

Some helpful tools I'm using in my research

Jay Sayre

Guest Lecture – ARE 202A, Introduction to Applied Research Methods
November 19th, 2024

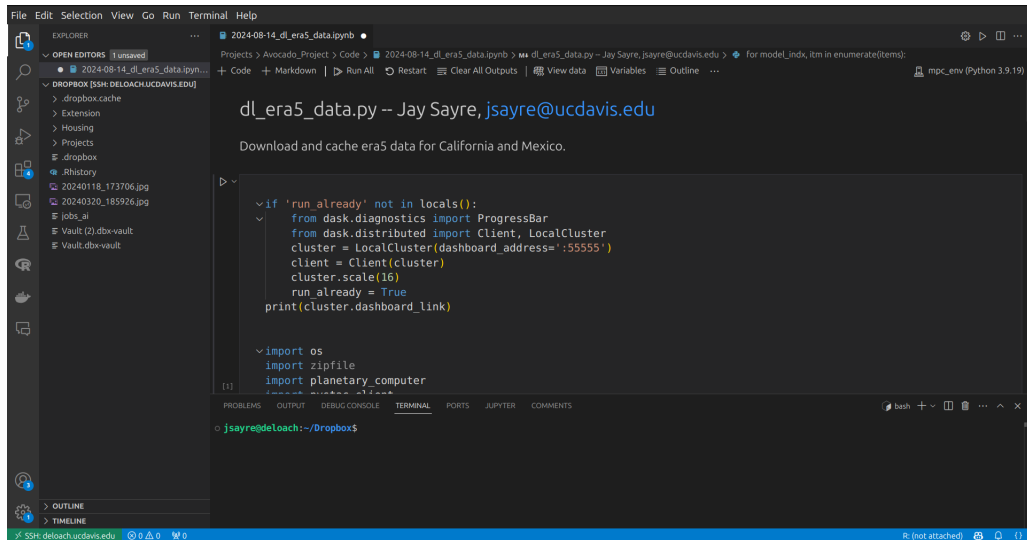
1. VS Code and the ARE Servers
2. GitHub Copilot
3. Makefiles
4. APIs for LLMs: Converting image, text, and audio inputs into quantitative data

Visual Studio Code (VS Code)

What is VS Code?

- A lightweight, open-source code editor
- Similar to Jupyter notebooks or RStudio, but supports multiple programming languages
- Most importantly, has vast ecosystem of plugin tools
 - Integrates with git and Github, as well as Github Copilot (more on this later)
 - Has extensions for running Python, R, and Julia code in neat blocks
- My favorite feature: lets you use the ARE departmental servers via SSH without needing to have a virtual window (RDP) to the server, which can have latency issues on slow connections
 - Arnon Erba can help you get set up with this

VS Code running resources on Deloach



What is GitHub Copilot?

- AI-powered code completion tool that suggests code snippets as you type
- Supports multiple programming languages (Julia, Python, R, Stata,...) and integrates in popular IDEs like VS Code and RStudio

Benefits:

- Accelerates repetitive coding tasks and provides suggestions based on project structure and comments
- The suggestions are often really good!

Warning:

- Makes it almost too easy to write code – can make sometimes large mistakes, should not be a substitute for careful thinking

GitHub Copilot Installation

The screenshot shows the Visual Studio Code interface with the GitHub Copilot extension installed. The left sidebar displays the 'EXTENSIONS' view, listing several installed extensions including 'GitHub Copilot', 'GitHub Copilot Chat', 'gms', 'isort', 'Jupyter Cell Tags', and 'Jupyter Kevman'. The main editor area shows the 'Extension: GitHub Copilot' details page. The page features the GitHub Copilot logo, the version number 'v1.245.0', and a star rating of 4.5 stars (1101 reviews). The description states 'Your AI pair programmer'. Below this, there are buttons for 'Disable', 'Uninstall', and 'Switch to Pre-Release Version', along with a checked 'Auto Update' option. The 'DETAILS' tab is selected, showing a description of the extension as an AI pair programmer tool that helps write code faster and smarter. A 'Sign up for a GitHub Copilot free trial' link is also present. The right sidebar shows 'Categories' (AI, Chat), 'Programming Languages', 'Machine Learning', 'Resources' (Marketplace, Issues, License, GitHub), and 'More Info' (Published: 2021-06-29, 07:26:17; Last: 2024-11-15).

