Data Science with Python and Pandas

Dylan Gregersen

gregersen.dylan@gmail.com

About Me

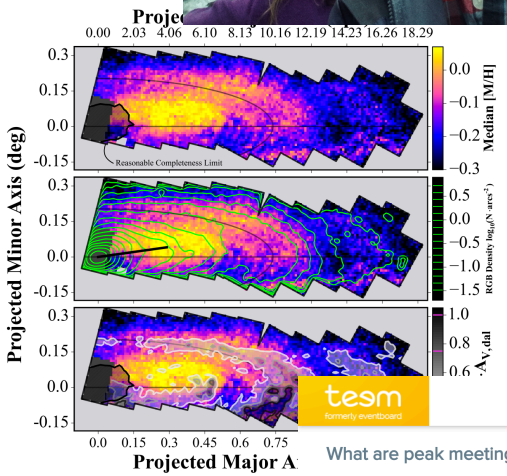
My name is Dylan Gregersen

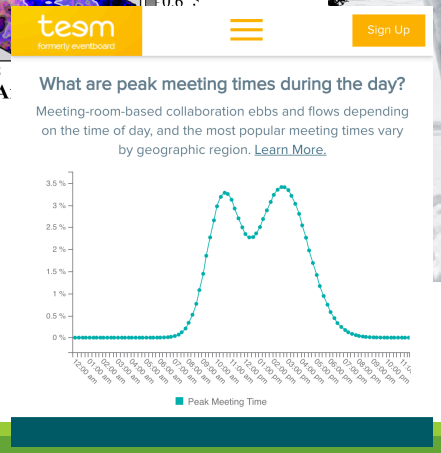
I have a degree in Physics and published research in Astronomy

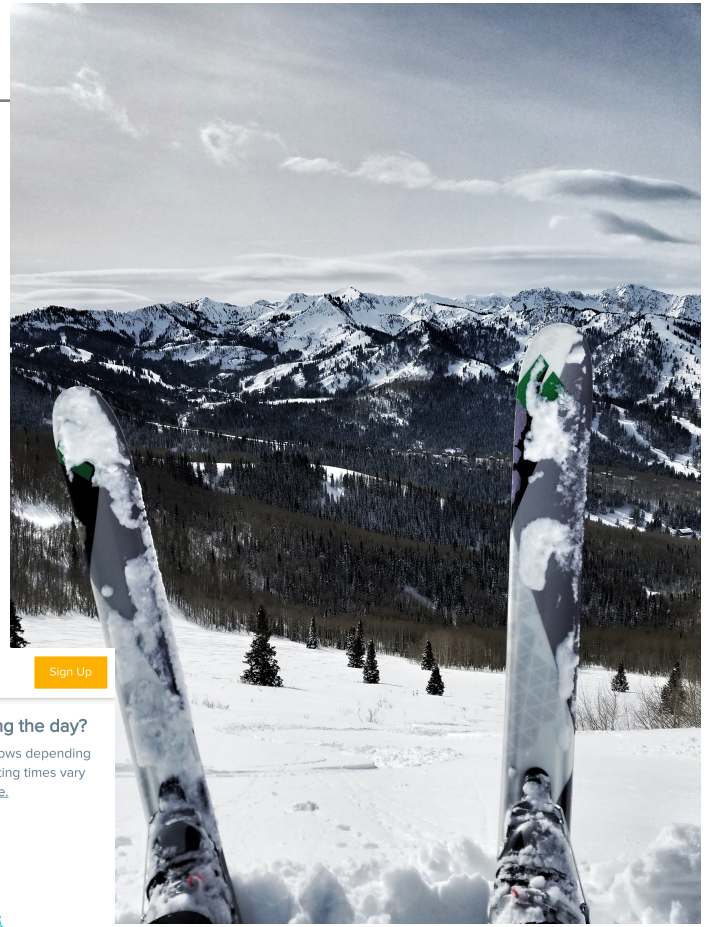
I currently work at Teem as a Data Scientist and Python Dev

I love getting outside by skiing, kayaking or hiking. I'm also progressing to a swing dance king!

gregersen.dylan@gmail.com
astrodsg.github.io

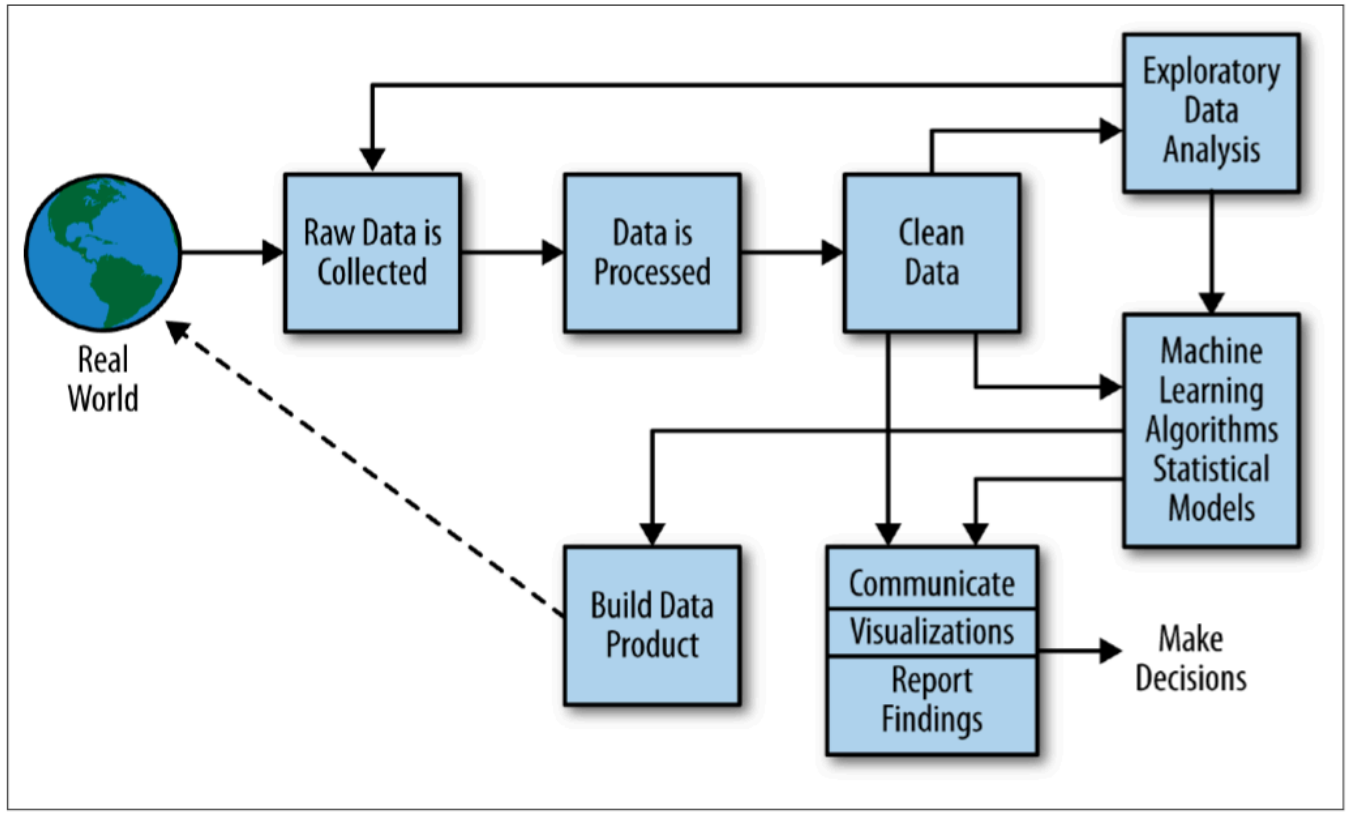






What is Data Science?

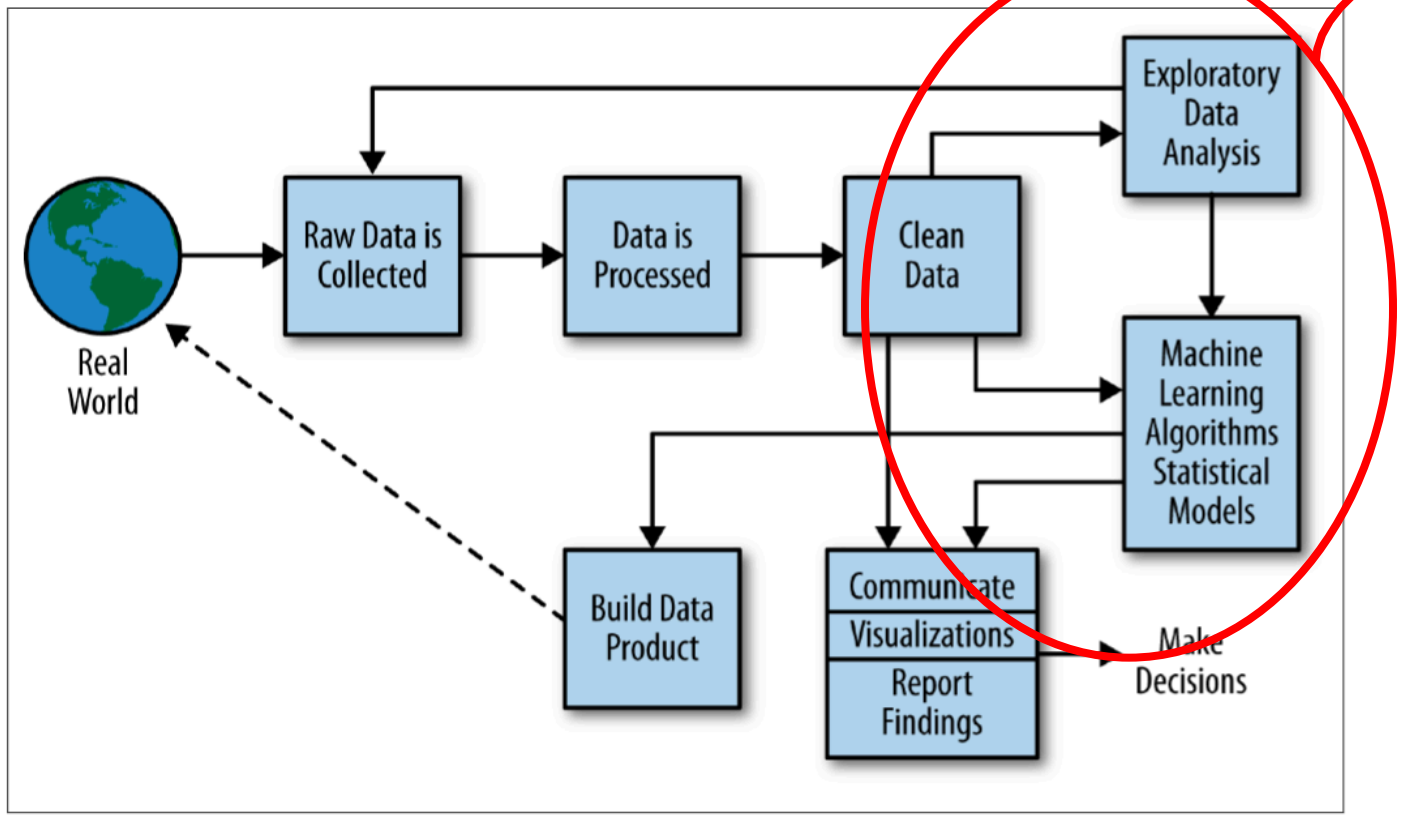
Data Science



Data Science is a combination of collecting, munging, analyzing, visualizing, and communicating data to **answer questions** about the world around us

Image Credit: Rachel Schutt & Cathy O'Neil in Doing Data Science: Straight Talk From the Frontline

Data Science Process

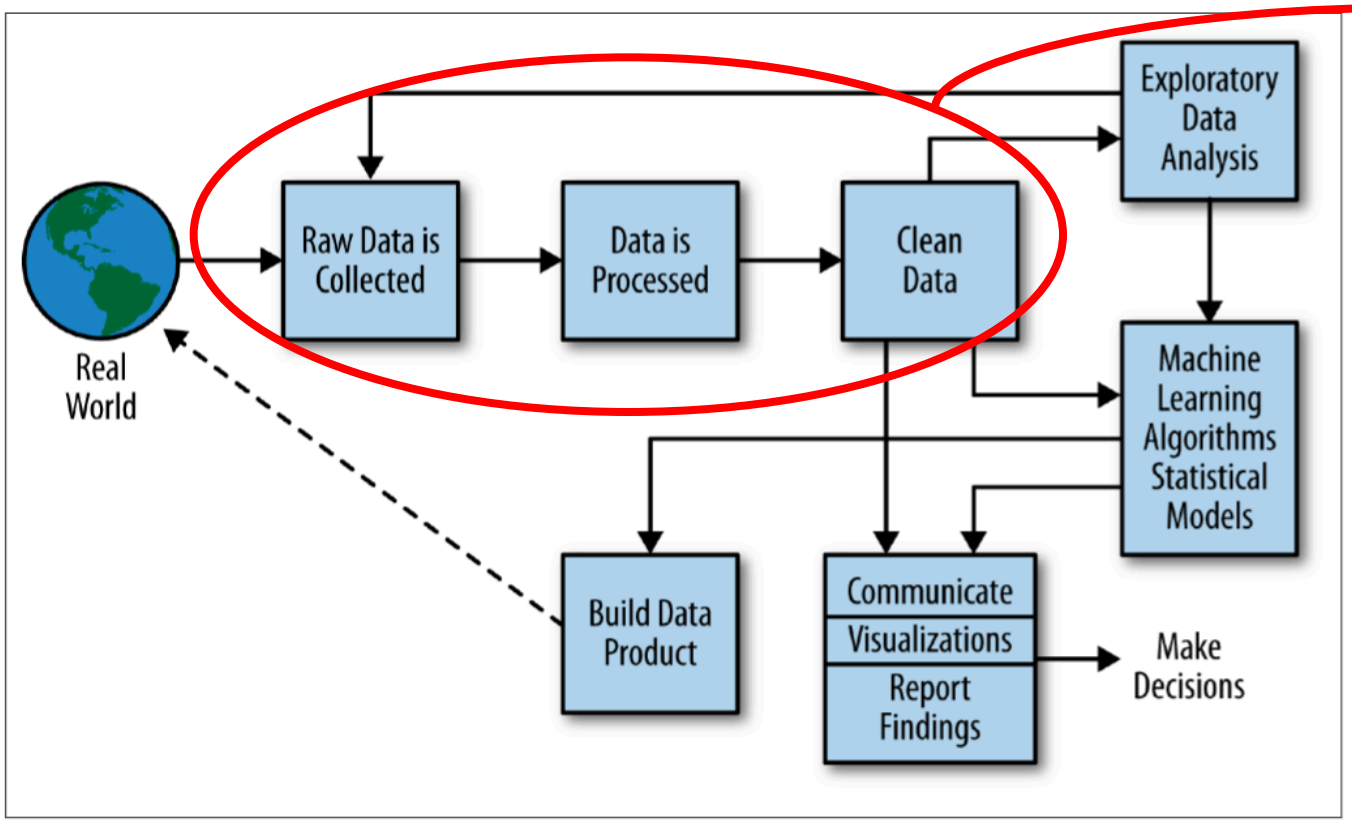


A common conception of what all **Data Science** is. Analysis.

