

# Ruffus: A Simple Python Pipeline Management Tool

Jorge Padial

AJC December 4, 2024

# “Typical Scientific Compute workload”

Where's my data?

Clean data

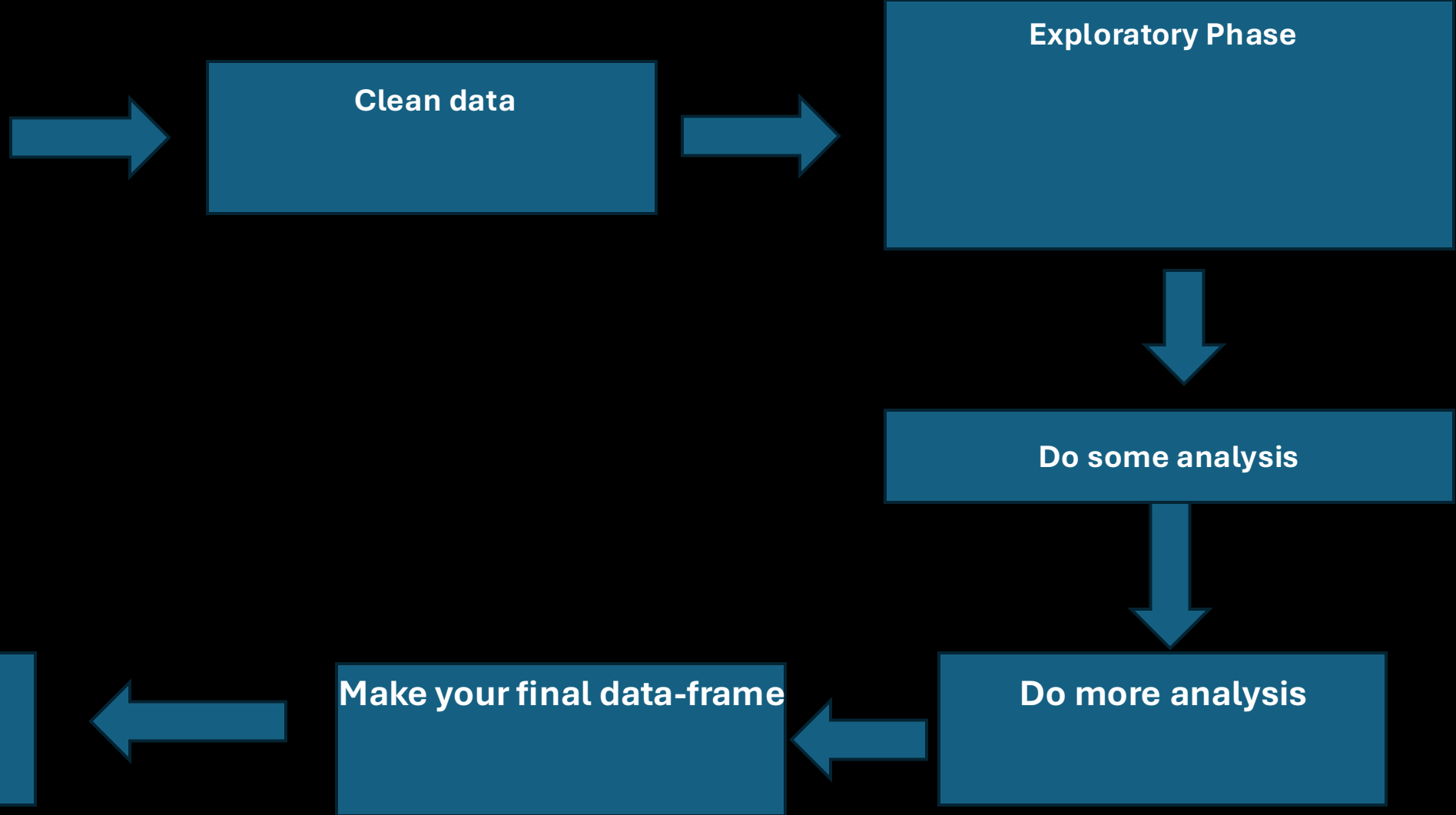
Exploratory Phase

Do some analysis

Do more analysis

Visualize result

Make your final data-frame



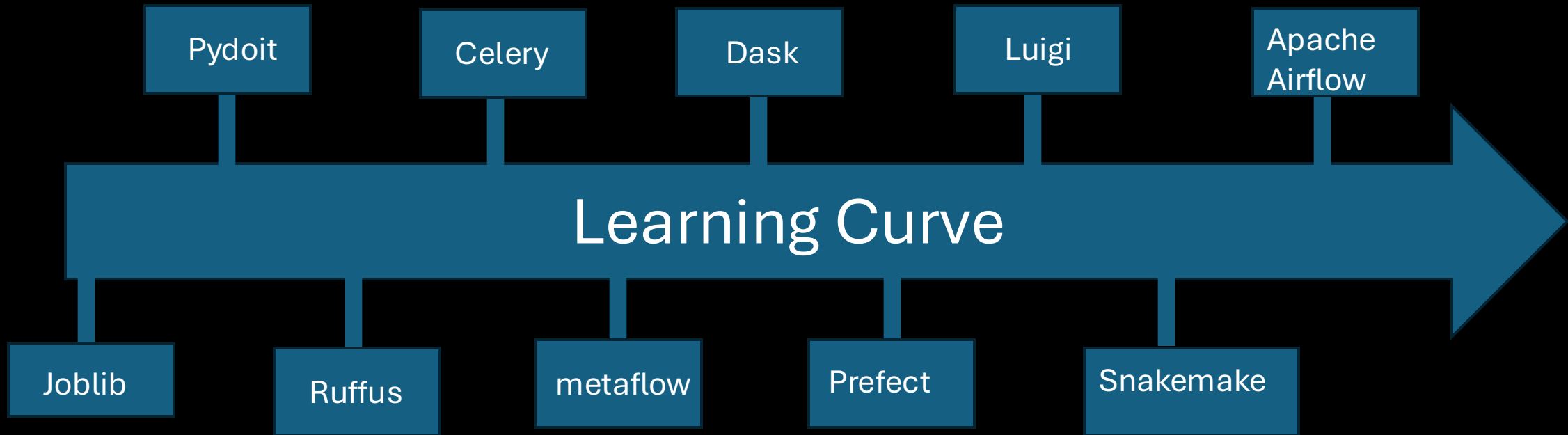
# Pipeline management tools: definition

A **pipeline management tool** is a software that helps automate, schedule, and orchestrate the execution of a series of tasks or processes, managing dependencies, retries, and resource allocation to ensure efficient and reliable workflow execution.

# Pipeline management tools: use cases


Goal	What they are	What they are not
<b>Task Scheduling &amp; Orchestration</b>	<ul style="list-style-type: none"><li>- Automating and scheduling workflows.</li><li>- Managing complex task dependencies with multiple steps.</li><li>- Defining explicit dependencies between tasks (e.g., task A must finish before task B starts).</li></ul>	<ul style="list-style-type: none"><li>- Simple, single-task scripts.</li><li>- Ad-hoc task execution without dependencies.</li><li>- Simple, independent tasks with no need for dependency chains.</li></ul>
