UC Berkeley
SCHOOL OF INFORMATION

Module 11

Pandas

Project 2 prep

Programming for Data Science

Pandas and activities.

Mar 12, 2020 [gb]; updated Mar 19, 2020

1. Today's activities

UC Berkeley
SCHOOL OF INFORMATION

- Intro to Project 2; Teams & coordination
- Overview of Data Exploration & Analysis; Architectures and analytics
- Pandas - really more of an exploration of possibilities
  - Demos in resources folder; class activity; SciKit
- Team Discussions (breakout rooms to get to share ideas and perhaps establish a team for the last project)
- Next class session: visualization!

9	3-Mar	4-Mar	5-Mar	7-Mar	Unit 9	Text and Binary Data					Exam 1		
10	10-Mar	11-Mar	12-Mar	14-Mar	Unit 10	NumPy - Vectors	Project 1 Presentation	HW unit 9					Project 1 Code
11	17-Mar	18-Mar	19-Mar	21-Mar	Unit 11	Pandas - Dataframes		HW unit 10	HW unit 9			Project 2	
	24-Mar	25-Mar	26-Mar	28-Mar		Spring Break - no classes!							
12	31-Mar	1-Apr	2-Apr	4-Apr	Unit 12	MatPlotLib - Data Visualization		HW on units 11-13	HW unit 10				Project 2 Proposal
13	7-Apr	8-Apr	9-Apr	11-Apr	Unit 13	Advanced Pandas - Aggregation & Groups			HW units 11-13	Exam 2			
14	14-Apr	15-Apr	16-Apr	18-Apr	Unit 14	Testing	Project 2 Presentation				Exam 2		Project 2 Report

Calendar

2. *Activity Room 1:*

UC Berkeley
SCHOOL OF INFORMATION

- Spend about 10-15 minutes in breakout rooms discussing ideas and building a team.
- You can create teams across sections but let all instructors know.
- If you have a team already, we'll have a breakout room just for you; others assigned however … just lmk.
- Tho obvious - plan times to meet, contact info, assign tasks.

optional tech tangent

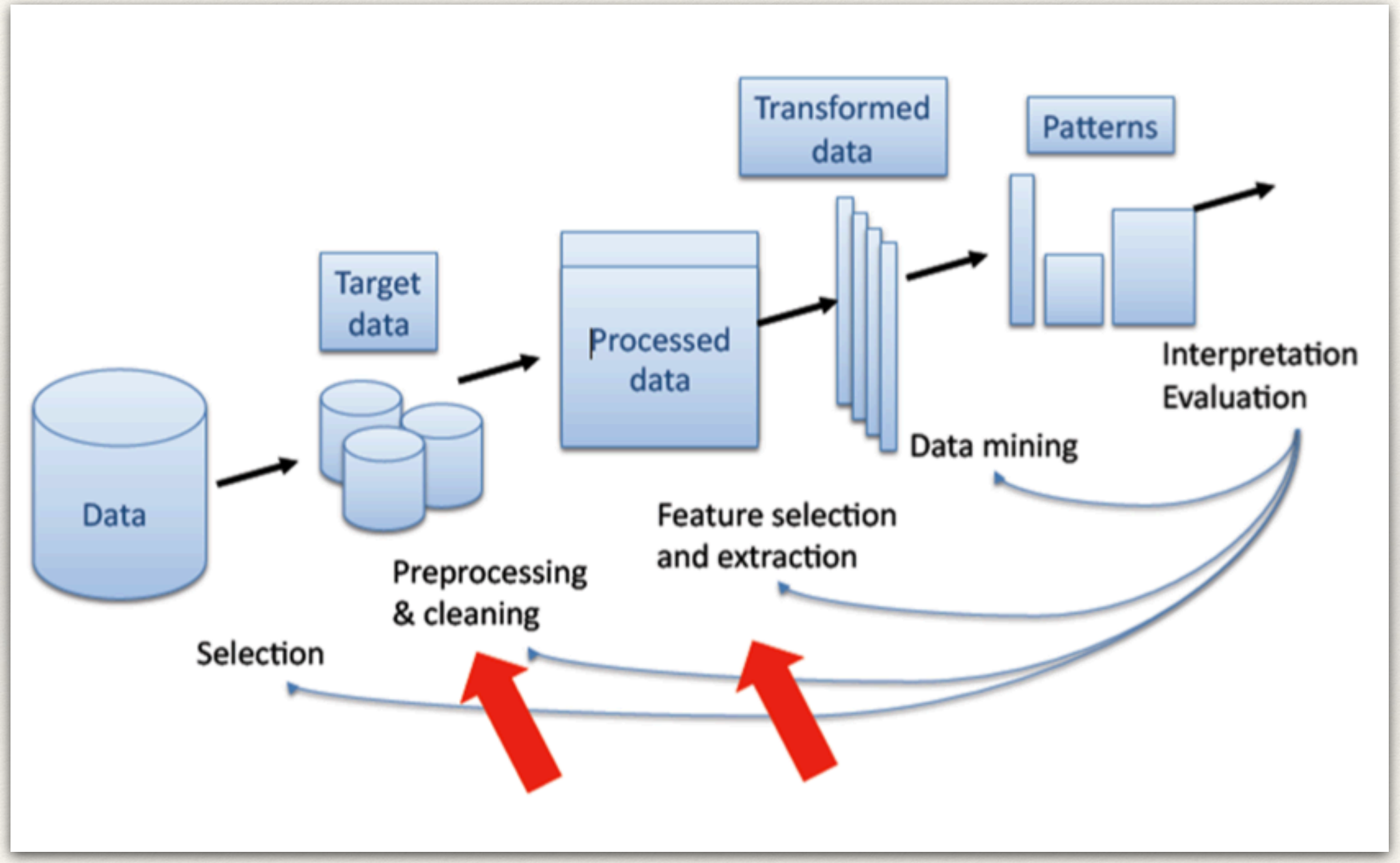
3. Analysis activities - overview

UC Berkeley
SCHOOL OF INFORMATION

- Identify data sources/streams
  - Get to know the content and structure
  - Dimensionality reduction
  - Harmonize integration; create new/delete/merge columns
  - Domain, range; MCT (usual stats)
- Start with a version of"data cleansing", "data cleaning", "warehousing", KDD (knowledge discovery in databases);
  - Optional data cleaning slides in resources; Data Mining article
- Common: researchers, others use Excel, R, Stata, SQL, .json
- Proceed from data, to intermediary sources, to special-purpose architectures

