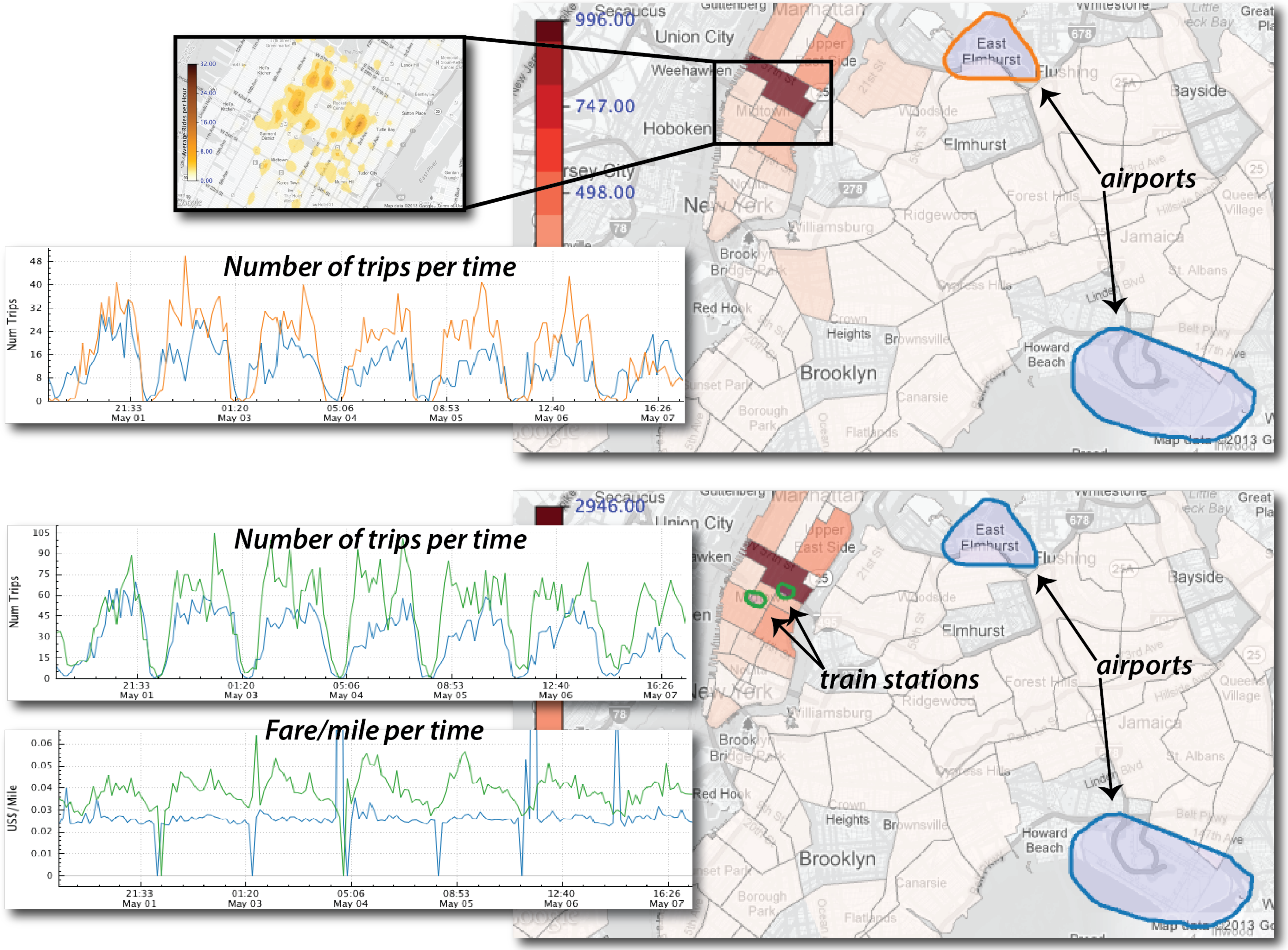
Advanced Data Management (CSCI 640/490)

Python

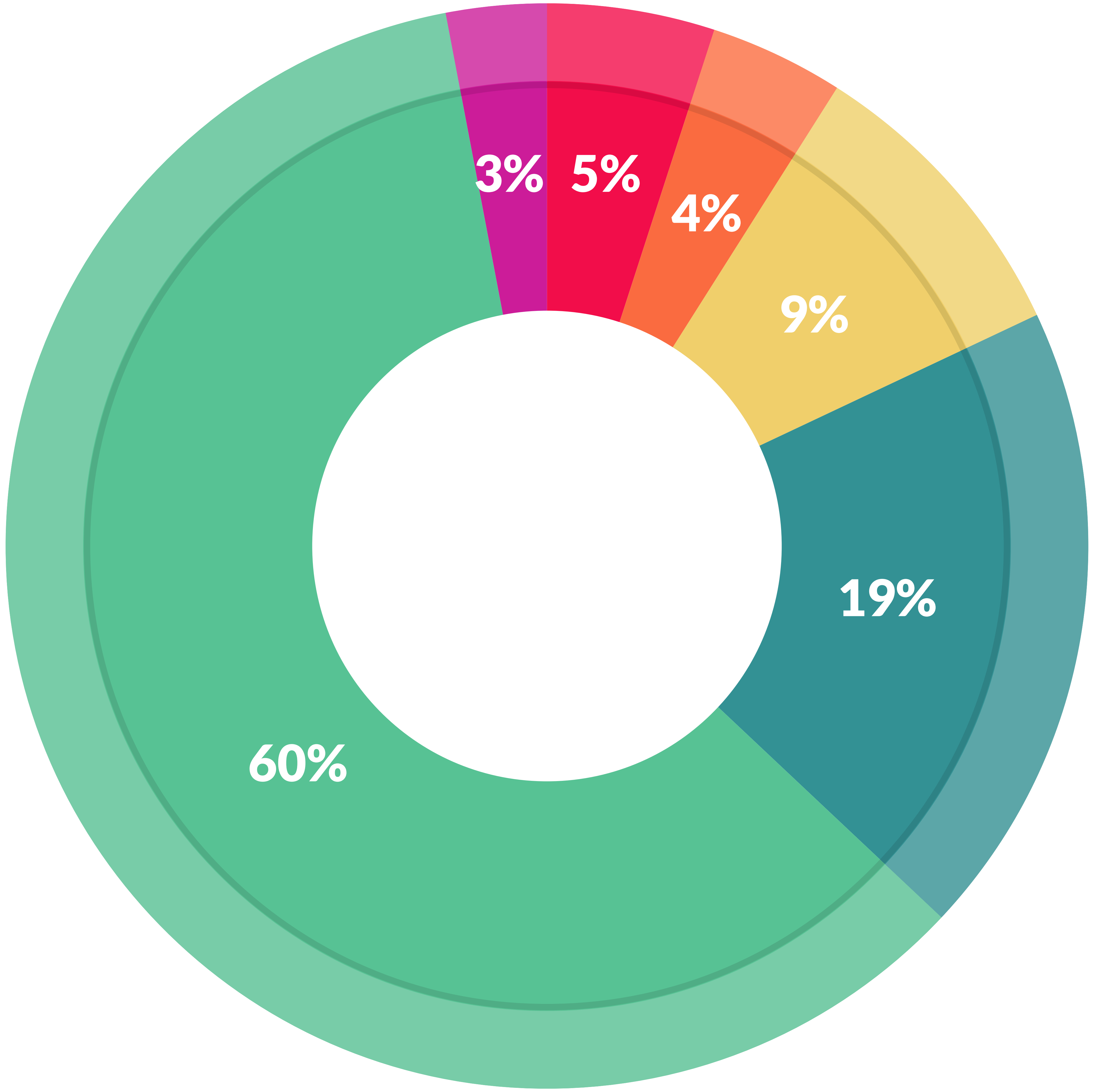
Dr. David Koop

Supporting Data Science



[Ferreira et al., 2013]

How do data scientists spend their time?



