Analyzing Data from a CSV in a Jupyter Notebook (Part 2)

Announcements

Re: Assignments:

- **EX09: Linked List Utility Functions** due at 11:59pm on LDOC (April 27)
    - Late submission deadline: 4pm on April 30th (start of the final exam)

Re: Quiz 03:

- Great job on this! We’ll have the graded quizzes released ASAP

Recall: We aim to analyze Raleigh weather data and Walter Wally's predictions to see how accurate his winter weather forecasts are!

(Walter Wally is Raleigh's version of Punxsutawney Phil!)

What is a CSV?

A .csv file stores tabular data as plain text. Each line is a row, and values in that row are separated by commas.

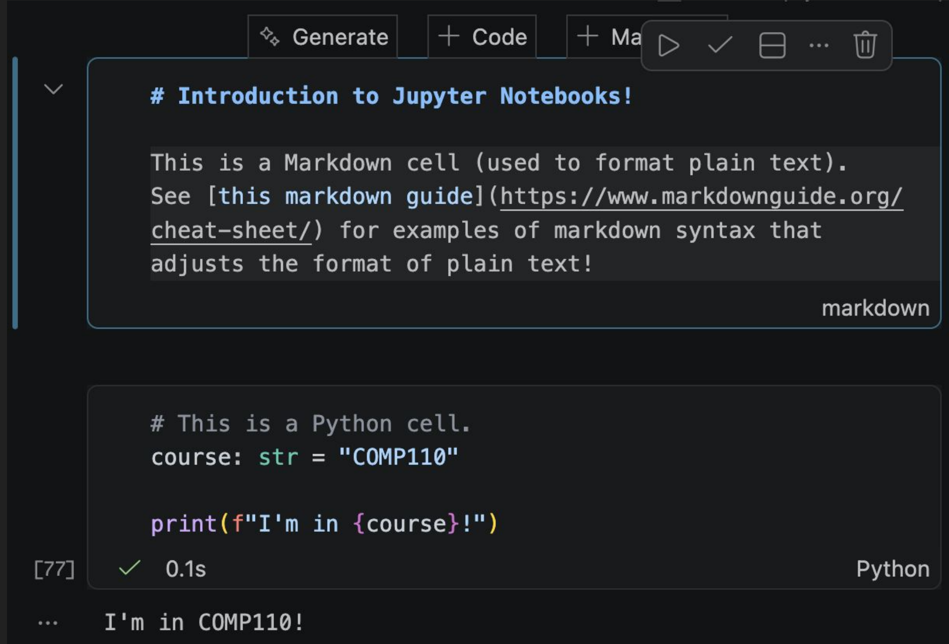
In your VSCode workspace:

1. Create a directory (folder) called **`data_analysis`**
2. In **`data_analysis`**, create a directory called **`data`**
3. In your **`data`** directory, create a file called **`walter_wally.csv`**
4. Copy the data to the right, and paste it into **`walter_wally.csv`**
5. Save the .csv file

```
year,temp,result
2018,48.24,no shadow
2019,46.63,shadow
2020,52.50,shadow
2021,45.86,no shadow
2022,50.36,shadow
```

What is a Jupyter Notebook?

An .ipynb file mixes live Python code, rich text, and output, all in one document. Instead of running an entire script at once, you execute small chunks (“cells”) and see output immediately below each cell.

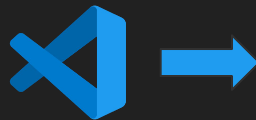


In your VSCode workspace:

1. In **`data_analysis`**, create a new file called **`analysis.ipynb`**

What is a Jupyter Notebook?

Let’s try it!



An .ipynb file mixes live Python code, rich text, and output, all in one document. Instead of running an entire script at once, you execute small chunks (“cells”) and see output immediately below each cell.

