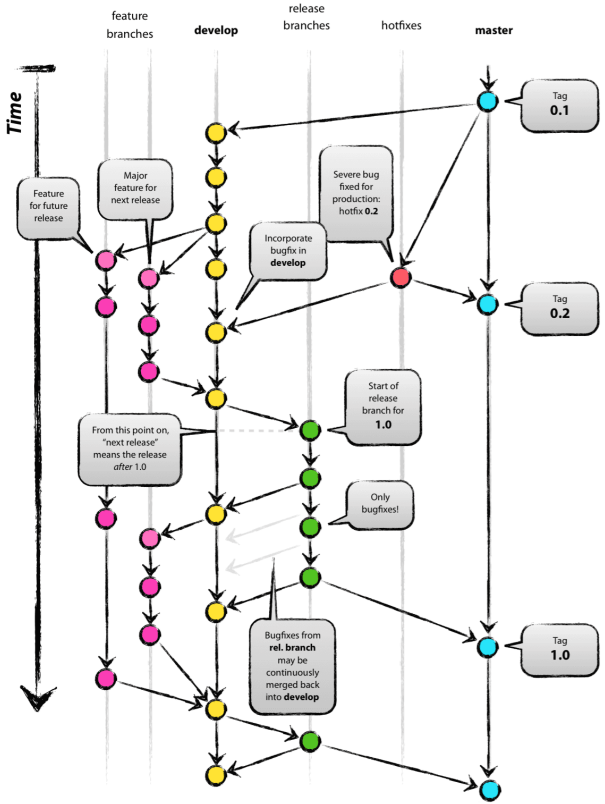
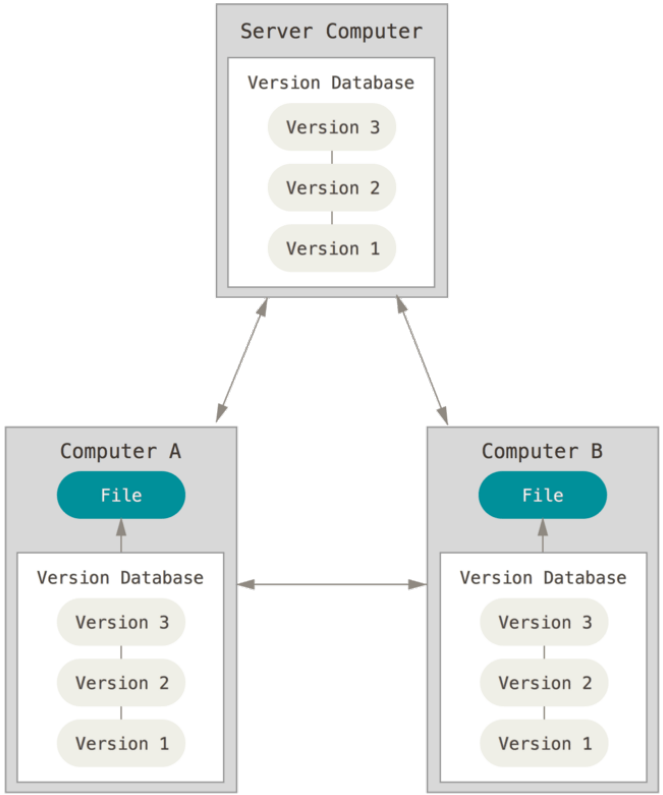


Image source: https://nvie.com/posts/a-successful-git-branching-model/

Code Versioning - Terminology

- **Merging**: When changes from one branch need to be incorporated into another branch, a process called merging is used. This can either be done automatically by Git or manually, with conflicts resolved by the developer.
- **Pull Requests**: A pull request is a way to propose changes to a repository's code. It allows other developers to review and discuss the changes before they are merged into the main branch.
- **Distributed Version Control:** Every clone is really a full backup of all the data. If the server dies, any client can restore the repo.