<b>Reproducibility &amp; Versioning</b>	<ul style="list-style-type: none"><li>- Ensuring workflows are reproducible (e.g., storing inputs, outputs, and intermediate steps).</li></ul>	<ul style="list-style-type: none"><li>- One-time or non-critical tasks where reproducibility is not necessary.</li></ul>
<b>Failure Handling &amp; Retries</b>	<ul style="list-style-type: none"><li>- Managing retries for failed tasks.</li><li>- Handling error propagation and task recovery.</li></ul>	<ul style="list-style-type: none"><li>- Simple, non-critical tasks without failure management.</li><li>- Workflows that do not need robustness or error handling.</li></ul>
<b>Parallel &amp; Distributed Computing</b>	<ul style="list-style-type: none"><li>- Distributing tasks across multiple machines or processes.</li><li>- Parallelizing independent tasks for faster execution.</li></ul>	<ul style="list-style-type: none"><li>- Simple sequential task processing.</li><li>- Tasks that cannot be parallelized or distributed.</li></ul>
<b>Data Integration &amp; ETL</b>	<ul style="list-style-type: none"><li>- Integrating with various data sources (e.g., databases, APIs, cloud storage).</li></ul>	<ul style="list-style-type: none"><li>- Workflows that don't require data integration with no need for data processing or transfer.</li></ul>

