Remind me again how we did that? Creating convenient and reproducible workflows using Jupyter Notebook

MORPC Data Day

March 1, 2023

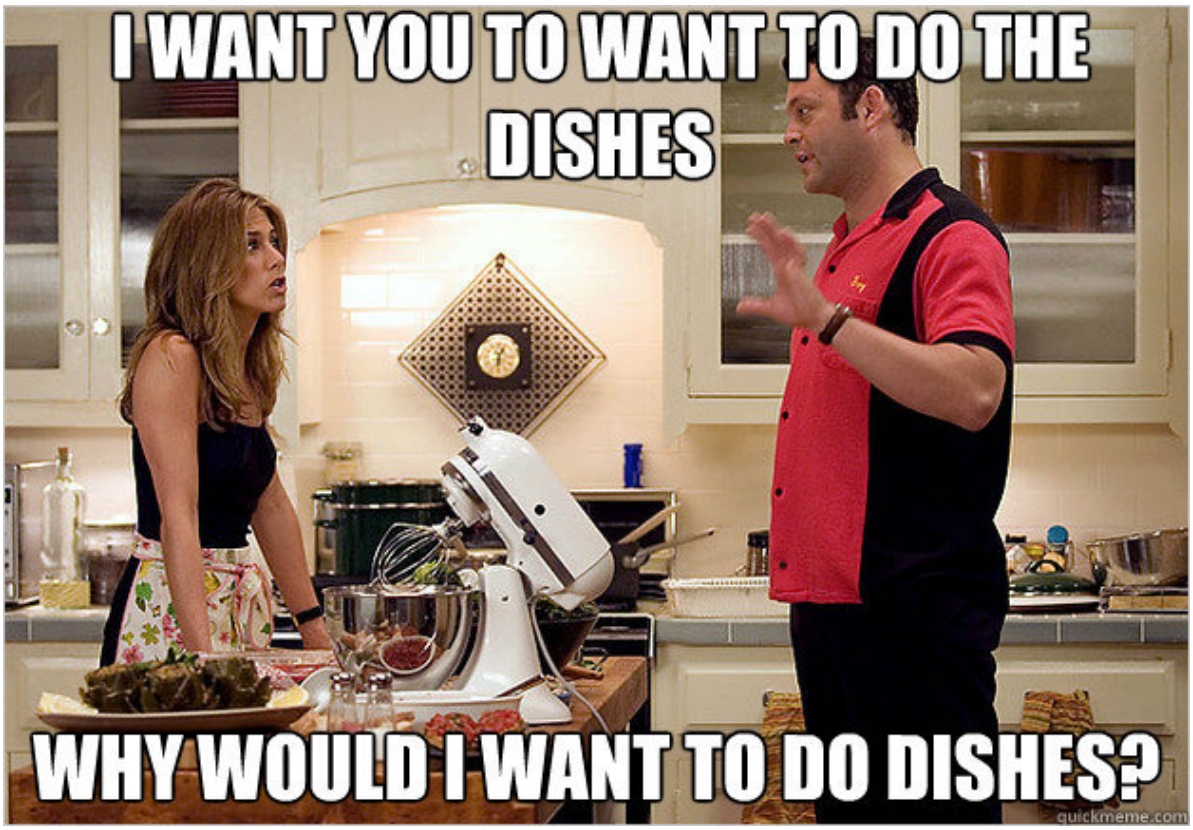
Adam Porr
Research & Data Officer
Mid-Ohio Regional Planning Commission

Abstract

Attendees will be introduced to Jupyter notebooks and how they can be used to produce well-documented automated workflows using Python code. We will walk through an existing Jupyter notebook that makes use of the popular Pandas data analysis and manipulation library (among others) and its geographic extensions (i.e. GeoPandas) to retrieve geographic data and attribute data from the Census website, integrate the retrieved data, and perform a simple common analysis task using the data. We anticipate that attendees will already have experience downloading, integrating, and analyzing Census data using other tools (e.g., data.census.gov, Excel, ArcGIS), therefore this presentation will focus on the automation of this workflow using Jupyter and the efficiency, transparency, and reproducibility benefits that can be realized from this strategy. All of tools demonstrated during the presentation will be free and open source, and the notebook will be made available so that attendees can experiment with the workflow after the presentation. Basic familiarity with Python or other programming languages is beneficial to understand the workflow implementation, but not necessary to appreciate the primary learning objectives.

Introduction

My goal for this presentation: I want you to **want** to create reproducible workflows.



How I'll make my case

- Rant about the problem as I see it.
- Introduce a shiny new tool (Jupyter)
- Show off a tiny fraction of the cool things you can do with Jupyter while...
  - Demonstrating how work done in Jupyter is inherently more reproducible
  - Suggesting practices to make your Jupyter workflows even more reproducible

Why we need reproducible workflows

Have you every been guilty of this?