- Source: Git

Code Versioning - Benefits

Why is code versioning important?

- **Collaboration**: Makes it easier for multiple people to work on the same codebase without stepping on each other's toes.
- **Backup and recovery**: Complete history of all changes made to the code, making it easier to revert to a previous version if necessary.
- **Transparency**: Makes it easier to see who made changes to the code, when they were made, and why.
- **Auditing**: Keeping track of code changes makes it easier to audit the codebase and ensure that it adheres to coding standards and security requirements.

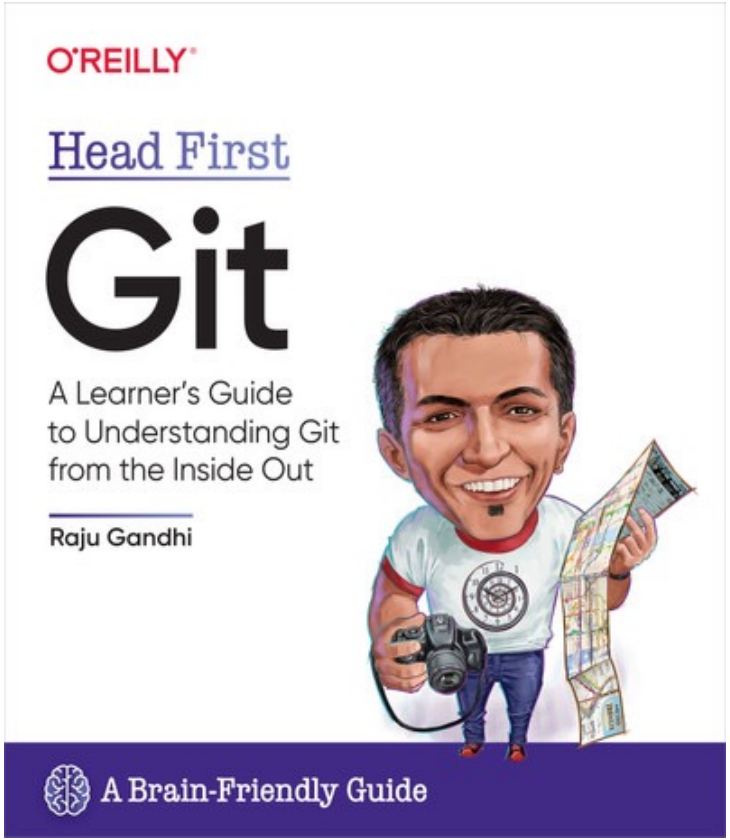
How to use Git

Using Git from the command line 101:

- **Install Git on your computer**: Download from https://git-scm.com/downloads
- **Create a Git repository**: To create a new Git repository, navigate to the directory where your code is located and run the command **`git init`**. This will initialize a new Git repository in that directory.
- **Add files to the repository**: Use the **`git add`** command to add files to the repository. For example, to add all the files in the current directory, you can run **`git add .`**
- **Commit changes:** Save changes by running **`git commit -m "Commit message"`**.
- **Push changes to a remote repository**: If you want to store your repository on a remote server (such as GitHub), you can push your changes to the remote repository using the **`git push`** command.
- **Clone a repository**: To start working with an existing Git repository, you can clone it to your local machine using the git clone command, followed by the URL of the repository. For example, to clone a repository hosted on GitHub, you can run **`git clone https://github.com/user/repo.git`**

