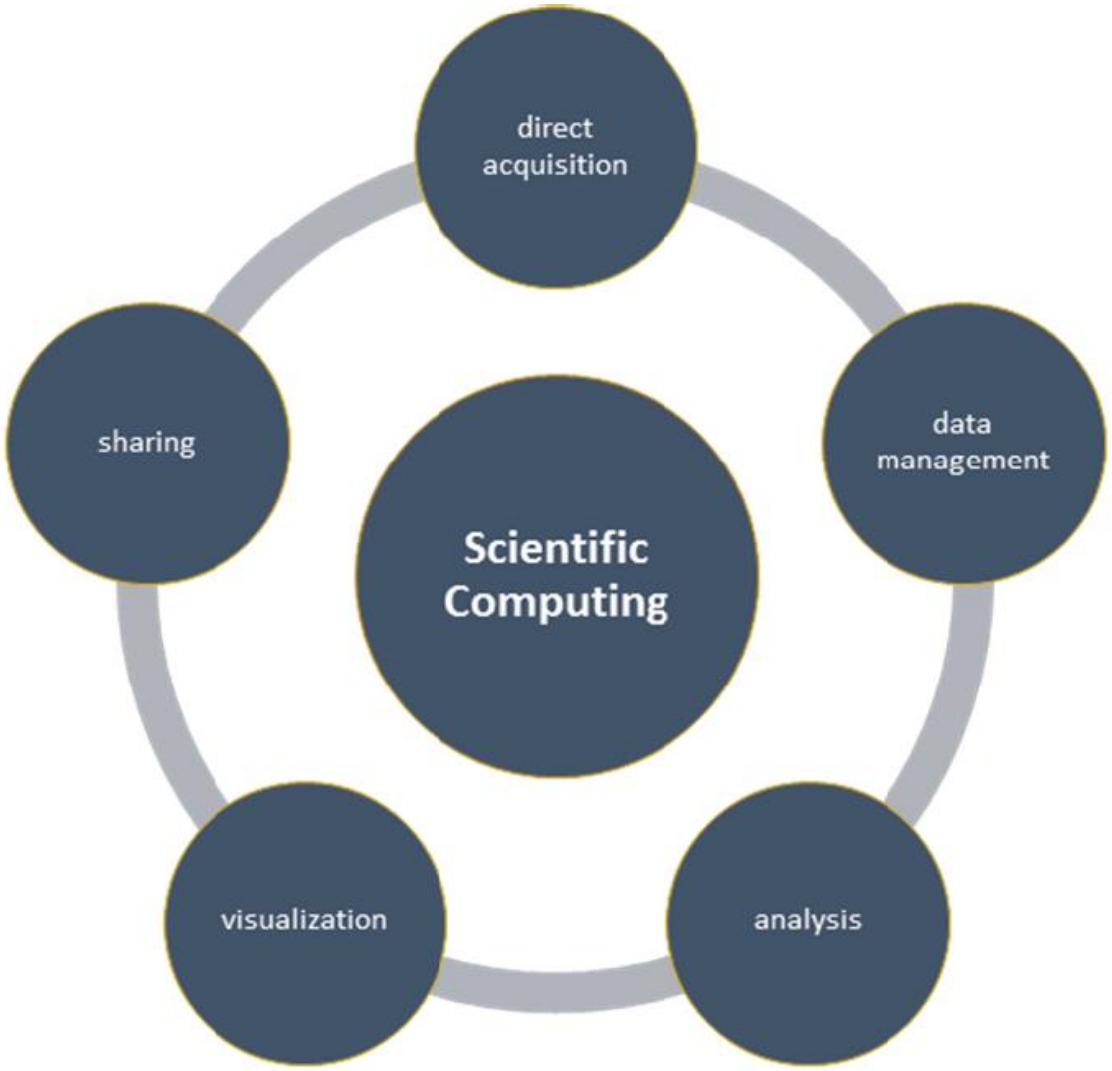
Scientific Computing, Data Science, Python & GIS