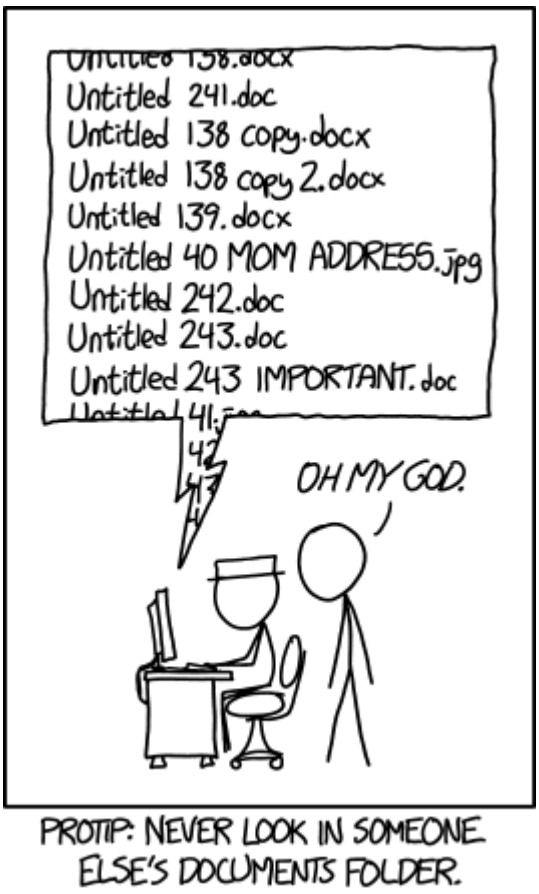


Image credit: Randall Monroe (XKCD)

My favorite is "Big official report FINAL v2 AMP comments.docx"

Or this?

File not found

Or this?

```
#TODO: Figure out what I’m doing here and comment accordingly.
```

```
// Peter wrote this, nobody knows what it does, don't change it!
```

```
// drunk, fix later
```

Why are we so bad a creating reproducible workflows?

- Creating reproducible workflows used to be difficult and tedious
- No wonder data scientists (and most other people) are bad at making workflows reproducible
- Creating reproducible workflows will lead to better:
  - Transparency
  - Efficiency
  - Sanity!

The case for programming your analysis

Code inherently provides documentation

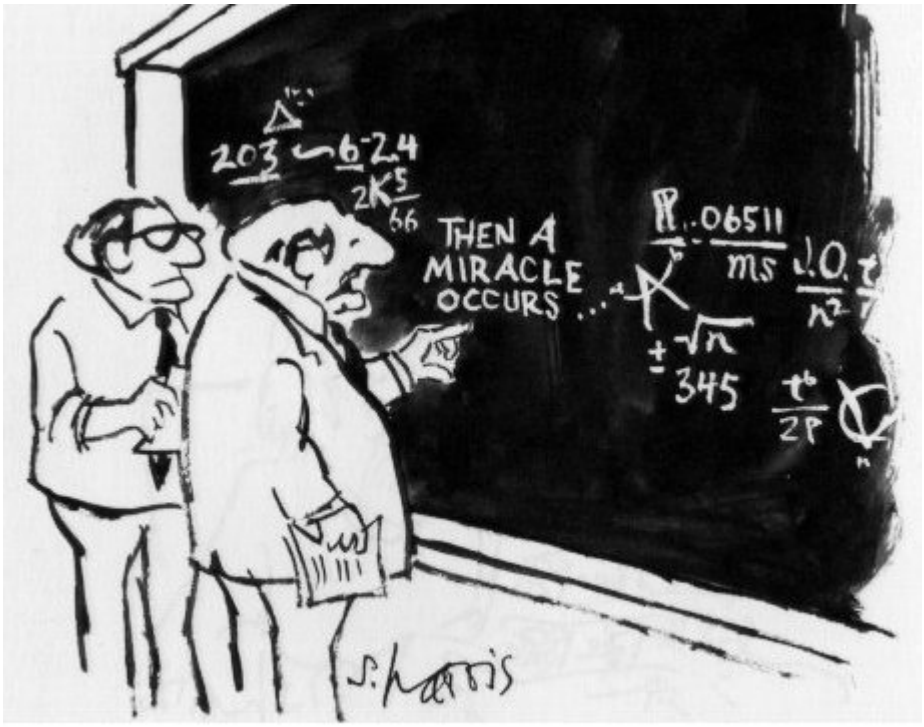
```
r = requests.get(DECENNIAL_API_SOURCE_URL, params=DECENNIAL_API_PARAMS)
decennialRaw = r.json()
columns = decennialRaw.pop(0)
decennialData = pd.DataFrame.from_records(decennialRaw, columns=columns) \
    .filter(items=morpc.avro_get_field_names(decennialSchema),
axis="columns") \
    .astype(morpc.avro_to_pandas_dtype_map(decennialSchema)) \
    .rename(columns=morpc.avro_map_to_first_alias(decennialSchema))
```

```
decennialData["GEOID"] = decennialData["GEOID"].apply(lambda x:x.split("US")
[1])
```


Example JSON


Example table

Code does only what you tell it to do



"I think you should be more explicit here in step two."

from *What's so Funny about Science?* by Sidney Harris (1977)

Code allows you to repeat steps with minimal effort


Example model

The case for literate programming

A few thoughtful, well-placed comments go a long way

```
<svg width="200" height="250" version="1.1"
xmlns="http://www.w3.org/2000/svg">
  <polygon points="50 160 55 180 70 180 60 190 65 205 50 195 35 205
40 190 30 180 45 180"
      stroke="green" fill="transparent" stroke-width="5"/>
</svg>
```

Draw a green five-pointed star

But a picture is even better!