How to use Git

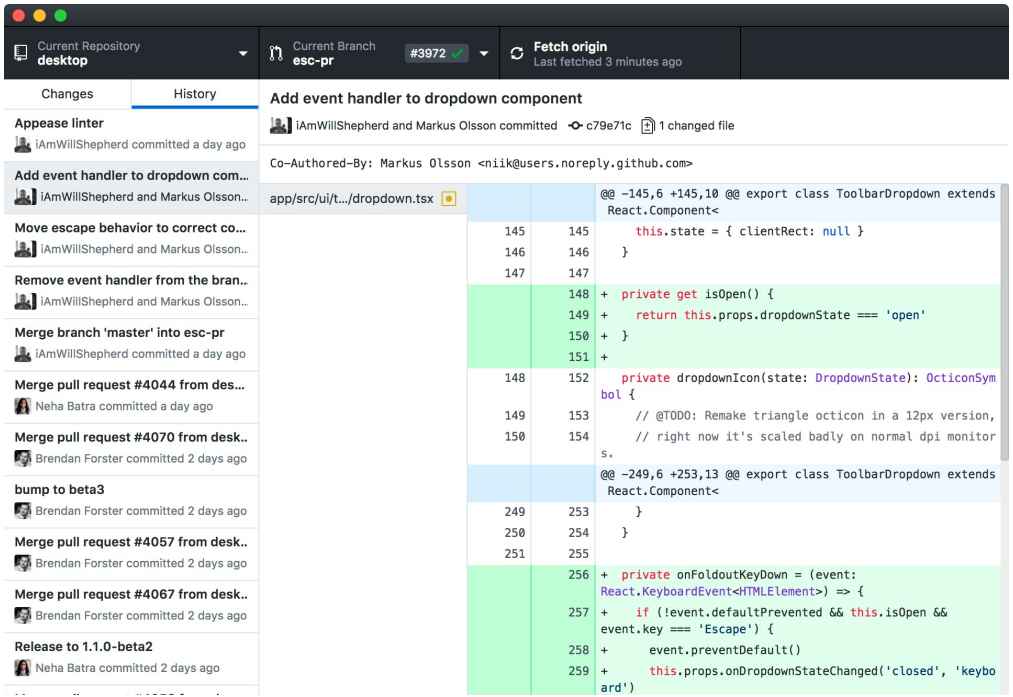
- https://learning.oreilly.com/library/view/head-first-git/9781492092506/



How to use Git

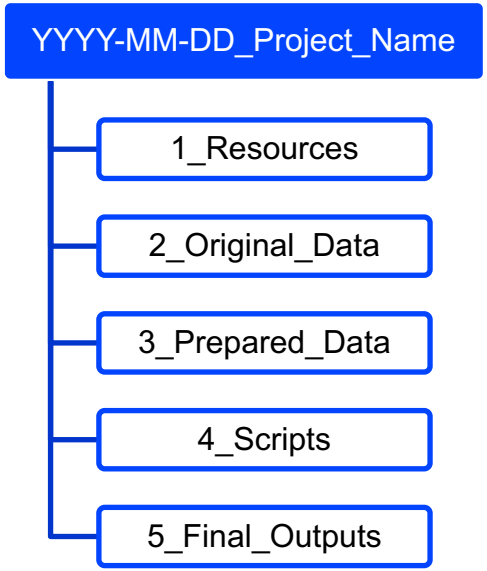
Git using Github Desktop

- Free and easy to use GUI for using Git (optional: with Github)
- Available for Mac and Windows
- https://desktop.github.com



Organizing your files

- Use a recurring structure for your projects
- Adapt to your needs
- KISS - Keep it simple and stupid
- Example:
  - **Resources:** Project proposal, e-mails, …
  - **Original data**: Contains the original, raw files → Never modify!
  - **Prepared data:** Any data artefacts you create, e.g. csv, pickle, …
  - **Scripts:** Your scripts and/or notebooks
  - **Final Outputs:** Deliverables, PPT slides, spreadsheets



Exercise: Build your first repo!