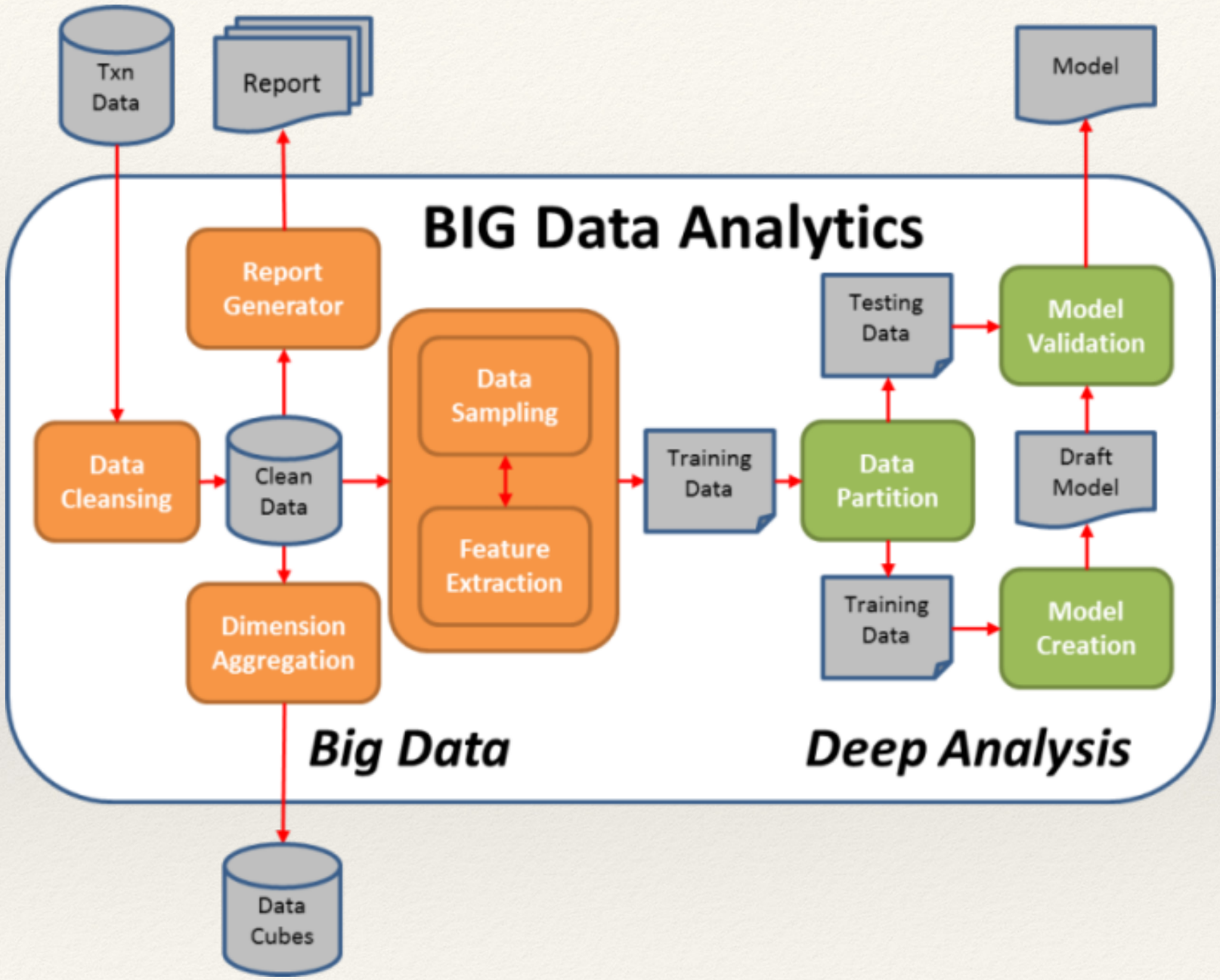
optional tech tangent

3. Analysis activities - overview



3. *Typical architecture for analysis*

UC Berkeley
SCHOOL OF INFORMATION

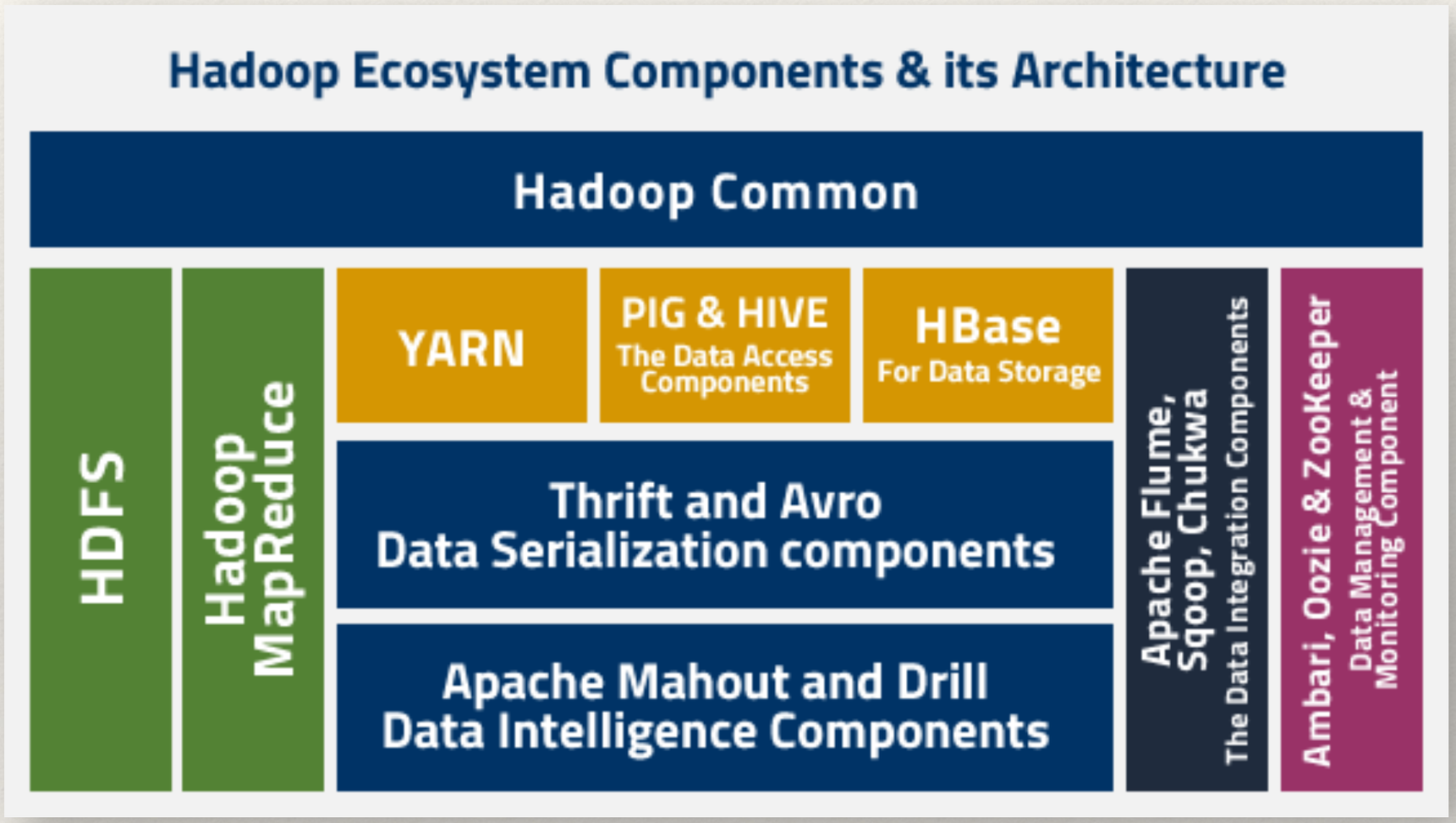


http://www.byteorigin.com/services/cloud-application-development/big-data-analytics/