ENV 859 – Advanced GIS

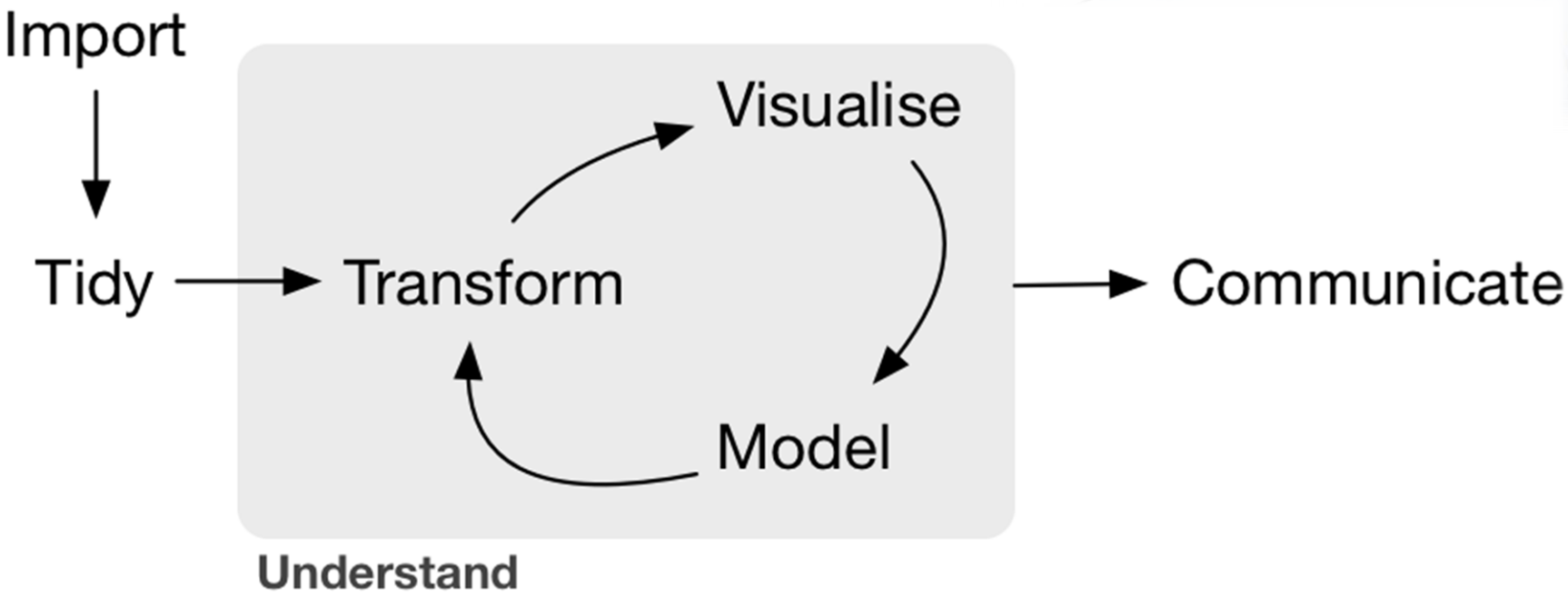
Fay 2017

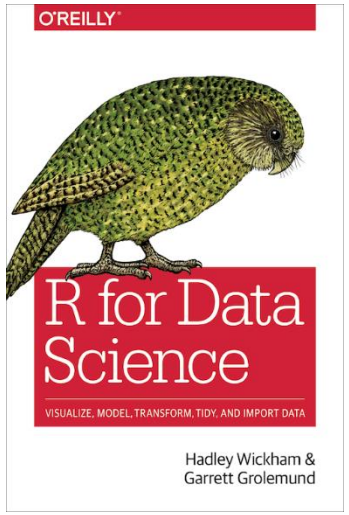
What is “Scientific Computing”



What is “Data Science?”

“Tidy Data”





[PDF] Tidy Data - Journal of Statistical Software
https://www.jstatsoft.org/article/view/v059i10/v59i10.pdf
by H Wickham - Cited by 171 - Related articles
Aug 20, 2014 - **Tidy Data**. **Hadley Wickham** ... The principles of **tidy data** are closely tied to those of relational databases and Codd's rela- ..... 20Traditions.**pdf** ...

Tidy Data Concept…

How data are stored in a table…

- **Variable**: *A measurement or and attribute*
  - Height, gender, weight, etc.
- **Value**: *The actual measurement/attribute:*
  - 25cm, female, 5kg
- **Observation**: *The set of measurements for an individual record:*
  - *Mouse 5* | 25 cm | female | 55g

	Height	Gender	Weight
Mouse 5	25 cm	Female	55 g
Mouse 10	21 cm	Male	35 g

Tidy Data Concept…

- Each **variable** forms a *column*;
- Each **observation** forms a *row*; and
- The collection of **observational units** forms a *table*.

	Height	Gender	Weight
Mouse 5	25 cm	Female	55 g
Mouse 10	21 cm	Male	35 g

Tidy Data Concept…

- Are these tables "tidy"?

	treatmenta	treatmentb
John Smith	—	2
Jane Doe	16	11
Mary Johnson	3	1

Table 1: Typical presentation dataset.

	John Smith	Jane Doe	Mary Johnson
treatmenta	—	16	3
treatmentb	2	11	1

Table 2: The same data as in Table 1 but structured differently.

1. `person`, with three possible values (John, Mary, and Jane).
2. `treatment`, with two possible values (a and b).
3. `result`, with five or six values depending on how you think of the missing value (-, 16, 3, 2, 11, 1).

http://vita.had.co.nz/papers/tidy-data.pdf

Tidy Data Concept…

- The “tidy” version of the data

name	trt	result
John Smith	a	—
Jane Doe	a	16
Mary Johnson	a	3
John Smith	b	2
Jane Doe	b	11
Mary Johnson	b	1

Table 3: The same data as in Table 1 but with variables in columns and observations in rows.

http://vita.had.co.nz/papers/tidy-data.pdf

Why tidy??

Facilitates:

- Manipulation of the data…
  - Query/subset
  - Computation of new values
  - Summarizing
  - Sorting
  - Joining
- Plotting…
  - “Grammar of graphics”
- Modeling…

	day	wolf	hare	fox
1	Monday	2	20	4
2	Tuesday	1	25	4
3	Wednesday	3	30	4

	day	species	count
1	Monday	wolf	2
2	Tuesday	wolf	1
3	Wednesday	wolf	3
4	Monday	hare	20
5	Tuesday	hare	25
6	Wednesday	hare	30
7	Monday	fox	4
8	Tuesday	fox	4
9	Wednesday	fox	4