Q&A

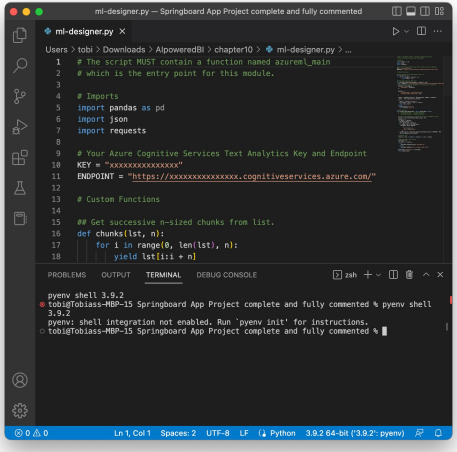
How do you feel?

Running your code

Recap: Running Code

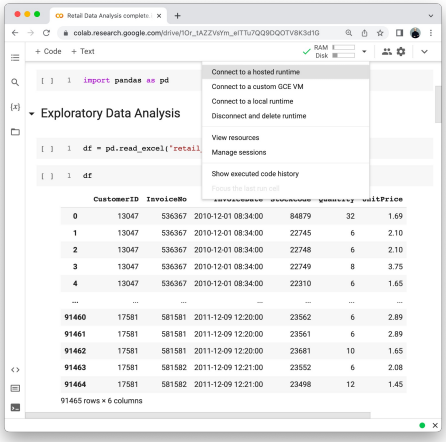
From an IDE

Jupyter Lab, VSCode, …



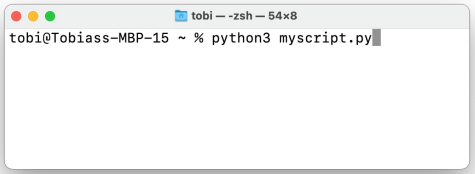
Within a notebook

Jupyter Notebooks, …



From the command line

Terminal, PowerShell, …

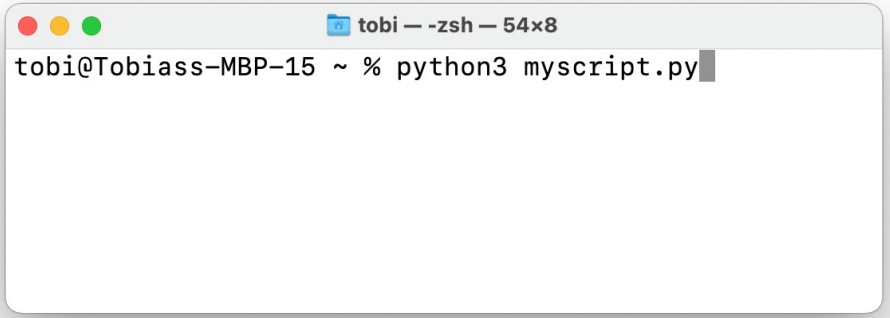


Automation

Running Python Code

Run a script from the terminal

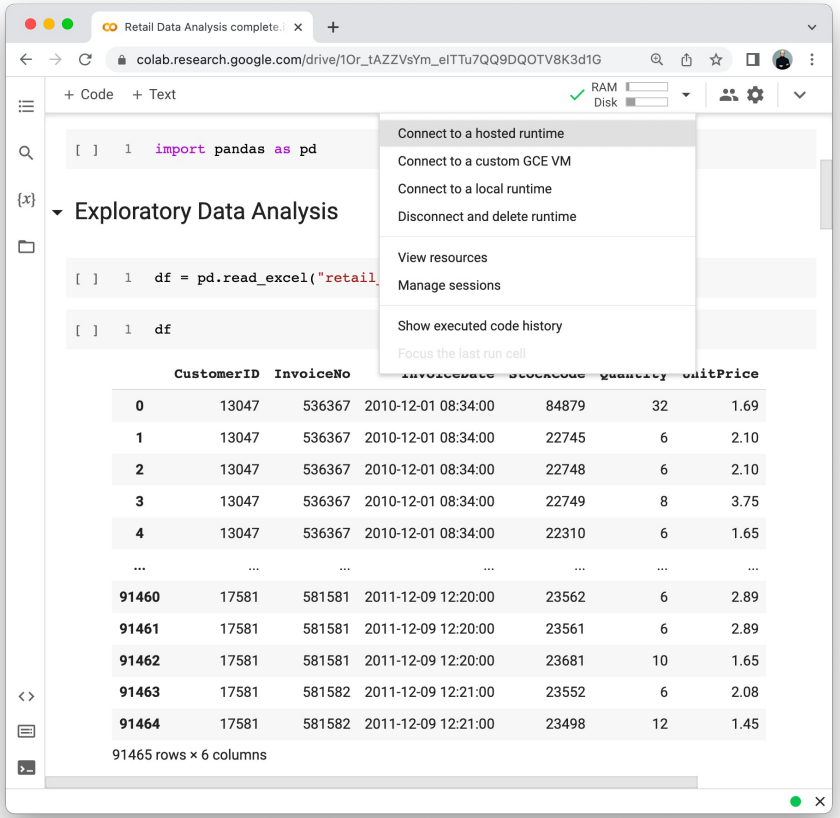
- Fastest way to run Python code
- Typically used for automation / scheduling
- Mac/Linux: Terminal
- Windows: Terminal / PowerShell / CMD.exe
- Runs the whole script (file)
- **Code in the script gets executed from top to bottom**