Generate | + Code | + Ma

```
# Introduction to Jupyter Notebooks!

This is a Markdown cell (used to format plain text).
See [this markdown guide](https://www.markdownguide.org/cheat-sheet/) for examples of markdown syntax that adjusts the format of plain text!
```

markdown

```
# This is a Python cell.
course: str = "COMP110"

print(f"I'm in {course}!")
```

[77] ✓ 0.1s Python

```
I'm in COMP110!
```

Using Notebooks in VS Code

1. **Open or create an .ipynb file**
   New file... → name it ***<anything>.ipynb***
2. **Select your Python kernel**
   Click the kernel picker (top right) → select a Python version
3. **Add a cell**
   Click "+ Code" or "+ Markdown" in the toolbar
4. **Run a cell**

   Click ▷ or press Shift + Enter
5. **Run all cells**
   Kernel → Restart & Run All (clears old outputs first)

Markdown in Jupyter Notebooks (Cheat Sheet here!)

Syntax	Renders as
`# Heading 1`	Large section title
`## Heading 2`	Subsection title
`**bold text**`	**bold text**
`*italic text*`	*italic text*
`- item\n- item`	• Bulleted list
`1. first\n2. sec`	1. Numbered list
`` `inline code` ``	`monospace snippet`

Let's write a function to help us read and wrangle the data!

`read_csv_rows`

`read_csv_rows` should read an entire CSV of data and return it as a list of rows, where each row represents a `dict[str, str]`.

- Function Name: `read_csv_rows`
- Parameter:
  - `str` path to CSV file
- Return Type: `list[dict[str, str]]`

Data in our .csv:

```
year,temp,result
2018,48.24,no shadow
2019,46.63,shadow
2020,52.50,shadow
2021,45.86,no shadow
2022,50.36,shadow
```

keys

values

```
{'year': '2018',
 'temp': '48.24',
 'result': 'no shadow'}
```

(append to the list)

The current contents of our `list[dict[str, str]]`:

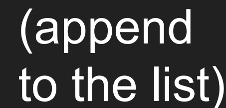
```
[{'year': '2018', 'temp': '48.24', 'result': 'no shadow'}]
```

Data in our .csv:

```
year,temp,result
2018,48.24,no shadow
2019,46.63,shadow
2020,52.50,shadow
2021,45.86,no shadow
2022,50.36,shadow
```

keys

values

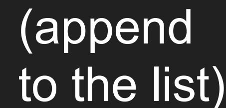


The current contents of our `list[dict[str, str]]`:

```
[{'year': '2018', 'temp': '48.24', 'result': 'no shadow'}]
{'year': '2019', 'temp': '46.63', 'result': 'shadow'}]
```

Data in our .csv:

```
year,temp,result        keys
2018,48.24,no shadow
2019,46.63,shadow
2020,52.50,shadow       values
2021,45.86,no shadow
2022,50.36,shadow
```



The current contents of our `list[dict[str, str]]`:

Data in our .csv:

```
year,temp,result
2018,48.24,no shadow
2019,46.63,shadow
2020,52.50,shadow
2021,45.86,no shadow
2022,50.36,shadow
```

keys

values

```
{'year': '2021',
 'temp': '45.86',
 'result': 'no shadow'}
```

(append to the list)

The current contents of our `list[dict[str, str]]`:

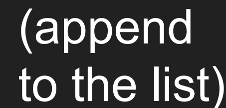
```
[{'year': '2018', 'temp': '48.24', 'result': 'no shadow'}
{'year': '2019', 'temp': '46.63', 'result': 'shadow'},
{'year': '2020', 'temp': '52.50', 'result': 'shadow'}]
{'year': '2021', 'temp': '45.86', 'result': 'no shadow'}]
```

Data in our .csv:

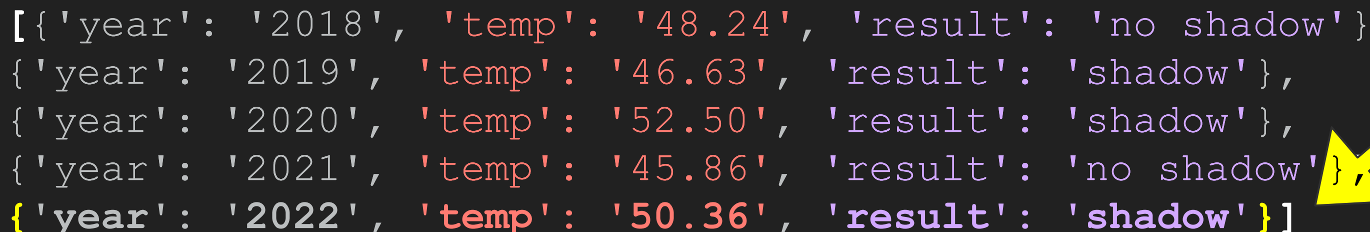
```
year,temp,result
2018,48.24,no shadow
2019,46.63,shadow
2020,52.50,shadow
2021,45.86,no shadow
2022,50.36,shadow
```

keys

values



The current contents of our `list[dict[str, str]]`:



`read_csv_rows` will employ a `DictReader` to convert each row of data into a `dict[str, str]`

Data in our .csv:

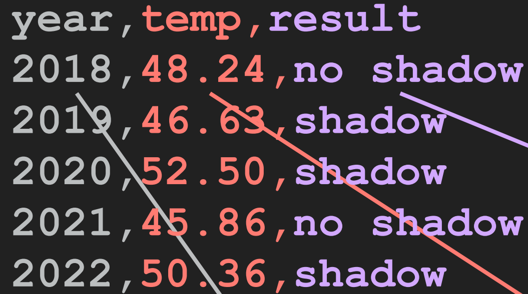
```
year,temp,result
2018,48.24,no shadow
2019,46.63,shadow
2020,52.50,shadow
2021,45.86,no shadow
2022,50.36,shadow
```

After calling read_csv_rows:

```
[{'year': '2018', 'temp': '48.24', 'result': 'no shadow'}
{'year': '2019', 'temp': '46.63', 'result': 'shadow'},
{'year': '2020', 'temp': '52.50', 'result': 'shadow'},
{'year': '2021', 'temp': '45.86', 'result': 'no shadow'},
{'year': '2022', 'temp': '50.36', 'result': 'shadow'}]
```

read_csv_rows

Before calling read_csv_rows (data in our .csv):



```
year,temp,result
2018,48.24,no shadow
2019,46.63,shadow
2020,52.50,shadow
2021,45.86,no shadow
2022,50.36,shadow
```

After calling read_csv_rows (Python objects in our .ipynb):

```
[{'year': '2018', 'temp': '48.24', 'result': 'no shadow'}
{'year': '2019', 'temp': '46.63', 'result': 'shadow'},
{'year': '2020', 'temp': '52.50', 'result': 'shadow'},
{'year': '2021', 'temp': '45.86', 'result': 'no shadow'},
{'year': '2022', 'temp': '50.36', 'result': 'shadow'}]
```

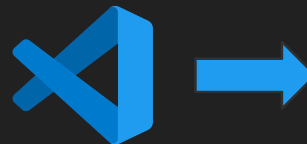
Let’s write a function to help us read and wrangle the data!

`read_csv_rows`

`read_csv_rows` should read an entire CSV of data and return it as a list of rows, where each row represents a **`dict[str, str]`**.

- Function Name: **`read_csv_rows`**
- Parameter:
  - **`str`** path to CSV file
- Return Type: **`list[dict[str, str]]`**

Let’s call it to read our CSV data and convert them to a **`list[dict[str, str]]`**



read_csv_rows (solution)

```
def read_csv_rows(filename: str) -> list[dict[str, str]]:
    """Read the rows of a CSV into a 'table'."""
    result: list[dict[str, str]] = []

    # Open a handle to the data file
    file_handle = open(filename, "r", encoding="utf8")

    # Prepare to read the data file as a CSV rather than just strings.
    csv_reader = DictReader(file_handle)

    # Read each row of the CSV line-by-line
    for row in csv_reader:
        result.append(row)

    # Close the file when done, to free its resources.
    file_handle.close()

    return result
```

The `read_csv_rows` function we just wrote together will be one of 5 functions provided in EX09.

The other 4 will be provided to you, along with descriptions (“documentation”).