What data scientists spend the most time doing

- *Building training sets: 3%*
- *Cleaning and organizing data: 60%*
- *Collecting data sets; 19%*
- *Mining data for patterns: 9%*
- *Refining algorithms: 4%*
- *Other: 5%*

[CrowdFlower Data Science Report, 2016]

Data Wrangling

	A	B	C	D
1	**Transaction Date**	**Customer Name**	**Phone Numbers**	**Address**
2	Wed, 12 Jan 2011	John K. Doe Jr.	(609)-993-3001	2196 184th Ave. NE, Redmond, 98052
3	Thu, 15 Sep 2011	Mr. Doe, John	609.993.3001 ext 2001	4297 148th Avenue NE, Bellevue, 98007
4	Mon, 17 Sep 2012	Jane A. Smith	+1-4250013981	2720 N Mesa St, El Paso, 79902, USA
5	2010-Nov-30 11:10:41	MS. Jane Smith	425 001 3981	3524 W Shore Rd APT 1002, Warwick
6	2011-Jan-11 02:27:21	Smith, Jane	tel: 4250013981	4740 N 132nd St Apt 417, Omaha, 68164
7	2011-Jan-12	Anthony R Von Fange II	650-384-9911	10508 Prairie Ln, Oklahoma City
8	2010-Dec-24	Mr. Peter Tyson	(405)123-3981	525 1st St, Marysville, WA 95901
9	9/22/2011	Dan E. Williams	1-650-1234183	211 W Ridge Dr, Waukon,52172
10	7/11/2012	James Davis Sr.	+1-425-736-9999	13120 Five Mile Rd, Brainerd
11	2/12/2012	Mr. James J. Davis	425.736.9999 x 9	602 Highland Ave, Shinnston, 26431
12	3/31/2013	Donald Edward Miller	(206) 309-8381	840 W Star St, Greenville, 27834
13	6/1/2009 12:01	Miller, Donald	206 309 8381	25571 Elba, Redford, 48239
14	2/26/2007 18:37	Rajesh Krishnan	206 456 8500 extension 1	539 Co Hwy 48, Sikeston, USA
15	1/4/2011 14:33	Daniel Chen	425 960 3566	1008 Whitlock Ave NW, Marietta, 30064

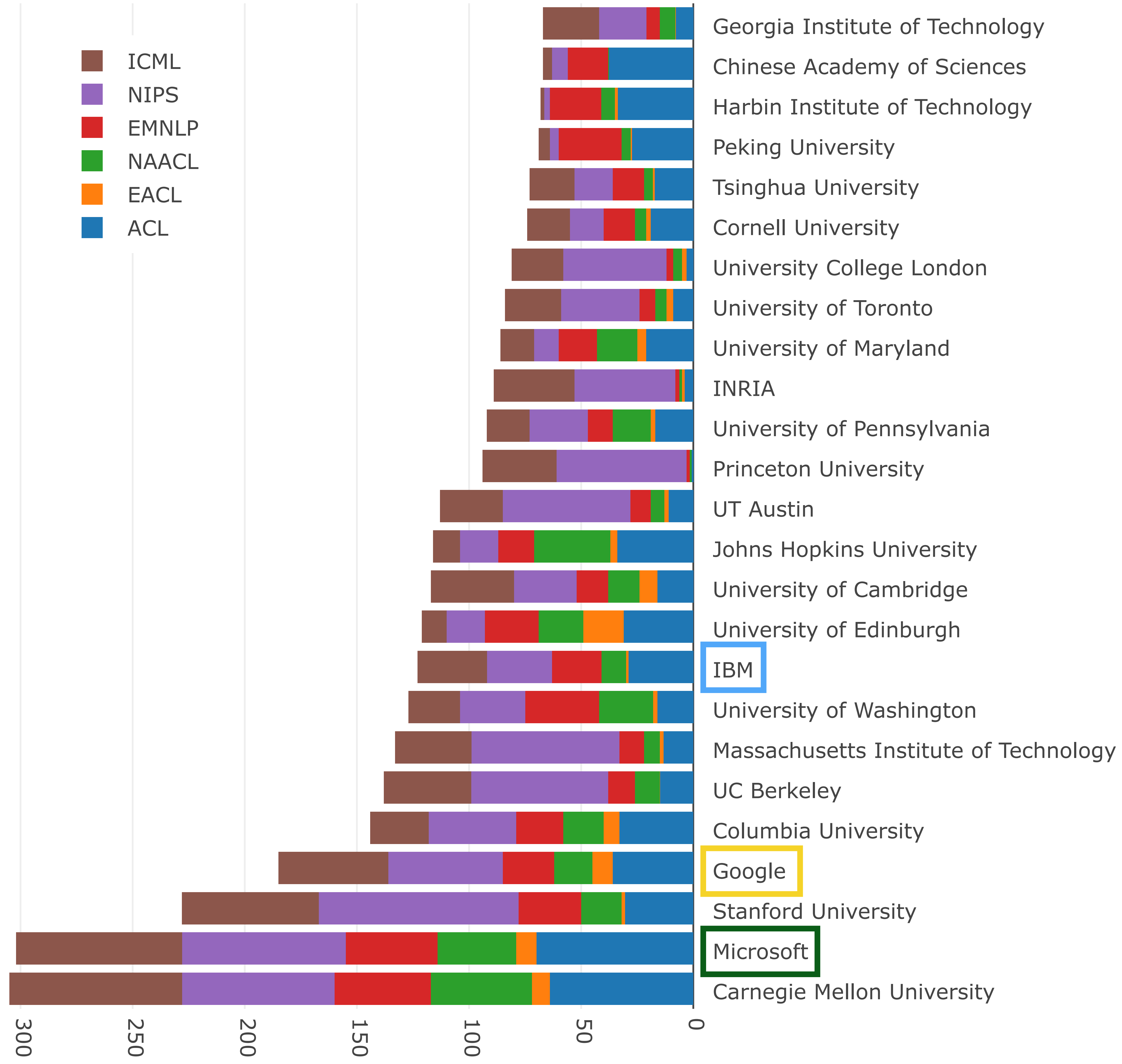
C	D
Transaction Date	**output**
Wed, 12 Jan 2011	2011-01-12-Wednesday
Thu, 15 Sep 2011	2011-09-15-Thursday
Mon, 17 Sep 2012	2012-09-17-Monday
2010-Nov-30 11:10:41	2010-11-30-Tuesday
2011-Jan-11 02:27:21	2011-01-11-Tuesday
2011-Jan-12	2011-01-12-Wednesday
2010-Dec-24	2010-12-24-Friday
9/22/2011	2011-09-22-Thursday
7/11/2012	2012-07-11-Wednesday
2/12/2012	2012-02-12-Sunday

C	D
Customer Name	**Output**
John K. **Doe** Jr.	Doe, John
Mr. **Doe, John**	Doe, John
Jane A. **Smith**	Smith, Jane
MS. **Jane Smith**	Smith, Jane
Smith, Jane	Smith, Jane
Dr **Anthony** R **Von Fange** III	Von Fange, Anthony
Peter Tyson	Tyson, Peter
Dan E. **Williams**	Williams, Dan
James Davis Sr.	Davis, James
James J. **Davis**	Davis, James
Mr. **Donald** Edward **Miller**	Miller, Donald

C	D
Address	**Output**
2196 184th Ave. NE Apt 417, **Redmond, 98052**	Redmond, WA, 98052
4297 148th Avenue NE L105, **Bellevue, WA 98007**	Bellevue, WA, 98007
2720 N Mesa St, **El Paso, 79902, USA**	El Paso, TX, 79902
3524 W Shore Rd APT 1002, **Warwick,02886**	Warwick, RI, 02886
4740 N 132nd St, **Omaha, 68164**	Omaha, NE, 68164
10508 Prairie Ln, **Oklahoma City**	Oklahoma City, OK, 73162
525 1st St, **Marysville, WA 95901**	Marysville, CA, 95901
211 W Ridge Dr, **Waukon,52172**	Waukon, IA, 52172
602 Highland Ave, **Shinnston, 26431**	Shinnston, WV, 26431
840 W Star St, **Greenville, 27834**	Greenville, NC, 27834

[Y. He et al., 2018]

Data Cleaning/Standardization (Aliases)




```
'google brain resident': 'google',
'google brain': 'google',
'google inc': 'google',
'google inc.':'google',
'google research nyc': 'google',
'google research': 'google',
'google, inc.': 'google’,
'deepmind @ google': 'deepmind',
'deepmind technologies': 'deepmind',
'google deepmind': 'deepmind’,
```

```
'ibm research - china':'ibm',
'ibm research':'ibm',
'ibm research, ny':'ibm',
'ibm research, usa':'ibm',
'ibm t. j. watson research center':'ibm',
'ibm t. j. watson research':'ibm',
'ibm t.j watson research center':'ibm',
'ibm t.j. watson research center':'ibm',
'ibm t.j.watson research center':'ibm',
'ibm thomas j. watson research center':'ibm',
'ibm tj watson research center':'ibm',
```