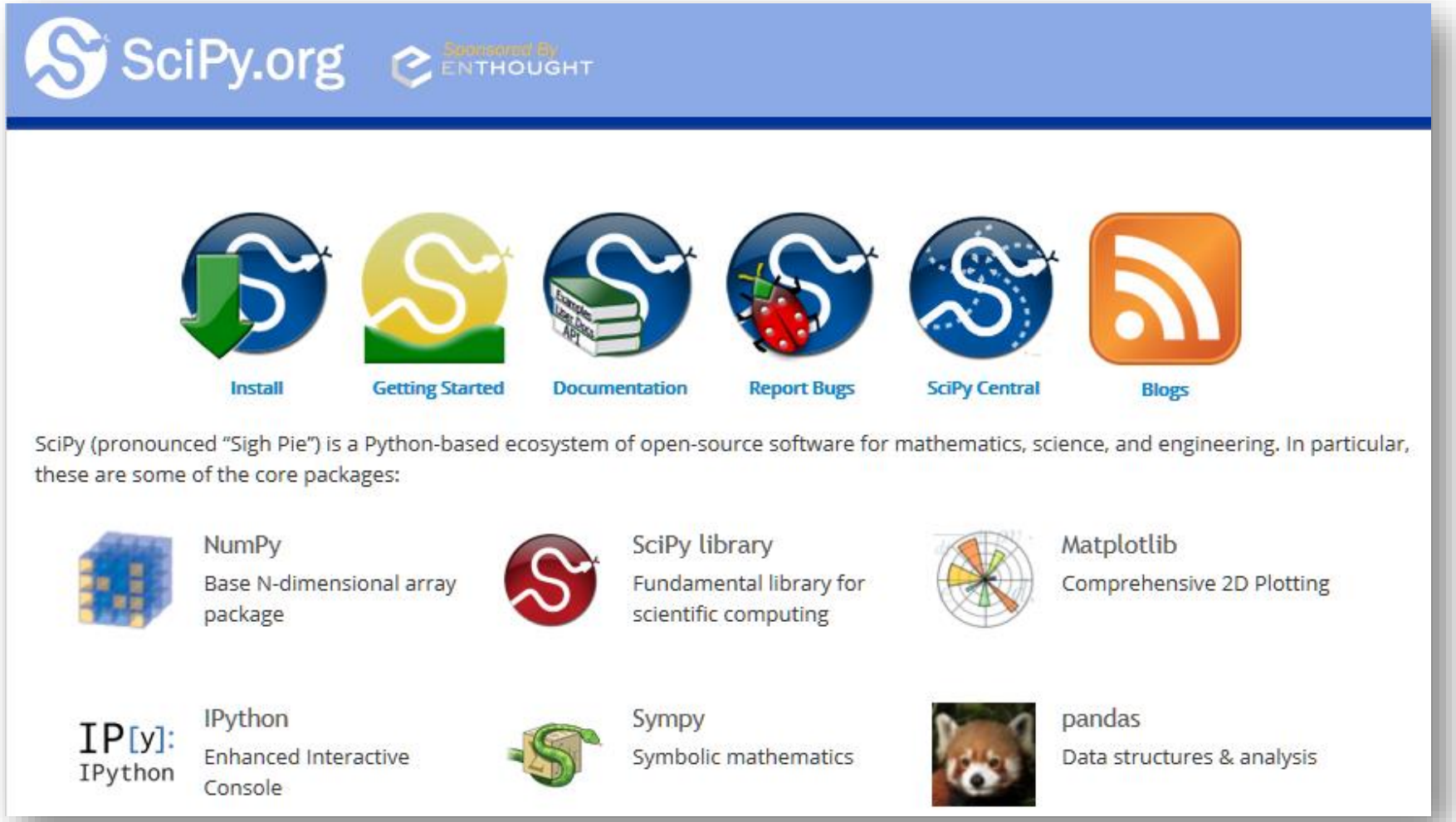
Data science – in R

- TidyVerse
  Set of R Tools for tidying data and working with tidy data
  - https://www.tidyverse.org/packages/

- Tools are designed to string – or “pipe” – commands together
  - Output of one tool becomes the input of another…

```
the_data <-
  read.csv('/path/to/data/file.csv') %>%
  subset(variable_a > x) %>%
  transform(variable_c = variable_a/variable_b) %>%
  head(100)
```

Data science – in Python



The SciPy 'stack'

Package	KLOC	Contributors	Stars
matplotlib	118	426	3359
Nose	7	79	912
NumPy	236	405	2683
Pandas	183	407	5834
SciPy	387	375	2150
SymPy	243	427	2672
Totals	1174	1784	

Unit testing

Algebraic computation

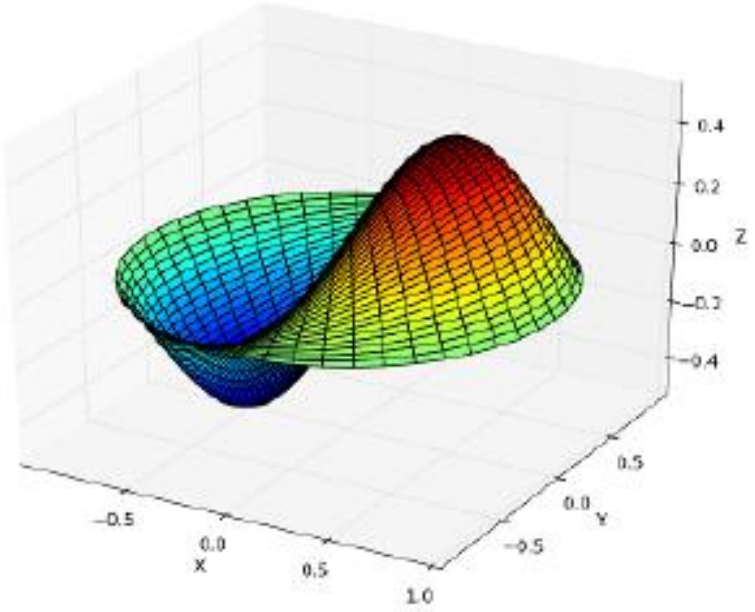
https://github.com/scw/scipy-devsummit-2016-talk/blob/master/slides/devsummit-2016-scipy-arcgis-presentation-full.pdf