Compare to Hadoop, Spark; data mining hypothesis confirmation or hypothesis generation

3. Typical architecture for analysis

UC Berkeley
SCHOOL OF INFORMATION



Overview of Hadoop: software components and architecture (master/slave and task submission/tracking) (con't next slide)

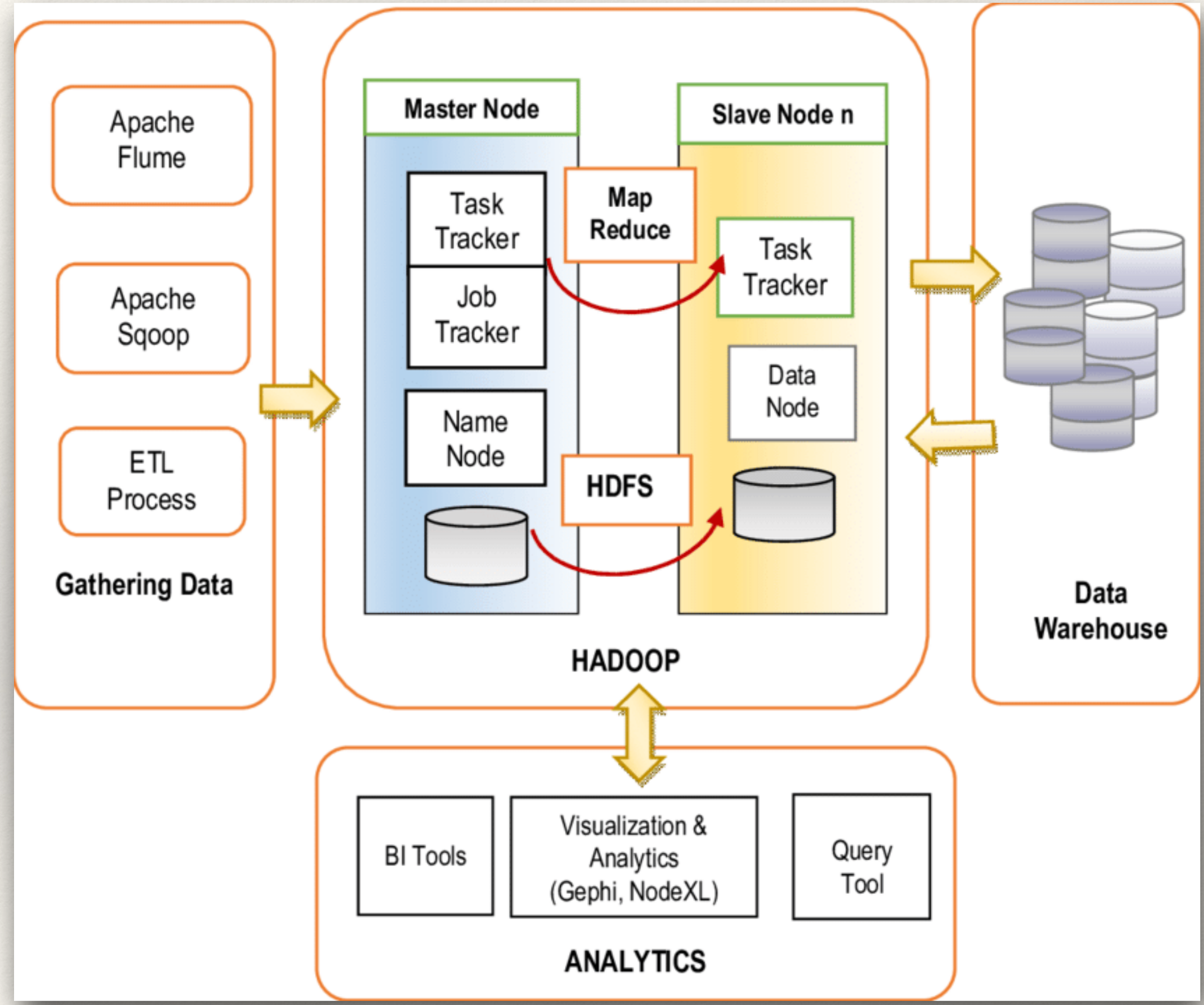
See the Hadoop book in the resources folder on github.