Requires skills Machine Learning, Mathematics, Statistics and Domain Expertise

Image Credit: Rachel Schutt & Cathy O'Neil in Doing Data Science: Straight Talk From the Frontline

Data Science Process



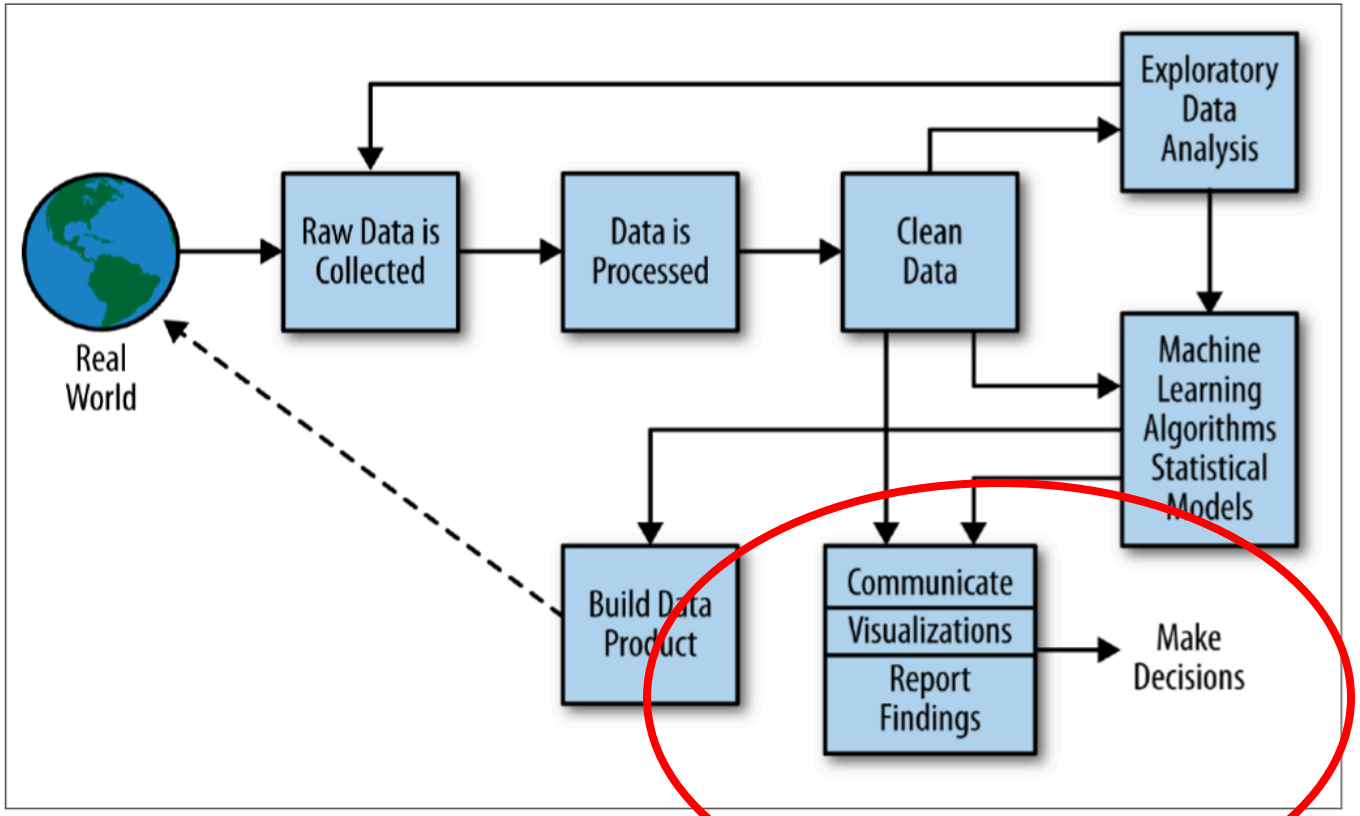
Data Science actually spends 80% of the time doing

Requires skills in Computer Science, Domain Expertise, and Mathematics

Image Credit: Rachel Schutt & Cathy O'Neil in Doing Data Science: Straight Talk From the Frontline

Image Credit: Rachel Schutt & Cathy O'Neil in Doing Data Science: Straight Talk From the Frontline

Data Science Process



Data Science tends to struggle when communicating and delivering actionable insights

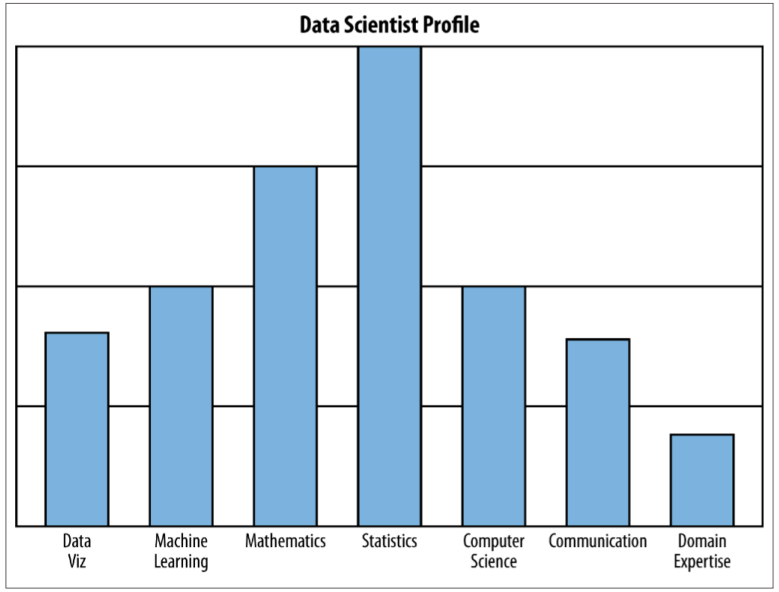
Requires skills in Communication, Data Visualization, Domain Expertise, Statistics

Data Science

Data Science really incorporates many different jobs and skills

Being a "**data scientist**" is very much like being a **"full stack developer"**

At Teem we now have a teem of 8 people with a variety of specialties - Exploring the data, creating a data warehouse, data pipeline creation, visualization creation



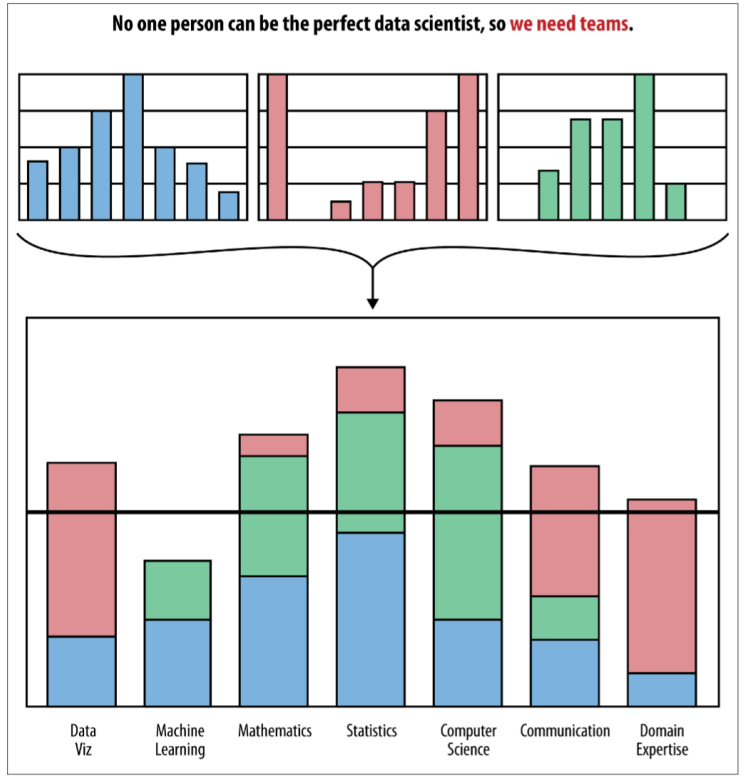
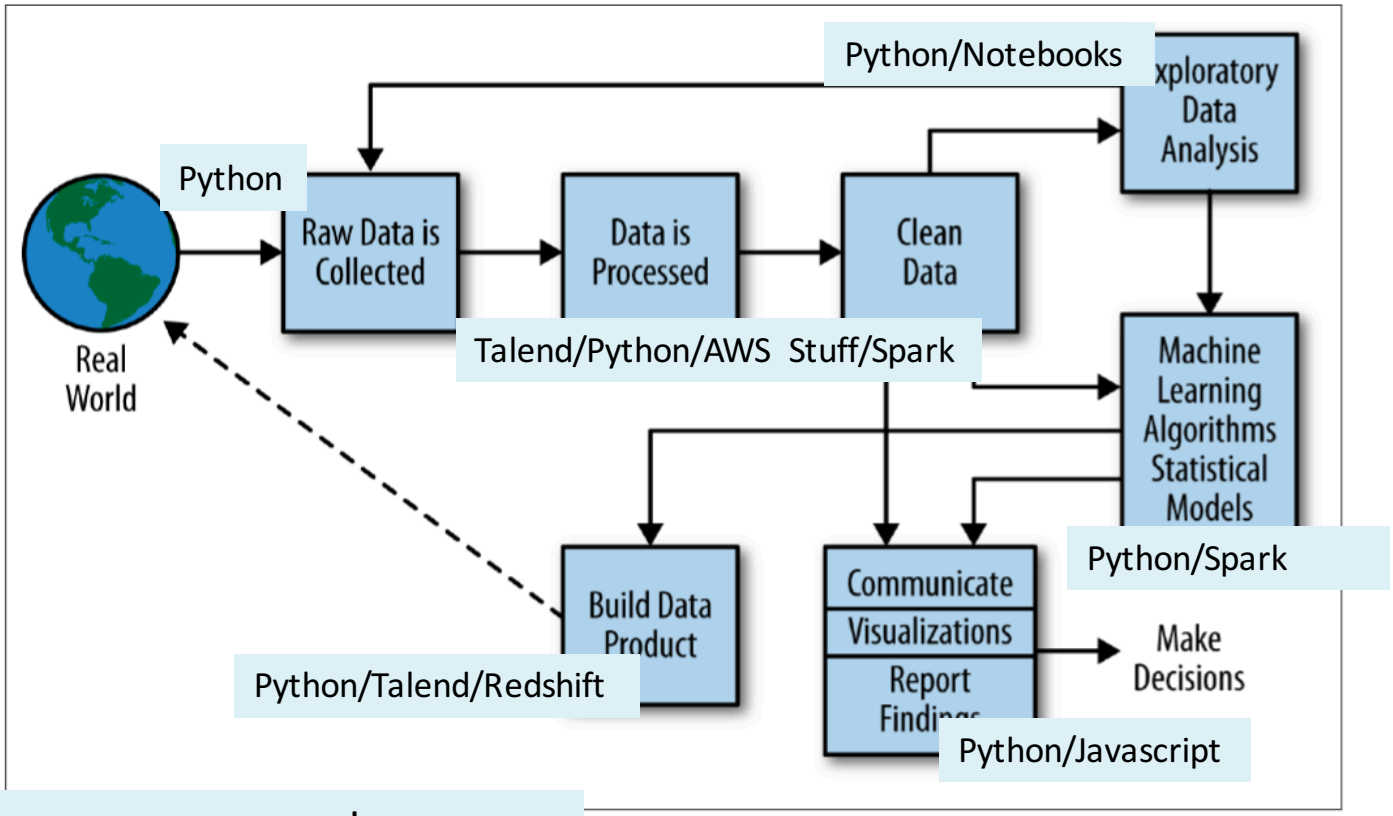


Image Credit: Rachel Schutt & Cathy O'Neil in Doing Data Science: Straight Talk From the Frontline


Data Science

So what about **Python**?