KLOC = Thousands of lines of [actual] code
Stars = # of people following projects on GitHub

SciPy modules

- ***matplotlib*** – object oriented plotting

- ***SciPy*** – Computational methods for:
  - Integration (scipy.integrate)
  - Optimization (scipy.optimize)
  - Interpolation (scipy.interpolate)
  - Fourier Transforms (scipy.fftpack)
  - Signal Processing (scipy.signal)
  - Linear Algebra (scipy.linalg)
  - **Spatial (scipy.spatial)**
  - Statistics (scipy.stats)
  - **Multidimensional image processing (scipy.ndimage)**



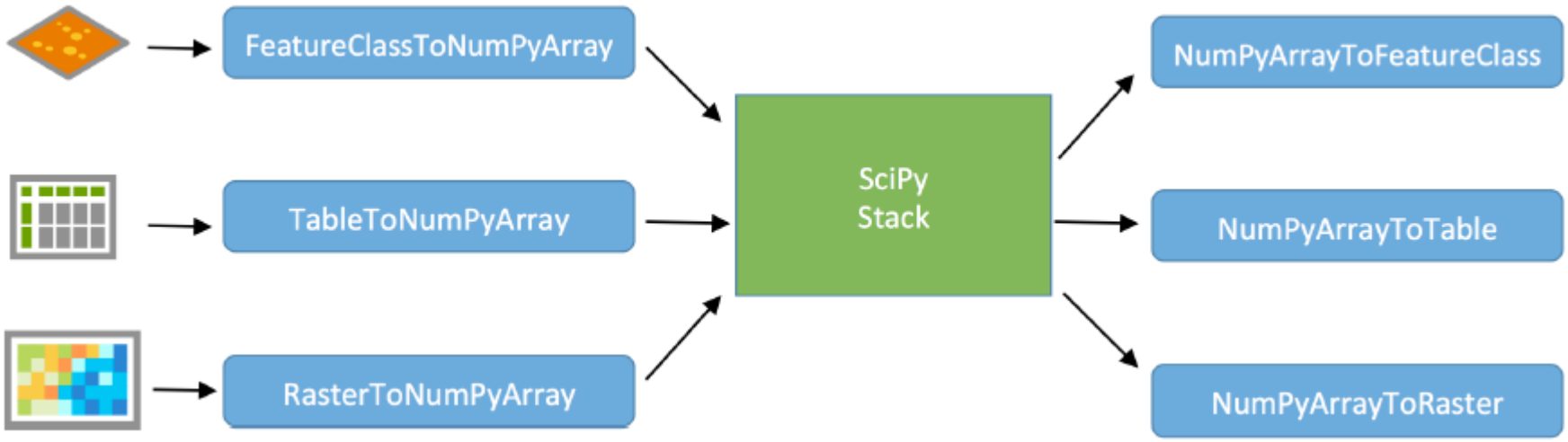
NumPy

- Provides an n-dimensional data structure: **Array**
  - Absence has been holding Python back as a rigorous scientific coding platform.
  - Allows for *array programming*

- *Why important??*
  - Easily extract specific data
  - Fast and efficient w/ large data sets

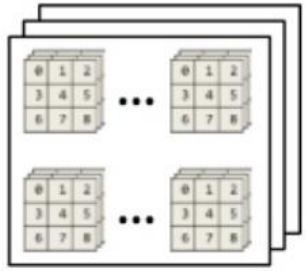
ArcGIS and NumPy

- NumPy ships with ArcGIS (since 9.x)
- Easy to switch between ArcGIS data types and NumPy arrays that work with SciPy Stack



http://desktop.arcgis.com/en/arcmap/latest/analyze/python/working-with-numpy-in-arcgis.htm

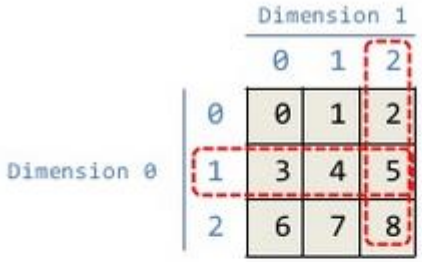
NumPy's *n-dimensional array*

Dimensions	Example	Terminology
1	0 1 2	Vector
2	0 1 2 3 4 5 6 7 8	Matrix
3	0 1 2 3 4 5 6 7 8	3D Array (3rd order Tensor)
N		ND Array