Running Python Code

Run a script from a notebook

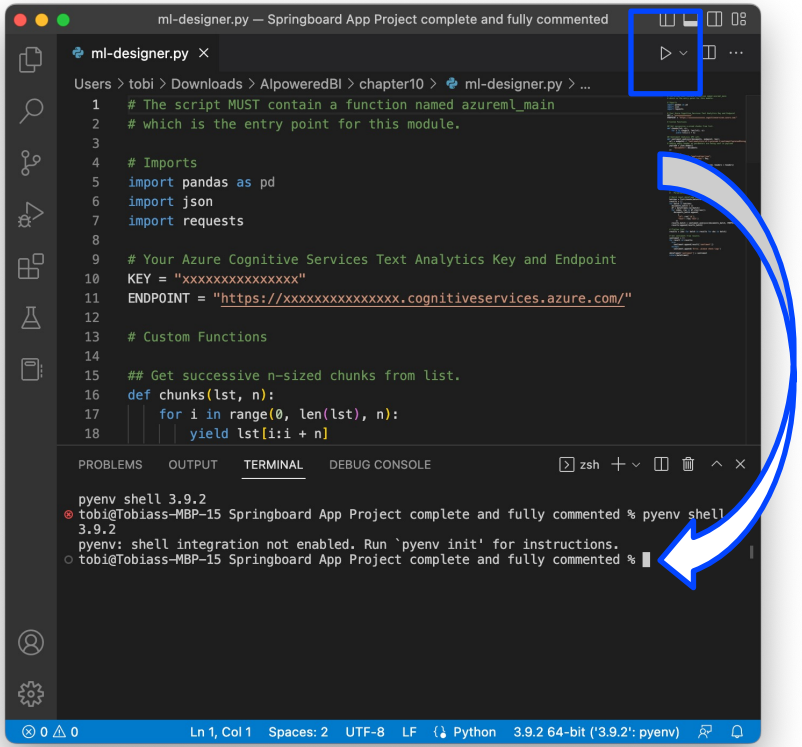
- Connect notebook to a (Python) kernel
- Execute code cell to run the code
- The results will stay in memory until the kernel gets restarted
- Run ”code cells” instead of whole script → High interactivity
- **Order of code execution can be defined by the user (best practice: run top to bottom)**