Extension: GitHub Copilot - Visual Studio Code

File Edit Selection View Go Run Terminal Help

2024-10-10_compute_temp_bins_era5.ipynb • Extension: GitHub Copilot x

Search Extensions in Mark...

INSTALLED 26

Open any folder or reposi...
Microsoft

Docker
Makes it easy to create, ma...
Microsoft

GitHub Copilot 263ms
Your AI pair programmer
GitHub

GitHub Copilot Chat 82ms
AI chat features powered b...
GitHub

gms
GAMS language
Laurent Drouet

isort 15ms
Import organization suppor...
Microsoft

Jupyter Cell Tags 2ms
Jupyter Cell Tags support F...
Microsoft

Jupyter Kevman 3

GitHub Copilot v1.245.0
GitHub [github.com](#) | 22,398,824 | ★★★★★ (1101)
Your AI pair programmer
Disable Uninstall Switch to Pre-Release Version ☒ Auto Update

DETAILS FEATURES CHANGELOG EXTENSION PACK

GitHub Copilot - Your AI pair programmer

GitHub Copilot is an AI pair programmer tool that helps you write code faster and smarter.

[Sign up for a GitHub Copilot free trial](#)

`@workspace what does this proje`

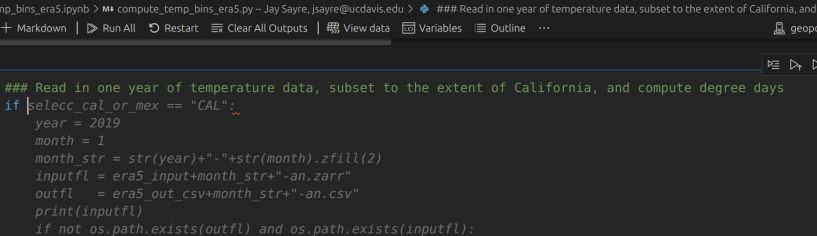
Categories
AI Chat
Programming Languages
Machine Learning

Resources
[Marketplace](#)
[Issues](#)
[License](#)
[GitHub](#)

More Info
Published 2021-06-29, 07:26:17
Last 2024-11-15

- Requires a GitHub account to access Copilot (educational plan is free)