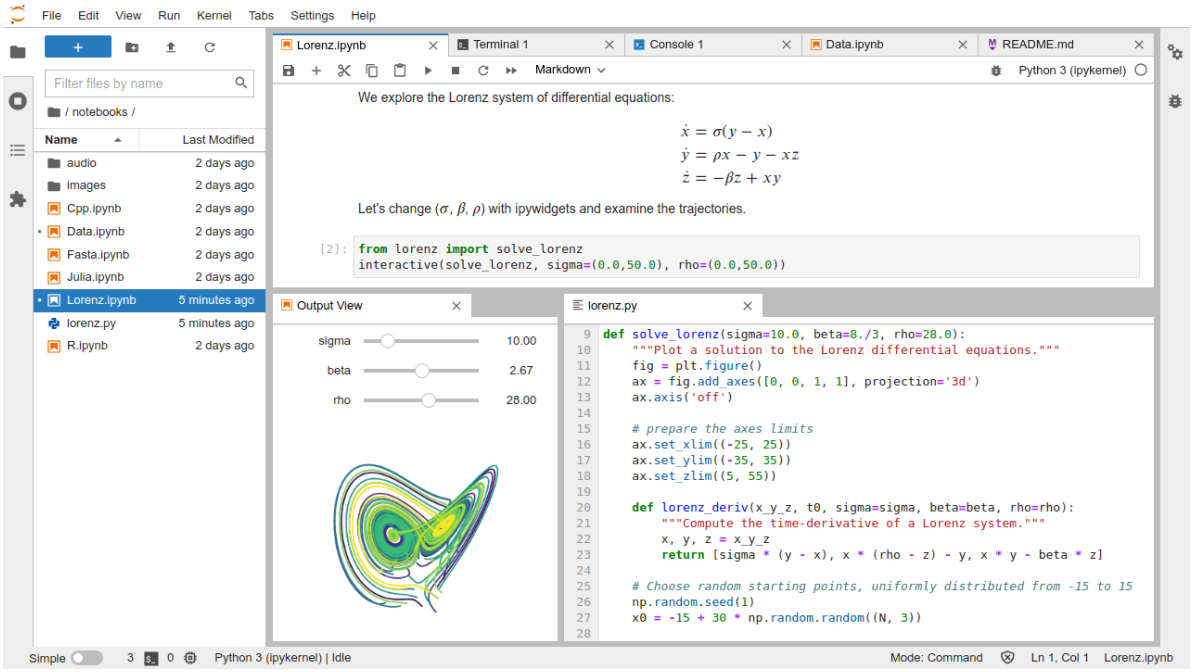
 Star

Sometimes you just need the gist

 Outline

Literate programming reads like a book but runs like code

Jupyter makes literate programming possible for mere mortals



Literate programming by example

Write an introduction

In the remainder of this notebook, I'll demonstrate a simple (but hopefully compelling) use case for Jupyter that highlights it's literate programming capabilities. It will use the popular pandas data analysis library and its spatially-enabled companion, geopandas. The notebook will fetch U.S. Census data from the web, do some common manipulations to it, plot it, map it, and perform a statistical analysis. Since I included an introduction, you can decide whether you care about any of that without having to try to interpret the rest of the document.

Specify the input and output data

It's good to specify all prerequisites at the beginning of the document, including data, parameters, and required libraries. This helps the reader understand the inputs and outputs and what steps they may have to take prior to running the script.

Input data

We'll retrieve population estimates and factors of population change from the Census Population Estimates Program (PEP) website and archive a local copy.

In [1]:
```
CENSUS_PEP_URL = "https://www2.census.gov/programs-surveys/popest/datasets/2010-2019/c
CENSUS_PEP_ARCHIVE_PATH = "./input_data/census_pep.csv"
```

We'll also need the county polygons Shapefile from the Census geography program.

In [2]:
```
CENSUS_COUNTY_POLYGONS_URL = "https://www2.census.gov/geo/tiger/TIGER2019/COUNTY/tl_20
CENSUS_COUNTY_POLYGONS_ARCHIVE_PATH = "./input_data/county_polygons.shp"
```

Output data

We'll produce a spatial database (Shapefile) that includes the geometries and numerical population change for Franklin County and the surrounding counties.

In [3]:
```
POPCHANGE_FEATURECLASS_PATH = "./output_data/popchange.shp"
```

Don't forget to specify the schemas!

Schemas help humans to understand how datasets are structured and what the variables represent. If the schema is specified in a machine readable format like this, then the computer can parse the data without additional effort from a human.

In [4]:
```
POPCHG_FEATURECLASS_SCHEMA = {
    "type": "record",
    "doc": "Numerical population change for Central Ohio counties for the years 2010 t
    "fields": [
        {"name":"CTYNAME", "type":"string", "doc":"Name of the county"},
        {"name":"2010", "type":"int", "doc":"Numeric change in resident total populati
        {"name":"2011", "type":"int", "doc":"Numeric change in resident total populati
        {"name":"2012", "type":"int", "doc":"Numeric change in resident total populati
        {"name":"2013", "type":"int", "doc":"Numeric change in resident total populati
```

```
        {"name":"2014", "type":"int", "doc":"Numeric change in resident total populati
        {"name":"2015", "type":"int", "doc":"Numeric change in resident total populati
        {"name":"2016", "type":"int", "doc":"Numeric change in resident total populati
        {"name":"2017", "type":"int", "doc":"Numeric change in resident total populati
        {"name":"2018", "type":"int", "doc":"Numeric change in resident total populati
        {"name":"2019", "type":"int", "doc":"Numeric change in resident total populati
    ]
}
```