# Pipeline management tools: examples



# Ruffus

- **Simple and lightweight:**
  - Easy way to go from notebook experimentation to a full pipeline.
- **Decorator functions that map from input file(s) to output file(s).**
  - well-suited for data processing and scientific workflows
- **Easy Parallelization**
- **Platform-agnostic:**
  - Windows, Linux, Mac
- **Caches:**
  - Only runs files not up to date
- **Requires regular expressions**
- **Not easy to run on multiple nodes**

 ruffus

Search docs

Installation

Ruffus Manual: List of Chapters and Example code

Chapter 1: An introduction to basic Ruffus syntax

Chapter 2: Transforming data in a pipeline with `@transform`

Chapter 3: More on `@transform`-ing data

Chapter 4: Creating files with `@originate`

Chapter 5: Understanding how your pipeline works with `pipeline_printout(...)`

Chapter 6: Running Ruffus from the command line with `ruffus.cmdline`

Chapter 7: Displaying the pipeline visually with `pipeline_printout_graph(...)`

Chapter 8: Specifying output file names with `formatter()` and `regex()`

Chapter 9: Preparing directories for output with `@mkdir()`

Chapter 10: Checkpointing: Interrupted Pipelines and Exceptions

Chapter 11: Pipeline topologies and a compendium of Ruffus decorators

Chapter 12: Splitting up large tasks / files with `@split`

Chapter 13: `@merge` multiple input into a single result

Chapter 14: Multiprocessing, `drmaa` and Computation Clusters

Chapter 15: Logging progress through a pipeline

[Docs](#) »

[Edit on GitHub](#)



# Ruffus

Ruffus is a Computation Pipeline library for python. It is open-sourced, powerful and user-friendly, and widely used in science and bioinformatics.

## Citation:

Please cite *Ruffus* as:

[Leo Goodstadt \(2010\)](#) : Ruffus: a lightweight Python library for computational pipelines.

*Bioinformatics* 26(21): 2778-2779

## Welcome

*Ruffus* is designed to allow scientific and other analyses to be automated with the minimum of fuss and the least effort.

These are *Ruffus*'s strengths:

**Lightweight:** Suitable for the simplest of tasks  
**Scalable:** Handles even fiendishly complicated pipelines which would cause *make* or *scons* to go cross-eyed and recursive.

**Standard python:** No "clever magic", no pre-processing.

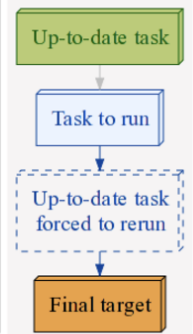
**Unintrusive:** Unambitious, lightweight syntax which tries to do this one small thing well.

Please join me (email: [ruffus\\_lib@llew.org.uk](mailto:ruffus_lib@llew.org.uk)) in setting the direction of this project if you are interested.