Apache Pig

https://pig.apache.org
https://en.wikipedia.org/wiki/Apache_Pig

https://project.carrot2.org
CARROT search

optional tech tangent



3. *Exploration & Analysis*

UC Berkeley
SCHOOL OF INFORMATION

- Data Exploration
  - For data integrity
  - Explore to develop questions based on the variables you have
  - Try to break your model … better now than in production!
- Data Analysis
  - to answer a research question or hypothesis*
  - Usually involves complex math, modeling, statistics
  - Likely to combine multiple datasets
  - Explore your data by collapsing in grouping in various ways
  - Some functions are useful in both exploration and analysis.

Be careful using research/stat terms in a casual way - some terms have specific implications, such as "correlation", "research question", "proved", and more.

3. *Exploration & Analysis*

UC Berkeley
SCHOOL OF INFORMATION

- Take the time to get familiar with the data! Consider the domain & range of the data; measures of central tendency; parametric or non-parametric? Use some handy basic commands:
  - `value_counts()`
  - `describe()`
  - `min(), max(), isnull()`. Check for NaN, empty vars, 0s.
  - Typical plots help explore the data (histogram, scatter, bar chart)
    - More on the visualization parts in Module 12.

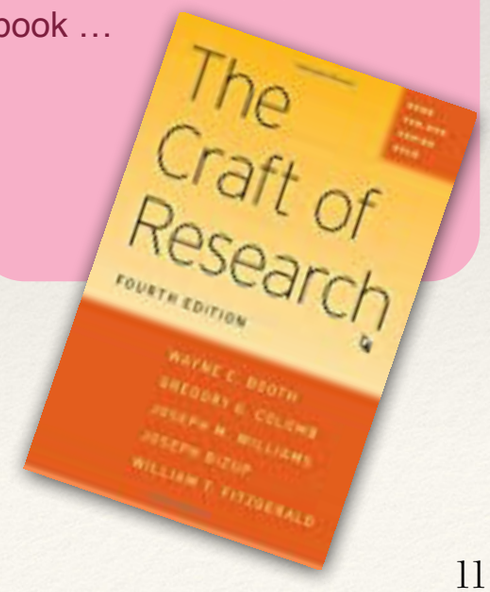
- A main point: to suggest a standardized set of actions for you ("what do I do now?!) when inspecting data sources, data themselves, tailored for your domain.

3. *Exploration & Analysis*

UC Berkeley
SCHOOL OF INFORMATION

- Consider the source(s) of your data (dataset documentation)
- Research the topic …
- Tho not a research or stats course, consider "threats to validity" [research bias, lack of precision in definitions]; cross-validation; are your data robust?
  - e.g., if the variable "hospital_los" means "length of stay", we should be able to compare this to other vars, such as "admit_date" and "discharge_date"
- [BTW, look to professional gold-standards that apply to your missing data or expectations? Are there typical confounding events or sources of error? Look for threats to validity.]
  - E.g., we expect electrical power to cost more when demand is - let's find the max power cost in the dataset and compare to daily temps to ensure it makes sense (would have to consider parametric/non-parametric and chi-square but that's for another course (grin))

By the way, for a good intro/ review of research, see this book …



3. *Data Exploration & Analysis*

UC Berkeley
SCHOOL OF INFORMATION

- Think about analysis as a series of dataset transformations … how to explore our data? Maybe …
  - filter rows based on some condition (are values too high, not applicable to the current task? are there duplicated data?)
  - create new columns when necessary
  - aggregate or collapse by groups (may be better to deal with x number of groups than a zillion individual data points; think of statistical error, too)
  - join two or more datasets together (not without its own problems, tho, e.g., NaN; may need intermediary "cross-walk", akin to a database cursor - also for another course; Zhang & Chan)

4. Pandas (finally!)

UC Berkeley
SCHOOL OF INFORMATION

- A quick visual reminder of some groups of commands, grouped by function, to help manage and analyze data.
- See the resources git folder.

Optional: see https://www.tutorialspoint.com/python_pandas/python_pandas_visualization.htm