Specify any parameters

In [5]:
```
# For COUNTY_NAMES, enter a list of strings representing the names of the counties of
COUNTY_NAMES = \
    ["Franklin","Fairfield","Pickaway","Madison",
     "Union", "Delaware", "Licking"]

# For STATE_NAME, enter a string representing name of the state where the counties of
# are located.  For STATE_FIPS, enter the two digit FIPS code assigned to the state by
STATE_NAME = "Ohio"
STATE_FIPS = "39"
```

Import required libraries

In [6]:
```
import os                 # Perform basic filesystem operations, such as creating dir
import pandas as pd       # Create and manipulate tabular data in the form of datafra
import geopandas as gpd   # Create and manipulate spatial data in the form of geodata
from scipy import stats   # Perform statistical analysis
import numpy as np        # General purpose numerical computation library
```

Prepare the environment

Create subdirectories to store the input data and output data if they don't already exist.

In [7]:
```
if not os.path.exists("./input_data"):
    os.makedirs("./input_data")

if not os.path.exists("./output_data"):
    os.makedirs("./output_data")
```

Finally, let's get some data! Pandas makes working tabular data a dream.

Read the CSV file directly from the Census website. This particular file uses an atypical text encoding format, so we have to specify it explicitly. Pandas can determine the encoding for most CSV files automatically. After downloading the file, save an archival copy and display some sample records.

In [8]:
```
censusPepRaw = pd.read_csv(CENSUS_PEP_URL, encoding="ISO-8859-1")
censusPepRaw.to_csv(CENSUS_PEP_ARCHIVE_PATH, index=False)
# If you need to load the archival copy, use the following line instead
#censusPepRaw = pd.read_csv(CENSUS_PEP_ARCHIVE_PATH, encoding="ISO-8859-1")
censusPepRaw.head()
```

Out[8]:

	SUMLEV	REGION	DIVISION	STATE	COUNTY	STNAME	CTYNAME	CENSUS2010POP	ESTIMATES
0	40	3	6	1	0	Alabama	Alabama	4779736	
1	50	3	6	1	1	Alabama	Autauga County	54571	
2	50	3	6	1	3	Alabama	Baldwin County	182265	
3	50	3	6	1	5	Alabama	Barbour County	27457	
4	50	3	6	1	7	Alabama	Bibb County	22915	

5 rows × 164 columns

Extract only the records that are county level (SUMLEV equal to 50) and whose state name (STNAME) and county name (CTYNAME) match those specified in the parameters. When specifying the county names, it is necessary to append " County" to match the format of CTYNAME.

In [9]:
```
censusPep = censusPepRaw.loc[ \
    (censusPepRaw["SUMLEV"] == 50) & \
    (censusPepRaw["STNAME"] == STATE_NAME) & \
    (censusPepRaw["CTYNAME"].isin(["{} County".format(x) for x in COUNTY_NAMES]))
].copy()
censusPep.head()
```

Out[9]:

	SUMLEV	REGION	DIVISION	STATE	COUNTY	STNAME	CTYNAME	CENSUS2010POP	ESTIMA
2099	50	2	3	39	41	Ohio	Delaware County	174214	
2101	50	2	3	39	45	Ohio	Fairfield County	146156	
2103	50	2	3	39	49	Ohio	Franklin County	1163414	
2123	50	2	3	39	89	Ohio	Licking County	166492	
2127	50	2	3	39	97	Ohio	Madison County	43435	

5 rows × 164 columns

The CSV includes **a lot** of variables, but we are only interested in numerical population change.