### Pipeline Flowchart:



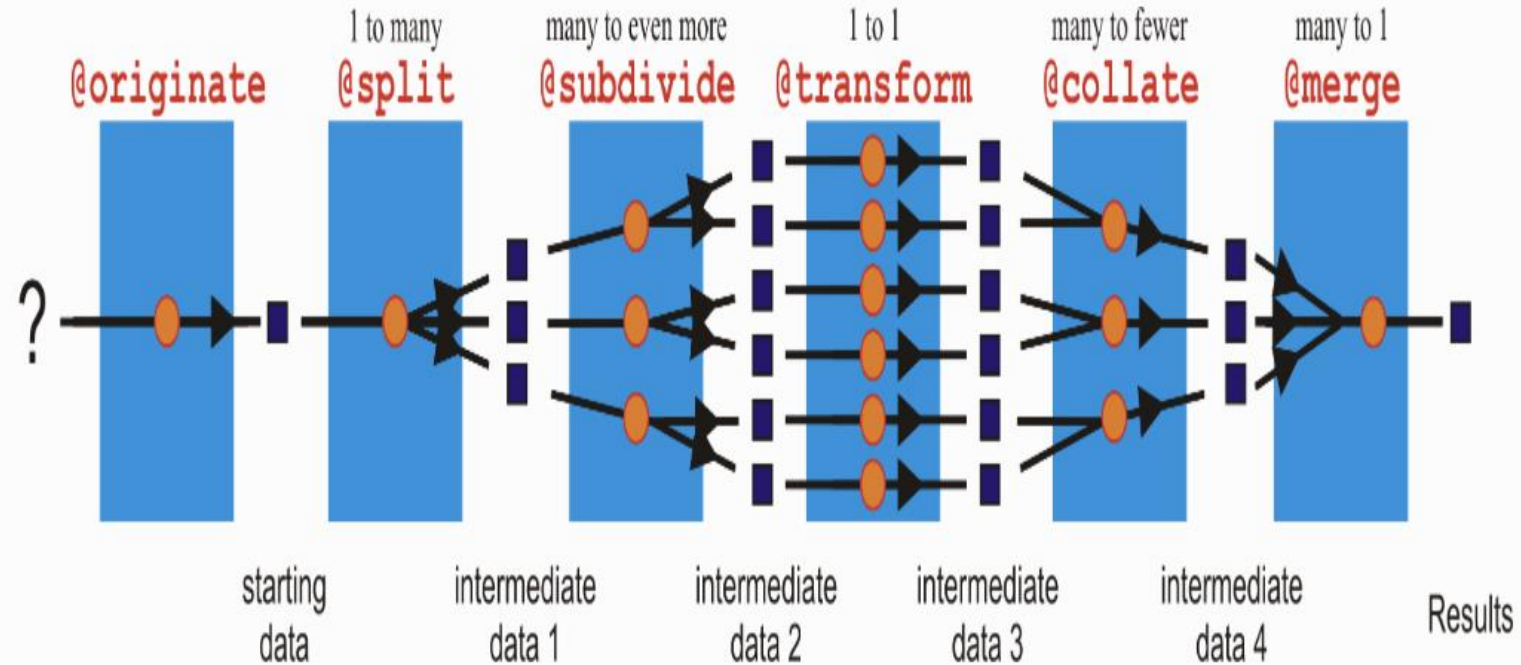
### Key:



# Ruffus: decorators

## A bestiary of *Ruffus* decorators

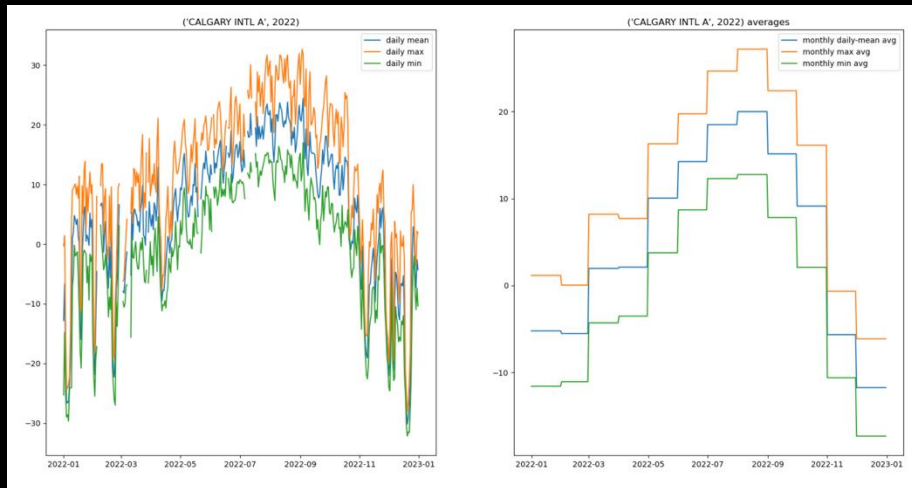
Very often, we would like to transform our data in more complex ways, this is where other *Ruffus* decorators come in.



# “Typical Scientific Compute workload”

**Where's my data?**  
Download yearly weather  
data from Canada

**Clean data**  
- Standardize column names

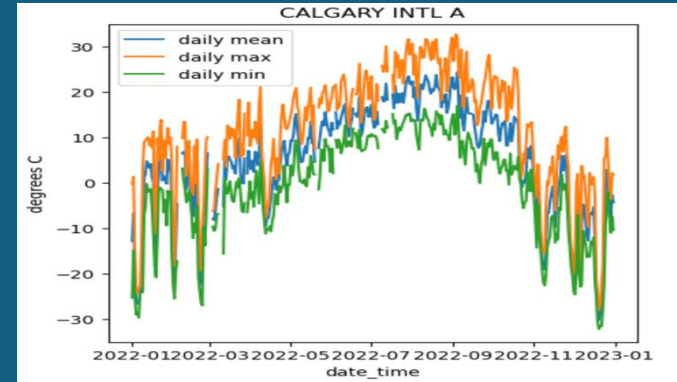


**Visualize result:**  
- Plot mean monthly  
temperatures

**Concatenate N-outputs  
into 1 data-frame.**

**Do some analysis:**  
- Calculate mean of monthly  
temperatures

**Exploratory Phase**



**Spawn N-amount of new jobs**



Let's view a basic Jupyter exploratory phase.

The same functions  
can be used to create  
a simple script from  
download to plot

```
# Start the timer
start_time = time.time()

city_dict = {"Calgary": 50430}

# city_dict = {"Calgary": 50430,
#             "Montreal": 30165,
#             "Vancouver": 51442,
#             "Winnipeg": 51097}

result = []
list_of_years = [2022]

# create query dictionary for each city/year
for city, city_code in city_dict.items():
    for year in list_of_years:
        result.append({"City": city, "City_Code": city_code, "year": year})

# download the data
data_downloaded_list = []
for entry in result:
    data_downloaded_list.append(analysis_module.download_data(entry))

# prep the columns of the dataframe
df_list_col_prepped = [analysis_module.prep_df(this_file) for this_file in data_downloaded_list]

# do some analysis

new_df = pd.concat([analysis_module.calculate_monthly_avgs(this_df) for this_df in df_list_col_prepped]).reset_index(drop = True)

# plot data

for label, group in new_df.groupby(['station_name', 'year']):
    analysis_module.plot_data(label, group)

# End the timer
end_time = time.time()

# Calculate elapsed time
elapsed_time = end_time - start_time

# Print the results
print(f"Elapsed time: {elapsed_time:.6f} seconds")
```

Let's create the same script with ruffus.

