

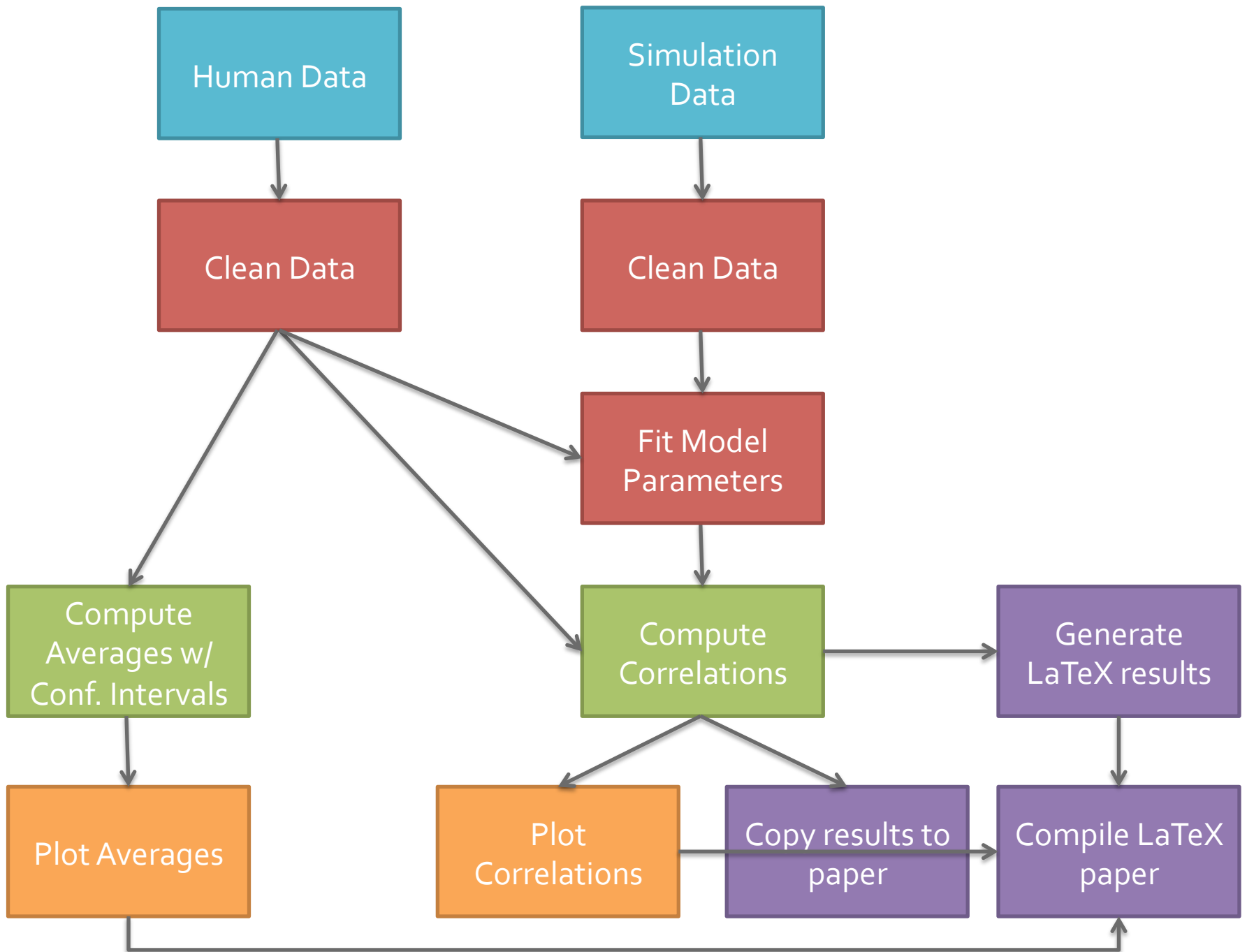
Reproducible, One-Button Workflows with the Jupyter Notebook and SCons

Jessica Hamrick

UC Berkeley & Project Jupyter

@jhamrick

An idealized analysis pipeline...



How do we achieve this
idealized workflow?

Approach 1: one big script

Drawback: no interactivity 😞

Notebooks are interactive!