https://pandas.pydata.org/pandas-docs/stable/10min.html

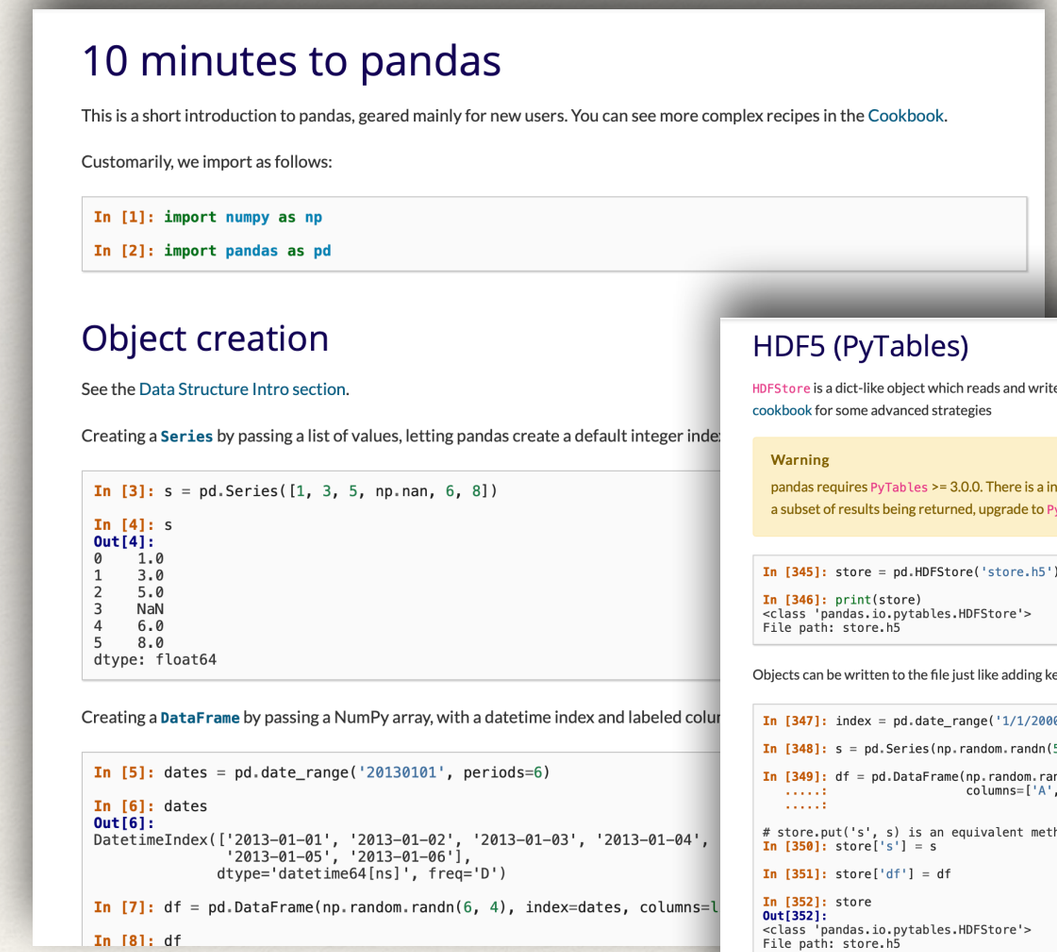
optional tech tangent

These optional tech tangent slides were added to suggest to students a variety of more-efficient problem-solving techniques when working with files and analyses. Pandas has a lots of tools for I/O with

https://pandas.pydata.org/pandas-docs/stable/getting_started/10min.html

https://pandas.pydata.org/pandas-docs/stable/user_guide/io.html#io-hdf5

- CSV & Text
- JSON
- HTML
- Excel
- OpenDoc
- Binary Excel
- Clipboard
- Pickling
- msgpack
- Feather
- Parquet
- ORC
- SQL
- GoogleBigQuery
- Stata
- SAS
- SPSS
- and more



10 minutes to pandas

This is a short introduction to pandas, geared mainly for new users. You can see more complex recipes in the Cookbook.

Customarily, we import as follows:

```
In [1]: import numpy as np
In [2]: import pandas as pd
```

Object creation

See the Data Structure Intro section.

Creating a `Series` by passing a list of values, letting pandas create a default integer inde

```
In [3]: s = pd.Series([1, 3, 5, np.nan, 6, 8])

In [4]: s
Out[4]:
0    1.0
1    3.0
2    5.0
3    NaN
4    6.0
5    8.0
dtype: float64
```

Creating a `DataFrame` by passing a NumPy array, with a datetime index and labeled colu

```
In [5]: dates = pd.date_range('20130101', periods=6)

In [6]: dates
Out[6]:
DatetimeIndex(['2013-01-01', '2013-01-02', '2013-01-03', '2013-01-04',
               '2013-01-05', '2013-01-06'],
              dtype='datetime64[ns]', freq='D')

In [7]: df = pd.DataFrame(np.random.randn(6, 4), index=dates, columns=l

In [8]: df
```

HDF5 (PyTables)

HDFStore is a dict-like object which reads and writes pandas using the high performance HDF5 format using the excellent PyTables library. See the cookbook for some advanced strategies

Warning

pandas requires PyTables >= 3.0.0. There is a indexing bug in PyTables < 3.2 which may appear when querying stores using an index. If you see a subset of results being returned, upgrade to PyTables >= 3.2. Stores created previously will need to be rewritten using the updated version.

```
In [345]: store = pd.HDFStore('store.h5')

In [346]: print(store)
<class 'pandas.io.pytables.HDFStore'>
File path: store.h5
```

Objects can be written to the file just like adding key-value pairs to a dict:

```
In [347]: index = pd.date_range('1/1/2000', periods=8)

In [348]: s = pd.Series(np.random.randn(5), index=['a', 'b', 'c', 'd', 'e'])

In [349]: df = pd.DataFrame(np.random.randn(8, 3), index=index,
   .....:                   columns=['A', 'B', 'C'])
   .....:

# store.put('s', s) is an equivalent method
In [350]: store['s'] = s

In [351]: store['df'] = df

In [352]: store
Out[352]:
<class 'pandas.io.pytables.HDFStore'>
File path: store.h5
```

In a current or later Python session, you can retrieve stored objects:

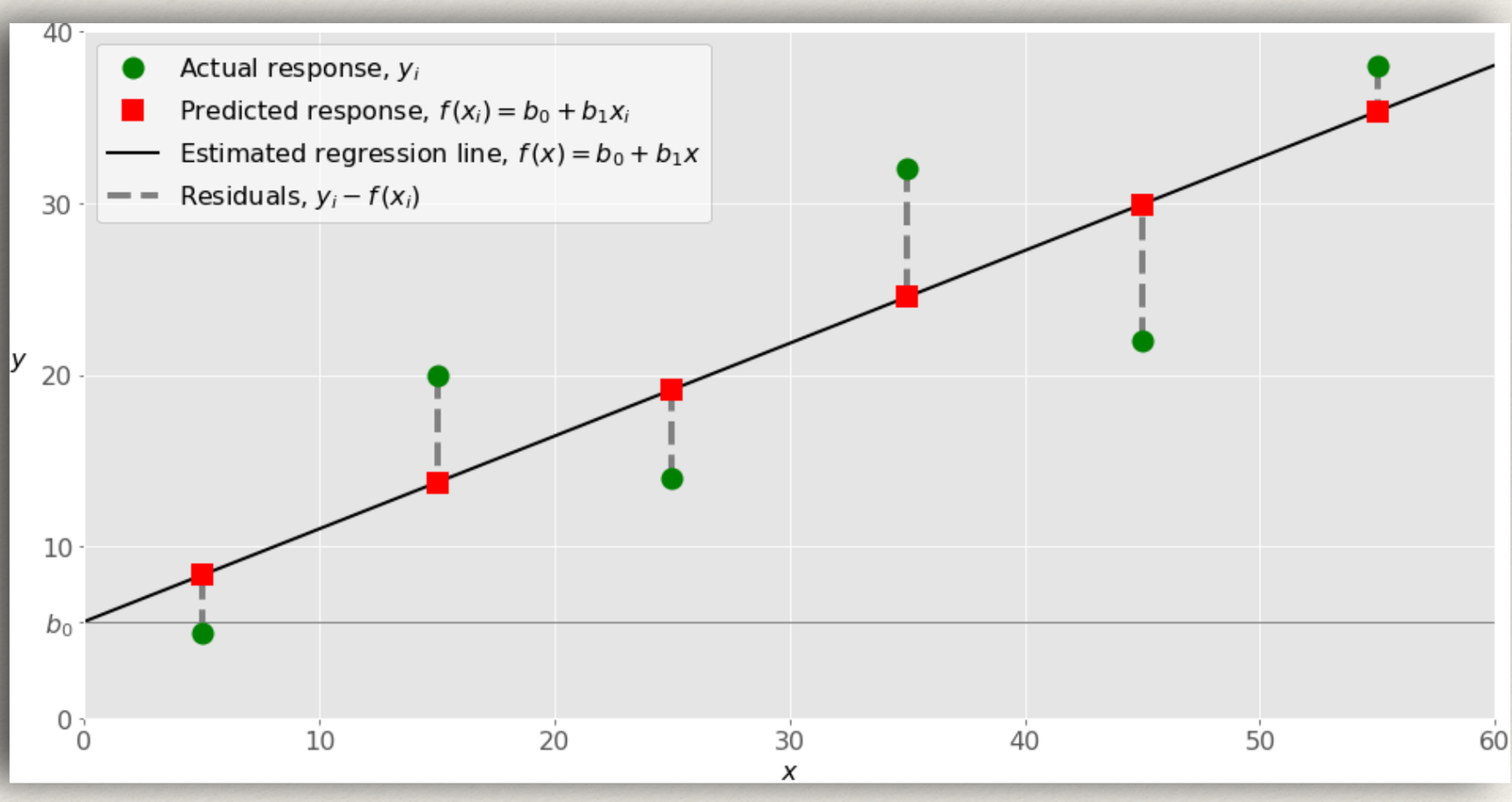
```
# store.get('df') is an equivalent method
In [353]: store['df']
Out[353]:
                   A         B         C
2000-01-01  1.334065  0.521036  0.930384
2000-01-02 -1.613932  1.088104 -0.632963
```

optional tech tangent

In the breakout activity, we'll look at a host of pandas and numpy methods for learning about data, extracting trends, and so on.

BTW, you might enjoy this site about vectorization.

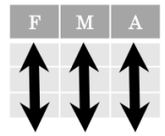
And this interesting intro to simple linear, multiple, and polynomial regression, as well as over- and under-fitting data models.



Data Wrangling
with pandas
Cheat Sheet
http://pandas.pydata.org

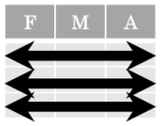
Tidy Data – A foundation for wrangling in pandas

In a tidy data set:



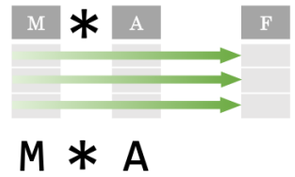
Each **variable** is saved in its own **column**

&



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.



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

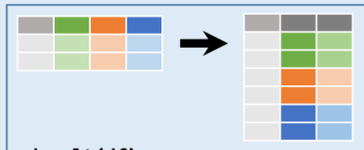
```
df = pd.DataFrame(
        {"a" : [4 ,5, 6],
         "b" : [7, 8, 9],
         "c" : [10, 11, 12]},
index = pd.MultiIndex.from_tuples(
        [('d',1),('d',2),('e',2)],
          names=['n','v'])))
```
Create DataFrame with a MultiIndex

Method Chaining

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

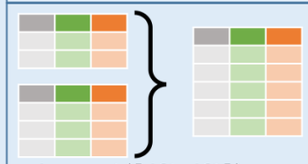
```
df = (pd.melt(df)
        .rename(columns={
                'variable' : 'var',
                'value' : 'val'})
        .query('val >= 200')
    )
```

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).

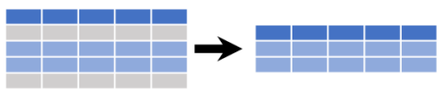
df.rename(columns = {'y':'year'})
Rename the columns of a DataFrame

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(columns=['Length','Height'])
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.

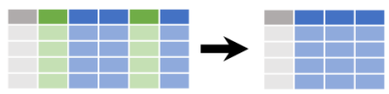
df.nlargest(n, 'value')
Select and order top n entries.

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 .

http://pandas.pydata.org/ This cheat sheet inspired by Rstudio Data Wrangling Cheatsheet (https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf) Written by Irv Lustig, Princeton Consultants

Summarize Data

`df['w'].value_counts()`
Count number of rows with each unique value of variable

`len(df)`
# of rows in DataFrame.

`df['w'].nunique()`
# of distinct values in a column.

`df.describe()`
Basic descriptive statistics for each column (or GroupBy)

pandas provides a large set of **summary functions** that operate on different kinds of pandas objects (DataFrame columns, Series, GroupBy, Expanding and Rolling (see below)) and produce single values for each of the groups. When applied to a DataFrame, the result is returned as a pandas Series for each column. Examples:

`sum()`
Sum values of each object.

`count()`
Count non-NA/null values of each object.

`median()`
Median value of each object.

`quantile([0.25,0.75])`
Quantiles of each object.

`apply(function)`
Apply function to each object.

`min()`
Minimum value in each object.

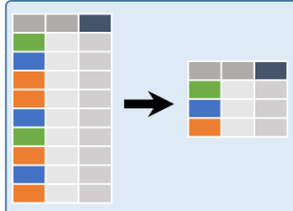
`max()`
Maximum value in each object.

`mean()`
Mean value of each object.

`var()`
Variance of each object.

`std()`
Standard deviation of each object.

Group Data



`df.groupby(by="col")`
Return a GroupBy object, grouped by values in column named "col".