GitHub Copilot in Action

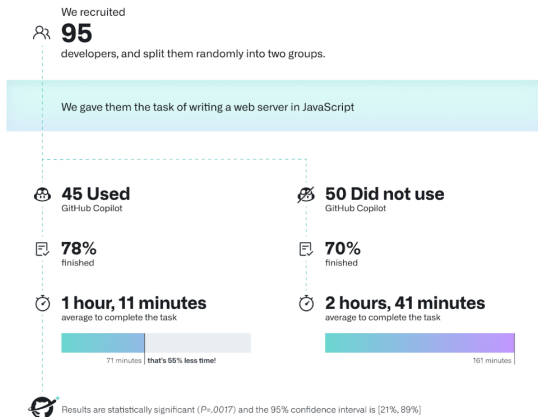


The screenshot shows a Jupyter Notebook window with a dark theme. The top bar includes a menu (File, Edit, Selection, View, Go, Run, Terminal, Help) and a toolbar with icons for file operations and execution. The notebook title is "2024-10-10_compute_temp_bins_era5.ipynb". The code cell is active, showing a script that reads temperature data from a Zarr store, filters for California, and computes degree days. The code is as follows:

```
### Read in one year of temperature data, subset to the extent of California, and compute degree days
if selecc_cal_or_mex == "CAL":
    year = 2019
    month = 1
    month_str = str(year)+"-"+str(month).zfill(2)
    inputfl = era5_input+month_str+"-an.zarr"
    outfl = era5_out_csv+month_str+"-an.csv"
    print(inputfl)
    if not os.path.exists(outfl) and os.path.exists(inputfl):
        print(month_str)
        era5_da = xr.open_zarr(inputfl)
        era5_da = era5_da[selecc_var].to_dataset()
        era5_da['lon'] = xr.where(era5_da['lon'] > 180, era5_da['lon'] - 360, era5_da['lon'])
        era5_da = era5_da.sortby(era5_da.lon)
        era5_da[selecc_var] = era5_da[selecc_var] - 273.15 ## Recall that this is in Kelvin natively
        era5_da = era5_da.sel(lat=slice(42.1,32.4), lon=slice(-124.5, -114)) ## California
        era5_da['degree_days'] = xr.where(era5_da[selecc_var] < 10, 10-era5_da[selecc_var], 0)
        era5_da['degree_days'] = xr.where(era5_da[selecc_var] > 30, era5_da[selecc_var]-30, era5_da['degree_days'])
        era5_da['num_grids'] = 1
        era5_da = era5_da.sum(dim='time')
        era5_da = era5_da.compute()
        era5_da = era5_da.reindex_like(geo_da, method='nearest')
```

- All of the code in gray is predicted based on preceding document

Research on GitHub Copilot



- Pandey et al. (2024) – 33-36% reduction in time spent in coding related tasks
- Song et al. (2024) – 6.5% increase in project level productivity with no decrease in code quality, larger increases for developers with *more* subject expertise

- Many journals now require code and some will even do basic replication of the package
- One common way to facilitate this is a script that will run all of your other scripts



AMERICAN
ECONOMIC
ASSOCIATION

Data and Code Availability Policy

Computer Code

A master script is strongly encouraged. When no master script is included, please provide sufficient and precise step-by-step instructions, allowing users to exactly reproduce the generated outputs with the least amount of effort.

- If your project is simple and in only one programming language (e.g. R), perhaps a quick R script that runs all your R scripts is sufficient

Arguments for makefiles

- Many projects now use multiple programming languages (Julia, Matlab, Python, R, Stata) and the replication code should be multimodal and agnostic to the type of program used
- A *makefile* accommodates this, and can help manage the research process, especially if you need to iteratively update parts of your analysis as new data comes in
- Suppose you collect new data from a survey, and you want to update your regression analyses accordingly, but don't need to rerun other parts of your analysis (e.g. pre-treatment balance tables)
- With a normal script, you'd either need to rerun everything or manually figure out what to rerun
- A *makefile* automates this process, tracking changes in inputs and which outputs depend on them

Example Makefile for a Research Project

main_makefile.mk

```
1  # Define directories
2  DATA_DIR := /home/user/project_dir/data
3  OUTPUT_DIR := /home/user/project_dir/output
4
5  # Step 1: Clean raw data using Python script
6  $(OUTPUT_DIR)/clean_data.csv: clean_data.py $(DATA_DIR)/raw_data.csv
7  |   python clean_data.py --input $< --output $@
8
9  # Step 2: Analyze the cleaned data using R
10 $(OUTPUT_DIR)/analysis_results.csv: $(OUTPUT_DIR)/clean_data.csv analyze_data.R
11 |   Rscript analyze_data.R --input $< --output $@
12
13 # Step 3: Generate the final PDF report using Stata
14 $(OUTPUT_DIR)/final_report.pdf: $(OUTPUT_DIR)/analysis_results.csv generate_report.do
15 |   stata -b do generate_report.do
16
17 all: $(OUTPUT_DIR)/final_report.pdf
18 ### Last line needed to ensure final report is always generated
19 ### Even if a file called "all" exists in directory
20 .PHONY: all
```

Makefile: defining inputs

main_makefile.mk

```
1  # Define directories
2  DATA_DIR := /home/user/project_dir/data
3  OUTPUT_DIR := /home/user/project_dir/output
4
5  # Step 1: Clean raw data using Python script
6  $(OUTPUT_DIR)/clean_data.csv: clean_data.py $(DATA_DIR)/raw_data.csv
7      python clean_data.py --input $< --output $@
8
9  # Step 2: Analyze the cleaned data using R
10 $(OUTPUT_DIR)/analysis_results.csv: $(OUTPUT_DIR)/clean_data.csv analyze_data.R
11     Rscript analyze_data.R --input $< --output $@
12
13 # Step 3: Generate the final PDF report using Stata
14 $(OUTPUT_DIR)/final_report.pdf: $(OUTPUT_DIR)/analysis_results.csv generate_report.do
15     stata -b do generate_report.do
16
17 all: $(OUTPUT_DIR)/final_report.pdf
18 ### Last line needed to ensure final report is always generated
19 ### Even if a file called "all" exists in directory
20 .PHONY: all
```