Running Python Code

Run a script from an IDE

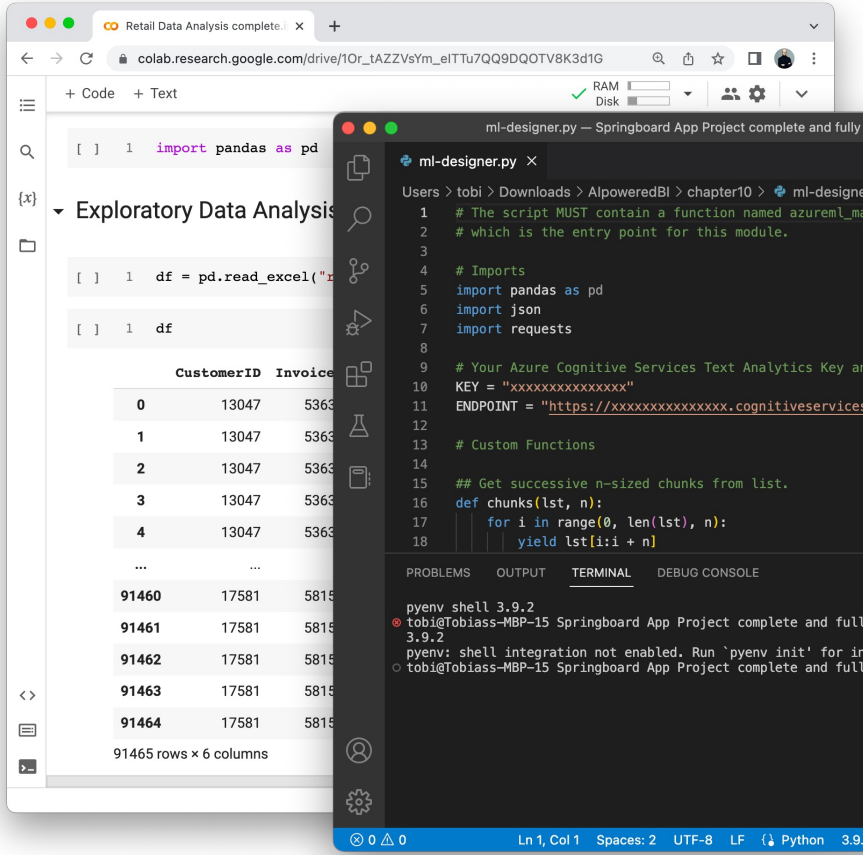
- Typically sends the whole script from the code editor to the terminal
- Can also run multiple files (threading)
- Some IDEs let you run code partially (e.g. highlight code and just run this selection)



Notebooks vs. Scripts

- Notebooks provide a more flexible / interactive experience for data analysis
- In notebooks, code can be run from top to bottom, but you can also jump around → Both biggest advantage and disadvantage of a notebook
- **Best practice:** Treat your notebook like a script file – If you want, you should be able to run your notebook at ONCE from top to bottom without any manual intervention
- Notebooks are harder to version control because of rich data types (markdown, images, widgets, …)
- Notebooks allow you to keep code, markdown, and output in a single document.

→ Most users find it easier to start with notebooks



Exercise: Work with VS Code and the command line

Q&A

How do you feel?

Wrap-up

What did we learn today?

- ❑ Getting started with Python – what is it, why is it so popular?
- ❑ Understanding the Python Ecosystem
- ❑ Knowing the most popular packages for data analysis
- ❑ Differentiating between Notebooks vs. script files
- ❑ Install Anaconda & Jupyter Notebook on your computer
- ❑ Learn essential coding best practices (versioning, clean code, functions)
- ❑ Built your first code repository
- ❑ Organizing to your code files
- ❑ Running scripts from the command line

Outlook for next week

Week 3: Descriptive Business Analytics with Python

- Introduction to Descriptive Analytics
- Introduction to exploratory data analysis (EDA)
- Data modeling (tidy data for data analysis)
- Data wrangling essentials with Python
- Loading data with and without SQL
- Writing / Exporting data
- Descriptive vs inferential statistics
- Summary statistics & Anscombe's quartet
- Data visualization techniques
- Visualization frameworks in Python
- Hands-on Exercises!

Thank you!

Keep in touch:

linkedin.com/in/tobias-zwingmann
tobias@rapyd.ai