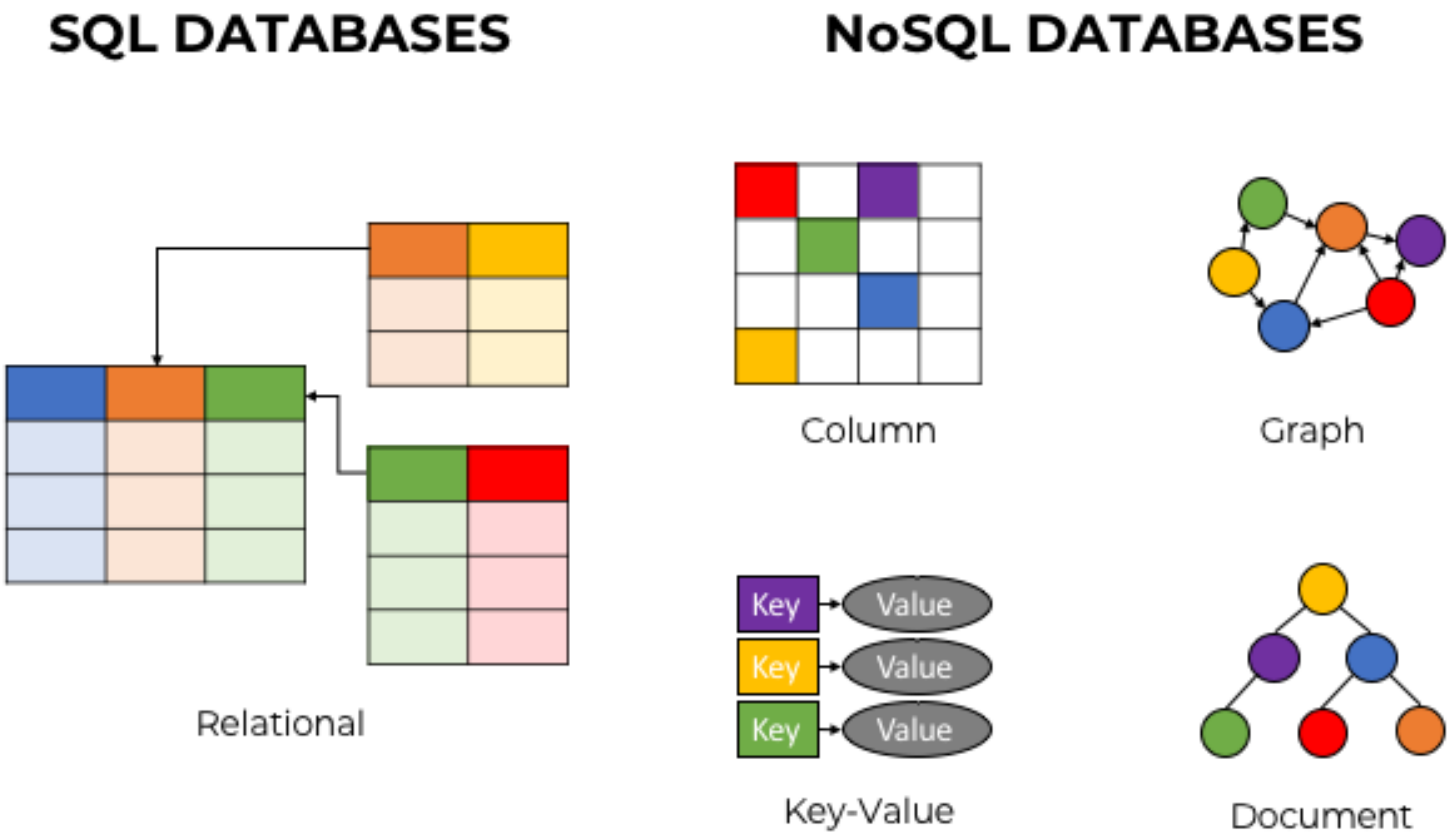
```
'microsoft research cambridge':'microsoft',
'microsoft research india':'microsoft',
'microsoft research maluuba':'microsoft',
'microsoft research new england':'microsoft',
'microsoft research':'microsoft',
'microsoft research, redmond, w':'microsoft',
'microsoft research, redmond, wa':'microsoft',
'miicrosoft research':'microsoft',
```

[NLP Publishing Stats, M. Rei & R. Allen]

Data Integration

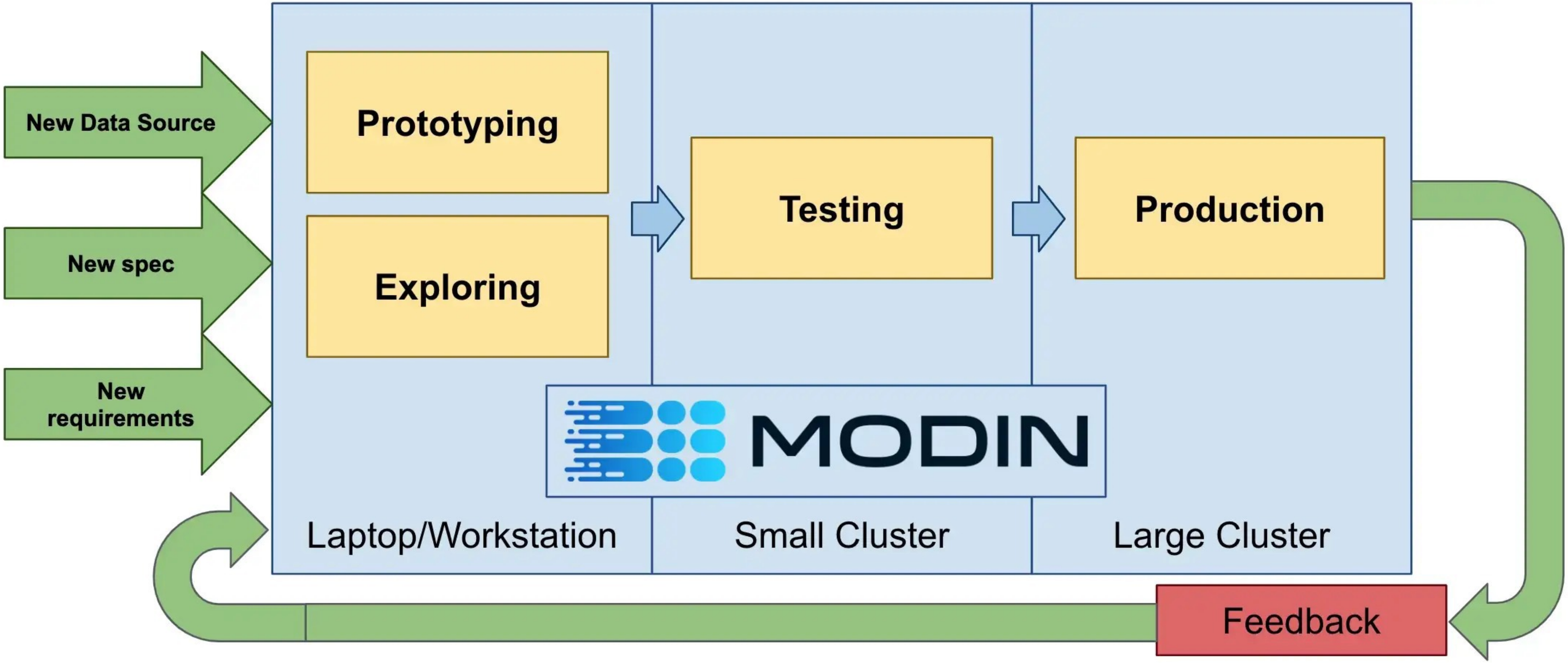
- [Google Thinks I'm Dead (I know otherwise.)] [R. Abrams, NYTimes, 2017]
- Not only Google, but also Alexa:
  - "Alexa replies that Rachel Abrams is a sprinter from the Northern Mariana Islands (which is true of someone else)."
  - "He asks if Rachel Abrams is deceased, and Alexa responds yes, citing information in the Knowledge Graph panel."