In [10]:
```
print(", ".join(list(censusPep.columns)))
```

```
SUMLEV, REGION, DIVISION, STATE, COUNTY, STNAME, CTYNAME, CENSUS2010POP, ESTIMATESBAS
E2010, POPESTIMATE2010, POPESTIMATE2011, POPESTIMATE2012, POPESTIMATE2013, POPESTIMAT
E2014, POPESTIMATE2015, POPESTIMATE2016, POPESTIMATE2017, POPESTIMATE2018, POPESTIMAT
E2019, NPOPCHG_2010, NPOPCHG_2011, NPOPCHG_2012, NPOPCHG_2013, NPOPCHG_2014, NPOPCHG_
2015, NPOPCHG_2016, NPOPCHG_2017, NPOPCHG_2018, NPOPCHG_2019, BIRTHS2010, BIRTHS2011,
BIRTHS2012, BIRTHS2013, BIRTHS2014, BIRTHS2015, BIRTHS2016, BIRTHS2017, BIRTHS2018, B
IRTHS2019, DEATHS2010, DEATHS2011, DEATHS2012, DEATHS2013, DEATHS2014, DEATHS2015, DE
ATHS2016, DEATHS2017, DEATHS2018, DEATHS2019, NATURALINC2010, NATURALINC2011, NATURAL
INC2012, NATURALINC2013, NATURALINC2014, NATURALINC2015, NATURALINC2016, NATURALINC20
17, NATURALINC2018, NATURALINC2019, INTERNATIONALMIG2010, INTERNATIONALMIG2011, INTER
NATIONALMIG2012, INTERNATIONALMIG2013, INTERNATIONALMIG2014, INTERNATIONALMIG2015, IN
TERNATIONALMIG2016, INTERNATIONALMIG2017, INTERNATIONALMIG2018, INTERNATIONALMIG2019,
DOMESTICMIG2010, DOMESTICMIG2011, DOMESTICMIG2012, DOMESTICMIG2013, DOMESTICMIG2014,
DOMESTICMIG2015, DOMESTICMIG2016, DOMESTICMIG2017, DOMESTICMIG2018, DOMESTICMIG2019,
NETMIG2010, NETMIG2011, NETMIG2012, NETMIG2013, NETMIG2014, NETMIG2015, NETMIG2016, N
ETMIG2017, NETMIG2018, NETMIG2019, RESIDUAL2010, RESIDUAL2011, RESIDUAL2012, RESIDUAL
2013, RESIDUAL2014, RESIDUAL2015, RESIDUAL2016, RESIDUAL2017, RESIDUAL2018, RESIDUAL2
019, GQESTIMATESBASE2010, GQESTIMATES2010, GQESTIMATES2011, GQESTIMATES2012, GQESTIMA
TES2013, GQESTIMATES2014, GQESTIMATES2015, GQESTIMATES2016, GQESTIMATES2017, GQESTIMA
TES2018, GQESTIMATES2019, RBIRTH2011, RBIRTH2012, RBIRTH2013, RBIRTH2014, RBIRTH2015,
RBIRTH2016, RBIRTH2017, RBIRTH2018, RBIRTH2019, RDEATH2011, RDEATH2012, RDEATH2013, R
DEATH2014, RDEATH2015, RDEATH2016, RDEATH2017, RDEATH2018, RDEATH2019, RNATURALINC201
1, RNATURALINC2012, RNATURALINC2013, RNATURALINC2014, RNATURALINC2015, RNATURALINC201
6, RNATURALINC2017, RNATURALINC2018, RNATURALINC2019, RINTERNATIONALMIG2011, RINTERNA
TIONALMIG2012, RINTERNATIONALMIG2013, RINTERNATIONALMIG2014, RINTERNATIONALMIG2015, R
INTERNATIONALMIG2016, RINTERNATIONALMIG2017, RINTERNATIONALMIG2018, RINTERNATIONALMIG
2019, RDOMESTICMIG2011, RDOMESTICMIG2012, RDOMESTICMIG2013, RDOMESTICMIG2014, RDOMEST
ICMIG2015, RDOMESTICMIG2016, RDOMESTICMIG2017, RDOMESTICMIG2018, RDOMESTICMIG2019, RN
ETMIG2011, RNETMIG2012, RNETMIG2013, RNETMIG2014, RNETMIG2015, RNETMIG2016, RNETMIG20
17, RNETMIG2018, RNETMIG2019
```

Create a new dataframe that includes only the county name (CTYNAME) and fields whose name includes "NPOPCHG"

In [11]:
```
censusPepPopChg = pd.concat([
    censusPep.filter(like="CTYNAME", axis="columns"),
    censusPep.filter(like="NPOPCHG", axis="columns")
], axis="columns")
censusPepPopChg.head()
```

Out[11]:

	CTYNAME	NPOPCHG_2010	NPOPCHG_2011	NPOPCHG_2012	NPOPCHG_2013	NPOPCHG_2014
2099	Delaware County	927	3436	2592	4253	4060
2101	Fairfield County	223	757	127	1495	1564
2103	Franklin County	2726	14598	18245	19833	19484
2123	Licking County	223	459	425	872	949
2127	Madison County	-4	-320	-123	265	724

Eliminate the " County" suffix from the county names.

In [12]:
```
censusPepPopChg["CTYNAME"] = censusPepPopChg["CTYNAME"].str.replace(" County", "")
```

Aside from CTYNAME, all of our columns include two parts separated by an underscore - a prefix "NPOPCHG" and a suffix representing the four-digit year. Rename the columns, retaining only the year. Index by county name.

In [13]:
```
censusPepPopChg = censusPepPopChg \
    .set_index("CTYNAME") \
    .rename(columns=(lambda x:x[-4:]))
censusPepPopChg.head()
```

Out[13]:

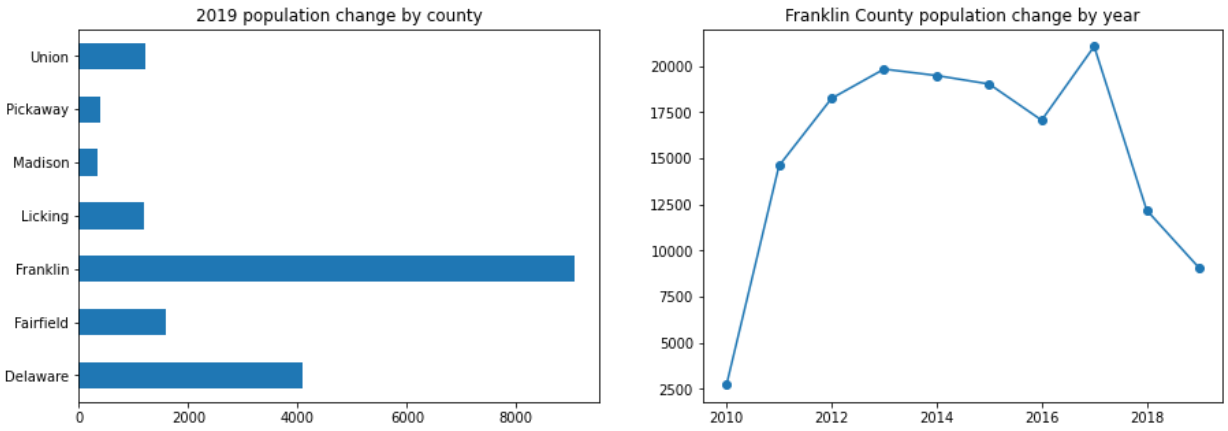
CTYNAME	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019
Delaware	927	3436	2592	4253	4060	3951	3753	3726	4221	4086
Fairfield	223	757	127	1495	1564	894	1535	1897	1296	1592
Franklin	2726	14598	18245	19833	19484	19024	17064	21060	12188	9058
Licking	223	459	425	872	949	1201	1382	1624	2049	1196
Madison	-4	-320	-123	265	724	159	-762	664	348	342