The image displays three overlapping Jupyter Notebook windows, illustrating the interactive nature of these environments. The topmost window, titled "Lorenz Differential Equations (autosaved)", is in the "Code" view and shows a Python cell with the following code:

```
In [7]: interact(Lorenz, N=fixed(10), angle=(0.,360.),
                sigma=(0.0,50.0), beta=(0.,5), rho=(0.0,50.0));
```

Below the code cell, a set of interactive sliders is visible, allowing users to adjust the parameters of the Lorenz system:

- angle: 308.2
- max_time: 12
- σ : 10
- β : 2.6
- ρ : 28

At the bottom of the window, a colorful plot of the Lorenz attractor is shown, with multiple trajectories in different colors (red, blue, green, yellow, purple) illustrating the system's chaotic behavior.

The middle window shows the "Welcome to the Jupyter Notebook" screen, featuring a "WARNING" message: "Don't rely on this server for production use." and instructions on how to run code.

The bottom window shows the "Welcome to the Jupyter Notebook" screen, featuring a "WARNING" message: "Don't rely on this server for production use." and instructions on how to run code.

Approach 2: one big notebook

Drawback: can be unwieldy 😞

Approach 3: multiple scripts
and / or notebooks

Drawback: dependency hell ☹️

Approach 4: multiple scripts
and / or notebooks with a
traditional build system

Traditional Build Systems

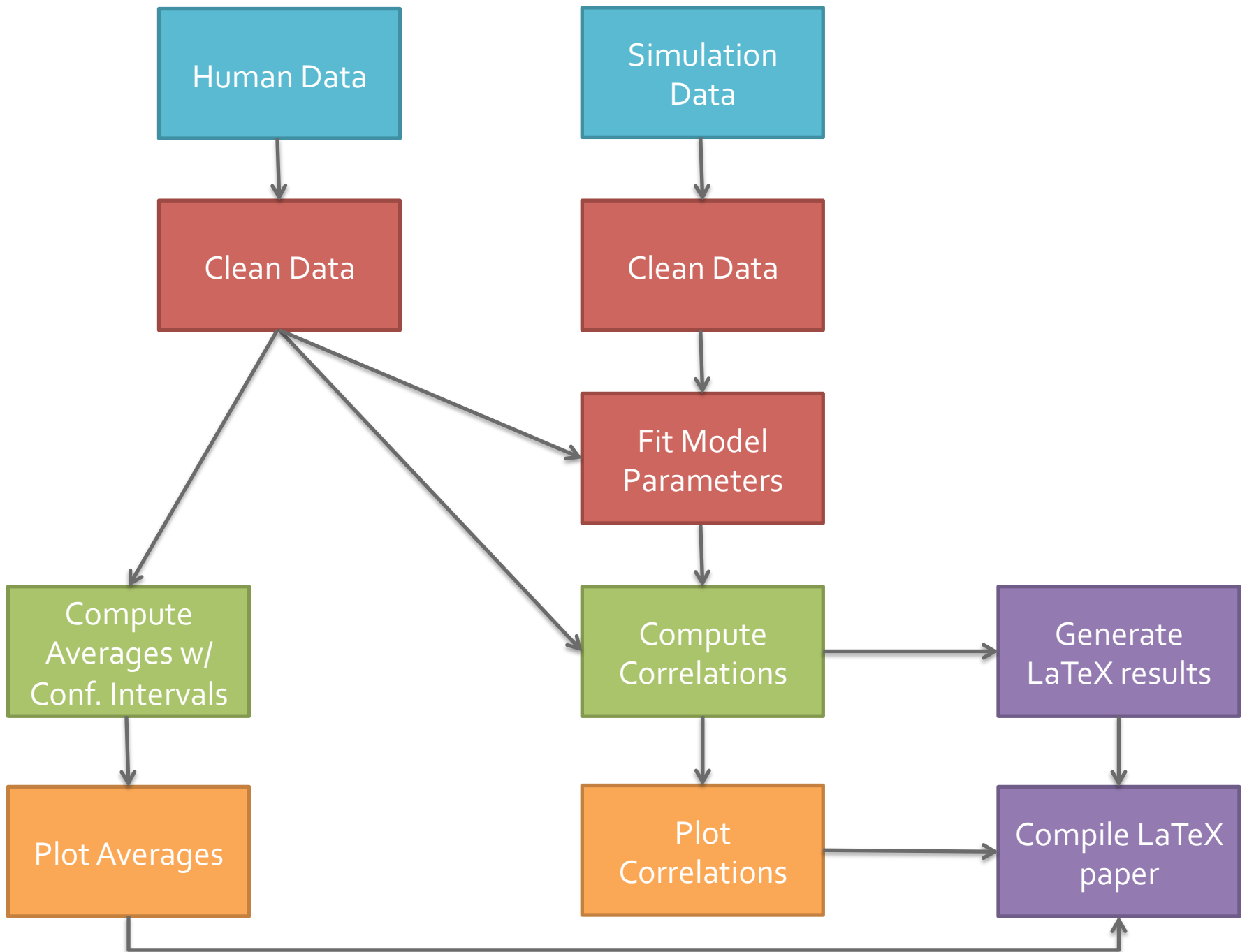
Make

CMake

SCons

...

**Idea: Run a series of commands in order,
ensuring dependencies are satisfied**



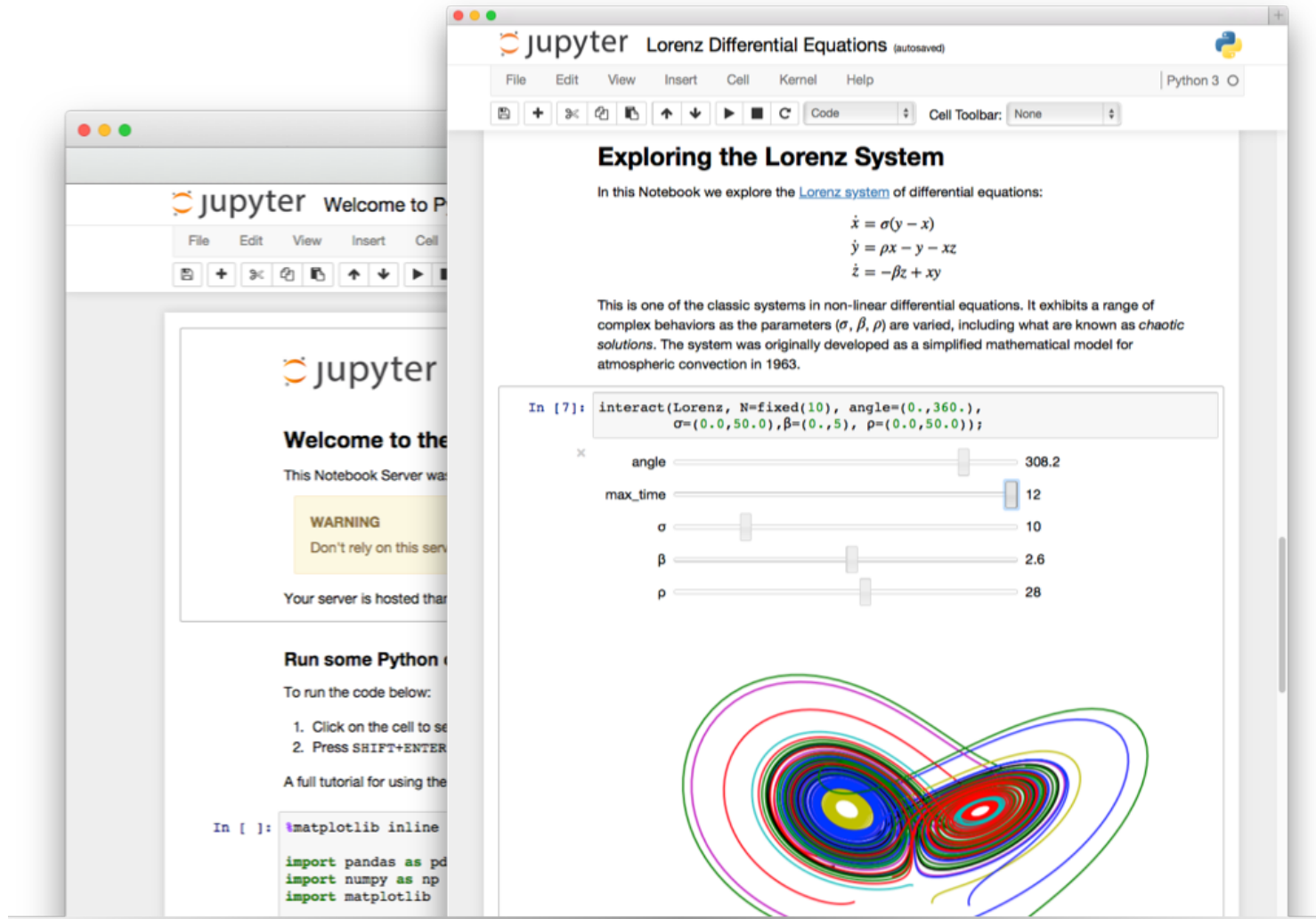
An example SConstruct file

```
import os
env = Environment(ENV=os.environ)
env.PDF("paper.pdf", "paper.tex")
env.Depends("paper.pdf", "fig1.pdf")
env.Depends("paper.pdf", "fig2.pdf")
env.Command(
    "fig1.pdf", "make_fig1.py", "make_fig1.py $TARGET")
env.Command(
    "fig2.pdf", "make_fig2.py", "make_fig2.py $TARGET")
```