So what about **Python**?



Python can be used during all stages of the data science process

Python reduces development time

Python can perform efficiently through c/c++ and fortran extensions

Python has a LARGE community

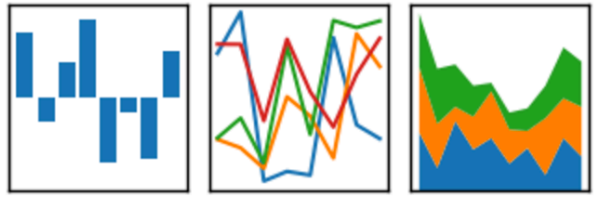
At Teem we use several technologies at each step but Python is a primary tool

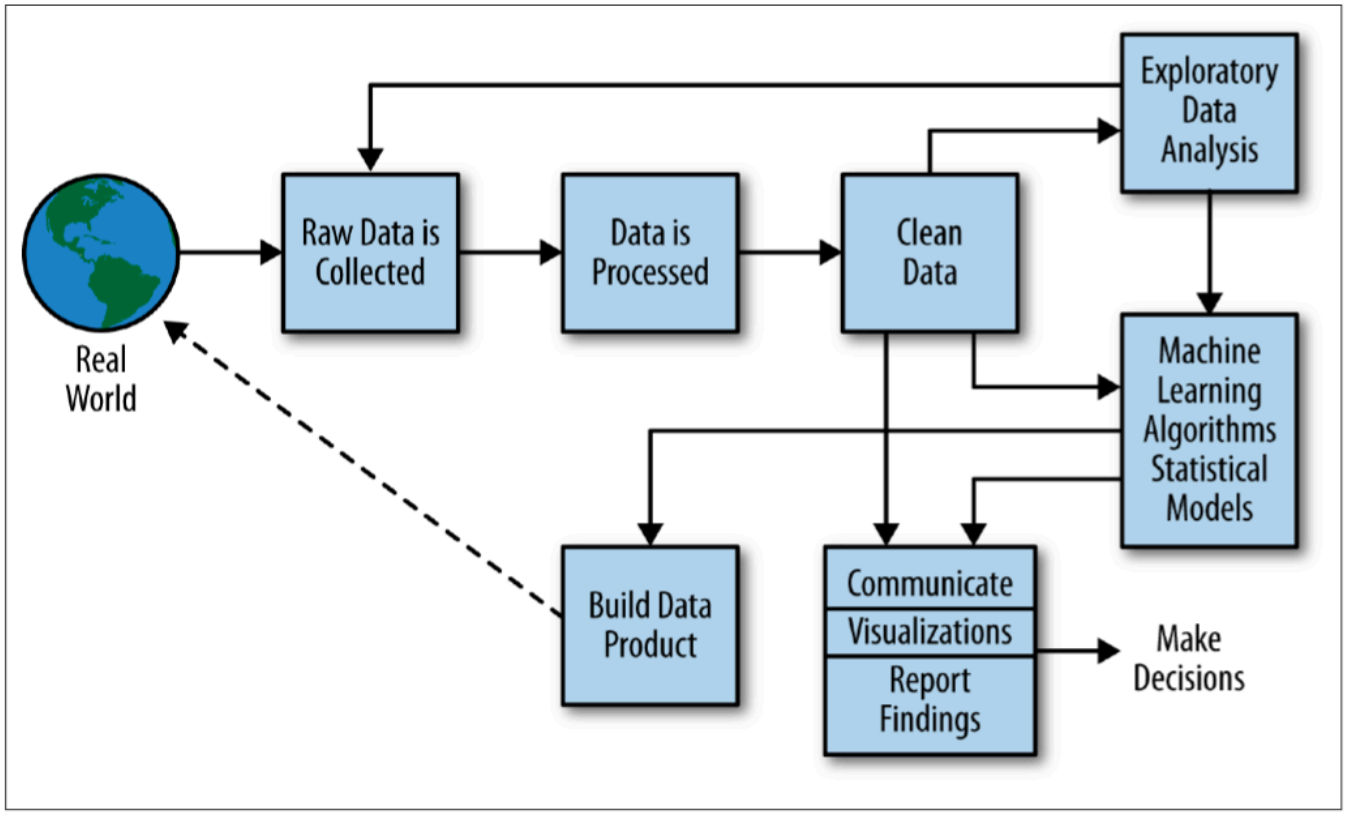
Ok, So what about this **Pandas** thing?

The **Pandas** thing

pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$



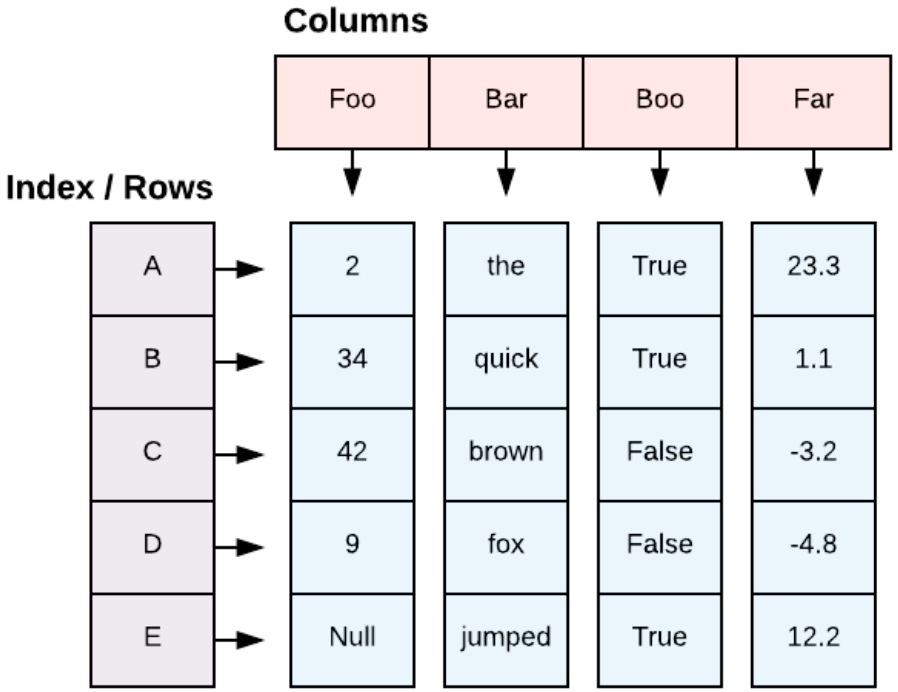


Pandas is a Python library that adds a **DataFrame** data structure to make the data science process easier

Pandas is performant, with many of it's core functionalities exported to C

Pandas plays well with other Python modeling and visualization libraries

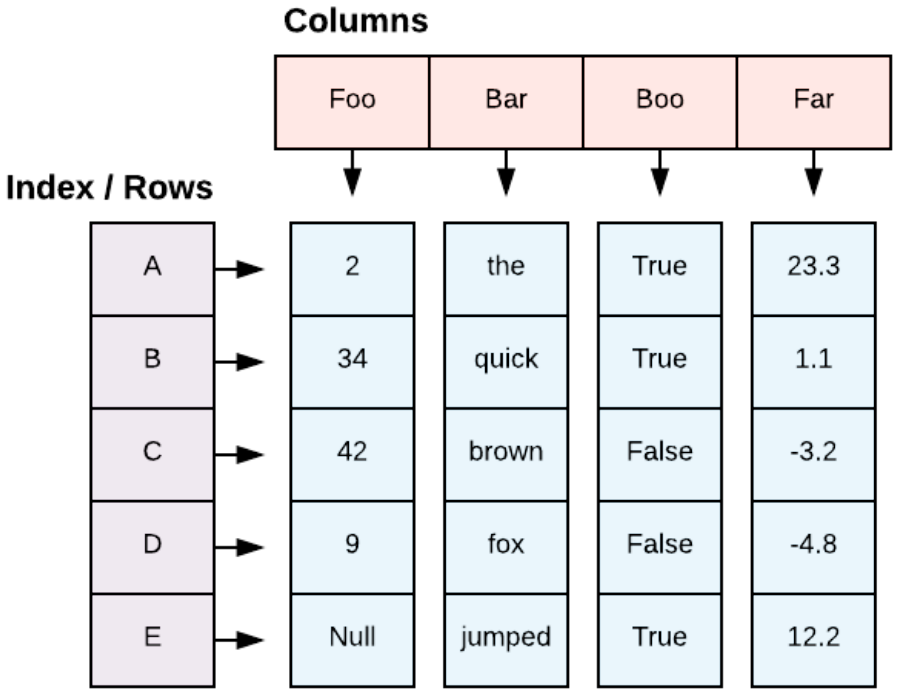
The **DataFrame**



DataFrame is the primary data structure in the Pandas library

Made popular in R, many languages are now adopting a similar data structure including Spark

The **DataFrame**

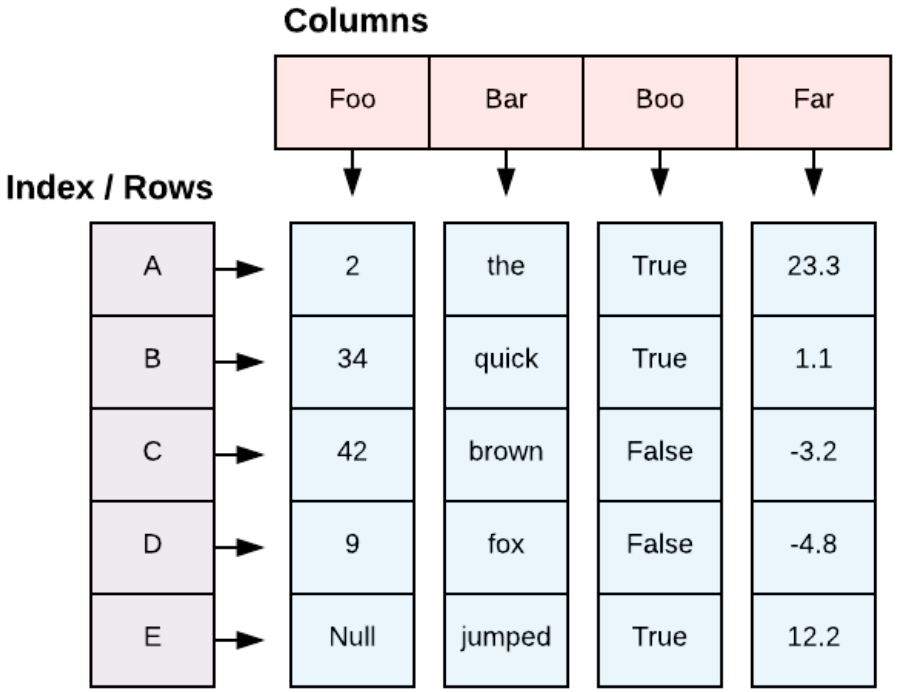


DataFrames are a tabular data structure with rows and columns and metadata about them

- Matrix like
- Each column can have different type
- Row and column metadata
- Size mutable: insert and delete columns and rows

Credit: Wes McKinney, PyCon 2012

The **DataFrame**



A Pandas DataFrame *can* be thought of as a Python dictionary of numpy arrays

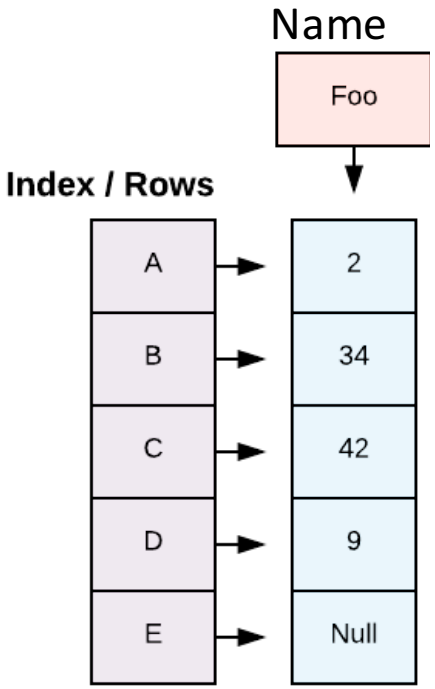
```
dataframe['Foo']
```

```
A     2.0
B    34.0
C    42.0
D     9.0
E     NaN
Name: Foo, dtype: float64
```

Aside: One column of a **DataFrame** is called a **Series**

Credit: Wes McKinney, PyCon 2012

Aside : The **Series**



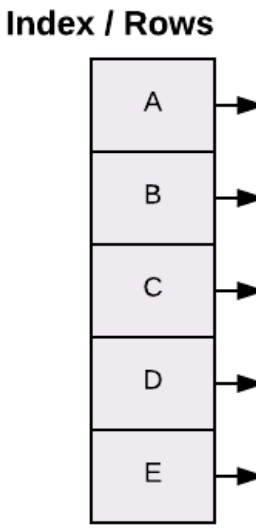

```
dataframe['Foo']
```

```
A     2.0
B    34.0
C    42.0
D     9.0
E     NaN
Name: Foo, dtype: float64
```

- Subclass of *numpy.ndarray*
- Data: any dtype
- Index labels
- Duplicate row indexes are possible (but some functions require unique)