Jupyter can create charts.

Plot the 2019 population change by county as a horizontal bar chart and Franklin County's population change by year as a line chart.

In [14]:
```
import matplotlib.pyplot as plt
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(15,5))
censusPepPopChg["2019"].plot.barh(ax=axes[0], title="2019 population change by county'
censusPepPopChg.loc["Franklin"].plot(ax=axes[1], title="Franklin County population cha
```

Out[14]:
```
<AxesSubplot:title={'center':'Franklin County population change by year'}>
```



Maps too!

Read the county geography Shapefile directly from the Census website. Note that GeoPandas can read the zipped Shapefile directly; it is not necessary to download it and unzip it first. Save an archival copy of the Shapefile.

In [15]:
```
censusCountyPolysRaw = gpd.read_file(CENSUS_COUNTY_POLYGONS_URL)
censusCountyPolysRaw.to_file(CENSUS_COUNTY_POLYGONS_ARCHIVE_PATH, driver="ESRI Shapefi
# If you need to load the archival copy, use the following line instead
# censusCountyPolysRaw = gpd.read_file(CENSUS_COUNTY_POLYGONS_ARCHIVE_PATH)
censusCountyPolysRaw.head()
```

Out[15]:

	STATEFP	COUNTYFP	COUNTYNS	GEOID	NAME	NAMELSAD	LSAD	CLASSFP	MTFCC	CSAF
0	31	039	00835841	31039	Cuming	Cuming County	06	H1	G4020	Nor
1	53	069	01513275	53069	Wahkiakum	Wahkiakum County	06	H1	G4020	Nor
2	35	011	00933054	35011	De Baca	De Baca County	06	H1	G4020	Nor
3	31	109	00835876	31109	Lancaster	Lancaster County	06	H1	G4020	33
4	31	129	00835886	31129	Nuckolls	Nuckolls County	06	H1	G4020	Nor

This Shapefile includes all counties nationwide. Filter by state using the state FIPS code (state name is not available). Then, as before, filter by county name after appending the " County" suffix. After filtering eliminate the suffix. Index by county name.

In [16]:
```
censusCountyPolys = censusCountyPolysRaw.loc[ \
    (censusCountyPolysRaw["STATEFP"] == STATE_FIPS) & \
    (censusCountyPolysRaw["NAMELSAD"].isin(["{} County".format(x) for x in COUNTY_NAME
].copy()
censusCountyPolys["NAMELSAD"] = censusCountyPolys["NAMELSAD"].str.replace(" County", "
censusCountyPolys = censusCountyPolys.set_index(["NAMELSAD"])
censusCountyPolys.head()
```

Out[16]:

	STATEFP	COUNTYFP	COUNTYNS	GEOID	NAME	LSAD	CLASSFP	MTFCC	CSAFP	CI
NAMELSAD										
Pickaway	39	129	01074077	39129	Pickaway	06	H1	G4020	198	
Delaware	39	041	01074033	39041	Delaware	06	H1	G4020	198	
Madison	39	097	01074061	39097	Madison	06	H1	G4020	198	
Franklin	39	049	01074037	39049	Franklin	06	H1	G4020	198	
Union	39	159	01074091	39159	Union	06	H1	G4020	198	

Join our population change data to the polygons, aligning on the index (i.e. the county name).

In [17]:
```
censusPepPolys = gpd.GeoDataFrame(data=censusPepPopChg.copy(), geometry=censusCountyPo
censusPepPolys.head()
```