Elements within are all the same data type…

NumPy's n-dimensional array

- Allow quick access to: rows, columns, cells

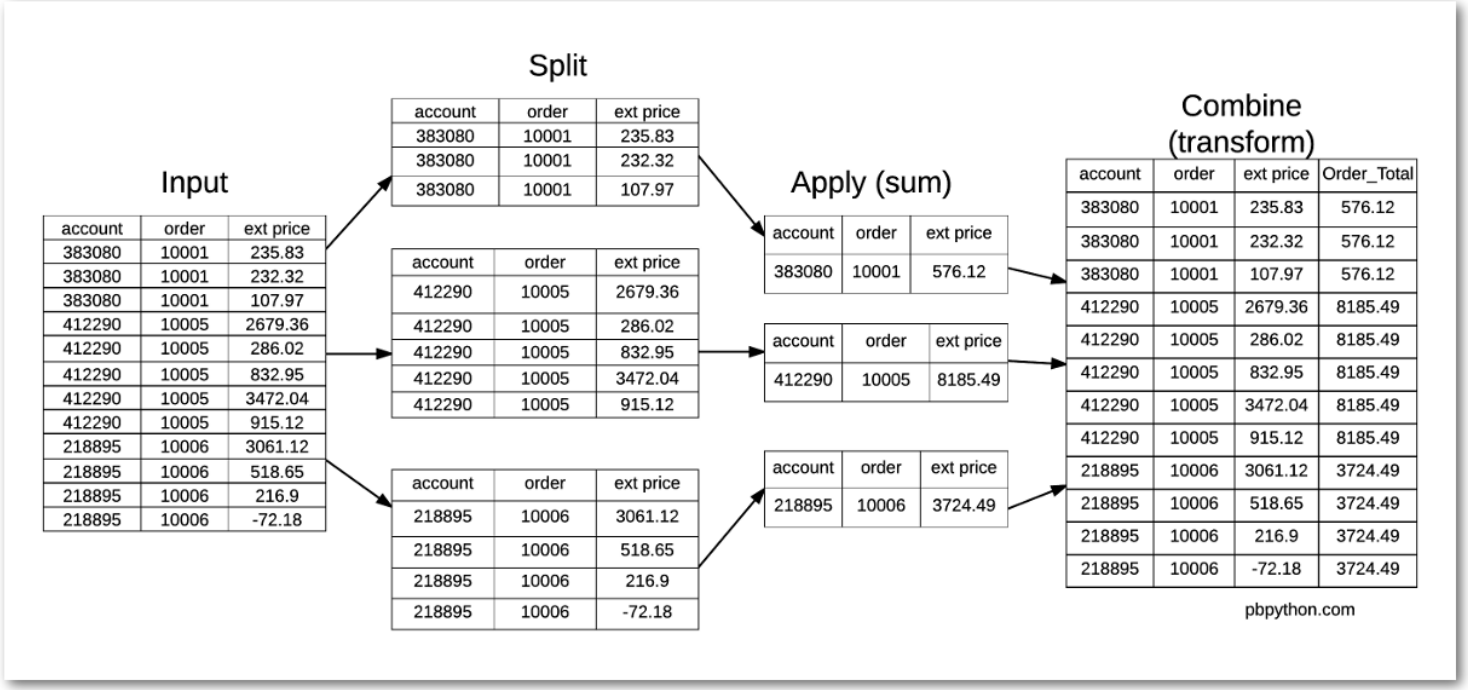


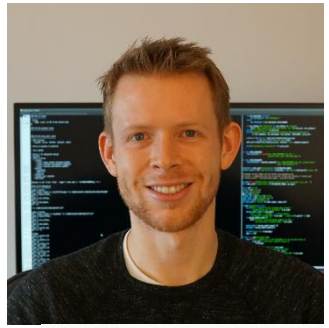

```
(def M [[0 1 2]
        [3 4 5]
        [6 7 8]])

(mget M 1 2)
=> 5
```

- Efficient computation (bulk operations)
- *Data driven* representation

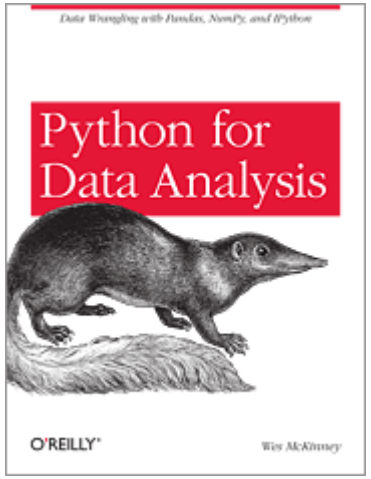
Pandas

- “Swiss-army knife” of data manipulation in Python
- Brings the “Data Frame” to Python
  - 2-dimensional (tabular) data structure (i.e. ‘tidy data’)





Wes McKinney



https://www.slideshare.net/wesm/a-look-at-pandas-design-and-development

Pandas

- "Swiss-army knife" of data manipulation in Python
- Brings the "Data Frame" to Python
  - 2-dimensional (tabular) data structure (i.e. 'tidy data')

  - Facilitates:
    - sorting/transforming/pivoting/melting of data
    - sub-setting/querying/selection of specific rows and/or columns
    - aggregation and summarizing of [selected] rows and columns
    - input & output; merging/appending/joining of multiple tables into one
    - plotting