Credit: Wes McKinney, PyCon 2012

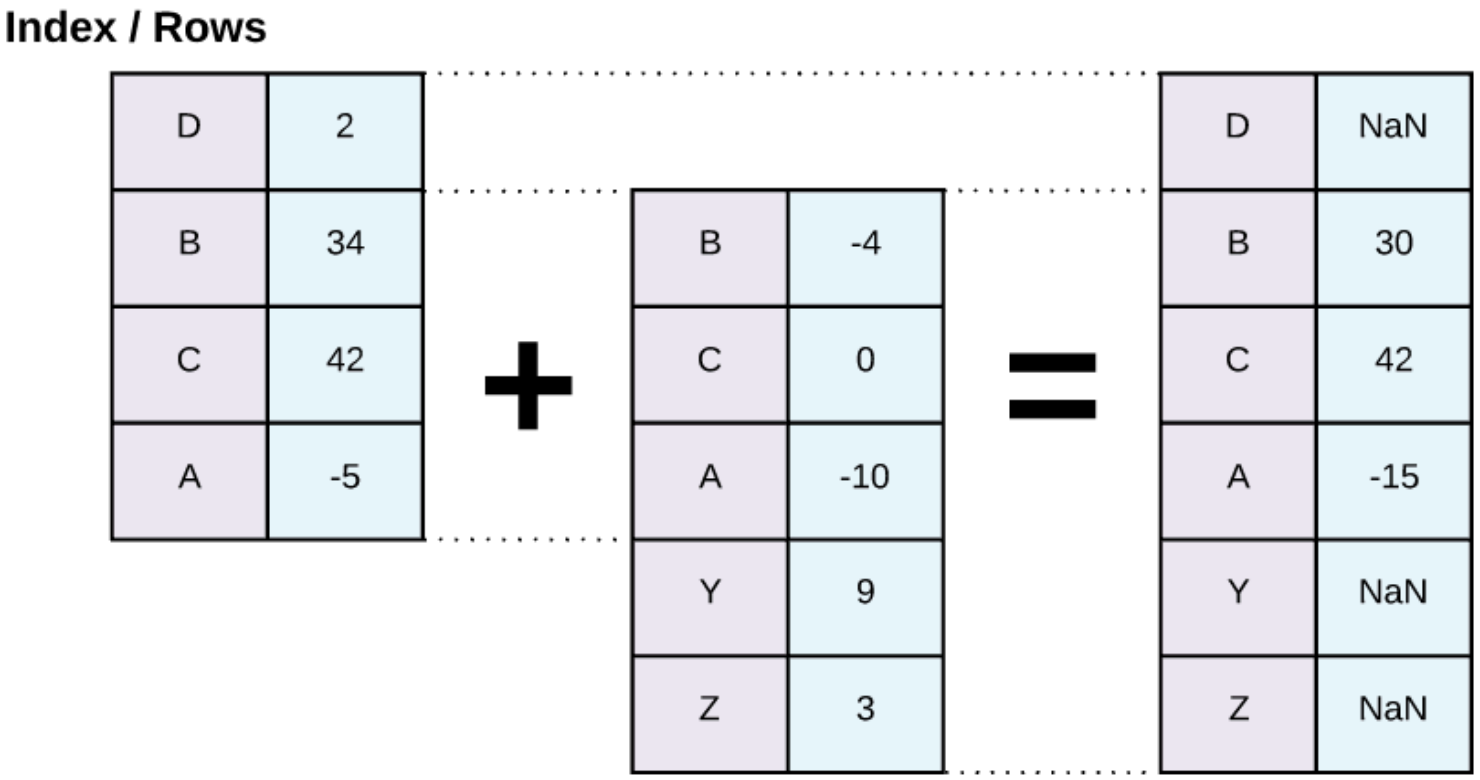
The **Index**



- Every axis has an index
- Used to implement:
  - Fast lookups
  - Data alignment and join operations
- Each axis index is a self-contained data structure

Credit: Wes McKinney, PyCon 2012

The **Index** : Data Alignment



The Index is used for data alignment when doing join actions (like + - * /)

Does an “Outer” join by default

Credit: Wes McKinney, PyCon 2012

The **Index** : Data Alignment

	foo	bar	fiz
a	-1.754	-0.967	-0.011
b	-1.003	1.487	-0.028
c	1.263	-0.330	0.327
d	-0.074	-0.846	-0.593
e	0.409	0.178	1.578

+

	foo	bar
a	-1.754	-0.967
b	-1.003	1.487
c	1.263	-0.337
d	-0.074	-0.846
e	0.409	0.178
f	-1.045	0.834
g	-0.660	-1.299

=

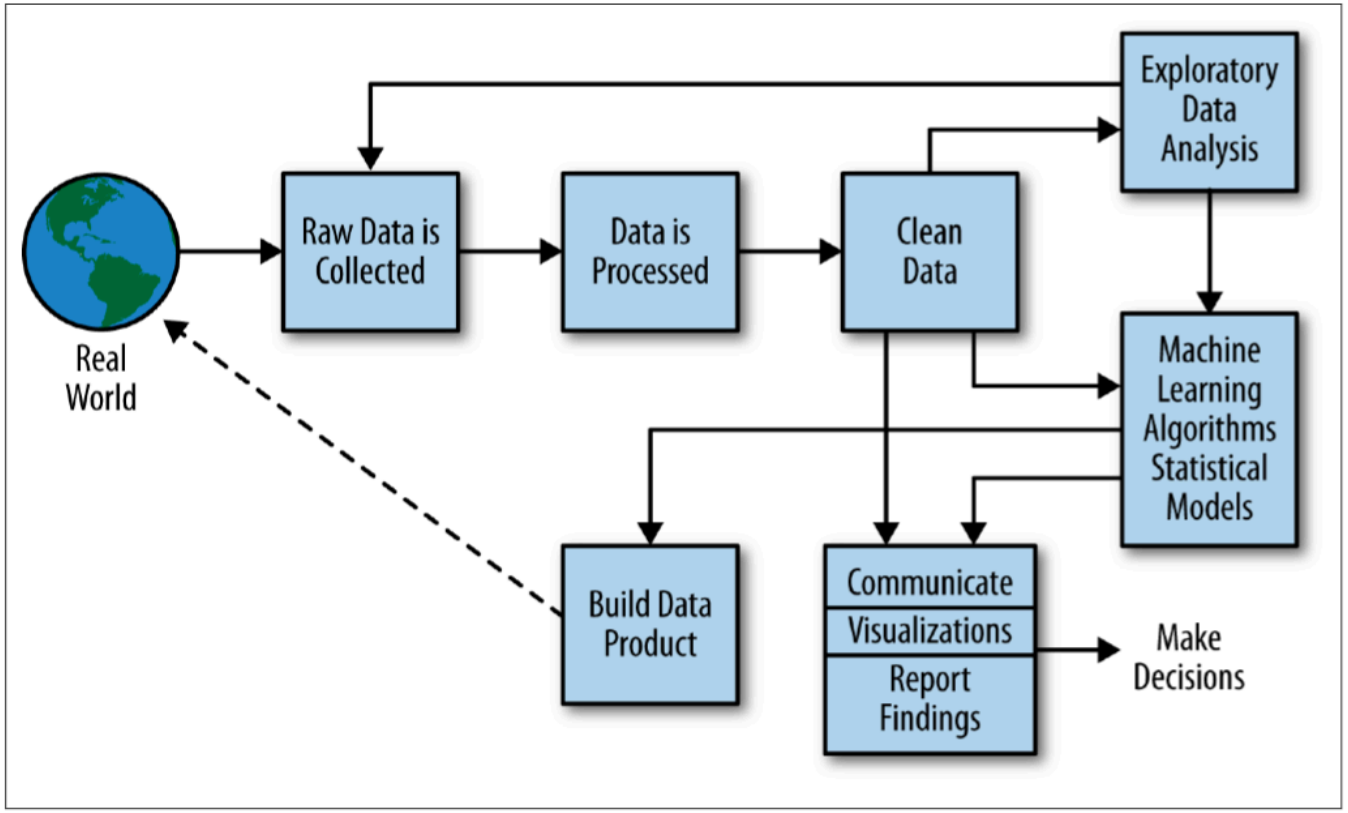
	bar	foo	fiz
a	-1.934	-3.508	NaN
b	2.974	-2.006	NaN
c	-0.667	2.526	NaN
d	-1.692	-0.148	NaN
e	0.356	0.818	NaN
f	NaN	NaN	NaN
g	NaN	NaN	NaN

DataFrame joins / align on both axes

Technically the columns are an index – which allows Pandas to have an N-dim data structure

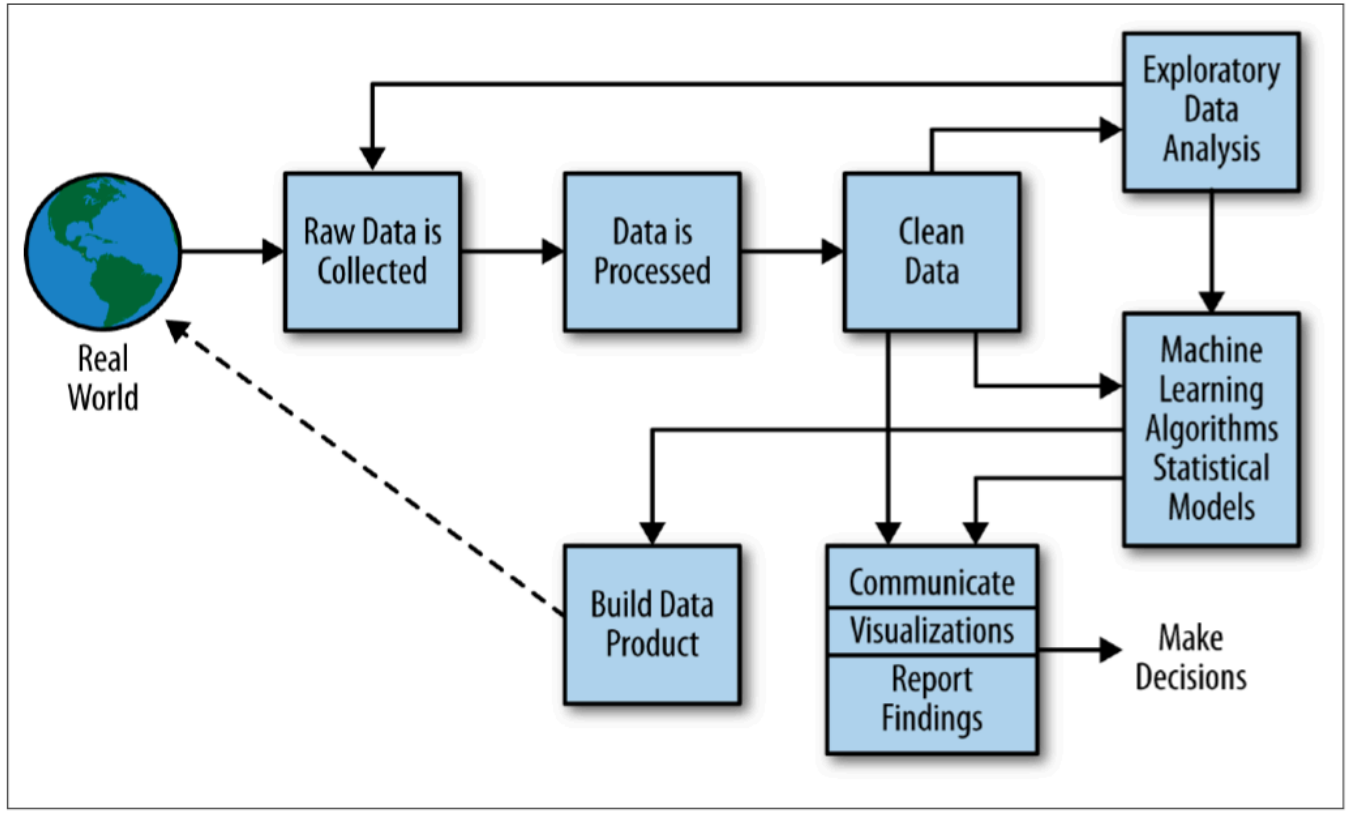
Credit: Wes McKinney, PyCon 2012

Data Science with Pandas



+

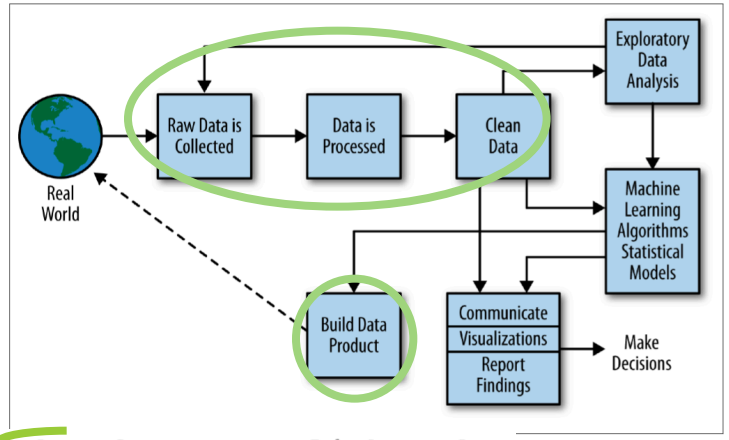
Data Science with Pandas



Pandas has tools which can help across all steps of data science

- IO and Format Conversions
- Descriptive Info
- Mutability
- Filtering
- Flexible Accessing
- Handling Missing Values
- Statistics and Math
- Groupby Aggregation
- Data Alignment
- Data Visualization
- Date Time Handling
- Library Compatibility