We are going to touch on 4 main things:

1. Creating a file from scratch using **@files**
2. Using **@transform** to map from 1-to-1
3. Using **@subdivide** to map from 1-to-many
4. Using **@collate** to map from many-to-1

# @files: from nothing to something

- Needs a list of initialization configuration information
- In our example, the config dictionaries are the same as in a normal script. They are just called within **@files**

- **Input file:**
  - None
- **Output file:**
  - Calgary\_2022.downloaded
- **Config:**
  - Individual dictionary inside results = []

```
# input list of years
list_of_years = [2022]

result = []
# Loop through the dictionary keys and list of years
for city, city_code in city_dict.items():
    for year in list_of_years:
        result.append({"City": city, "City_Code": city_code, "year": year})

# function that takes every config in result and passes it to
# the "file creator" (AKA @files decorator) in order to download
def dl_params():
    for config in result:
        infile = None

        city, year = config['City'], config['year']

        # establish the first ruffus outfile that will be created from scratch

        outfile = f'{city}_{year}.downloaded'

        yield(infile, outfile, config)

@files(dl_params) # @files will create a file from no-files made
def download_data(infile, outfile, config):

    downloaded_file_name = analysis_module.download_data(config)

    output_df = pd.DataFrame([{'downloaded_file_name': downloaded_file_name}])

    pickle.dump(output_df, open(outfile, 'wb'))

if __name__ == '__main__':

    pipeline_run([download_data], verbose = 2)
```

# @transform: 1-to-1 to fix the column names:

- @transform('1<sup>st</sup> arg', '2nd arg', '3rd arg')
  - '1<sup>st</sup>': what was the previous step
  - '2<sup>nd</sup>': what will you change from the previous functions output file.
  - '3<sup>rd</sup>': what is the new extension
- 
- **Input file:**
    - Calgary\_2022.downloaded
  - **Output file:**
    - Calgary\_2022.fixed\_columns.pickle

```
# if __name__ == '__main__':  
  
#     pipeline_run([download_data], verbose = 2)  
  
|  
@transform(download_data, suffix('downloaded'), 'fixed_columns.pickle')  
def fix_columns(infile, outfile):  
  
    infile_df = pickle.load(open(infile, 'rb'))  
  
    csv_file_name = infile_df.iloc[0].downloaded_file_name  
  
    prepped_df = analysis_module.prep_df(csv_file_name)  
  
    pickle.dump(prepped_df, open(outfile, 'wb'))  
  
if __name__ == '__main__':  
  
    pipeline_run([fix_columns], verbose = 2)
```

# @subdivide: 1-to-many split full year into months

- @subdivide('1<sup>st</sup> arg', '2nd arg', '3rd arg', '4<sup>th</sup> )
- '1<sup>st</sup>': what was the previous step
- '2<sup>nd</sup>': formatter()
- '3<sup>rd</sup>': specify files to be checked
- '4<sup>th</sup>': specify path to be created
- **Input file:**
  - Calgary\_2022.fixed\_columns.pickle
- **Output file:**
  - Calgary\_2022.fixed\_columns.#.subdivide\_monthly.pickle

```
# if __name__ == '__main__':

#     pipeline_run([fix_columns], verbose = 2)

@subdivide(fix_columns, formatter(),
           # Output parameter: Glob matches any number of output file names
           "{path[0]}/{basename[0]}*.subdivide_monthly.pickle",
           # Extra parameter: Append to this for output file names
           "{path[0]}/{basename[0]}")
def divide_by_month(infile, outfiles, output_file_name_root):

    infile_df = pickle.load(open(infile, 'rb'))

    print(infile)

    for month_number in infile_df.month.unique():

        copy_df = infile_df.copy()

        mask_df_for_output_by_month = copy_df[copy_df.month == month_number]

        # output_file_name_root == /path/to/where/you/are/working/{city}_{year}.fixed_columns
        # note: the output_file_name_root is the previous outfile without the extension "pickle"

        output_file_name = f'{output_file_name_root}.{month_number}.subdivide_monthly.pickle'

        print(output_file_name)

        pickle.dump(mask_df_for_output_by_month, open(output_file_name, 'wb'))

if __name__ == '__main__':

    pipeline_run([divide_by_month], verbose = 2)
```