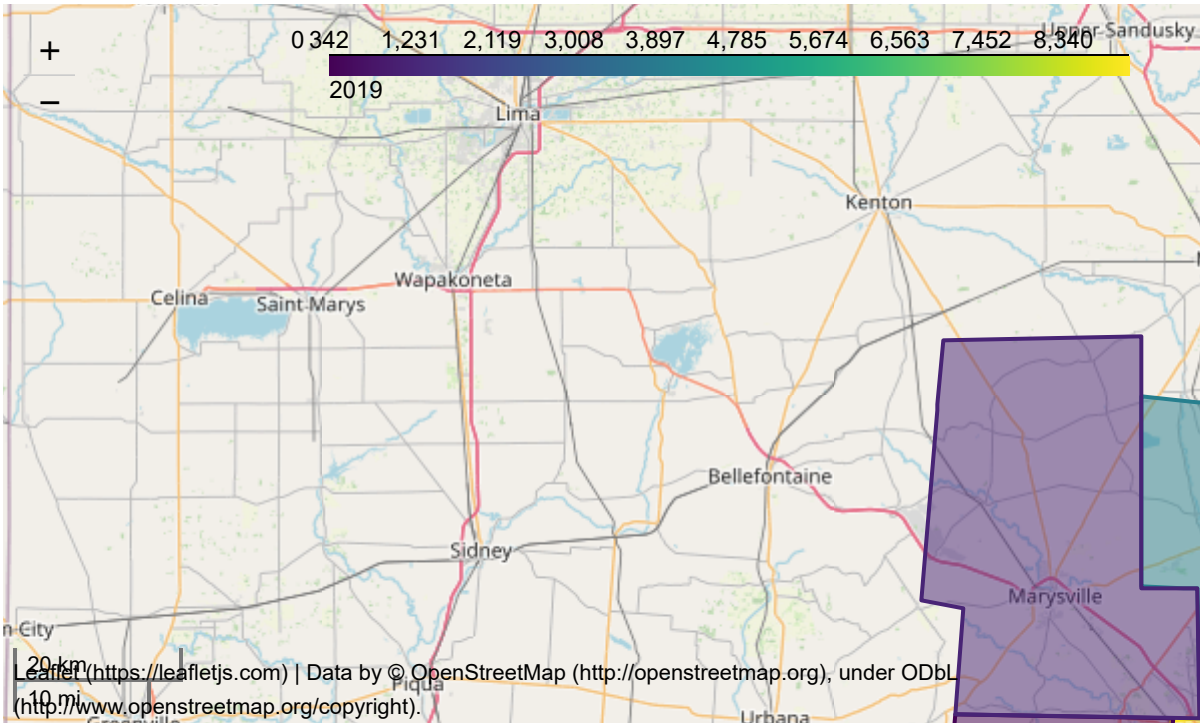
Out[17]:

CTYNAME	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019	geometry
Delaware	927	3436	2592	4253	4060	3951	3753	3726	4221	4086	POLYGON ((-83.19229 40.24440, -83.19672 40.244...
Fairfield	223	757	127	1495	1564	894	1535	1897	1296	1592	POLYGON ((-82.80248 39.82295, -82.80242 39.823...
Franklin	2726	14598	18245	19833	19484	19024	17064	21060	12188	9058	POLYGON ((-83.01188 40.13656, -83.01171 40.136...
Licking	223	459	425	872	949	1201	1382	1624	2049	1196	POLYGON ((-82.76183 40.12586, -82.76181 40.125...
Madison	-4	-320	-123	265	724	159	-762	664	348	342	POLYGON ((-83.54053 39.91715, -83.54039 39.917...

Make a nice interactive map. Yes, this is all it takes!

In [18]:
```
censusPepPolys.explore(column="2019")
```

Out[18]:



And statistical analyses? You bet.

Support for statistical analysis in pandas is limited, so we'll use the statsmodels library instead. Convert our time index (column names) and time series values (Franklin County population change) to individual variables for convenience. Note that the column names are strings, so it is necessary to convert them to integers.

In [19]:
```
x = censusPepPopChg.columns.to_series().astype("int")
y = censusPepPopChg.loc["Franklin"]
```

Perform linear regression and calculate slope and standard error

In [20]:
```
slope, intercept, r_value, p_value, std_err = stats.linregress(x, y)
trendline = slope * x + intercept
print("Slope: {:1.2f}".format(slope))
print("Intercept: {:1.2f}".format(intercept))
print("Standard error: {:1.2f}".format(std_err))
print("Standard error as percent of mean: {:1.2f}%".format(std_err/y.mean()*100))
```

```
Slope: 275.31
Intercept: -539282.16
Standard error: 671.80
Standard error as percent of mean: 4.38%
```

Calculate t-statistic and p-value for one-tailed test (significance level = 0.05) and print results

In [21]:
```
def print_trend_assessments(p_value, alpha, slope):
    if p_value < alpha and slope < 0:
        print("There is significant evidence (p = {:.4f}) of a negative trend".format(
    else:
        print("There is no significant evidence of a negative trend")
    if p_value < alpha and slope > 0:
        print("There is significant evidence (p = {:.4f}) of a positive trend".format(
```

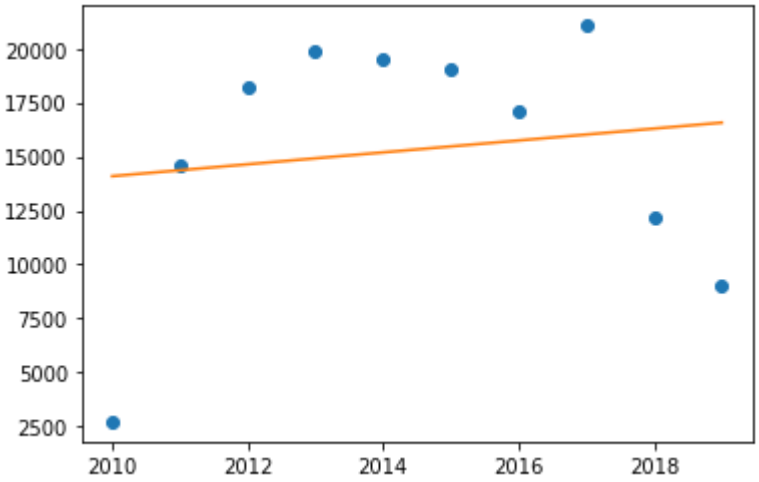
```
    else:
        print("There is no significant evidence of a positive trend")
```