1. Dependencies

Makefile: defining outputs

main_makefile.mk

```
1  # Define directories
2  DATA_DIR := /home/user/project_dir/data
3  OUTPUT_DIR := /home/user/project_dir/output
4
5  # Step 1: Clean raw data using Python script
6  $(OUTPUT_DIR)/clean_data.csv: clean_data.py $(DATA_DIR)/raw_data.csv
7      python clean_data.py --input $< --output $@
8
9  # Step 2: Analyze the cleaned data using R
10 $(OUTPUT_DIR)/analysis_results.csv: $(OUTPUT_DIR)/clean_data.csv analyze_data.R
11     Rscript analyze_data.R --input $< --output $@
12
13 # Step 3: Generate the final PDF report using Stata
14 $(OUTPUT_DIR)/final_report.pdf: $(OUTPUT_DIR)/analysis_results.csv generate_report.do
15     stata -b do generate_report.do
16
17 all: $(OUTPUT_DIR)/final_report.pdf
18     ### Last line needed to ensure final report is always generated
19     ### Even if a file called "all" exists in directory
20 .PHONY: all
```

2. Target

Makefile: defining scripts

main_makefile.mk

```
1  # Define directories
2  DATA_DIR := /home/user/project_dir/data
3  OUTPUT_DIR := /home/user/project_dir/output
4
5  # Step 1: Clean raw data using Python script
6  $(OUTPUT_DIR)/clean_data.csv: clean_data.py $(DATA_DIR)/raw_data.csv
7      python clean_data.py --input $< --output $@
8
9  # Step 2: Analyze the cleaned data using R
10 $(OUTPUT_DIR)/analysis_results.csv: $(OUTPUT_DIR)/clean_data.csv analyze_data.R
11     Rscript analyze_data.R --input $< --output $@
12
13 # Step 3: Generate the final PDF report using Stata
14 $(OUTPUT_DIR)/final_report.pdf: $(OUTPUT_DIR)/analysis_results.csv generate_report.do
15     stata -b do generate_report.do
16
17 all: $(OUTPUT_DIR)/final_report.pdf
18 ### Last line needed to ensure final report is always generated
19 ### Even if a file called "all" exists in directory
20 .PHONY: all
```

3. Commands

Cons of makefiles

- Makefiles can be slow to write, and tracking all of the inputs to each file can be time-intensive ex-post
- For simple projects probably not worthwhile
- If you start every project by defining inputs and outputs in the makefile and then use it to pass these arguments into your code, this cost is greatly reduced
- Much of the documentation you'd need to do anyway is provided by the steps in the makefile

Using APIs for LLMs in research

- Most likely you (and your students) are already quite familiar with Large Language Models (LLMs)
- However, researchers are realizing how LLMs like GPT-4 and Claude can be used to convert unstructured inputs into structured, quantitative data that can be used for analysis
- Applications range from text analysis, image recognition, and audio transcription
- You can usually test these features one by one in a browser window
- APIs (application programming interfaces) provide a way to use LLMs with large data sources

Some examples: classification of text data



United States Department of Agriculture
National Agricultural Statistics Service

California Crop Progress & Condition



WEEK ENDING: September 24, 2023

FRUIT CROPS

Table **grape** and wine grape harvests continued. **Peach** and **pluot** harvests were winding down and **nectarine** harvest completed. **Plum** harvest was ongoing. **Cherry** and **apricot** orchards were pruned and treated for nematodes. **Figs** and **prunes** were harvested. **Pomegranates** and **kiwifruit** continued to develop. **Persimmons** were reaching mature size and starting to show color. **Apple** and **pear** harvests continued. Table **olive** harvest continued. **Strawberries** continued to develop despite poor weather conditions. **Blackberry** and **raspberry** harvests were ongoing. Valencia **oranges** and **grapefruit** were harvested. District 2 **lemon** harvest was winding down. Some orange trees were topped and skirted.