Data Storage



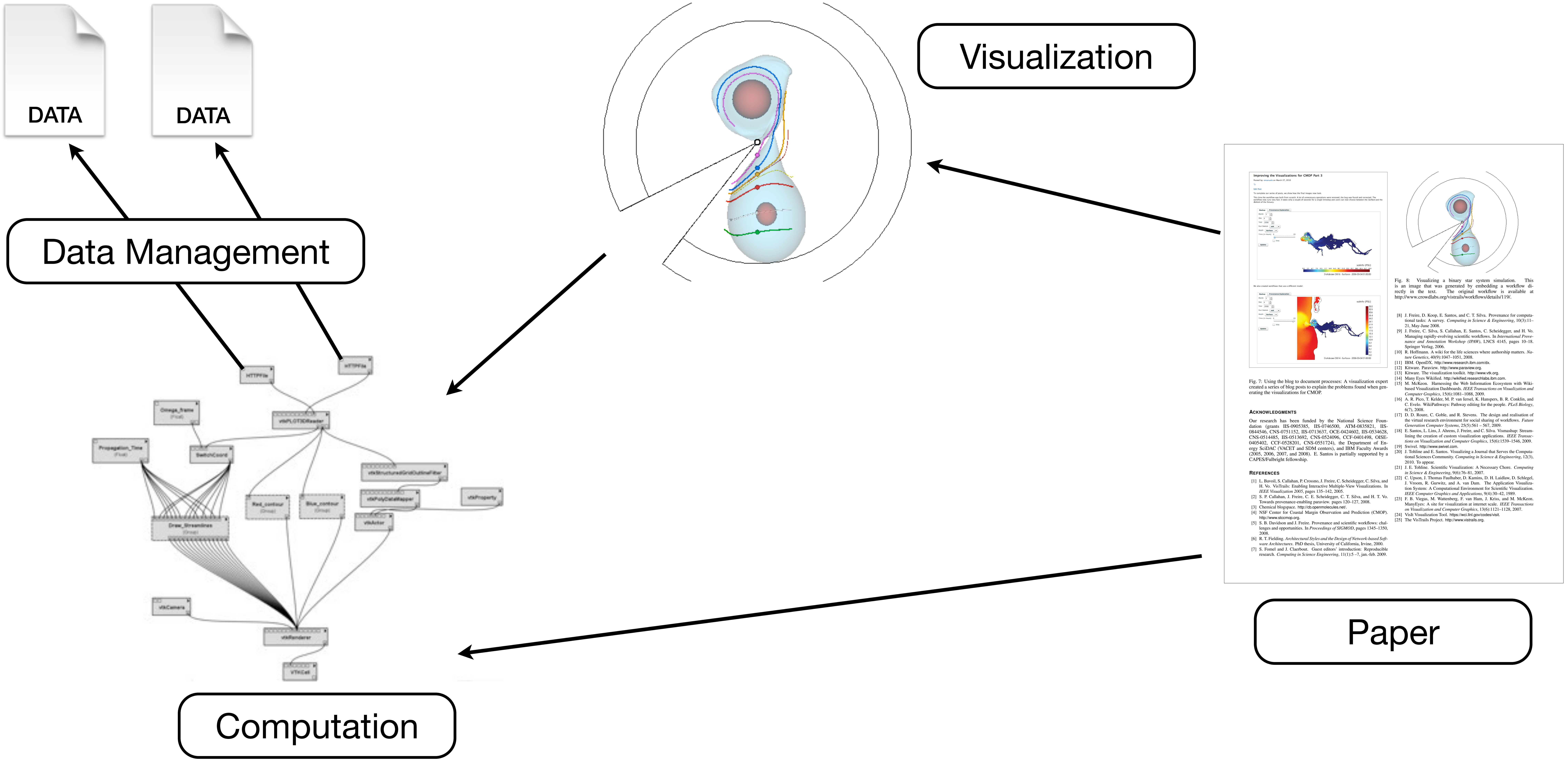
[V. Wilkinson]

Scaling Dataframes

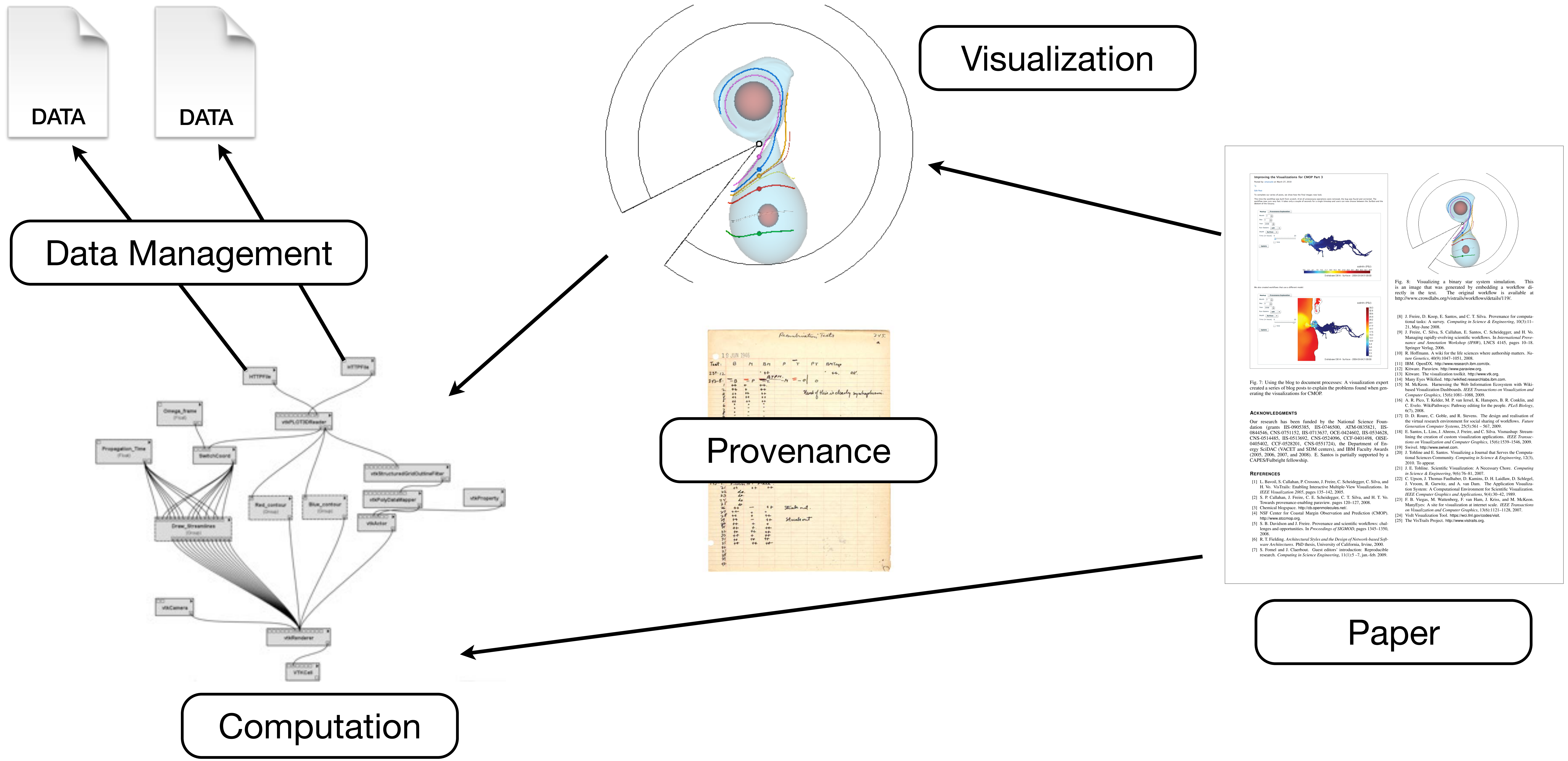


[D. Petersohn]