In [22]:
```
t_stat = slope / (std_err / np.sqrt(len(x)))
p_value = stats.t.sf(np.abs(t_stat), len(x)-2)

alpha = 0.05
print_trend_assessments(p_value, alpha, slope)

plt.plot(x,y,"o")
plt.plot(x,trendline)
plt.show()
```

```
There is no significant evidence of a negative trend
There is no significant evidence of a positive trend
```



Need some nicely-formatted output for the boss? Jupyter has you covered.

- HTML
- PDF
- Microsoft Word
- Presentations
- Interactive dashboards
- Websites
- Books

Note: Some of these export options are built into Jupyter Notebook and JupyterLab. For others, you need a more sophisticated export tool like Quarto.

Closing thoughts

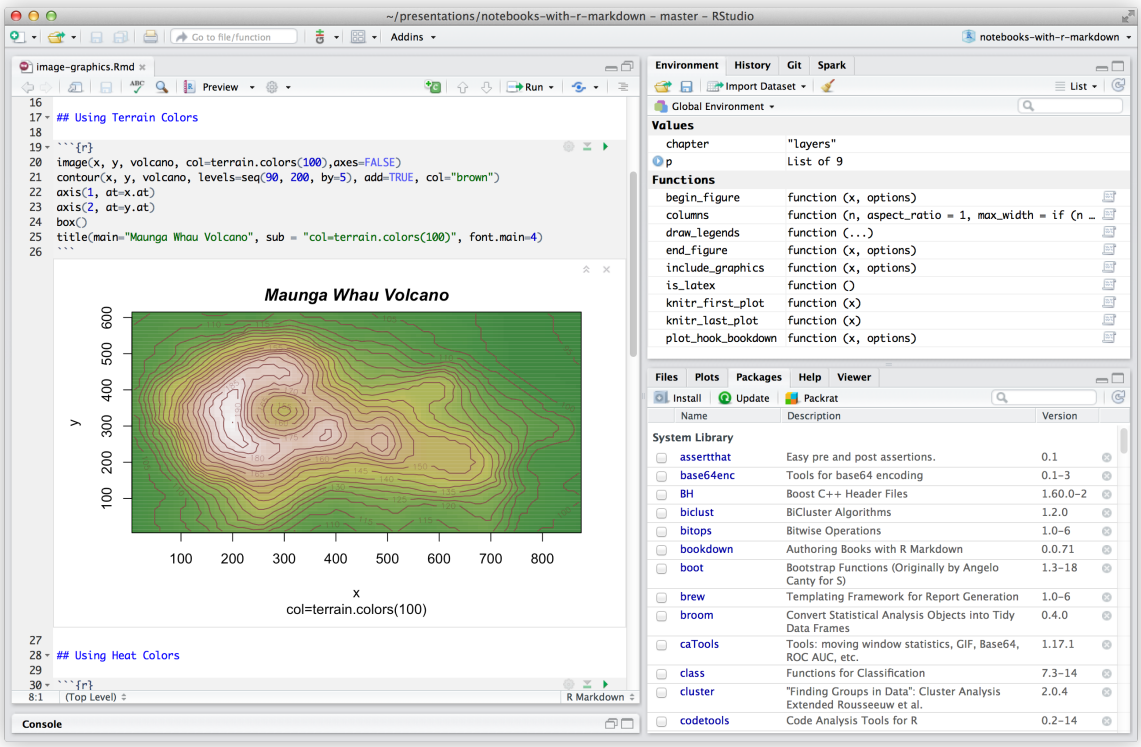
Your environment should be reproducible too

- Python virtual environments (for Python-only work)
- Docker (for more complex workflows)
- Documentation (if you like doing extra work)

Use version control

- Lots of choices. Git (often via GitHub) is most popular.
- Control versions of your data too!
- Text formats allow you to see diffs

You can do all of this in R too



No Jupyter? No problem. Binder lets you run Jupyter in the cloud.

Jupyter is Free Software (so are Python and R)

10 commandments of reproducible data science

1. Thou shalt **provide a succint and simple introduction**
2. Thou shalt **capture every step and assumption**
3. Thou shalt **capture metadata** for inputs and outputs
4. Thou shalt **explain complex operations** in words thine peers can understand
5. Thou shalt **collect prerequisite steps in one place**
6. Thou shalt **preserve versions** of thine process
7. Thou shalt **get thine input data directly from the source**
8. Thou shalt **save a copy of thine input data**
9. Thou shalt **document thine analysis environment**
10. Thou shalt use **freely-available tools** that thine peers have access to

Questions?

Accessing the content from this presentation

All of the content presented today is publicly available in GitHub:

https://github.com/aporr/jupyter-reproducible-workflows

The slides are available directly from the following URL:

https://aporr.github.io/jupyter-reproducible-workflows/slides.html

The slides are implemented using Reveal.js, which arranges slides in a 2D layout. Press `PGDN` to move to the next slide or `PGUP` to move to previous slide, or press `ESC` to see an overview and move through the slides non-linearly.