# @transform: 1-to-1 do each month individually

- @transform('1<sup>st</sup> arg', '2nd arg', '3rd arg')
- '1<sup>st</sup>': what was the previous step
- '2<sup>nd</sup>': what will you change from the previous functions output file.
- '3<sup>rd</sup>': what is the new extension

- **Input file:**  
Calgary\_2022.fixed\_columns.#.subdivide\_monthly.pickle
- **Output files:**  
Calgary\_2022.fixed\_columns.#.monthly\_analysis.pickle

```
# if __name__ == '__main__':  
  
#     pipeline_run([divide_by_month], verbose = 2)  
  
@transform(divide_by_month, suffix('.subdivide_monthly.pickle'), '.monthly_analysis.pickle')  
def monthly_analysis(infile, outfile):  
    infile_df = pickle.load(open(infile, 'rb'))  
  
    output_df = analysis_module.calculate_monthly_avgs(infile_df)  
  
    pickle.dump(output_df, open(outfile, 'wb'))  
  
if __name__ == '__main__':  
    pipeline_run([monthly_analysis], verbose = 2)
```



# @collate: many-to-1 join each monthly file

- @collate('1<sup>st</sup> arg', '2nd arg', '3rd arg')
- '1<sup>st</sup>': what was the previous step
- '2<sup>nd</sup>': regex to ID all files to join.
- '3<sup>rd</sup>': what is the new extension
- **Input files:**
  - Calgary\_2022.fixed\_columns.#.monthly\_analysis.pickle
- **Output file:**
  - Calgary\_2022.joined\_monthly\_ananalysis.pickle

```
# if __name__ == '__main__':  
  
#     pipeline_run([monthly_analysis], verbose = 2)  
  
@collate(monthly_analysis, regex(r'([A-Za-z]+\d{4}).fixed_columns.\d{0,2}.monthly_analysis.pickle'), r'\1.joined_monthly_analysis.pickle')  
def join_monthly_analysis(infiles, outfile):  
  
    joined_input_df = pd.concat([pickle.load(open(infile, 'rb')) for infile in infiles])  
  
    sorted_df = joined_input_df.sort_values(by = 'date_time')  
  
    pickle.dump(sorted_df, open(outfile, 'wb'))  
  
if __name__ == '__main__':  
  
    pipeline_run([join_monthly_analysis], verbose = 2)
```