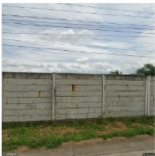
	A	B	C	D	E	F	G	H	I	J
1	crop	planting	harvesting	developing/growing/maturing	soil preparation	irrigation	pruning	county	date	type_crop
1362	table grape	0	1	0	0	0	0		2023-09-24	fruit_crops
1363	wine grape	0	1	0	0	0	0		2023-09-24	fruit_crops
1364	peach	0	1	0	0	0	0		2023-09-24	fruit_crops
1365	pluot	0	1	0	0	0	0		2023-09-24	fruit_crops
1366	nectarine	0	1	0	0	0	0		2023-09-24	fruit_crops
1367	plum	0	1	0	0	0	0		2023-09-24	fruit_crops
1368	cherry	0	0	0	0	0	1		2023-09-24	fruit_crops
1369	apricot	0	0	0	0	0	1		2023-09-24	fruit_crops
1370	fig	0	1	0	0	0	0		2023-09-24	fruit_crops
1371	prune	0	1	0	0	0	0		2023-09-24	fruit_crops
1372	pomegranate	0	0	1	0	0	0		2023-09-24	fruit_crops
1373	kiwifruit	0	0	1	0	0	0		2023-09-24	fruit_crops
1374	persimmon	0	0	1	0	0	0		2023-09-24	fruit_crops
1375	apple	0	1	0	0	0	0		2023-09-24	fruit_crops
1376	pear	0	1	0	0	0	0		2023-09-24	fruit_crops
1377	table olive	0	1	0	0	0	0		2023-09-24	fruit_crops
1378	strawberry	0	0	1	0	0	0		2023-09-24	fruit_crops
1379	blackberry	0	1	0	0	0	0		2023-09-24	fruit_crops
1380	raspberry	0	1	0	0	0	0		2023-09-24	fruit_crops
1381	valencia orange	0	1	0	0	0	0		2023-09-24	fruit_crops
1382	grapefruit	0	1	0	0	0	0		2023-09-24	fruit_crops
1383	lemon	0	1	0	0	0	0	District 2	2023-09-24	fruit_crops
1384	orange	0	0	0	0	0	1		2023-09-24	fruit_crops

1a. FECHA DE INSPECCIÓN DATE INSPECTED MARZO 02 DE 2020	1b. FECHA DE EXPEDICIÓN DATE ISSUED MARZO 10 DE 2020	1c. LUGAR DE EXPEDICIÓN PLACE OF ISSUE TIJUANA, B.C., MÉXICO
DESCRIPCIÓN / DESCRIPTION		
2. NOMBRE Y DIRECCIÓN DEL EXPORTADOR NAME AND ADDRESS OF EXPORTER LIMONES MONICA S.A. DE C.V. ANDADOR DEL REY #20051-C COL. RANCHO EL AGUILA, TIJUANA, B.C., MEXICO		3. NOMBRE Y DIRECCIÓN DECLARADOS DEL DESTINATARIO DECLARED NAME AND ADDRESS OF CONSIGNEE LIMONIK PRODUCE INC. 3200 E. GUASTI RD, SUITE 100 ONTARIO CA. 91761, E.U.A.
4. CANTIDAD DECLARADA Y NOMBRE DEL PRODUCTO 568.75 KG DE LIMON MEXICANO		NAME OF PRODUCE AND QUANTITY DECLARED
5. NOMBRE BOTÁNICO DE LAS PLANTAS BOTANICAL NAME OF PLANTS <i>Citrus aurantifolia</i>	6. LUGAR DE ORIGEN PLACE OF ORIGIN TECOMAN, COLIMA	
7. NÚMERO Y DESCRIPCIÓN DE LOS EMPAQUES NUMBER AND DESCRIPTION OF PACKAGES 35 CAJAS	8. MARCAS DISTINTIVAS DISTINGUISHING MARKS FANCY LIMES	
9. MEDIOS DE TRANSPORTE DECLARADOS DECLARED MEANS OF CONVEYANCE TERRESTRE/TRAILER	10. PUNTO DE ENTRADA DECLARADO DECLARED POINT OF ENTRY MESA DE OTAY, CA.	