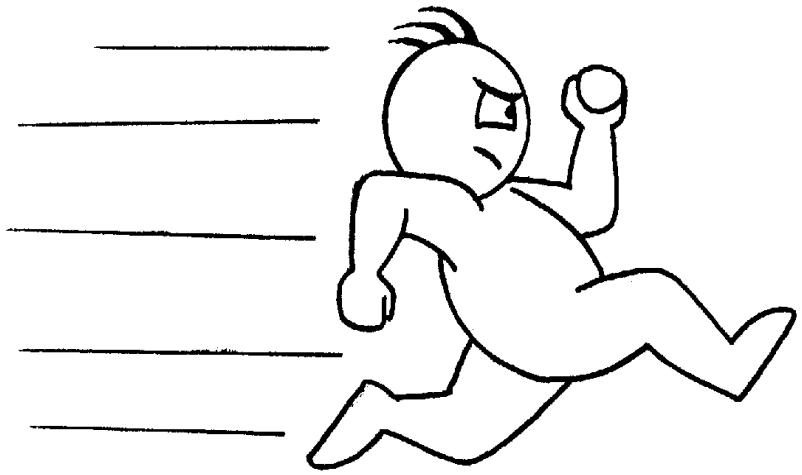
https://www.slideshare.net/wesm/a-look-at-pandas-design-and-development

Pandas' DataFrame

- *Similar* to attribute tables in ArcGIS
  - Multiple data types (but same in each column)
  - All columns contain equal # of rows
  - Indexed: Rows are like Python dictionaries
- Allows for easy selection of: rows, columns, values
  - Slicing and query
- Can be sorted, subset, transposed, re-shaped easily
- Can be merged and joined with other data frames

Pandas' DataFrame

- Filter rows meeting a criteria
- Select specific columns
- Sort rows on values in one/many columns
- Merge/append/join other arrays or frames
- Group and summarize values
- Reshape tables
- Time series
- Plotting



Diving In

NumPy

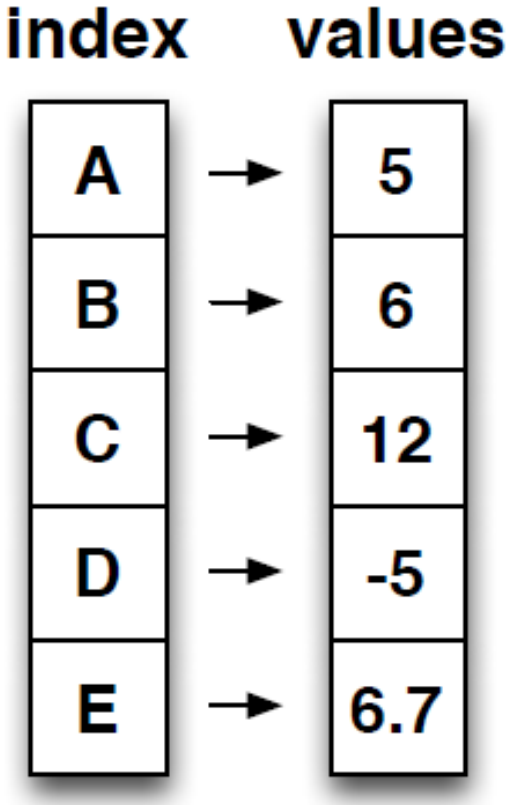
- Intro to NumPy – Why NumPy's array is useful
- Using NumPy with feature classes
- Using NumPy with Raster datasets

Pandas

- SQL vs Pandas
- Sara-the-Turtle *redux*: How indices work
- Getting to know Pandas
- *more research examples…*

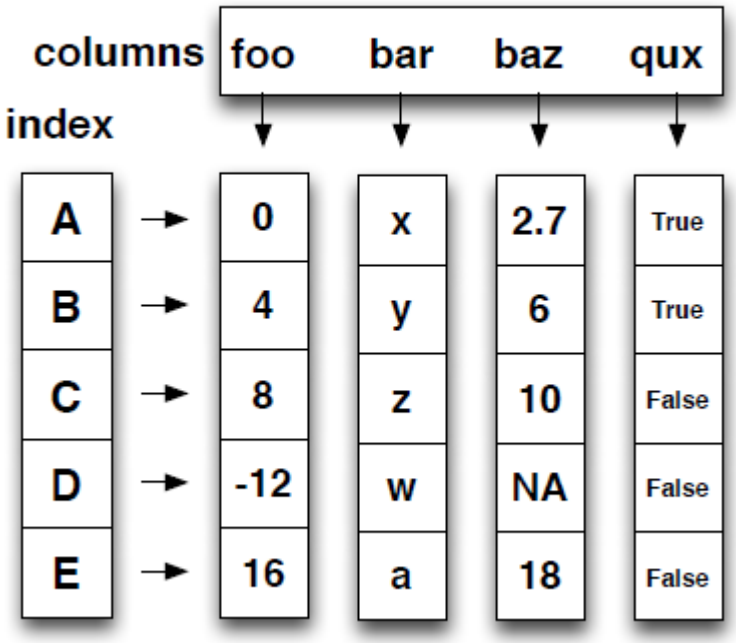
[DEMO]

Recap: Pandas' *Series* object



- 1-dimensional data collection
- Data can be of any type,
  but all members are of that type
- Indexed values
  - Need not be sequential numbers!
  - Can be anything?
  - Duplicates possible
    (but reduces functionality)

Recap: Pandas' DataFrame object



- Each column is a *series*
  - *A column can be any data type, but contents must all be of the same data type*
- Rows and columns have *implicit* & *explicit* indices
  - Can reference values by row & column number…
  - Or by row index and column name…
- The size is mutable: can append rows, columns.
- Can join to other tables

More info on NumPy & Pandas

- https://jakevdp.github.io/PythonDataScienceHandbook/index.html
- http://wesmckinney.com/
- http://www.datacarpentry.org/

- Cheat Sheets…
- https://www.datacamp.com/community/blog/python-pandas-cheat-sheet

Data Wrangling with pandas Cheat Sheet

http://pandas.pydata.org

Tidy Data – A foundation for wrangling in pandas

In a tidy data set:

Each variable is saved in its own column

Each observation is saved in its own row

Tidy data complements pandas's vectorized operations. pandas will automatically preserve observations as you manipulate variables. No other format works as intuitively with pandas.