IO and Format Conversion



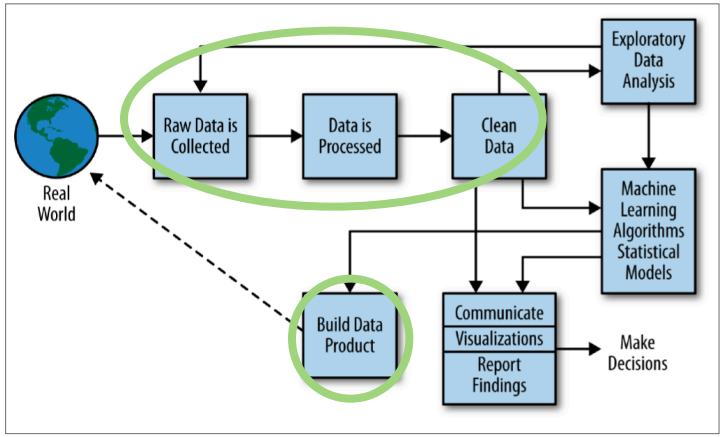
These will construct a Pandas DataFrame from the specified storage format

```
pandas.read_clipboard
      .read_csv
      .read_excel
      .read_fwf
      .read_gbq
      .read_hdf
      .read_html
      .read_json
      .read_msgpack
      .read_pickle
      .read_sas
      .read_sql
      .read_sql_query
      .read_sql_table
      .read_stata
      .read_table
```

These will output a Pandas DataFrame to a specified format

```
dataframe.to_clipboard
         .to_csv
         .to_dense
         .to_dict
         .to_excel
         .to_gbq
         .to_hdf
         .to_html
         .to_json
         .to_latex
         .to_msgpack
         .to_panel
         .to_period
         .to_pickle
         .to_records
         .to_sparse
         .to_sql
         .to_stata
         .to_string
         .to_timestamp
         .to_wide
         .to_xarray
```

IO and Format Conversion



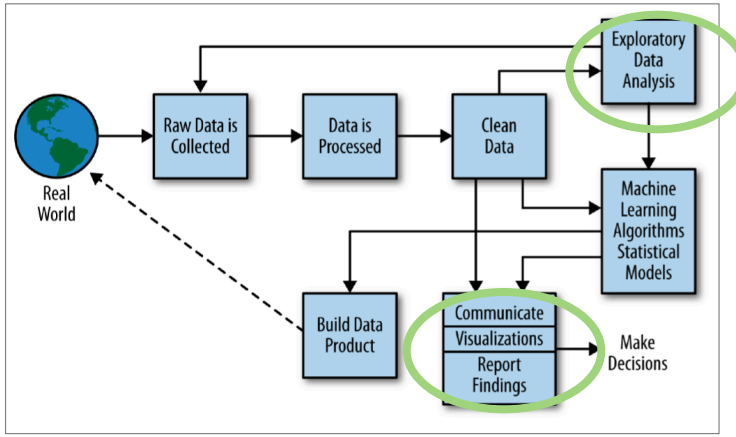

```
url = (
    'https://raw.github.com/pandas-dev/'
    'pandas/master/pandas/tests/data/tips.csv'
)
df = pd.read_csv(url)
```

```
fn = 'tips.jsoin'
df.to_json(fn)

with open(fn) as f:
    print(f.read()[:500] + '...')
```

```
{"total_bill":{"0":16.99,"1":10.34,"2":21.01,"3":23.68,"4":24.5
9,"5":25.29,"6":8.77,"7":26.88,"8":15.04,"9":14.78,"10":10.27,"11":
35.26,"12":15.42,"13":18.43,"14":14.83,"15":21.58,"16":10.33,"17":1
6.29,"18":16.97,"19":20.65,"20":17.92,"21":20.29,"22":15.77,"23":3
9.42,"24":19.82,"25":17.81,"26":13.37,"27":12.69,"28":21.7,"29":19.
65,"30":9.55,"31":18.35,"32":15.06,"33":20.69,"34":17.78,"35":24.0
6,"36":16.31,"37":16.93,"38":18.69,"39":31.27,"40":16.04,"41":17.4
6,"42":13.94,"43":9.68,"44":30.4,"45":...
```

Descriptive Tools




```
df.head()
```

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 244 entries, 0 to 243
Data columns (total 7 columns):
total_bill    244 non-null float64
tip           244 non-null float64
sex           244 non-null object
smoker        244 non-null object
day           244 non-null object
time          244 non-null object
size          244 non-null int64
dtypes: float64(2), int64(1), object(4)
memory usage: 13.4+ KB
```

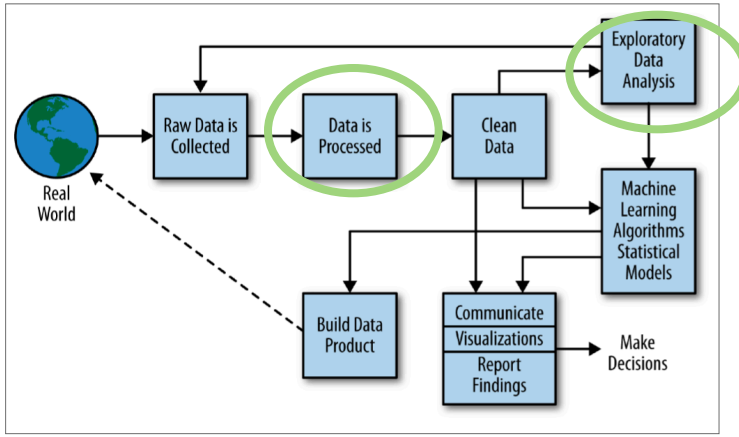
```
df.describe()
```

	total_bill	tip	size
count	244.000000	244.000000	244.000000
mean	19.785943	2.998279	2.569672
std	8.902412	1.383638	0.951100
min	3.070000	1.000000	1.000000
25%	13.347500	2.000000	2.000000
50%	17.795000	2.900000	2.000000
75%	24.127500	3.562500	3.000000
max	50.810000	10.000000	6.000000

```
df.describe(include=['object'])
```

	sex	smoker	day	time
count	244	244	244	244
unique	2	2	4	2
top	Male	No	Sat	Dinner
freq	157	151	87	176

Mutability




```
df['tip_percent'] = df['tip'] / df['total_bill']
df['totally_dumb_column'] = np.random.random(df.shape[0])
```

```
df.head()
```

	total_bill	tip	sex	smoker	day	time	size	tip_percent	totally_dumb_column
0	16.99	1.01	Female	No	Sun	Dinner	2	0.059447	0.465515
1	10.34	1.66	Male	No	Sun	Dinner	3	0.160542	0.743808
2	21.01	3.50	Male	No	Sun	Dinner	3	0.166587	0.847701
3	23.68	3.31	Male	No	Sun	Dinner	2	0.139780	0.354460
4	24.59	3.61	Female	No	Sun	Dinner	4	0.146808	0.776685

```
del df['totally_dumb_column']
```

```
list(df.columns)
```

```
['total_bill', 'tip', 'sex', 'smoker', 'day', 'time', 'size', 'tip_percen
t']
```

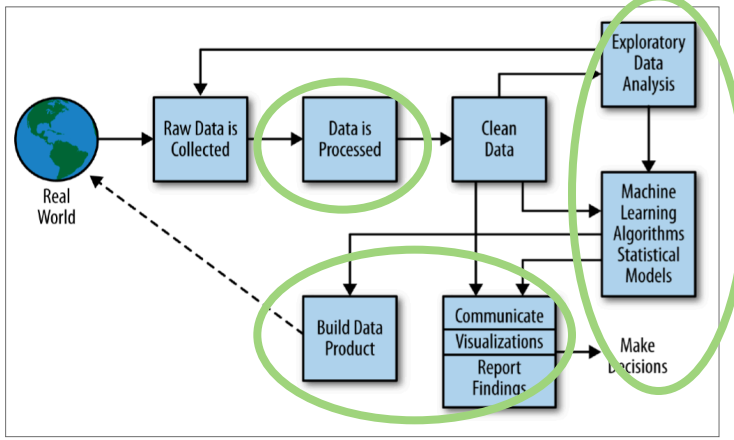
Filtering

```
mask = (df['sex'] == 'Male') & (df['total_bill'] > 10.00)
df[mask].head(3)
```

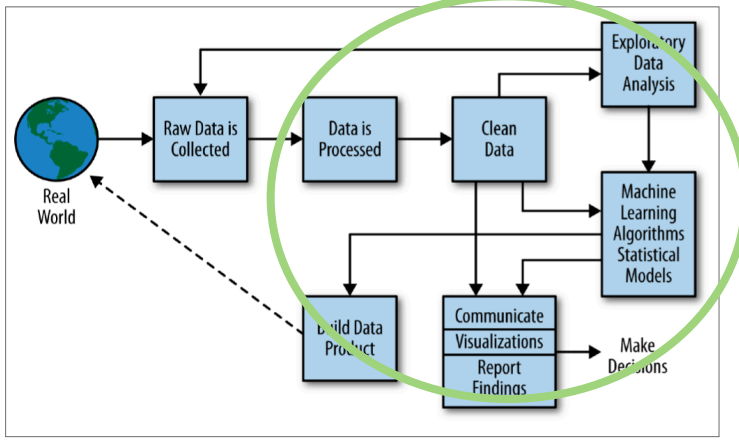
	total_bill	tip	sex	smoker	day	time	size	tip_percent
1	10.34	1.66	Male	No	Sun	Dinner	3	0.160542
2	21.01	3.50	Male	No	Sun	Dinner	3	0.166587
3	23.68	3.31	Male	No	Sun	Dinner	2	0.139780

```
df[mask].describe()
```

	total_bill	tip	size	tip_percent
count	146.000000	146.000000	146.000000	146.000000
mean	21.657123	3.175205	2.684932	0.151715
std	8.940516	1.482820	0.966711	0.046967
min	10.070000	1.000000	2.000000	0.035638
25%	15.435000	2.000000	2.000000	0.119297
50%	19.465000	3.000000	2.000000	0.149856
75%	25.492500	3.880000	3.000000	0.184942
max	50.810000	10.000000	6.000000	0.291990



Flexible Accessing



[ ], **iloc**, **loc**, **ix** exist to access the data using **integer position** and **labels**

```
df['Foo'][:3]
```

```
A     2.0
B    34.0
C    42.0
```

```
df.iloc[1:4]
```

```
df.loc['B':'D']
```

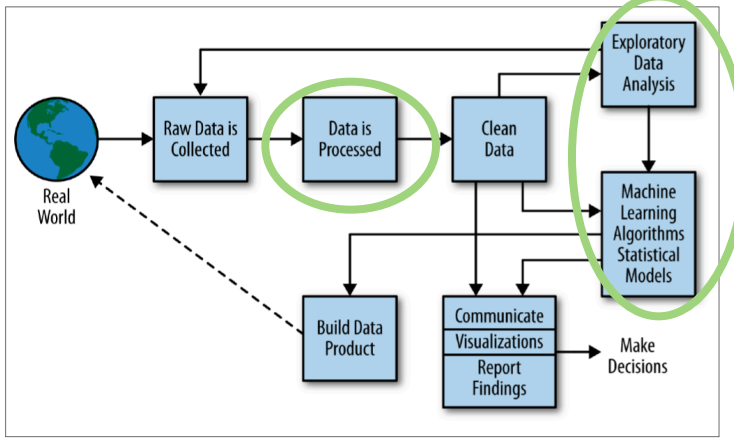
	Foo	Bar	Boo	Far
B	34.0	quick	True	1.1
C	42.0	brown	False	-3.2
D	9.0	fox	False	-4.8

```
df.ix[1:4, 'Bar']
```

```
df.ix['B':'D', 1]
```

```
B    quick
C    brown
D      fox
```

Handling Missing Values




```
df.ix[df['size'] == 2, 'size'] = None
```

```
df.head(3)
```

	total_bill	tip	sex	smoker	day	time	size	tip_percent
0	16.99	1.01	Female	No	Sun	Dinner	NaN	0.059447
1	10.34	1.66	Male	No	Sun	Dinner	3.0	0.160542
2	21.01	3.50	Male	No	Sun	Dinner	3.0	0.166587

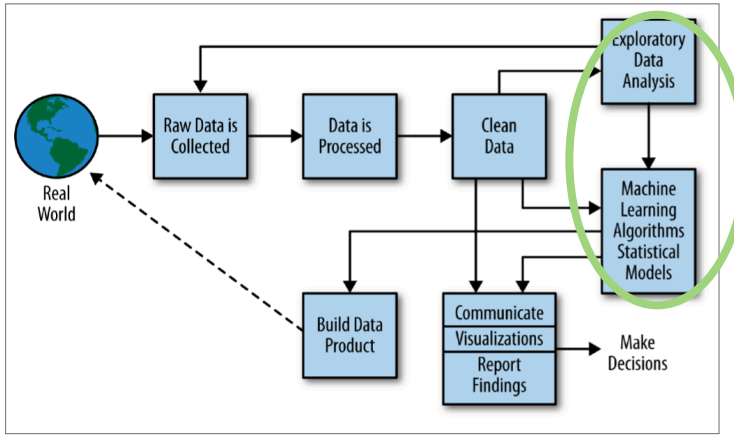
```
df.fillna(-99).head(3)
```

	total_bill	tip	sex	smoker	day	time	size	tip_percent
0	16.99	1.01	Female	No	Sun	Dinner	-99.0	0.059447
1	10.34	1.66	Male	No	Sun	Dinner	3.0	0.160542
2	21.01	3.50	Male	No	Sun	Dinner	3.0	0.166587