`df.groupby(level="ind")`
Return a GroupBy object, grouped by values in index level named "ind".

All of the summary functions listed above can be applied to a group. Additional GroupBy functions:

`size()`
Size of each group.

`agg(function)`
Aggregate group using function.

The examples below can also be applied to groups. In this case, the function is applied on a per-group basis, and the returned vectors are of the length of the original DataFrame.

`shift(1)`
Copy with values shifted by 1.

`rank(method='dense')`
Ranks with no gaps.

`rank(method='min')`
Ranks. Ties get min rank.

`rank(pct=True)`
Ranks rescaled to interval [0, 1].

`rank(method='first')`
Ranks. Ties go to first value.

`shift(-1)`
Copy with values lagged by 1.

`cumsum()`
Cumulative sum.

`cummax()`
Cumulative max.

`cummin()`
Cumulative min.

`cumprod()`
Cumulative product.

Windows

`df.expanding()`
Return an Expanding object allowing summary functions to be applied cumulatively.

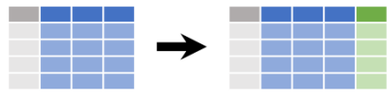
`df.rolling(n)`
Return a Rolling object allowing summary functions to be applied to windows of length n.

Handling Missing Data

`df.dropna()`
Drop rows with any column having NA/null data.

`df.fillna(value)`
Replace all NA/null data with value.

Make New Columns



`df.assign(Area=lambda df: df.Length*df.Height)`
Compute and append one or more new columns.

`df['Volume'] = df.Length*df.Height*df.Depth`
Add single column.

`pd.qcut(df.col, n, labels=False)`
Bin column into n buckets.

pandas provides a large set of **vector functions** that operate on all columns of a DataFrame or a single selected column (a pandas Series). These functions produce vectors of values for each of the columns, or a single Series for the individual Series. Examples:

`max(axis=1)`
Element-wise max.

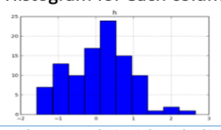
`clip(lower=-10,upper=10)`
Trim values at input thresholds

`min(axis=1)`
Element-wise min.

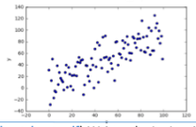
`abs()`
Absolute value.

Plotting

`df.plot.hist()`
Histogram for each column



`df.plot.scatter(x='w',y='h')`
Scatter chart using pairs of points



Combine Data Sets

adf

x1	x2
A	1
B	2
C	3

+

bdf

x1	x3
A	T
B	F
D	T

=

Standard Joins

x1	x2	x3
A	1	T
B	2	F
C	3	NaN

`pd.merge(adf, bdf, how='left', on='x1')`
Join matching rows from bdf to adf.

x1	x2	x3
A	1.0	T
B	2.0	F
D	NaN	T

`pd.merge(adf, bdf, how='right', on='x1')`
Join matching rows from adf to bdf.

x1	x2	x3
A	1	T
B	2	F

`pd.merge(adf, bdf, how='inner', on='x1')`
Join data. Retain only rows in both sets.

x1	x2	x3
A	1	T
B	2	F
C	3	NaN
D	NaN	T

`pd.merge(adf, bdf, how='outer', on='x1')`
Join data. Retain all values, all rows.

Filtering Joins

x1	x2
A	1
B	2

`adf[adf.x1.isin(bdf.x1)]`
All rows in adf that have a match in bdf.

x1	x2
C	3

`adf[~adf.x1.isin(bdf.x1)]`
All rows in adf that do not have a match in bdf.

ydf

x1	x2
A	1
B	2
C	3

+

zdf

x1	x2
B	2
C	3
D	4

=

Set-like Operations

x1	x2
B	2
C	3

`pd.merge(ydf, zdf)`
Rows that appear in both ydf and zdf (Intersection).

x1	x2
A	1
B	2
C	3
D	4

`pd.merge(ydf, zdf, how='outer')`
Rows that appear in either or both ydf and zdf (Union).

x1	x2
A	1

```
pd.merge(ydf, zdf, how='outer',
         indicator=True)
.query('_merge == "left_only"')
.drop(columns=['_merge'])
```
Rows that appear in ydf but not zdf (Setdiff).

http://pandas.pydata.org/ This cheat sheet inspired by Rstudio Data Wrangling Cheatsheet (https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf) Written by Irv Lustig, Princeton Consultants

		Description	example

Series	1	1D labeled homogeneous array, sizeimmutable.	`#import the pandas`
			`import pandas as pd`
			`import numpy as np`
			`data =`
			`np.array(['a','b','c','d'])`
			`s = pd.Series(data)`
			`print s`

Data Frames	2	General 2D labeled, size-mutable tabular structure with potentially heterogeneously typed columns.	`import pandas as pd`
			`data = [1,2,3,4,5]`
			`df = pd.DataFrame(data)`
			`print df`

Panel	3	General 3D labeled, size-mutable array.	`# creating an empty panel`
			`import pandas as pd`
			`import numpy as np`
			
			`data = np.random.rand(2,4,5)`
			`p = pd.Panel(data)`
			`print p`

Think of univariate, bi-variate, and multi-variate data.

4. Pandas

UC Berkeley
SCHOOL OF INFORMATION

- In the breakout, start with asking

1. what's the maximum donation?
2. the most common occupation who donated this amount?
3. in what zip code area do the most people identified in #2 live?

The subsequent slides compare questions we'd ask with pandas commands.

Think of some scenarios - why check the head() of the dataset?