M * A

Syntax – Creating DataFrames

	a	b	c
1	4	7	10
2	5	8	11
3	6	9	12

```
df = pd.DataFrame(
        {"a" : [4 ,5, 6],
         "b" : [7, 8, 9],
         "c" : [10, 11, 12]},
       index = [1, 2, 3])
```

Specify values for each column.

```
df = pd.DataFrame(
     [[4, 7, 10],
      [5, 8, 11],
      [6, 9, 12]],
     index=[1, 2, 3],
     columns=['a', 'b', 'c'])
```

Specify values for each row.

		a	b	c
n	v			
d	1	4	7	10
	2	5	8	11
e	2	6	9	12

```
df = pd.DataFrame(
        {"a" : [4 ,5, 6],
         "b" : [7, 8, 9],
         "c" : [10, 11, 12]},
index = pd.MultiIndex.from_tuples(
        [('d',1),('d',2),('e',2)],
          names=['n','v']))
```

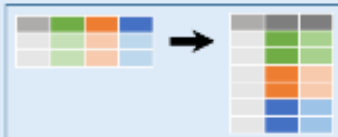
Create DataFrame with a MultiIndex

Method Chaining

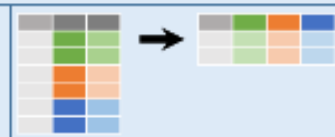
Most pandas methods return a DataFrame so that another pandas method can be applied to the result. This improves readability of code.

```
df = (pd.melt(df)
       .rename(columns={
               'variable' : 'var',
               'value' : 'val'})
       .query('val >= 200')
     )
```

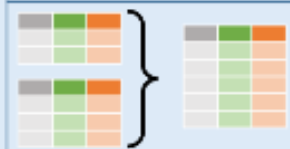
Reshaping Data – Change the layout of a data set



`pd.melt(df)`
Gather columns into rows.



`df.pivot(columns='var', values='val')`
Spread rows into columns.



`pd.concat([df1,df2])`
Append rows of DataFrames

`pd.concat([df1,df2], axis=1)`
Append columns of DataFrames

`df.sort_values('mpg')`
Order rows by values of a column (low to high).

`df.sort_values('mpg',ascending=False)`
Order rows by values of a column (high to low).

`df.rename(columns = {'y':'year'})`
Rename the columns of a DataFrame

`df.sort_index()`
Sort the index of a DataFrame

`df.reset_index()`
Reset index of DataFrame to row numbers, moving index to columns.

`df.drop(['Length','Height'], axis=1)`
Drop columns from DataFrame

Subset Observations (Rows)

`df[df.Length > 7]`
Extract rows that meet logical criteria.

`df.drop_duplicates()`
Remove duplicate rows (only considers columns).

`df.head(n)`
Select first n rows.

`df.tail(n)`
Select last n rows.

`df.sample(frac=0.5)`
Randomly select fraction of rows.

`df.sample(n=10)`
Randomly select n rows.

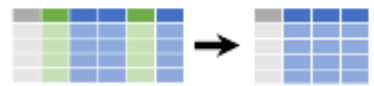
`df.iloc[10:20]`
Select rows by position.

`df.nlargest(n, 'value')`
Select and order top n entries.

`df.nsmallest(n, 'value')`
Select and order bottom n entries.

Logic in Python (and pandas)			
<	Less than	!=	Not equal to
>	Greater than	df.column.isin(values)	Group membership
==	Equals	pd.isnull(obj)	Is NaN
<=	Less than or equals	pd.notnull(obj)	Is not NaN
>=	Greater than or equals	&,|,~,^,df.any(),df.all()	Logical and, or, not, xor, any, all

Subset Variables (Columns)



`df[['width','length','species']]`
Select multiple columns with specific names.

`df['width']` or `df.width`
Select single column with specific name.

`df.filter(regex='regex')`
Select columns whose name matches regular expression regex.

regex (Regular Expressions) Examples	
'\.'	Matches strings containing a period '.'
'Length$'	Matches strings ending with word 'Length'
'^Sepal'	Matches strings beginning with the word 'Sepal'
'^x[1-5]$'	Matches strings beginning with 'x' and ending with 1,2,3,4,5
'^(?!Species$).*'	Matches strings except the string 'Species'

`df.loc[:,'x2':'x4']`
Select all columns between x2 and x4 (inclusive).

`df.iloc[:,[1,2,5]]`
Select columns in positions 1, 2 and 5 (first column is 0).

`df.loc[df['a'] > 10, ['a','c']]`
Select rows meeting logical condition, and only the specific columns .

https://github.com/pandas-dev/pandas/blob/master/doc/cheatsheet/Pandas_Cheat_Sheet.pdf

Python For Data Science *Cheat Sheet*

Pandas Basics

Learn Python for Data Science Interactively at www.DataCamp.com

Pandas

The **Pandas** library is built on NumPy and provides easy-to-use **data structures** and **data analysis** tools for the Python programming language.