Provenance and Reproducibility



Provenance and Reproducibility

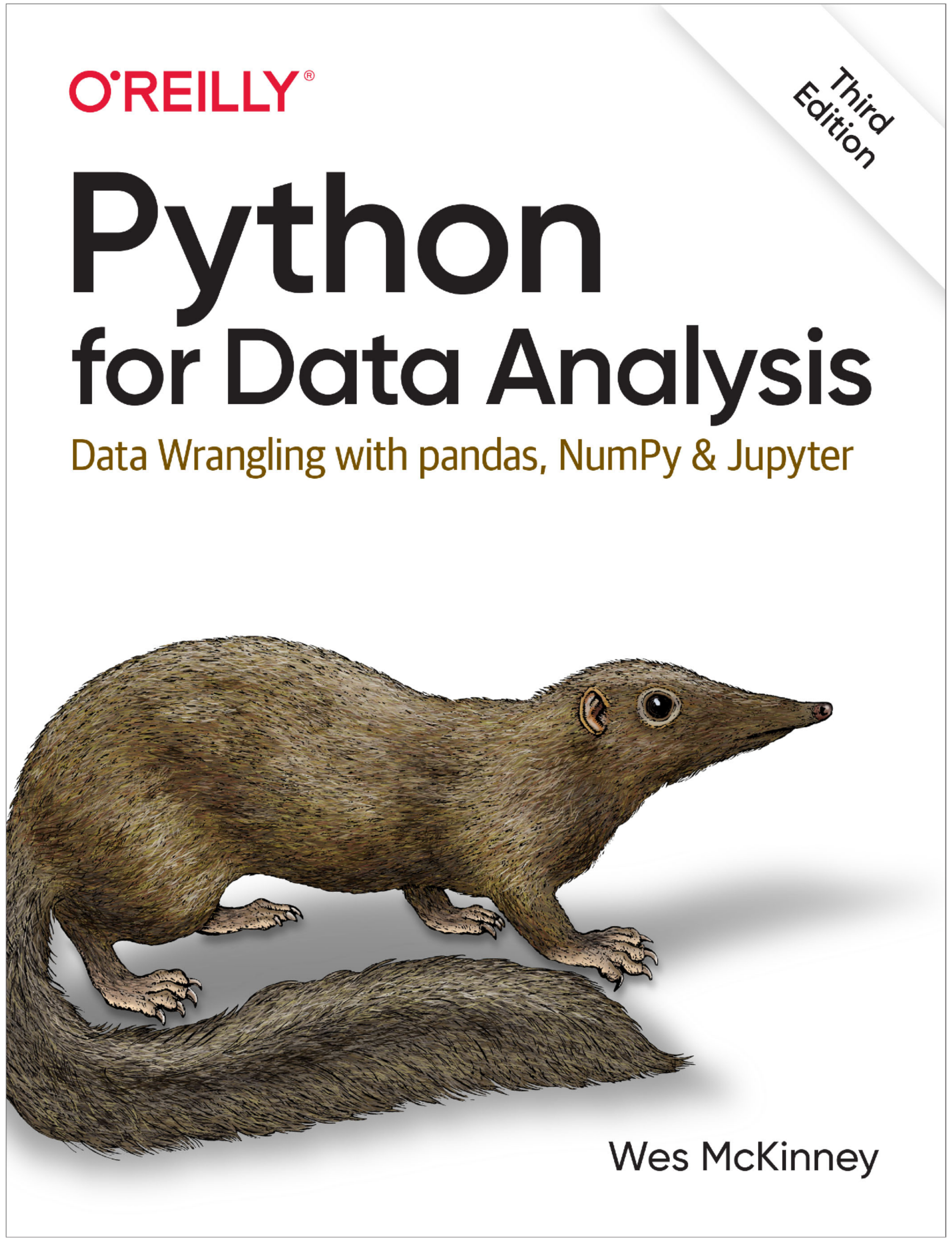


About this course

- Course web page is authoritative:
  - faculty.cs.niu.edu/~dakoop/cs640-2024sp/
  - Schedule, Readings, Assignments will be posted online
  - Check the web site before emailing me
- Course is meant to be more "cutting edge"
  - Still focus on building skills related to data management
  - Tune into current research and tools
- Requires student participation: readings and discussions

Course Material

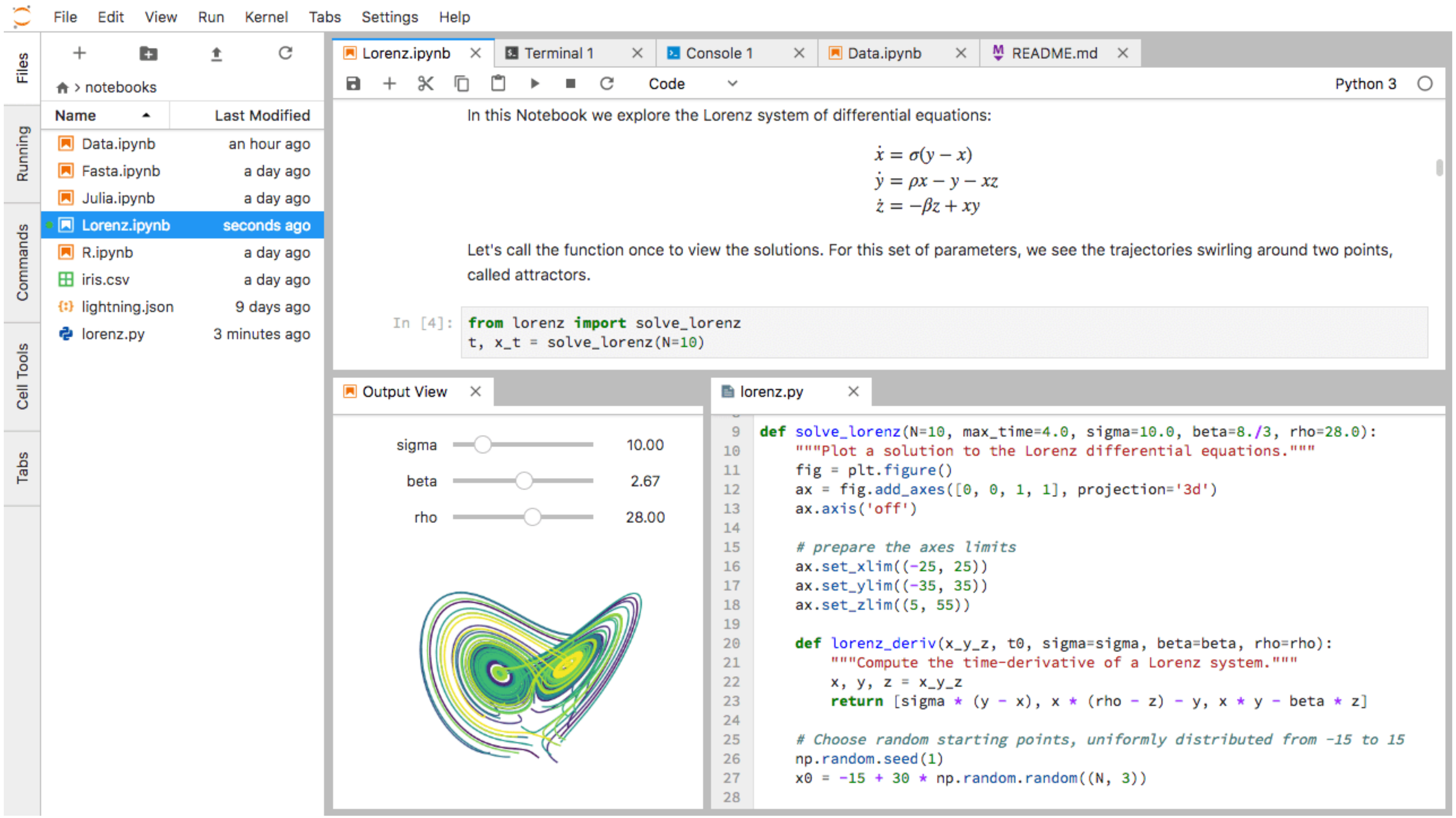
- Helpful Books:
  - Python for Data Analysis, W. McKinney
  - Effective Pandas, M. Harrison
  - Intro to Python, Deitel & Deitel
  - Python Data Science Handbook, J. VanderPlas
- Research papers
- Many websites



Syllabus Questions?