Some examples: classification of image data

Soler, Friedel, and Wang (2024): “Combining Deep Learning and Street View Imagery to Map Smallholder Crop Types”

Roadside Obstructions



Walls



Trees

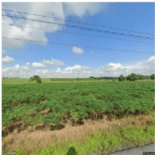


Vegetation



Buildings

Filtered GSV Images



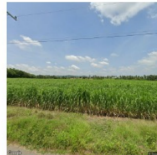
Cassava



Maize



Rice



Sugarcane

Soler, Friedel, and Wang (2024): “Combining Deep Learning and Street View Imagery to Map Smallholder Crop Types”

Training Dataset	Street-view Test Set Metrics						
	Overall Acc	Overall F1	Rice F1	Cassava F1	Maize F1	Sugarcane F1	Other F1
Baseline: Most common	0.67	0.54	0.80	0.00	0.00	0.00	0.00
WebCC	0.82 ± 0.04	0.82 ± 0.03	0.90 ± 0.03	0.73 ± 0.07	0.62 ± 0.09	0.62 ± 0.04	0.49 ± 0.07
iNaturalist	0.81 ± 0.04	0.82 ± 0.03	0.89 ± 0.02	0.63 ± 0.06	0.68 ± 0.06	0.76 ± 0.04	0.51 ± 0.08
iNaturalist + WebCC	0.85 ± 0.02	0.85 ± 0.02	0.91 ± 0.02	0.75 ± 0.04	0.71 ± 0.06	0.78 ± 0.03	0.62 ± 0.06
GPT-4V (zero-shot)[†]	0.95 ± 0.00	0.95 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.89 ± 0.00	0.93 ± 0.00	0.87 ± 0.00
GPT-4V labeled	0.92 ± 0.01	0.92 ± 0.01	0.94 ± 0.01	0.86 ± 0.03	0.81 ± 0.02	0.90 ± 0.02	0.77 ± 0.06
Expert labeled	0.93 ± 0.01	0.93 ± 0.01	0.94 ± 0.01	0.87 ± 0.03	0.82 ± 0.03	0.92 ± 0.02	0.79 ± 0.05
Combined	0.93 ± 0.01	0.93 ± 0.01	0.95 ± 0.01	0.86 ± 0.03	0.81 ± 0.03	0.92 ± 0.02	0.77 ± 0.05

Table 4: Performance on crop type classification from street-view images for the four major crops in Thailand. Models were trained on combinations of three different training datasets: WebCC, iNaturalist, GPT-4V labeled, and Expert labeled GSV images. The MHP threshold was set to 0.9. [†]GPT-4V refers to GPT-4V zero-shot performance on the test set. Each model is trained 5 times with different seeds and bootstrapped training sets. Datasets are ordered by increasing labeling cost.

Using a LLM API

- Need to obtain an API key, and set up payment

```
def analyze_gpt_text(prompt, structured_functions):  
    # Define the content with text information (without the function call)  
    content = {  
        "role": "user",  
        "content": prompt  
    }  
  
    # Create the response using OpenAI's client  
    response = openai.chat.completions.create(  
        model="gpt-4o",  
        temperature=0,  
        messages=[content],  
        functions=structured_functions,  
        function_call={"name": "extract_soil_certificate_info"}  
    )
```

- If an option like "temperature" exists, you should always keep it at 0 (default value)! This influences how random the results are -> 0 is deterministic
- Warning: you probably shouldn't upload sensitive/restricted information to these APIs. Many say that they won't store your information, but...

Conclusion

- You can find the code used in this presentation and to access the OpenAI API here:
<https://github.com/jaysayre/ResearchTools>



- Thanks!
- Contact: jsayre@ucdavis.edu