```
df.dropna().head(3)
```

	total_bill	tip	sex	smoker	day	time	size	tip_percent
1	10.34	1.66	Male	No	Sun	Dinner	3.0	0.160542
2	21.01	3.50	Male	No	Sun	Dinner	3.0	0.166587
4	24.59	3.61	Female	No	Sun	Dinner	4.0	0.146808

Statistics and Math




```
df.min()
```

```
total_bill          3.07
tip                    1
sex               Female
smoker                No
day                  Fri
time              Dinner
size                   1
tip_percent    0.0356381
dtype: object
```

```
df.max()
```

```
total_bill         50.81
tip                   10
sex                 Male
smoker               Yes
day                 Thur
time               Lunch
size                   6
tip_percent     0.710345
dtype: object
```

```
df.mean()
```

```
total_bill     19.785943
tip             2.998279
size            2.569672
tip_percent     0.160803
dtype: float64
```

```
df.std()
```

```
total_bill     8.902412
tip            1.383638
size           0.951100
tip_percent    0.061072
dtype: float64
```

```
df.quantile(0.3)
```

```
total_bill     14.249000
tip             2.000000
size            2.000000
tip_percent     0.137238
dtype: float64
```

```
df.cov()
```

	total_bill	tip	size	tip_percent
total_bill	79.252939	8.323502	5.065983	-0.184107
tip	8.323502	1.914455	0.643906	0.028931
size	5.065983	0.643906	0.904591	-0.008298
tip_percent	-0.184107	0.028931	-0.008298	0.003730

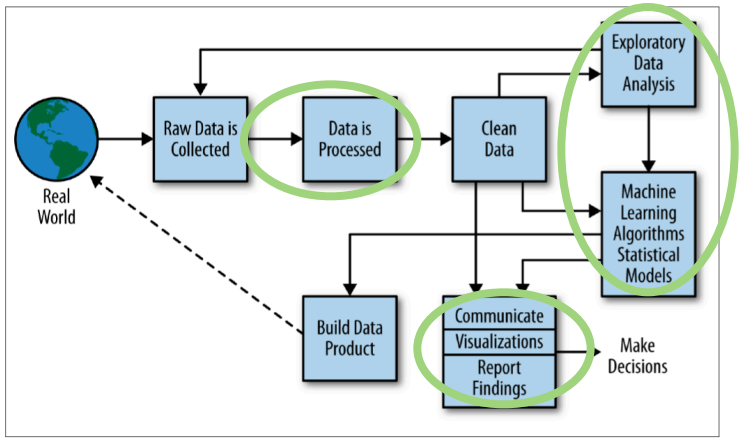
```
df.corr()
```

	total_bill	tip	size	tip_percent
total_bill	1.000000	0.675734	0.598315	-0.338624
tip	0.675734	1.000000	0.489299	0.342370
size	0.598315	0.489299	1.000000	-0.142860
tip_percent	-0.338624	0.342370	-0.142860	1.000000

.min
.max
.sum
.diff
.abs
.add
.sub
.product
.pow
.rank

.mean
.mode
.std
.skew
.quantile
.median
.corr
.cov

Groupby Aggregation



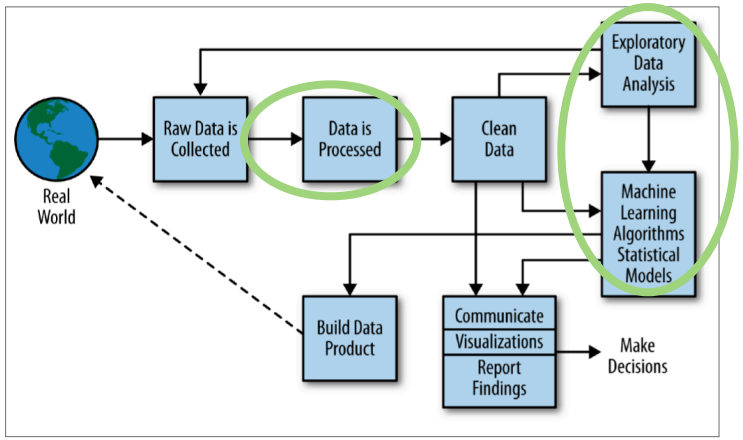

```
gb = df.groupby('sex')
gb['tip_percent'].mean()
```

```
sex
Female    0.166491
Male      0.157651
Name: tip_percent, dtype: float64
```

```
gb = df.groupby(('sex', 'day'))
gb.mean()
```

		total_bill	tip	size	tip_percent
sex	**day**				
Female	**Fri**	14.145556	2.781111	2.111111	0.199388
	Sat	19.680357	2.801786	2.250000	0.156470
	Sun	19.872222	3.367222	2.944444	0.181569
	Thur	16.715312	2.575625	2.468750	0.157525
Male	**Fri**	19.857000	2.693000	2.100000	0.143385
	Sat	20.802542	3.083898	2.644068	0.151577
	Sun	21.887241	3.220345	2.810345	0.162344
	Thur	18.714667	2.980333	2.433333	0.165276

Data Alignment




```
mean_by_day = df.groupby('day').mean()
mean_by_day.head(3)
```

	total_bill	tip	size	tip_percent
day				
Fri	17.151579	2.734737	2.105263	0.169913
Sat	20.441379	2.993103	2.517241	0.153152
Sun	21.410000	3.255132	2.842105	0.166897

```
index = [
    'Mon', 'Tue', 'Wed',
    'Thur', 'Fri', 'Sat', 'Sun'
]
data = {
    'takehome_percent': np.random.random(len(index)),
}

df_percent_takehome = pd.DataFrame(data, index=index)
df_percent_takehome
```

	takehome_percent
Mon	0.169295
Tue	0.675815
Wed	0.457629
Thur	0.482166
Fri	0.483651
Sat	0.912389
Sun	0.741331

```
df_percent_takehome.join(mean_by_day)
```

	takehome_percent	total_bill	tip	size	tip_percent
Mon	0.284653	NaN	NaN	NaN	NaN
Tue	0.232131	NaN	NaN	NaN	NaN
Wed	0.998957	NaN	NaN	NaN	NaN
Thur	0.256711	17.682742	2.771452	2.451613	0.161276
Fri	0.659085	17.151579	2.734737	2.105263	0.169913
Sat	0.365878	20.441379	2.993103	2.517241	0.153152
Sun	0.909365	21.410000	3.255132	2.842105	0.166897

Date Time Handling

```
t0 = pd.Timestamp('2010-01-01T10:30:01+06:00')
dt = pd.Timedelta(days=6 * 365)
```

```
t0
```

```
Timestamp('2010-01-01 10:30:01+0600', tz='pytz.FixedOffset(360)')
```

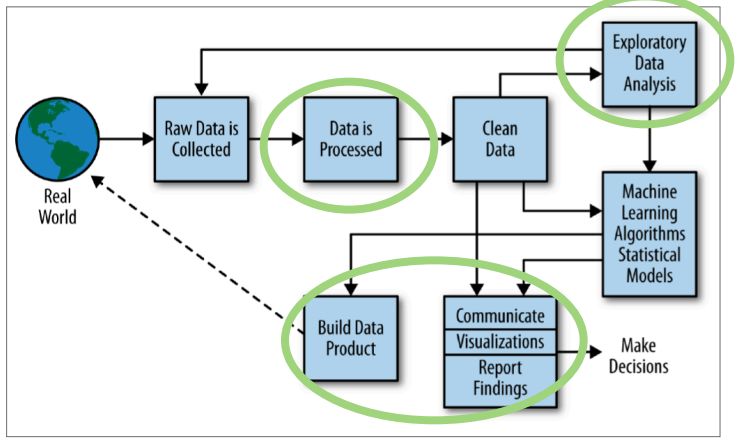
```
dt
```

```
Timedelta('2190 days 00:00:00')
```

```
index = pd.date_range(t0, t0 + dt, freq='W')
df = pd.DataFrame(
    pd.np.random.random(len(index)),
    index=index,
    columns=['rando']
)
```

```
df.head(3)
```

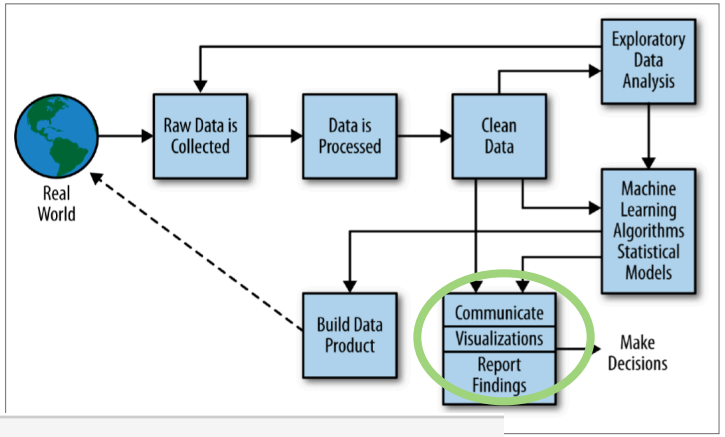
	rando
2010-01-03 10:30:01+06:00	0.416742
2010-01-10 10:30:01+06:00	0.872412
2010-01-17 10:30:01+06:00	0.183701



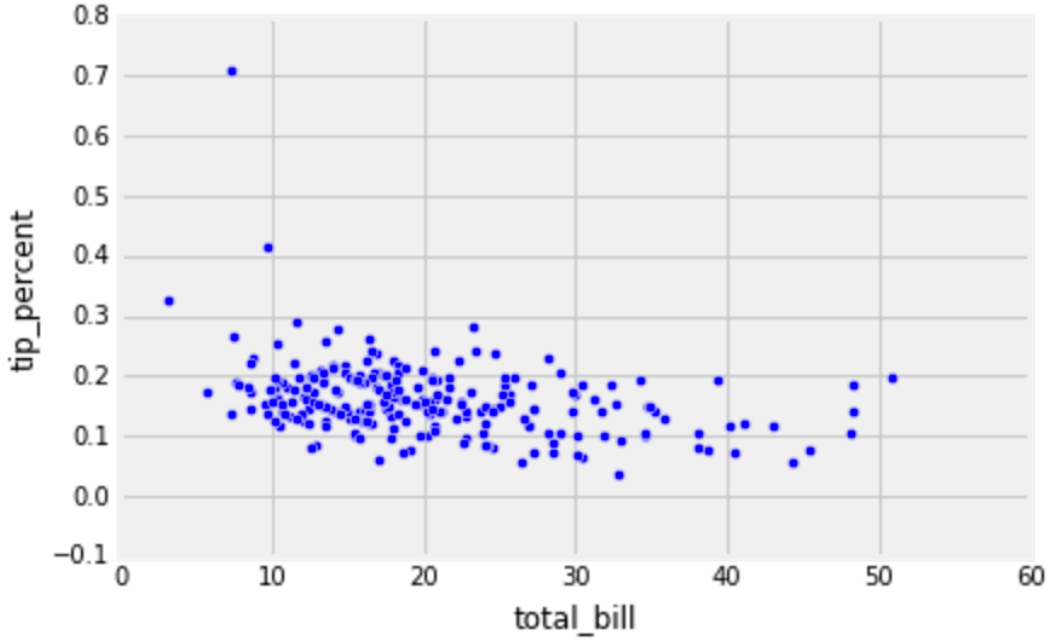

```
df.resample('QS').max().head(10)
```

	rando
2010-01-01 00:00:00+06:00	0.872412
2010-04-01 00:00:00+06:00	0.993424
2010-07-01 00:00:00+06:00	0.871019
2010-10-01 00:00:00+06:00	0.802668
2011-01-01 00:00:00+06:00	0.992361
2011-04-01 00:00:00+06:00	0.854063
2011-07-01 00:00:00+06:00	0.996082
2011-10-01 00:00:00+06:00	0.805820
2012-01-01 00:00:00+06:00	0.938627
2012-04-01 00:00:00+06:00	0.824218

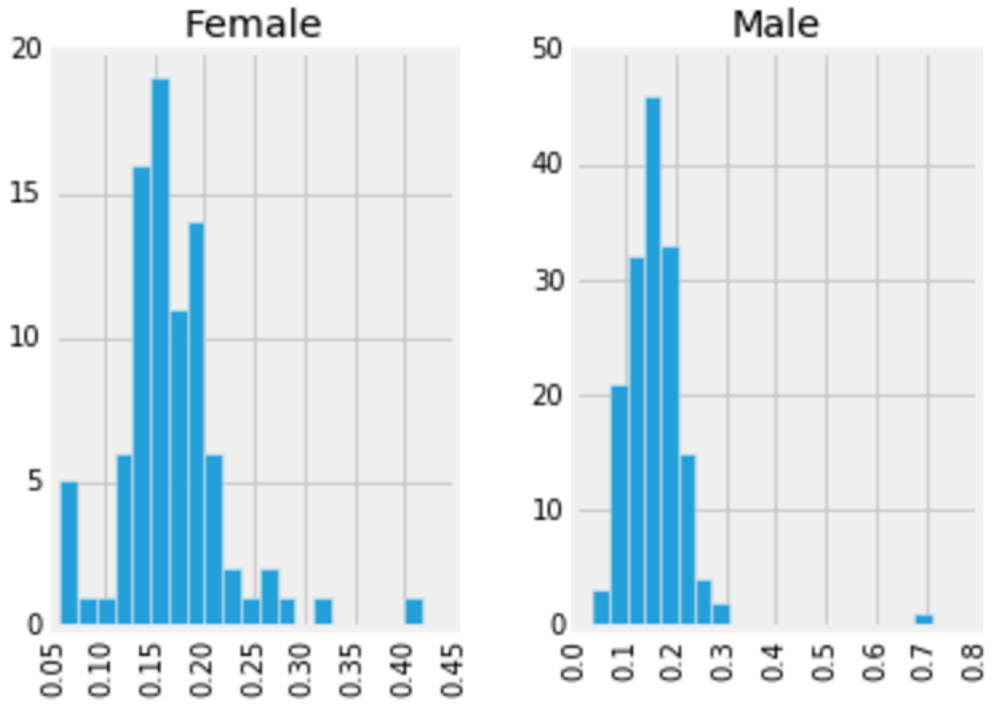
Data Visualization