Class Roster Check

JupyterLab


File Edit View Run Kernel Tabs Settings Help
Files
Running
Commands
Cell Tools
Tabs
notebooks
Name
Last Modified
Data.ipynb an hour ago
Fasta.ipynb a day ago
Julia.ipynb a day ago
Lorenz.ipynb seconds ago
R.ipynb a day ago
iris.csv a day ago
lightning.json 9 days ago
lorenz.py 3 minutes ago
Lorenz.ipynb
Terminal 1
Console 1
Data.ipynb
README.md
Code
Python 3
In this Notebook we explore the Lorenz system of differential equations:
ẋ = σ(y − x)
ẏ = ρx − y − xz
ż = −βz + xy
Let's call the function once to view the solutions. For this set of parameters, we see the trajectories swirling around two points, called attractors.
In [4]:
from lorenz import solve_lorenz
t, x_t = solve_lorenz(N=10)
Output View
sigma 10.00
beta 2.67
rho 28.00
lorenz.py
def solve_lorenz(N=10, max_time=4.0, sigma=10.0, beta=8./3, rho=28.0):
"""Plot a solution to the Lorenz differential equations."""
fig = plt.figure()
ax = fig.add_axes([0, 0, 1, 1], projection='3d')
ax.axis('off')
# prepare the axes limits
ax.set_xlim((-25, 25))
ax.set_ylim((-35, 35))
ax.set_zlim((5, 55))
def lorenz_deriv(x_y_z, t0, sigma=sigma, beta=beta, rho=rho):
"""Compute the time-derivative of a Lorenz system."""
x, y, z = x_y_z
return [sigma * (y - x), x * (rho - z) - y, x * y - beta * z]
# Choose random starting points, uniformly distributed from -15 to 15
np.random.seed(1)
x0 = -15 + 30 * np.random.random((N, 3))

JupyterLab

- An interactive, configurable programming environment
- Supports many activities including notebooks
- Runs in your web browser
- Notebooks:
  - Originally designed for Python
  - Supports other languages, too
  - Displays results (even interactive maps) inline
  - You decide how to divide code into executable cells
  - Shift+Enter to execute a cell

Installing Python & JupyterLab

- www.anaconda.com/download/
- Anaconda has Jupyter Lab
- Use Python 3.12 version
- Anaconda Navigator
  - GUI application for managing Python environment
  - Can install packages
  - Can start JupyterLab
- Can also use the shell to do this:
  - `$ jupyter lab`
  - `$ conda install <pkg_name>`

JupyterLab Notebook Tips

- Starts with a directory view
- Create new notebooks using the Launcher (+ icon on the left)
  - New notebooks have the name "Untitled"
  - File → Rename Notebook… (or right-click) to change the name
- Save a notebook using the command under the File menu
- Shutting down the notebook requires quitting the kernel
  - Web browser is **interface** to display code and results
  - **Kernel** runs the code: may see messages in a console/terminal window
  - Closing the browser window does not stop Jupyter
  - Use File → Shut Down to shut down everything

JupyterLab Notebooks

- Open a notebook using the left panel like you would in a desktop view
- Past results are displayed—does not mean they are loaded in memory
- Use "Run All" or "Run All Above" to re-execute past work
  - If you shut down the kernel, all of the data and variables you defined need to be redefined (so you need to re-run all)
  - **Watch Out—Order Matters**: If you went back and re-executed cells in a different order than they are shown, doing "Run All" may not produce the same results!
- Edit mode (green) versus Command mode (blue == **Be Careful**)

JupyterLab Notebooks

- Can write code or plain text (can be styled Markdown)
  - Choose the type of cell using the dropdown menu
- Cells break up your code, but all data is **global**
  - Defining a variable `a` in one cell means it is available in **any** other cell
  - This includes cells **above** the cell `a` was defined in!
- Remember **Shift+Enter** to execute
- Enter just adds a new line
- Use `?<function_name>` for help
- Use Tab for **auto-complete** or suggestions
- Tab also indents, and Shift+Tab unindents

JupyterLab Outputs

- stdout: where print commands go
- stderr: where error messages go
- display: special output channel used to show rich outputs
- output: same as display but used to display the value of the last line of a cell

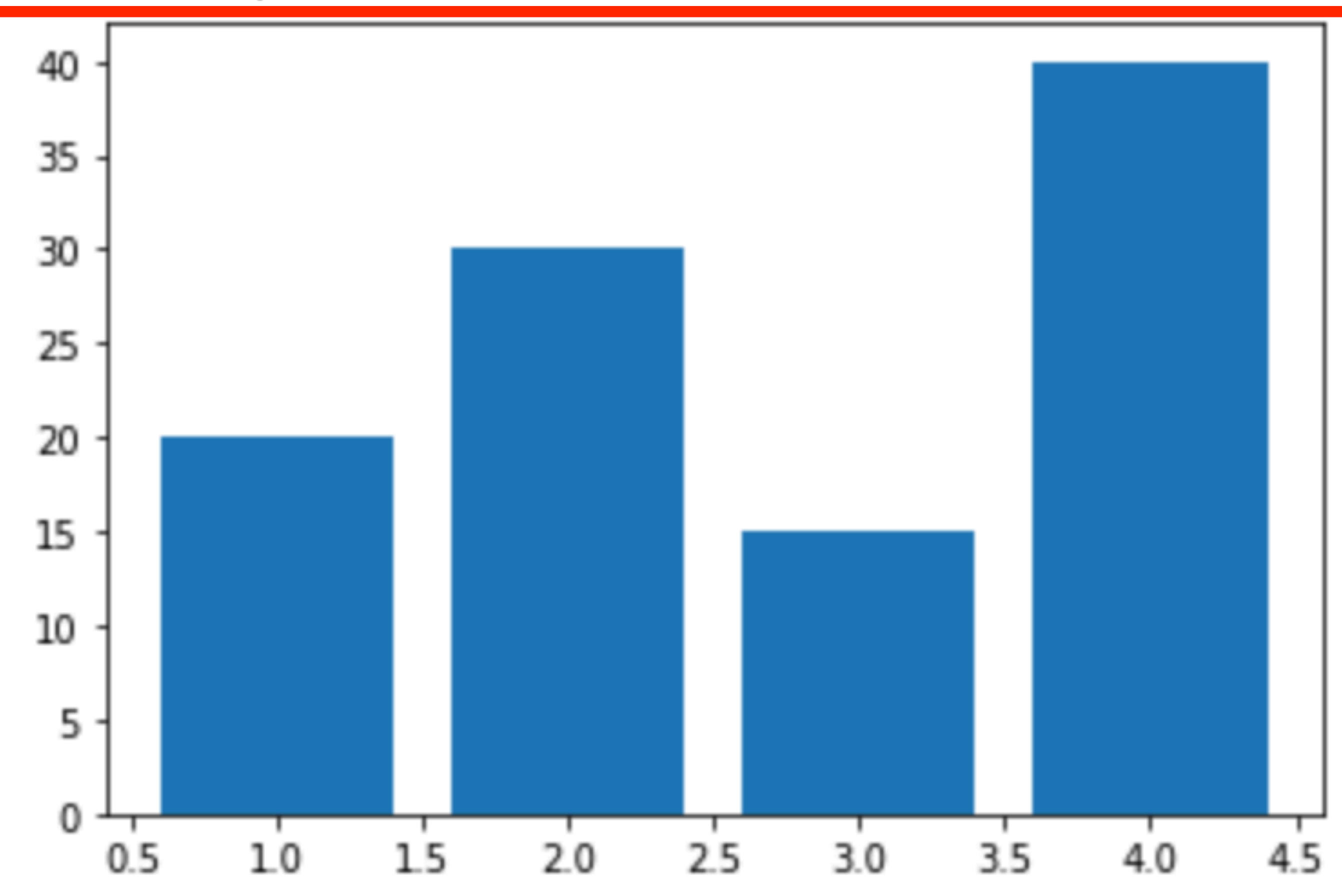
JupyterLab Output Types

[2]:
```
a = 12
for i in range(3):
    print("Some output")
plt.bar([1,2,3,4],[20,30,15,40])
plt.show()
a + 3
```