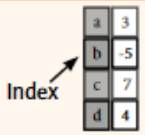
Use the following import convention:

```
>>> import pandas as pd
```

Pandas Data Structures

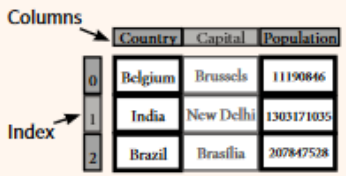
Series

A **one-dimensional** labeled array capable of holding any data type




```
>>> s = pd.Series([3, -5, 7, 4], index=['a', 'b', 'c', 'd'])
```

DataFrame



A **two-dimensional** labeled data structure with columns of potentially different types

```
>>> data = {'Country': ['Belgium', 'India', 'Brazil'],
            'Capital': ['Brussels', 'New Delhi', 'Brasília'],
            'Population': [11190846, 1303171035, 207847528]}

>>> df = pd.DataFrame(data,
                      columns=['Country', 'Capital', 'Population'])
```

Asking For Help

```
>>> help(pd.Series.loc)
```

Selection

Also see NumPy Arrays

Getting

`>>> s['b']` `-5`	Get one element

`>>> df[1:]`
`  Country  Capital  Population`
`1  India  New Delhi  1303171035`
`2  Brazil  Brasília  207847528`
	Get subset of a DataFrame

Selecting, Boolean Indexing & Setting

By Position	

`>>> df.iloc([0],[0])`
`'Belgium'`
`>>> df.iat([0],[0])`
`'Belgium'`
	Select single value by row & column

By Label	

`>>> df.loc([0], ['Country'])`
`'Belgium'`
`>>> df.at([0], ['Country'])`
`'Belgium'`
	Select single value by row & column labels

By Label/Position	

`>>> df.ix[2]`
`Country  Brazil`
`Capital  Brasília`
`Population  207847528`
	Select single row of subset of rows

`>>> df.ix[:,'Capital']`
`0  Brussels`
`1  New Delhi`
`2  Brasília`
	Select a single column of subset of columns

`>>> df.ix[1,'Capital']` `'New Delhi'`	Select rows and columns
Boolean Indexing	
`>>> s[~(s > 1)]`	Series s where value is not >1
`>>> s[(s < -1) | (s > 2)]`	s where value is <-1 or >2
`>>> df[df['Population']>1200000000]`	Use filter to adjust DataFrame
Setting	
`>>> s['a'] = 6`	Set index a of Series s to 6

Dropping

`>>> s.drop(['a', 'c'])`	Drop values from rows (axis=0)
`>>> df.drop('Country', axis=1)`	Drop values from columns(axis=1)

Sort & Rank

`>>> df.sort_index()`	Sort by labels along an axis
`>>> df.sort_values(by='Country')`	Sort by the values along an axis
`>>> df.rank()`	Assign ranks to entries

Retrieving Series/DataFrame Information

Basic Information

`>>> df.shape`	(rows,columns)
`>>> df.index`	Describe index
`>>> df.columns`	Describe DataFrame columns
`>>> df.info()`	Info on DataFrame
`>>> df.count()`	Number of non-NA values

Summary

`>>> df.sum()`	Sum of values
`>>> df.cumsum()`	Cummulative sum of values
`>>> df.min()/df.max()`	Minimum/maximum values
`>>> df.idxmin()/df.idxmax()`	Minimum/Maximum index value
`>>> df.describe()`	Summary statistics
`>>> df.mean()`	Mean of values
`>>> df.median()`	Median of values

Applying Functions

`>>> f = lambda x: x*2`	
`>>> df.apply(f)`	Apply function
`>>> df.applymap(f)`	Apply function element-wise

Data Alignment

Internal Data Alignment

NA values are introduced in the indices that don't overlap:

```
>>> s3 = pd.Series([7, -2, 3], index=['a', 'c', 'd'])
>>> s + s3
a    10.0
b     NaN
c     5.0
d     7.0
```

Arithmetic Operations with Fill Methods

You can also do the internal data alignment yourself with the help of the fill methods:

```
>>> s.add(s3, fill_value=0)
a    10.0
b    -5.0
c    5.0
d    7.0
>>> s.sub(s3, fill_value=2)
>>> s.div(s3, fill_value=4)
>>> s.mul(s3, fill_value=3)
```

I/O

Read and Write to CSV

```
>>> pd.read_csv('file.csv', header=None, nrows=5)
>>> df.to_csv('myDataFrame.csv')
```

Read and Write to Excel

```
>>> pd.read_excel('file.xlsx')
>>> pd.to_excel('dir/myDataFrame.xlsx', sheet_name='Sheet1')
```

Read multiple sheets from the same file

```
>>> xlsx = pd.ExcelFile('file.xls')
>>> df = pd.read_excel(xlsx, 'Sheet1')
```

Read and Write to SQL Query or Database Table

```
>>> from sqlalchemy import create_engine
>>> engine = create_engine('sqlite:///:memory:')
>>> pd.read_sql("SELECT * FROM my_table;", engine)
>>> pd.read_sql_table('my_table', engine)
>>> pd.read_sql_query("SELECT * FROM my_table;", engine)
```

`read_sql()` is a convenience wrapper around `read_sql_table()` and `read_sql_query()`

```
>>> pd.to_sql('myDf', engine)
```

DataCamp
Learn Python for Data Science Interactively

https://www.coursehero.com/file/19883028/PandasPythonForDataScience/