\$ scons

```
scons: Reading SConscript files ...
scons: done reading SConscript files.
scons: Building targets ...
... a whole bunch of output ...
scons: done building targets.
```

But what if we want to run *notebooks*?



The image displays three overlapping Jupyter Notebook windows, illustrating the process of running and interacting with a notebook.

The background window shows the Jupyter "Welcome to the Jupyter Notebook" screen. It includes a "WARNING" box stating "Don't rely on this server" and a section titled "Run some Python code" with instructions on how to execute code cells.

The middle window shows a notebook titled "Lorenz Differential Equations (autosaved)". It contains the following text:

Exploring the Lorenz System

In this Notebook we explore the [Lorenz system](#) of differential equations:

$$\begin{aligned}\dot{x} &= \sigma(y - x) \\ \dot{y} &= \rho x - y - xz \\ \dot{z} &= -\beta z + xy\end{aligned}$$

This is one of the classic systems in non-linear differential equations. It exhibits a range of complex behaviors as the parameters (σ, β, ρ) are varied, including what are known as *chaotic solutions*. The system was originally developed as a simplified mathematical model for atmospheric convection in 1963.

The foreground window shows the same notebook with an interactive interface. It displays the following code cell:

```
In [7]: interact(Lorenz, N=fixed(10), angle=(0.,360.),
                sigma=(0.0,50.0), beta=(0.,5), rho=(0.0,50.0));
```

Below the code cell, there are five sliders for the parameters:

- angle: 308.2
- max_time: 12
- σ : 10
- β : 2.6
- ρ : 28

At the bottom of the foreground window, a plot of the Lorenz system's trajectory is shown, displaying the characteristic butterfly shape. The plot is rendered with multiple colored lines (red, green, blue, yellow, magenta) to show different trajectories.

Notebooks can be commands!

```
$ jupyter nbconvert \  
  --execute \  
  --to notebook \  
  --output notebook.ipynb \  
  notebook.ipynb
```

Notebooks can be SCons commands!

```
def build_notebook(target, source, env):
    notebook = str(source[0])
    cmd = [
        "jupyter", "nbconvert", "--execute", "--inplace",
        "--to", "notebook", "--output", notebook, notebook
    ]
    code = sp.call(cmd)
    if code != 0:
        raise RuntimeError("Error executing notebook")

    return None

env.Command(
    ["notebook.ipynb"],          # targets
    ["notebook.ipynb", "data.csv"], # sources
    build_notebook)              # command
```