```
ax = df.plot.scatter('total_bill', 'tip_percent')
```

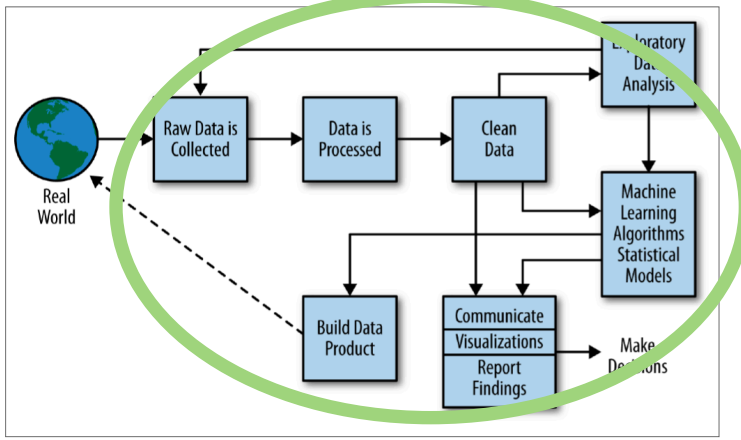



```
ax = df.hist('tip_percent', bins=20, by='sex')
```



For more check out http://pandas.pydata.org/pandas-docs/version/0.18.1/visualization.html

Library Compatibility



HDF5 for Python

SQLAlchemy

matplotlib

SciPy.org

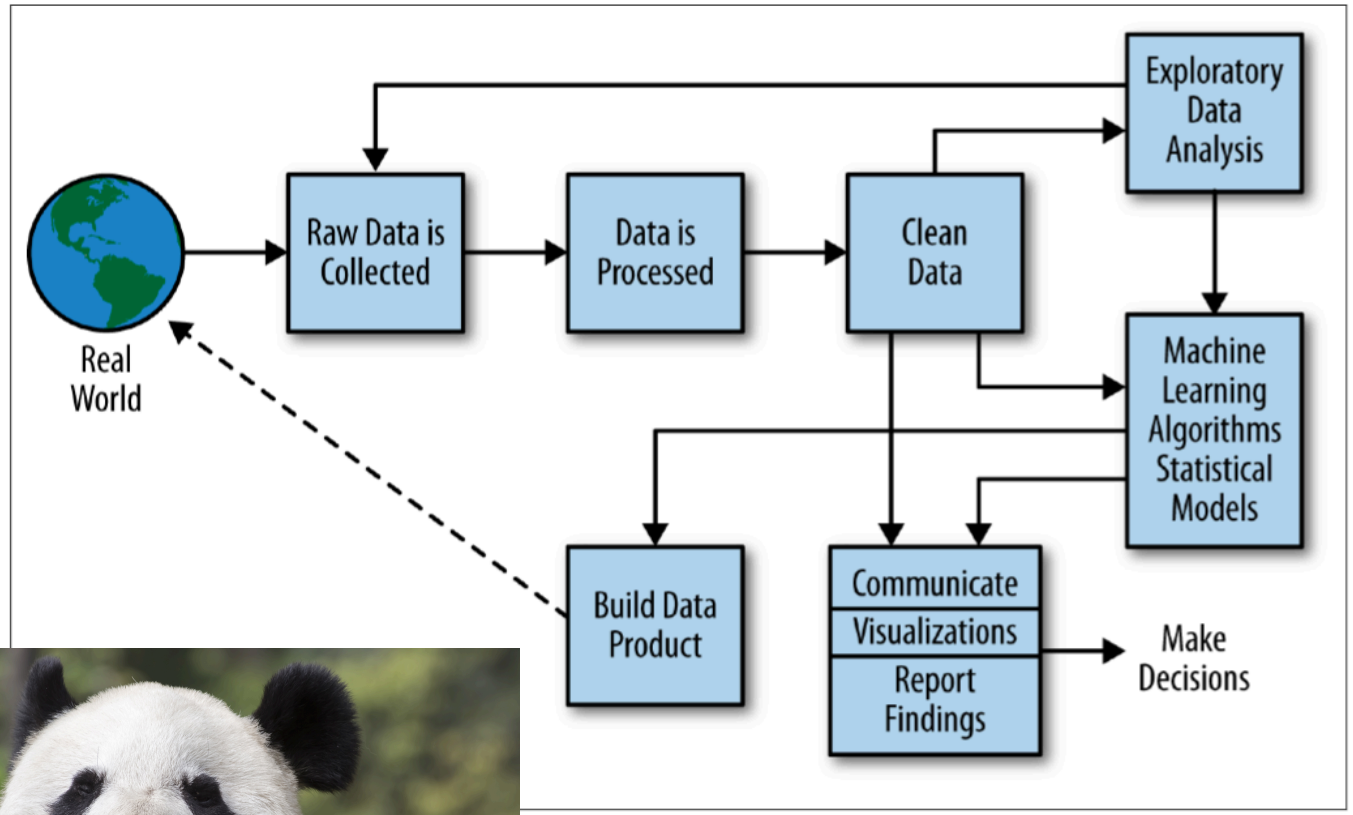
www.python-excel.org

Excel

scikit learn

theano

Data Science with Pandas

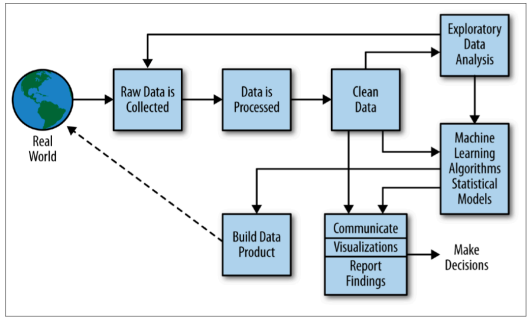


Pandas has tools which can help across all steps of data science

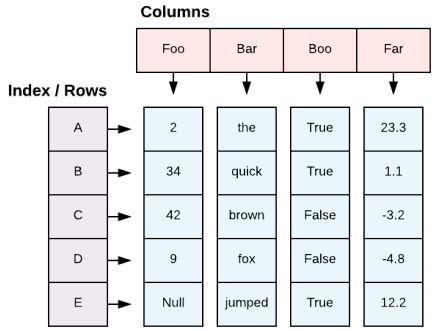
- IO and Format Conversions
- Descriptive Info
- Mutability
- Filtering
- Flexible Accessing
- Handling Missing Values
- Statistics and Math
- Groupby Aggregation
- Data Alignment
- Data Visualization
- Date Time Handling
- Library Compatibility

In Summary…

Data Science roughly describes the multi-disciplinary process of using data to answering questions about the world around us



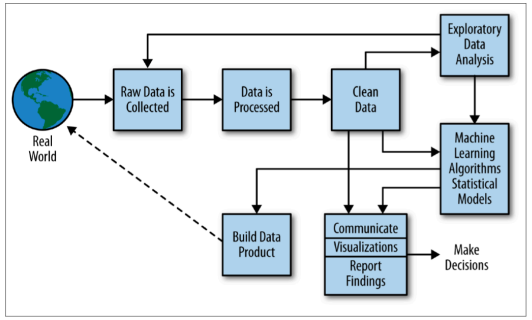
Python is a powerful language for data science and the library Pandas provides a powerful **DataFrame** data structure which aids data science's quest to answer questions



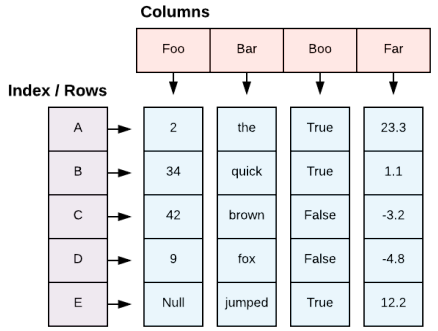
gregersen.dylan@gmail.com
astrodsg.github.io

Thank You!

Data Science roughly describes the multi-disciplinary process of using data to answering questions about the world around us



Python is a powerful language for data science and the library Pandas provides a powerful **DataFrame** data structure which aids data science's quest to answer questions



gregersen.dylan@gmail.com
astrodsg.github.io

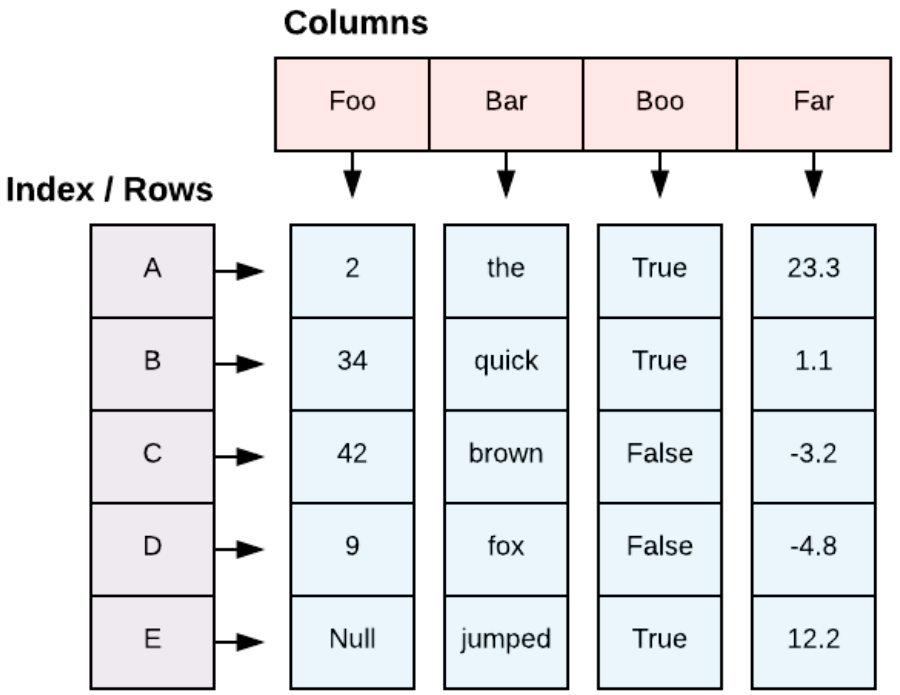
Extra Slides this way…

Links and References

Doing Data Science：Straight Talk From the Frontline
http://pandas.pydata.org/pandas-docs/stable/tutorials.html
https://pythonprogramming.net/python-pandas-data-analysis/
https://www.youtube.com/watch?v=yzIMircGU5I&list=PL5-da3qGB5ICCsgW1MxlZ0Hq8LL5U3u9y
https://github.com/brandon-rhodes/pycon-pandas-tutorial
http://pandas.pydata.org/talks.html
https://github.com/pandas-dev/pandas/blob/master/doc/cheatsheet/Pandas_Cheat_Sheet.pdf

Creating a DataFrame

The **DataFrame** : Creation



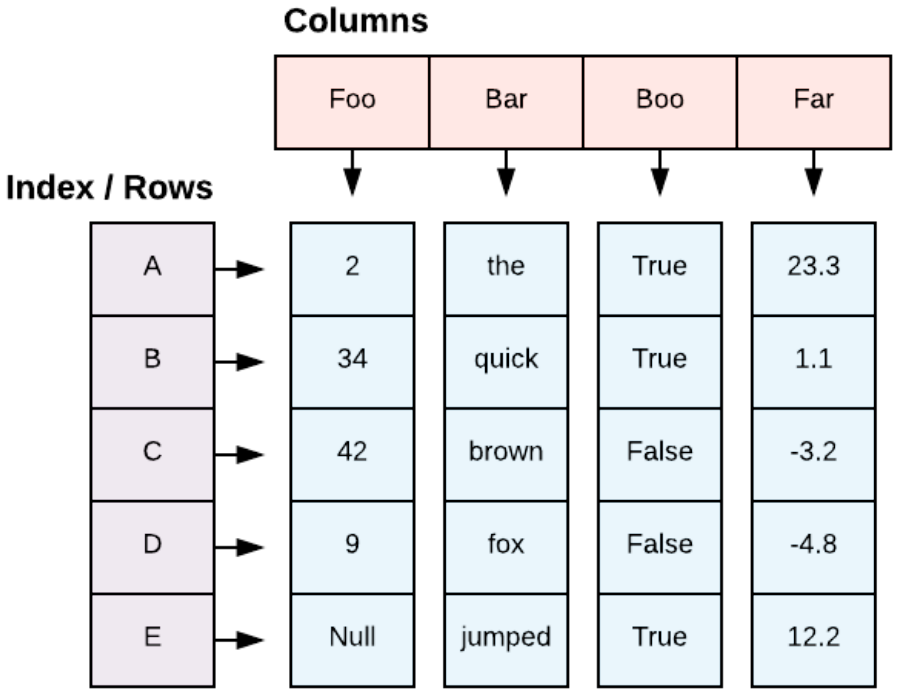