stdout

```
Some output
Some output
Some output
```

display



output

[2]: 15

[3]:
```
1 / 0
```

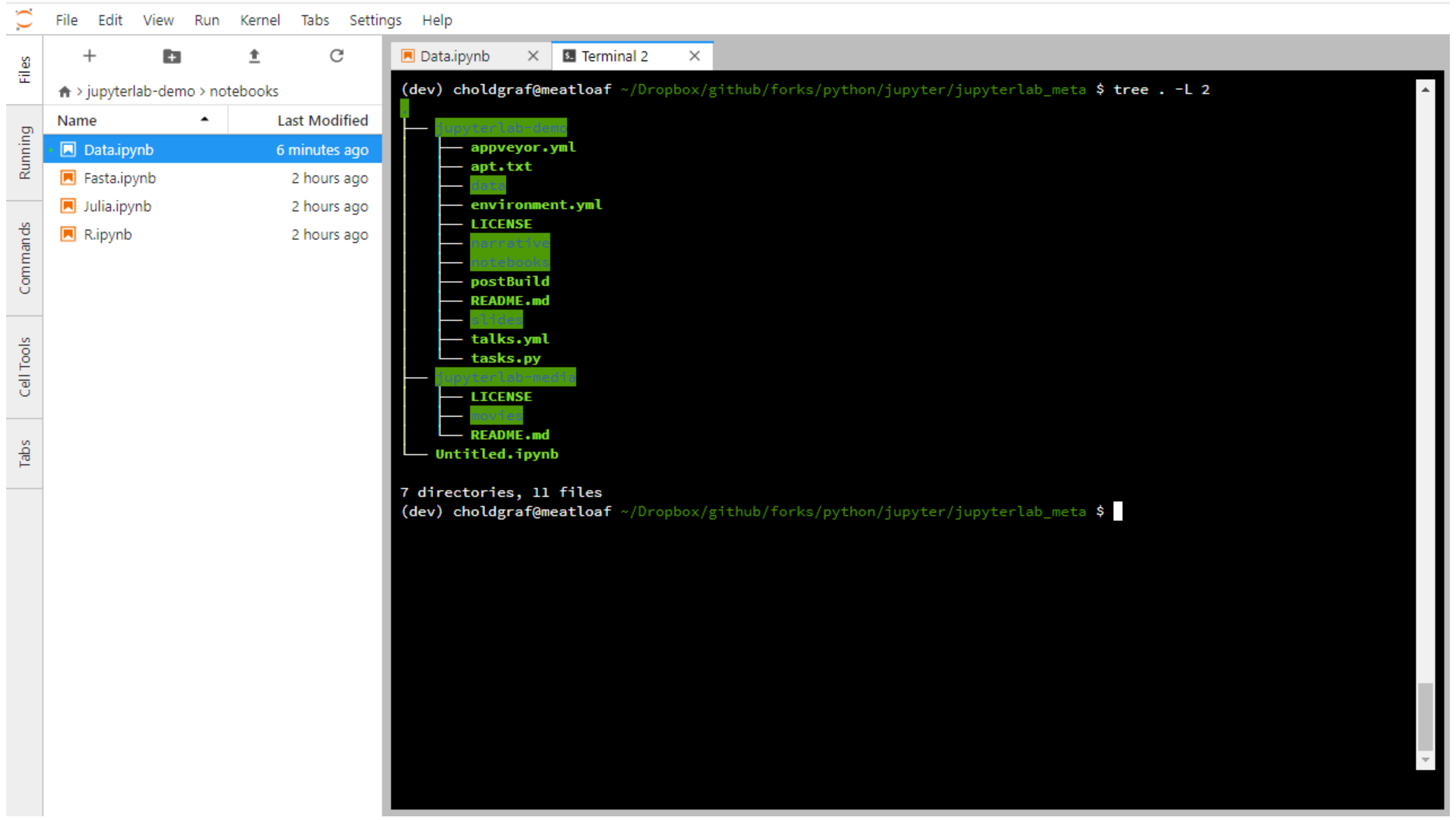
stderr

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-3-bc757c3fda29> in <module>
----> 1 1 / 0

ZeroDivisionError: division by zero
```

Other JupyterLab Features

- Terminal
  - Similar to what you see on turing/hopper but for your local machine
- File Viewers
  - CSV
  - Plugins available
- Console
  - Can be linked to notebooks



JupyterLab Documentation

- JupyterLab Tutorial Video
- JupyterLab Documentation

Python

- Started in December 1989 by Guido van Rossum
- "Python has surpassed Java as the top language used to introduce U.S. students to programming…" (ComputerWorld, 2014)
- Python and R are the two top languages for data science
- High-level, interpreted language
- Supports multiple paradigms (OOP, procedural, imperative)
- Help programmers write **readable** code, Use less code to do more
- Lots of libraries for python
  - Designed to be extensible
  - Easy to wrap code from other languages like C/C++
- Open-source with a large, passionate community

Learning Python Resources

- Python for Programmers
- https://wiki.python.org/moin/BeginnersGuide
- https://wiki.python.org/moin/IntroductoryBooks
- http://www.pythontutor.com
- https://www.python-course.eu
- https://software-carpentry.org/lessons/

Python Compared to C++ and Java

- Dynamic Typing
  - A variable does not have a fixed type
  - Example: `a = 1; a = "abc"`
- Indentation
  - Braces define blocks in Java, good style is to indent but not required
  - Indentation is **critical** in Python

```
z = 20
if x > 0:
    if y > 0:
        z = 100
else:
    z = 10
```

Notebook

Print function

- `print("Hello World")`
- Can also print variables:

```
name = "Jane"
print("Hello,", name)
```

Python Variables and Types

- No type declaration necessary
- Variables are names, not memory locations

```
a = 0
a = "abc"
a = 3.14159
```

- Don't worry about types, but think about types
- Strings are a type
- Integers are as big as you want them
- Floats can hold large numbers, too (double-precision)

Python Math and String "Math"

- Standard Operators: `+`, `-`, `*`, `/`, `%`
- Division "does what you want" (new in v3)
  - `5 / 2 = 2.5`
  - `5 // 2 = 2 # use // for integer division`
- Shortcuts: `+=, -=, *=`
- No `++, --`
- Exponentiation (Power): `**`
- Order of operations and parentheses: (`4 - 3 - 1` vs. `4 - (3 - 1)`)
- `"abc" + "def"`
- `"abc" * 3`

Python Strings

- Strings can be delimited by single or double quotes
  - `"abc"` and `'abc'` are exactly the same thing
  - Easier use of quotes in strings: `"Joe's"` or `'He said "Stop!"'`
- String concatenation: `"abc" + "def"`
- Repetition: `"abc" * 3`
- Special characters: `\n \t` like Java/C++

Python Strings

- Indexing:

```
a = "abcdef"
a[0]
```

- Slicing: `a[1:3]`
- Format:

```
name = "Jane"
print("Hello, {}".format(name))
```

  - or

```
print(f"Hello, {name}")
```

Loops

- ```
  while <condition>:
      <indented block>
  # end of while block (indentation done)
  ```
- Remember the colon!
- ```
  a = 5
  while a > 0:
      print(a)
      a -= 2
  ```
- `a > 0` is the condition
- Python has standard boolean operators (`<, >, <=, >=, ==, !=`)
  - What does a boolean operation return?
  - Linking boolean comparisons (`and, or`)

Conditionals

- `if, else`
  - Again, indentation is required
- `elif`
  - Shorthand for `else: if:`
- Same type of boolean expressions (`and or`)

break and continue

- `break` stops the execution of the loop
- `continue` skips the rest of the loop and goes to the next iteration

- ```
  a = 7
  while a > 0:
      a -= 2
      if a < 4:
          break
      print(a)
  ```

- ```
  a = 7
  while a > 0:
      a -= 2
      if a < 4 and a > 2:
          continue
      print(a)
  ```

True and False

- `True` and `False` (**captialized**) are defined values in Python
- `v == 0` will evaluate to either `True` or `False`

Why do we create and use functions?

Functions

- Calling functions is as expected:

```
mul(2,3) # computes 2*3 (mul from operator package)
```

  - Values passed to the function are parameters
  - May be variables!