# @transfer: 1-to-1 make visualization

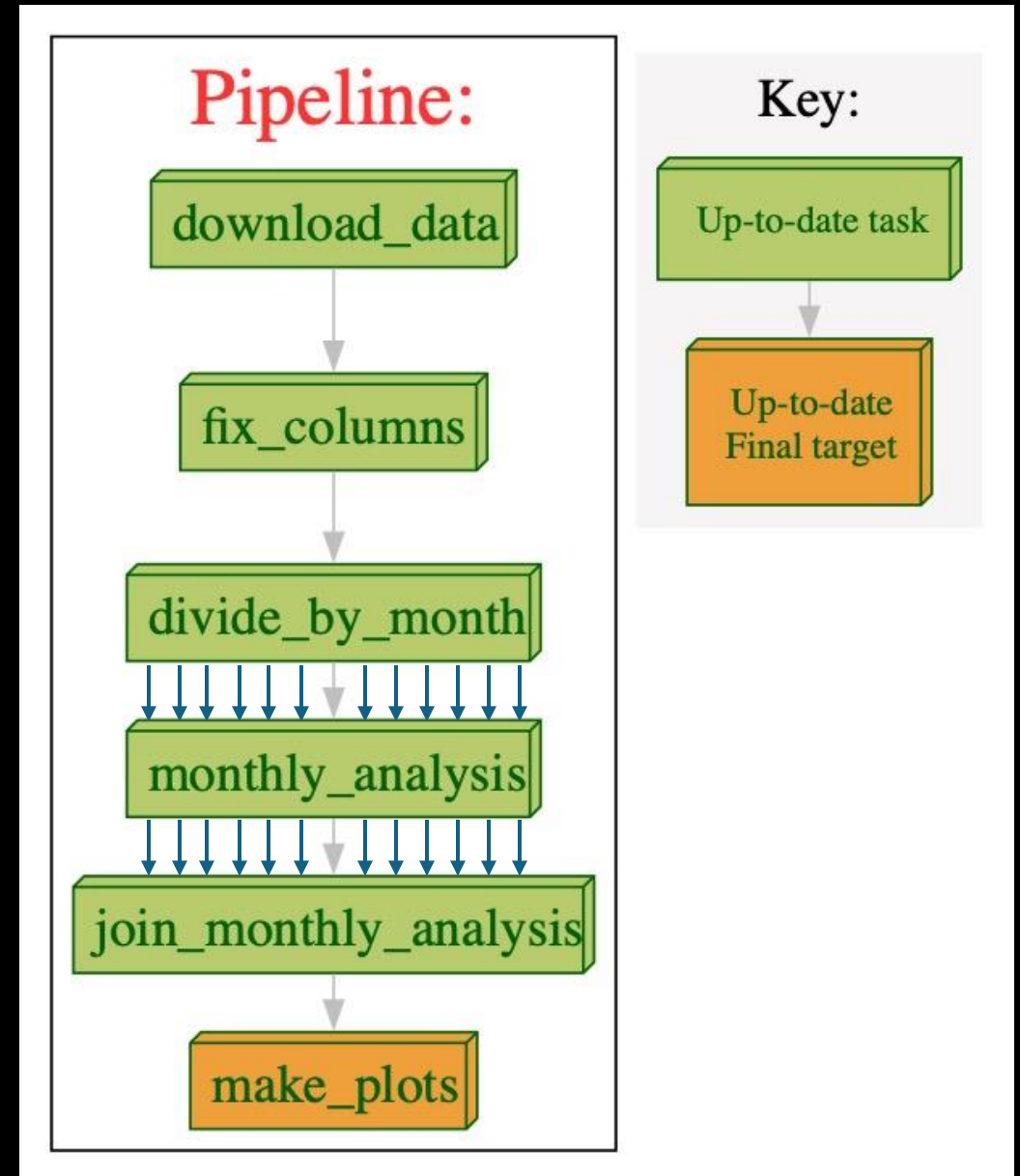
- @transform('1<sup>st</sup> arg', '2nd arg', '3<sup>rd</sup> arg')
  - '1<sup>st</sup>': what was the previous step
  - '2<sup>nd</sup>': regex to ID all files to join.
  - '3<sup>rd</sup>': what is the new extension
- **Input files:**
    - Calgary\_2022.joined\_monthly\_ananalysis.pickle
  - **Output file:**
    - Calgary\_2022.jpg

```
# if __name__ == '__main__':  
  
#     pipeline_run([join_monthly_analysis], verbose = 2)  
  
@transform(join_monthly_analysis, suffix('joined_monthly_analysis.pickle'), 'plots_made.pickle')  
def make_plots(infile, outfile):  
    df = pickle.load(open(infile, 'rb'))  
  
    for label, group in df.groupby(['station_name', 'year']):  
        analysis_module.plot_data_ruffus(label, group)  
  
    output_df = pd.DataFrame([{'plots_made': True}])  
  
    pickle.dump(output_df, open(outfile, 'wb'))  
  
if __name__ == '__main__':  
    pipeline_run([make_plots], verbose = 2)
```

# Yay! you created your first Ruffus script!

- Everything is in 1 script!
- Every infile/outfile is cached
- Combinations of different @ operators can be made to suit your needs
- BUT:
  - "Jorge, you said we can multi-process"

```
if __name__ == '__main__':  
    pipeline_run([make_plots], verbose = 2, multiprocess= 8)
```



# Links to resources

Ruffus:

<https://ruffus.readthedocs.io/en/latest/>

Regular expression builder:

<https://regex101.com/>