The rest of the slideset is just a review of the many options of Pandas - too many to know off the top of your head - but a useful setup for exploring data and prep for tonight's activities.

https://github.prod.oc.2u.com/UCB-INFO-PYTHON/assignments_upstream_spring20/tree/master/resources/pandas-lots-of-demos

`data = pd.Series([1,2,4,6,0,85,45,7,53,321,4,32,2355, 6])`	# Let's experiment with this series
`data[data < 10]`	# select values < 10
`data[(data < 10) & (data > 5)]`	# how 'bout a range with keywords

`data[data < 10][data > 5]`	# same result but "chained"
	`>>> data[(data < 10) & (data > 5)]`
	`3  6`
	`7  7`
	`dtype: int64`
	`>>> data[data < 10][data > 5]`
	`3  6`
	`7  7`
	`dtype: int64`

`# using booleans as counter: (data > 5).sum()`	# True = 1; False = 0

Slicing & Manipulating	
`d5 = data.head().copy()`	# make a copy … for safety!
`d6 = data.head(70).copy()`	# specify the size of the head (ceiling, e.g., up to 70)
`data[0:10:2]`	$ standard slicing - [start:end (exclusive):step]
`data[0] = 10000`	# value replacement
`d5_6 = pd.concat([d5, d6])`	

any and **all** tests for series	
`data[data < 10].any()`	# true
`data[data < 10].all()`	# false
`(data > 10000).any()`	# Alternatives … false
`(data > 0).all()`	# false

`len(data)`	

`data.mean()`
`data.mode()`
`data.median()`
`data.count()`
`data.std()`
`data.unique()`
	# can get individual “measures of
	# central tendency … or use
	# **describe()** for the whole thing!
	`>>> data.describe()`
	`count  16.000000`
	`mean  640.312500`
	`std  2496.070718`
	`min  0.000000`
	`25%  2.000000`
	`50%  4.000000`
	`75%  16.500000`
	`max  10000.000000`
	`dtype: float64`

`data.value_counts()`	# super useful … experiment
`data.shape`	# note that this is an *attribute*, # not a *method*.

`data[10]`	# by single index names
`data[ [10,20] ]`	# by multiple index names
	# if the index is not numeric, pandas interprets numbers as row number
`d5.iloc[ [0,3] ]`	# lookup by **index location** “dict style[]”
`data.index = range(100, len(data) + 100)`	# set new **index names**
`data.loc[ [100, 103] ]`	# we can use **`.loc`** to call by **index names**
`data.iget( [0,3] ), data.ix( [0,3] )`	# Depreciated - don’t use

`data.index = New_Index`	# overwrite the existing index
`new_data = data.reindex([0,2,15,21])`	# slide out rows and use their indices; missing values get “NaN”
`data.reindex([0,2,15,21], fill_value=0)`	# specify the missing values

`combo.reindex([0,2,15,21], fill_value=0)`	# set fill value
`new_combo.fillna(0)`	# fill those NaNs!
Forward and backward fill - guess at missing values - common in practice	
`new_combo.ffill()`	# take *forward* fills …
`new_combo.bfill()`	# take *backward* fills
`new_combination.interpolate()`	# fills missing values with linear interpolation *Note! There are several important techniques for missing values.

`s1 = pd.Series(['1', '3'])`	
`s1 = s1.astype(int)`	# what does this do to s1?
`s1.map(lambda x: x**2)`	# pass a series to a lambda function
`s1.map({'1':2, '2':3, '3': 12})`	# basic mapping with a dictionary

`pd.DataFrame([upcase, lcase])`	# make a DataFrame from a series
`pd.DataFrame({'lowercase': class, 'uppercase':upcase})`	# can pass column names
`letters.columns = ['LowerCase', 'UpperCase']`	# explicitly set the columns
`letters.index = lcase`	# change the indices
`letters.sort_values('Number'), letters.sort_index()`	# sort by column or by index
`letters[ ['LowerCase', 'UpperCase'] ]`	# slide columns by name

Viewing your data by a category can yield critical insights groupby

```
In [114]: df.groupby('Day_of_week').mean()
```

Out[114]:

	Miles	Minutes	Min_per_mile
Day_of_week			
Friday	2.786000	24.308333	7.747657
Monday	2.607143	22.243333	7.463291
Saturday	3.246429	46.708333	8.184961
Sunday	2.422727	19.762500	7.463840
Thursday	6.315000	84.530000	8.039543
Tuesday	2.428182	21.770833	7.659706
Wednesday	3.315000	28.021429	7.829348

Summary

UC Berkeley
SCHOOL OF INFORMATION

Today we looked at trends towards working in data science by considering …

- architecture for large data processing/analysis
- exploring and analyzing data practices
- introduced the bridge to research practices and converting our questions into code
  - using pandas
- And getting ready for your final project.

Good luck with your projects!

Your presentations focus on your communication of your project, not the tech notes.

がんばろう！

حظا سعيدا!

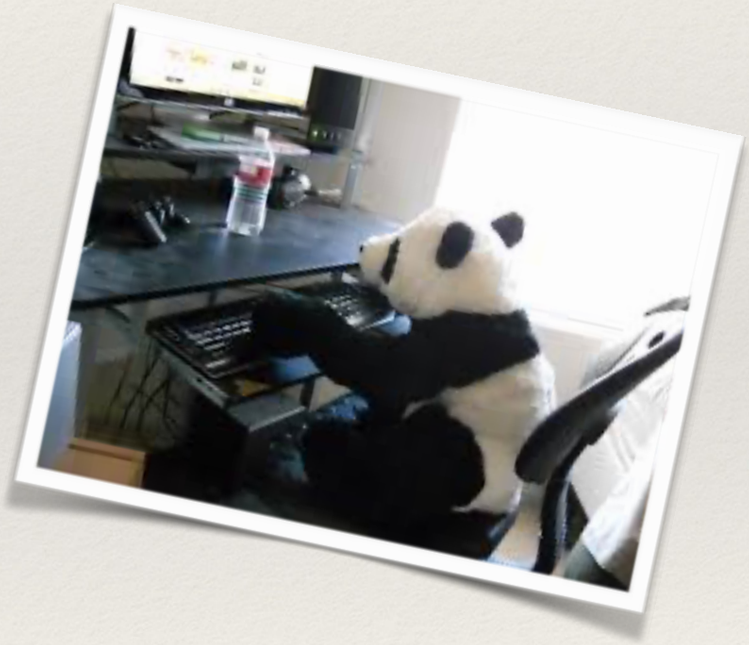
¡Buena suerte!

सौभाग्य!

Удачи!

ਖੁਸ਼ਕਿਸਮਤੀ!

اچھی قسمت!



સારા નસીબ!

Viel Glück!

祝好运！

Du courage!

Keep going!