Approach 4: multiple scripts
and / or notebooks with SCons

Drawback: keeping the
SConstruct file up-to-date
is a hassle 😞

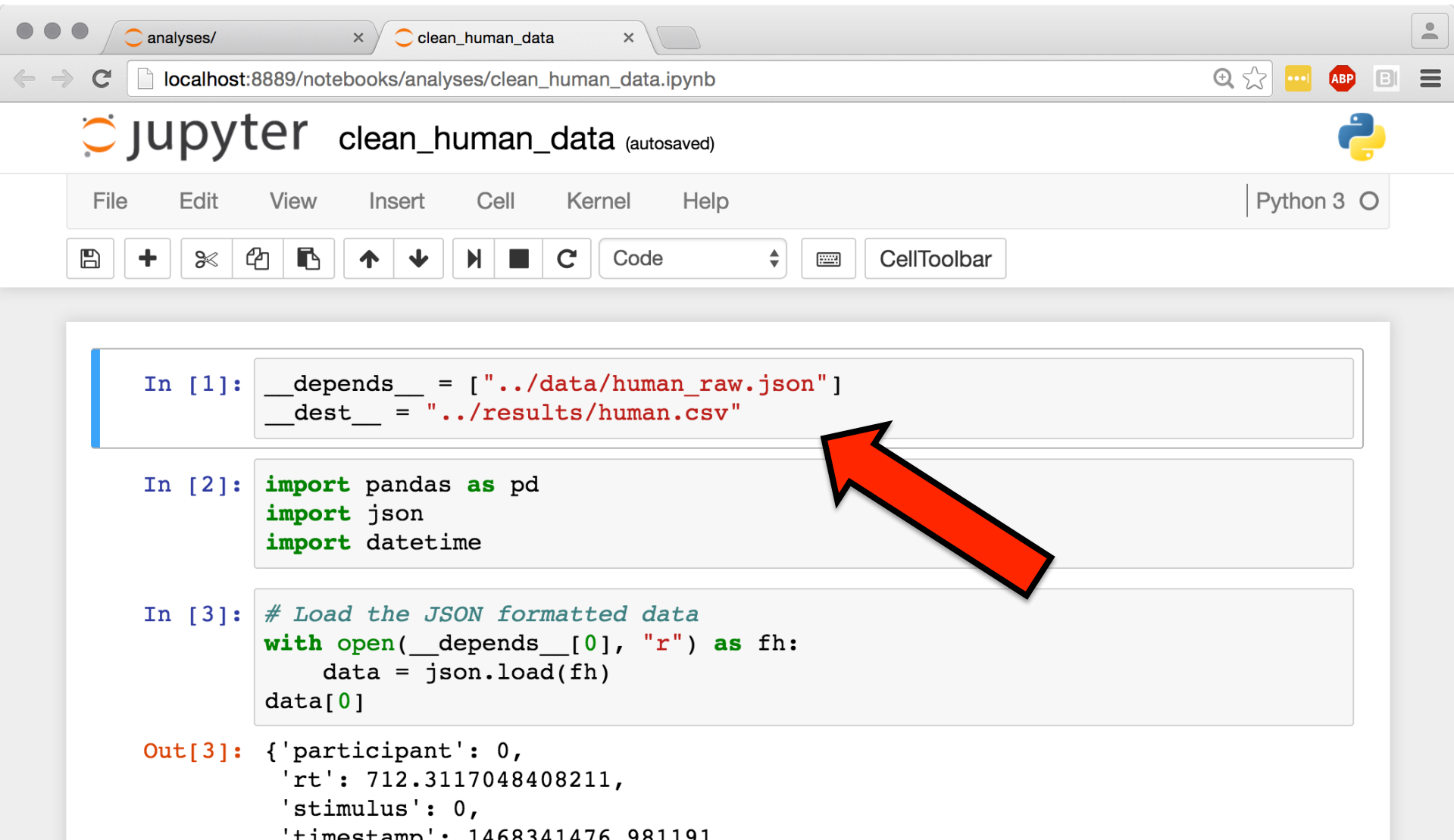
Approach 4: multiple scripts
and / or notebooks with nbflow

Drawback: notebooks must be
Python, requires Python 2 ☹️
(but notebooks can be Python 3)

Installing nbflow

```
$ pip2 install \
  git+git://github.com/jhamrick/nbflow.git
```

Notebooks with nbflow



The screenshot shows a Jupyter Notebook interface in a web browser. The browser tabs are labeled 'analyses/' and 'clean_human_data'. The address bar shows 'localhost:8889/notebooks/analyses/clean_human_data.ipynb'. The Jupyter logo and the notebook title 'clean_human_data (autosaved)' are visible. The menu bar includes File, Edit, View, Insert, Cell, Kernel, and Help. The toolbar shows icons for saving, adding, deleting, and running cells, along with a dropdown menu set to 'Code' and a 'CellToolbar' button. The notebook content area displays three input cells and one output cell. A large red arrow points to the first input cell, which contains the following code:

```
In [1]: __depends__ = ["../data/human_raw.json"]
        __dest__ = "../results/human.csv"
```

The second input cell contains the following code:

```
In [2]: import pandas as pd
import json
import datetime
```

The third input cell contains the following code:

```
In [3]: # Load the JSON formatted data
with open(__depends__[0], "r") as fh:
    data = json.load(fh)
data[0]
```