This mirrors real-world development: using functions written by others and interpreting their code and documentation to use them correctly.

Let’s read through the code and documentation of another function that will be provided to you in EX09! →

`column_values` function definition

```
def column_values(table: list[dict[str, str]], column: str) -> list[str]:
    """Produce a list[str] of all values in a single column."""
    result: list[str] = []

    for row in table:
        item: str = row[column]
        result.append(item)

    return result
```

With a neighbor, discuss:

1. Consider the **`read_csv_rows`**' and **`column_values`**' function signatures. What do they have in common?
    a. Could the return value of one function call be an argument in the other's function call?
2. Inside the for loop, what type of data would **`row`** store?
3. Does the literal value of **`column`** ever change?

`column_values` function definition and call

```
def column_values(table: list[dict[str, str]], column: str) -> list[str]:
    """Produce a list[str] of all values in a single column."""
    result: list[str] = []

    for row in table:
        item: str = row[column]
        result.append(item)

    return result

data_rows: list[dict[str, str]] = read_csv_rows(<your/path/to/walter_wally.csv>)

years: list[str] = column_values(data_rows, "year")
```

With a neighbor, discuss:

1. What is the length of `years`?
2. What list literal does `years` store?
3. Where would the variable name `years` and its contents be written in a memory diagram, respectively?

`column_values` function definition & documentation

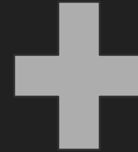
```
def column_values(table: list[dict[str, str]], column: str) -> list[str]:
    """Produce a list[str] of all values in a single column."""
    result: list[str] = []

    for row in table:
        item: str = row[column]
        result.append(item)

    return result
```

- Purpose: Produce a `list[str]` of all values in a single `column` whose name is the second parameter.
- Function Name: `column_values`
- Parameters:
    1. `list[dict[str, str]]` - a list of rows representing a _table_
    2. `str` - the name of the column (key) whose values are being selected
- Return Type: `list[str]`

`column_values` function

Example input:

```
[{'year': '2018', 'temp': '48.24', 'result': 'no shadow'}
{'year': '2019', 'temp': '46.63', 'result': 'shadow'},
{'year': '2020', 'temp': '52.50', 'result': 'shadow'},
{'year': '2021', 'temp': '45.86', 'result': 'no shadow'},
{'year': '2022', 'temp': '50.36', 'result': 'shadow'}]
```



“temp”

Returns:

```
['48.24','46.63','52.50','45.86','50.36']
```

`columnar` function definition

```python
def columnar(row_table: list[dict[str, str]]) -> dict[str, list[str]]:
    """Transform a row-oriented table to a column-oriented table."""
    result: dict[str, list[str]] = {}

    first_row: dict[str, str] = row_table[0]
    for column in first_row:
        result[column] = column_values(row_table, column)

    return result
```

- Purpose: *Transform* a table represented as a list of rows (e.g. `list[dict[str, str]]`) into one represented as a dictionary of columns (e.g. `dict[str, list[str]]`).
- Function Name: `columnar`
- Parameter: `list[dict[str, str]]` - a "table" organized as a list of rows
- Return Type: `dict[str, list[str]]` - a "table" organized as a dictionary of columns

`columnar` function purpose

Before calling **`columnar`**:

```
[{'year': '2018', 'temp': '48.24', 'result': 'no shadow'}
{'year': '2019', 'temp': '46.63', 'result': 'shadow'},
{'year': '2020', 'temp': '52.50', 'result': 'shadow'},
{'year': '2021', 'temp': '45.86', 'result': 'no shadow'},
{'year': '2022', 'temp': '50.36', 'result': 'shadow'}]
```

After calling **`columnar`**:

```
{'year': ['2018', '2019', '2020', '2021', '2022'],
'temp': ['48.24', '46.63', '52.50', '45.86', '50.36'],
'result': ['no shadow', 'shadow', 'shadow', 'no shadow', 'shadow']}
```

With a partner, discuss:

1. What are some of the benefits/drawbacks of column-wise data (in general, and in this example in Python)?