```
a = 5
b = 7
mul(a,b)
```

- `print` is a function

```
print("This line doesn't end.", end=" ")
print("See it continues")
```

  - `end` is also a parameter, but this has a different syntax (keyword argument!)

Defining Functions

- `def` keyword
- Arguments have names but **no types**

```
def hello(name):
    print(f"Hello {name}")
```

- Can have defaults:

```
def hello(name="Jane Doe"):
    print(f"Hello {name}")
```

- With defaults, we can skip the parameter: `hello()` or `hello("John")`
- Also can pick and choose arguments:

```
def hello(name1="Joe", name2="Jane"):
    print(f"Hello {name1} and {name2}")
hello(name2="Mary")
```

Return statement

- Return statement gives back a value:

```
def mul(a,b):
    return a * b
```

- Variables changed in the function won't be updated:

```
def increment(a):
    a += 1
    return a
b = 12
c = increment(b)
print(b,c)
```

Python Containers

- Container: store more than one value
- Mutable versus immutable: Can we update the container?
  - Yes → mutable
  - No → immutable
  - Lists are mutable, tuples are immutable
- Lists and tuples may contain values of different types:
- List: `[1,"abc",12.34]`
- Tuple: `(1, "abc", 12.34)`
- You can also put functions in containers!
- `len` function: number of items: `len(l)`

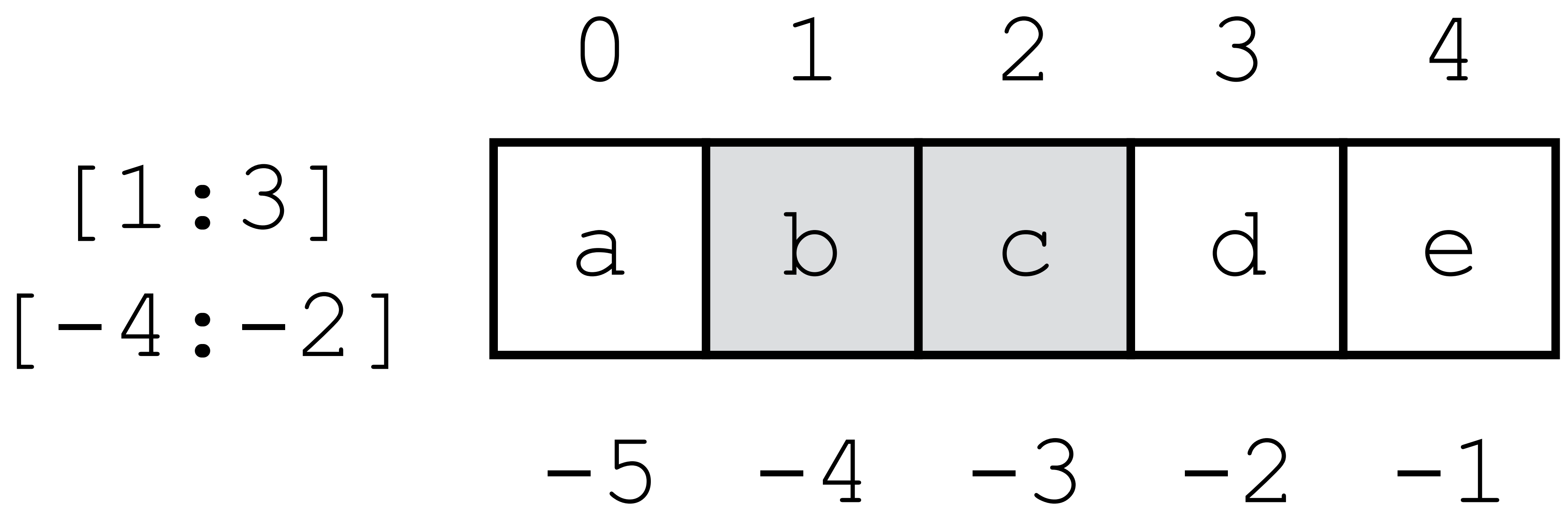
Indexing (Positive and Negative)

- Positive indices start at zero, negative at -1
- `my_str = "abcde"; my_str[1] # "b"`
- `my_list = [1,2,3,4,5]; my_list[-3] # 3`
- `my_tuple = (1,2,3,4,5); my_tuple[-5] # 1`

0	1	2	3	4
a	b	c	d	e
-5	-4	-3	-2	-1

Slicing

- Positive or negative indices can be used at any step
- `my_str = "abcde"; my_str[1:3] # ["b", c"]`
- `my_list = [1,2,3,4,5]; my_list[3:-1] # [4]`
- Implicit indices
  - `my_tuple = (1,2,3,4,5); my_tuple[-2:] # (4,5)`
  - `my_tuple[:3] # (1,2,3)`



Tuples

- `months = ('January','February','March','April', 'May','June','July','August','September','October', 'November','December')`
- Useful when you know you're not going to change the contents or add or delete values
- Can index and slice
- Also, can create new tuples from existing ones:
  - `t = (1,2,3)`
    `u = (4,5,6)`
  - `v = t + u # v` points to a **new** object
  - `t += u # t` is a **new** object

Modifying Lists

- Add to a list l:
  - `l.append(v)`: add one value (`v`) to the end of the list
  - `l.extend(vlist)`: add multiple values (`vlist`) to the end of `l`
  - `l.insert(i, v)`: add one value (`v`) at index `i`
- Remove from a list `l`:
  - `del l[i]`: deletes the value at index `i`
  - `l.pop(i)`: removes the value at index `i` (and returns it)
  - `l.remove(v)`: removes the **first** occurrence of value `v` (careful!)
- Changing an entry:
  - `l[i] = v`: changes the value at index `i` to `v` (Watch out for `IndexError`!)

Modifying a list

- `v = [1,2,3]`
  `w = [4,5,6]`
- `x = v + w # x` is a **new** list `[1,2,3,4,5,6]`
- `v.extend(w) # v` is mutated to `[1,2,3,4,5,6]`
- `v += w # v` is mutated to `[1,2,3,4,5,6]`
- `v.append(w) # v` is mutated to `[1,2,3,[4,5,6]]`
- `x = v + 4 #` **error**
- `v += 4 #` **error**
- `v += [4] # v` is mutated to `[1,2,3,4]`

in: Checking for a value

- The `in` operator:
  - `'a' in l`
  - `'a' not in l`
- Not very fast for lists

For loops

- Used much more frequently than while loops
- Is actually a "for-each" type of loop
- In Java, this is:
  - ```
    for (String item : someList) {
        System.out.println(item);
    }
    ```
- In Python, this is:
  - ```
    for item in someList:
        print(item)
    ```
- Grabs each element of `someList` in order and puts it into `item`
- Be careful modifying container in a for loop! (e.g. `someList.append(new_item)`)

What about counting?

- In C++:
- ```
  for(int i = 0; i < 100; i++) {
      cout << i << endl;
  }
  ```
- In Python:
- ```
  for i in range(0,100): # or range(100)
      print(i)
  ```
- `range(100)` vs. `list(range(100))`
- What about only even integers?

Dictionaries

- One of the most useful features of Python
- Also known as associative arrays
- Exist in other languages but a core feature in Python
- Associate a key with a value
- When I want to find a value, I give the dictionary a key, and it returns the value
- Example: InspectionID (key) → InspectionRecord (value)
- Keys must be immutable (technically, hashable):
  - Normal types like numbers, strings are fine
  - Tuples work, but lists do not (TypeError: unhashable type: 'list')
- There is only one value per key!

Dictionaries

- Defining a dictionary: curly braces
- `states = {'MA': 'Massachusetts, 'RI': 'Road Island', 'CT': 'Connecticut'}`
- Accessing a value: use brackets!
- `states['MA']` or `states.get('MA')`
- Adding a value:
- `states['NH'] = 'New Hampshire'`
- Checking for a key:
- `'ME' in states` → returns `True` or `False`
- Removing a value: `states.pop('CT') or del states['CT']`
- Changing a value: `states['RI'] = 'Rhode Island'`

Dictionaries

- Combine dictionaries: `d1.update(d2)`
  - `update` overwrites any key-value pairs in `d1` when the same key appears in `d2`
  - `d1 | d2`
- `len(d)` is the number of entries in `d`

Extracting Parts of a Dictionary

- `d.keys()`: the keys only
- `d.values()`: the values only
- `d.items()`: key-value pairs as a collection of tuples:
  `[(k1, v1), (k2, v2), …]`
- Unpacking a tuple or list
  - ```
    t = (1,2)
    a, b = t
    ```
- Iterating through a dictionary:

```
for (k,v) in d.items():
    if k % 2 == 0:
        print(v)
```

- Important: keys, values, and items are in added order!
```

Sets

- Just the keys from a dictionary
- Only one copy of each item
- Define like dictionaries without values
  - `s = {'a','b','c','e'}`
  - `'a' in s # True`
- Mutation
  - ```
    s.add('f')
    s.add('a') # only one copy
    s.remove('c')
    ```
- One gotcha:
  - `{}` is an empty **dictionary** not an empty set
    ```

Exercises

Exercise

- Given variables `x` and `y`, print the long division answer of `x` divided by `y` with the remainder.
- Examples:
  - `x = 11, y = 4` should print `"2R3"`
  - `x = 15, y = 2` should print `"7R1"`

Exercise

- Suppose I want to write Python code to print the numbers from 1 to 100. What errors do you see?

```
// print the numbers from 1 to 100
int counter = 1
while counter < 100 {
    print counter
    counter++
}
```