The output cell shows the following JSON object:

```
Out[3]: {'participant': 0,
         'rt': 712.3117048408211,
         'stimulus': 0,
         'timestamp': 1468341476.981191}
```

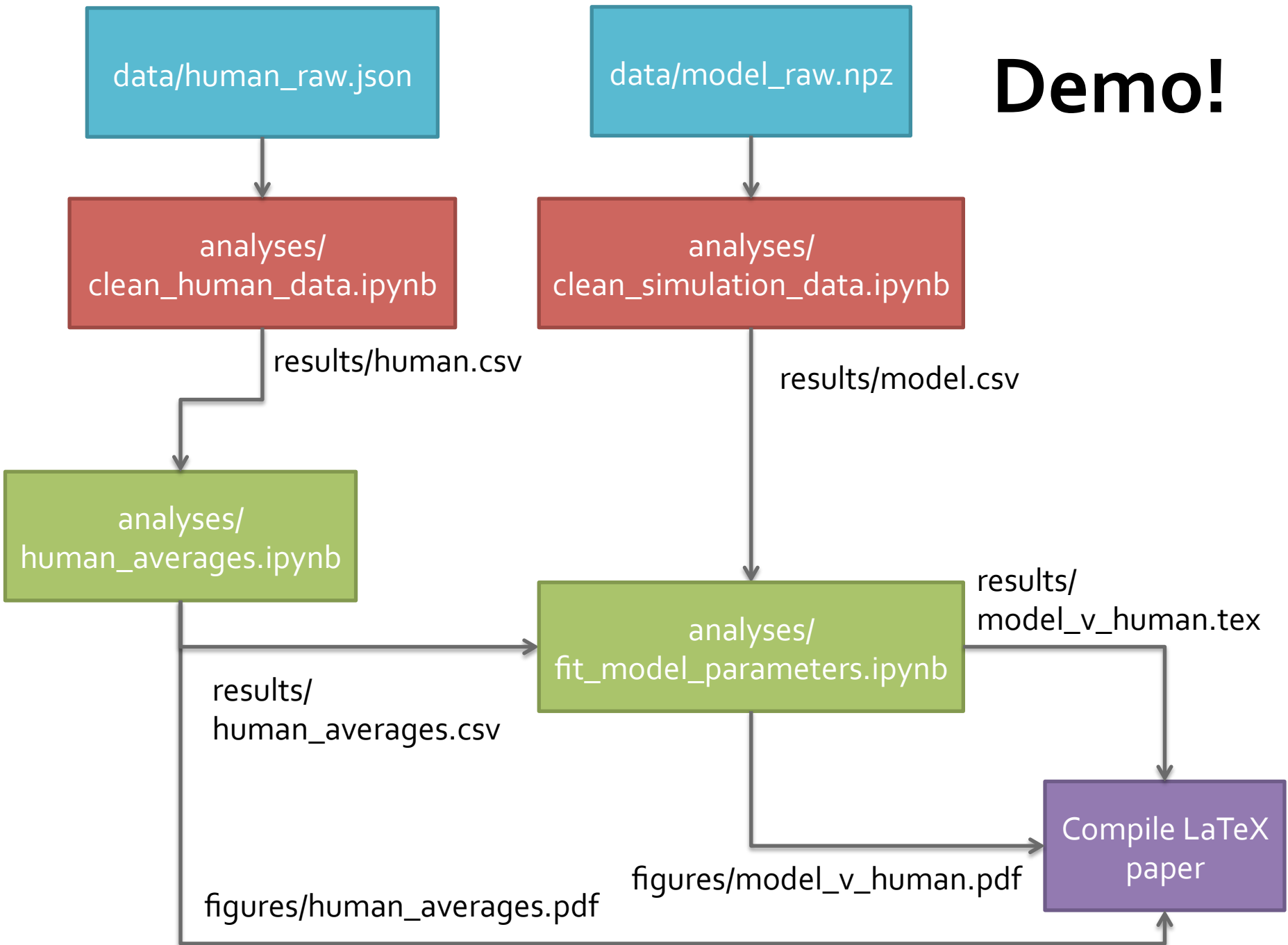
SConstruct file with nbflow

```
import os
from nbflow.scons import setup

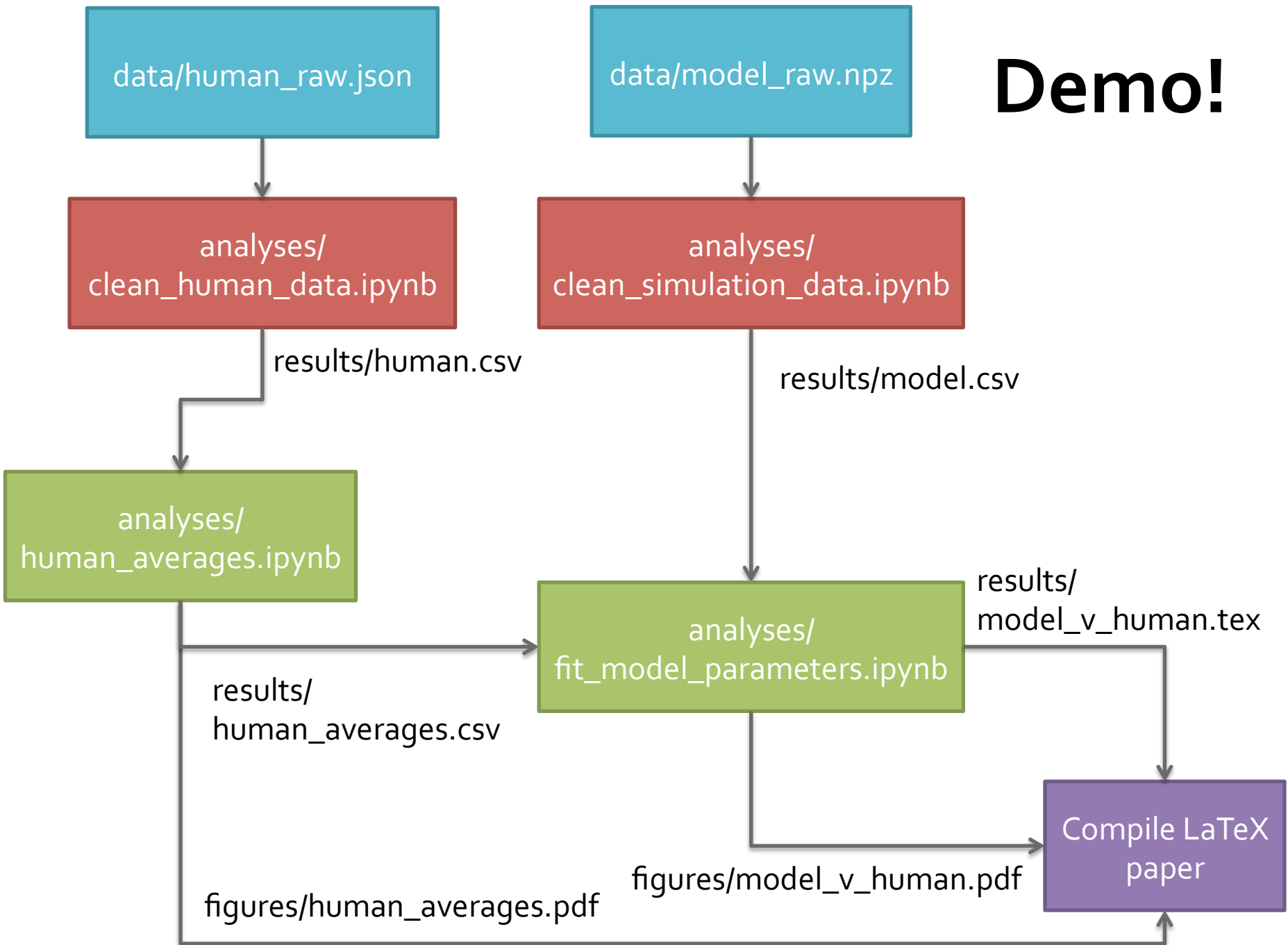
env = Environment(ENV=os.environ)
setup(env, ["dir_containing_notebooks"])
```

```
$ scons
scons: Reading SConscript files ...
scons: done reading SConscript files.
scons: Building targets ...
... a whole bunch of output ...
scons: done building targets.
```

Demo!



Demo!



Future Improvements

1. Supporting non-Python kernels
2. Supporting other build systems
3. Python 3

<http://tinyurl.com/nbflow-example>