```
import pandas as pd

arrays = [
    (2, 'the', True, 23.3),
    (34, 'quick', True, 1.1),
    (42, 'brown', False, -3.2),
    (9, 'fox', False, -4.8),
    (None, 'jumped', True, 12.2),
]
columns = ['Foo', 'Bar', 'Boo', 'Far']
index = ['A', 'B', 'C', 'D', 'E']

pd.DataFrame(arrays, columns=columns, index=index)
```

	Foo	Bar	Boo	Far
A	2.0	the	True	23.3
B	34.0	quick	True	1.1
C	42.0	brown	False	-3.2
D	9.0	fox	False	-4.8
E	NaN	jumped	True	12.2

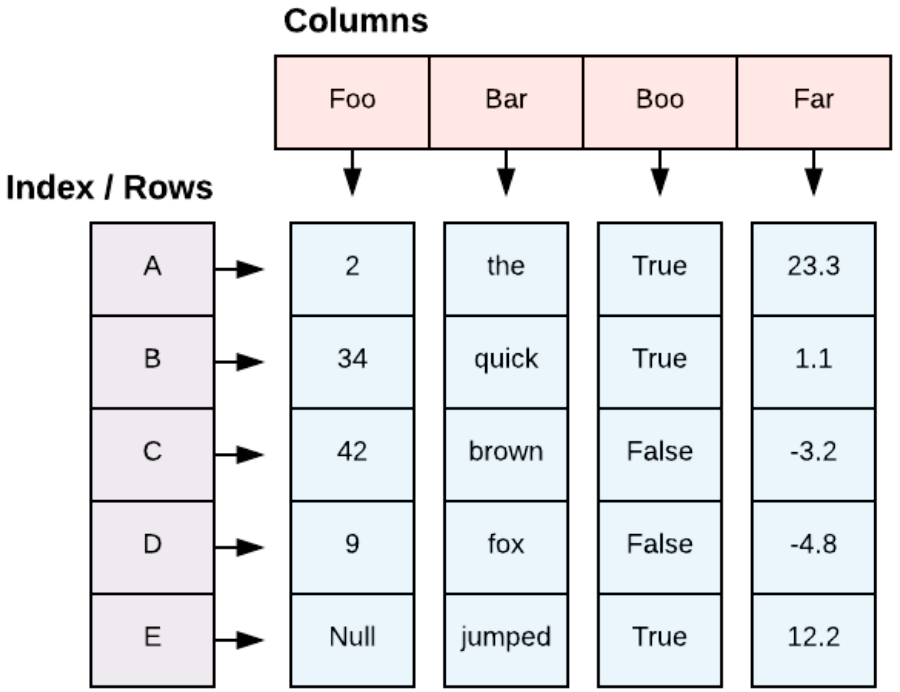
The **DataFrame** : Creation




```
data = {
    'Foo': [2, 34, 42, 9, None],
    'Bar': ['the', 'quick', 'brown', 'fox', 'jumped'],
    'Boo': [True, True, False, False, True],
    'Far': [23.3, 1.1, -3.2, -4.8, 12.2],
}
index = ['A', 'B', 'C', 'D', 'E']
pd.DataFrame(data, index=index)
```

	Bar	Boo	Far	Foo
A	the	True	23.3	2.0
B	quick	True	1.1	34.0
C	brown	False	-3.2	42.0
D	fox	False	-4.8	9.0
E	jumped	True	12.2	NaN

The **DataFrame** : Creation

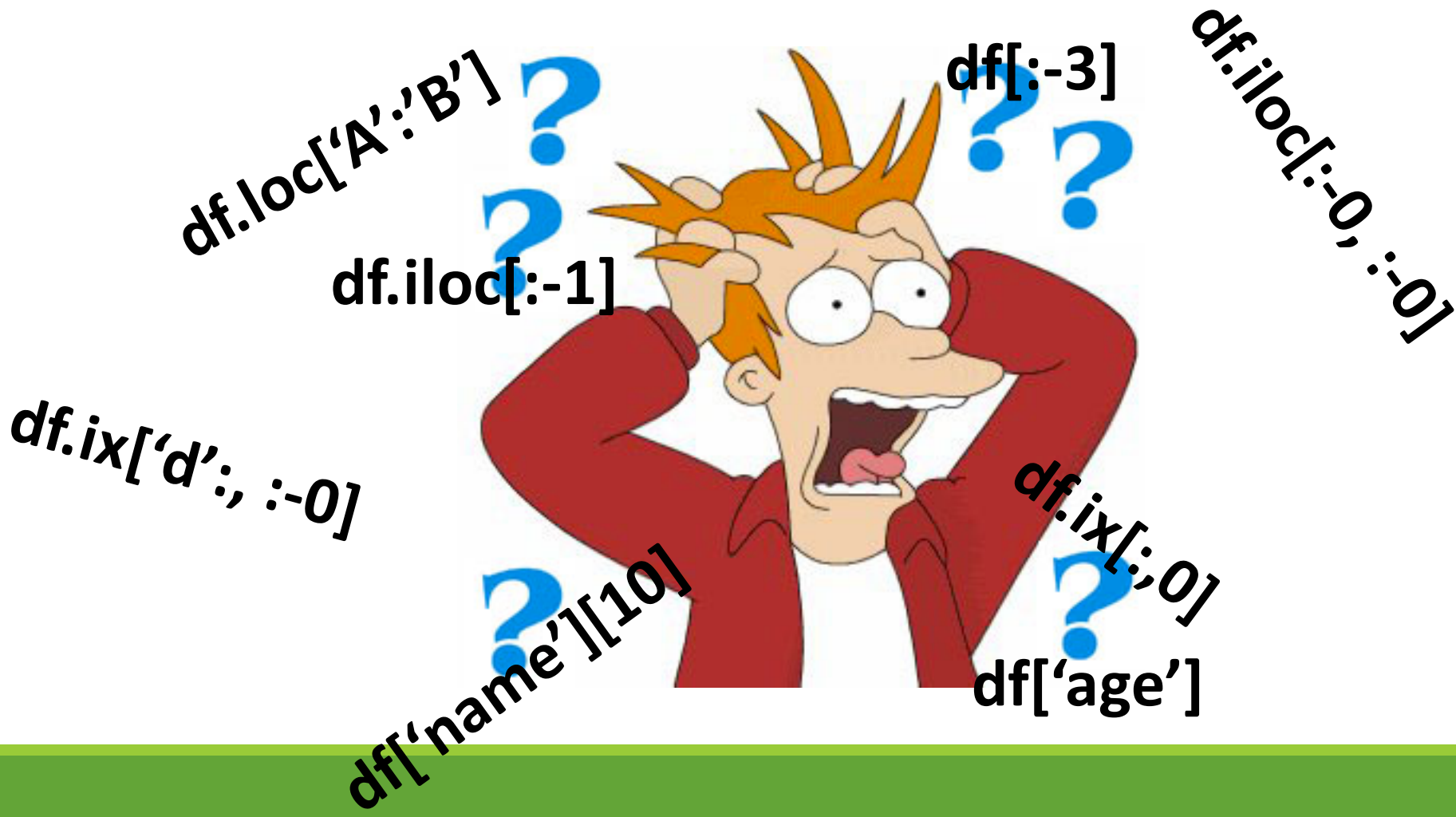


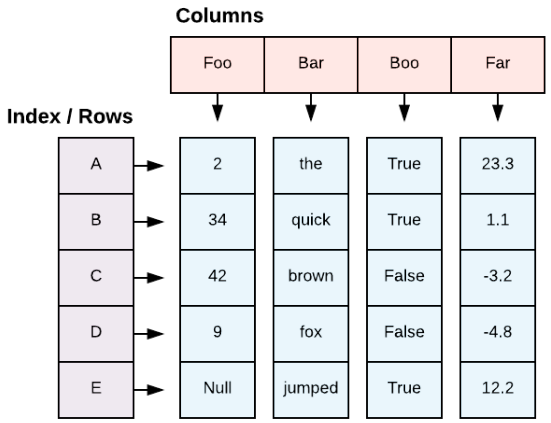

```
import io
csv = io.StringIO("""
,Bar,Boo,Far,Foo
A,the,True,23.3,2.0
B,quick,True,1.1,34.0
C,brown,False,-3.2,42.0
D,fox,False,-4.8,9.0
E,jumped,True,12.2,
""")
pd.read_csv(csv, index_col=0)
```

	Bar	Boo	Far	Foo
A	the	True	23.3	2.0
B	quick	True	1.1	34.0
C	brown	False	-3.2	42.0
D	fox	False	-4.8	9.0
E	jumped	True	12.2	NaN

Accessing a DataFrame

The **DataFrame** : Access Methods





The DataFrame : []

Standard getitem method for accessing the data frame

Pandas does a lot of guessing. Either experiment or look at the source code

Below you shows accessing by a column like a dictionary

```
df['Foo']
```

```
A     2.0
B    34.0
C    42.0
D     9.0
E     NaN
```

```
df['Foo'][:3]
```

```
A     2.0
B    34.0
C    42.0
```

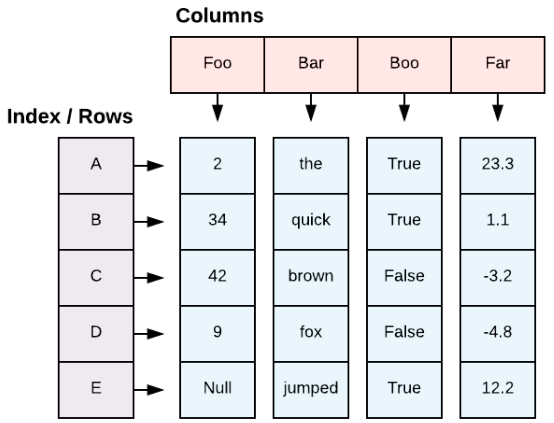
```
df['Bar'][df['Boo']]
```

```
A      the
B    quick
E   jumped
```

For help on slicing magic
https://stackoverflow.com/questions/509211/explain-pythons-slice-notation

For more on boolean array indexing see Numpy
https://docs.scipy.org/doc/numpy/user/basics.indexing.html

The DataFrame : []



Accessing the rows

Note: with **integers** it is **exclusive** and with **labels** it is **inculsive** of the end value

```
df[:2]
```

	Foo	Bar	Boo	Far
A	2.0	the	True	23.3
B	34.0	quick	True	1.1

```
df[:'C']
```

	Foo	Bar	Boo	Bar
A	2.0	the	True	23.3
B	34.0	quick	True	1.1
C	42.0	brown	False	-3.2

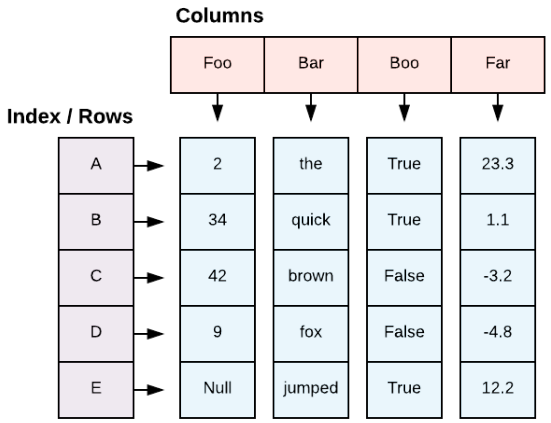
```
df[:2]['Bar']
```

```
A      the
B    quick
```

```
df[df['Boo']]
```

	Foo	Bar	Boo	Far
A	2.0	the	True	23.3
B	34.0	quick	True	1.1
E	NaN	jumped	True	12.2

The DataFrame : []

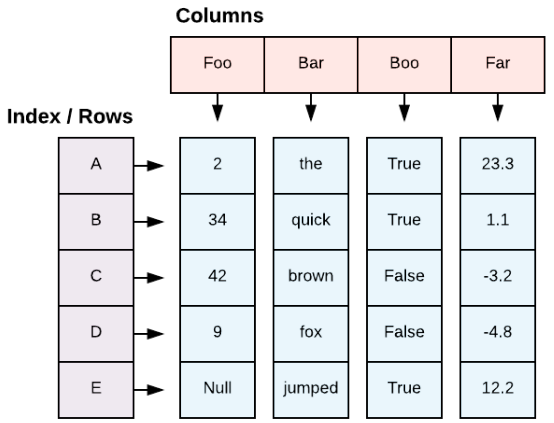


But what if you want to access the 3rd row?

You can't with [] because it's looking for **2** as a **column**

```
df[2]
```

```
KeyError
```



The DataFrame : **loc** and **iloc**

iloc is for accessing **by integer** & integers are **exclusive** of the end value

loc is for accessing **by label** & labels are **inclusive** of the end value

```
df.iloc[2]
```

```
df.loc['C']
```

```
Foo      42
Bar   brown
Boo   False
Far    -3.2
```

```
df.iloc[1:4]
```

```
df.loc['B':'D']
```

	Foo	Bar	Boo	Far
B	34.0	quick	True	1.1
C	42.0	brown	False	-3.2
D	9.0	fox	False	-4.8

```
df.iloc[1:4, 1]
```

```
df.loc['B':'D', 'Bar']
```

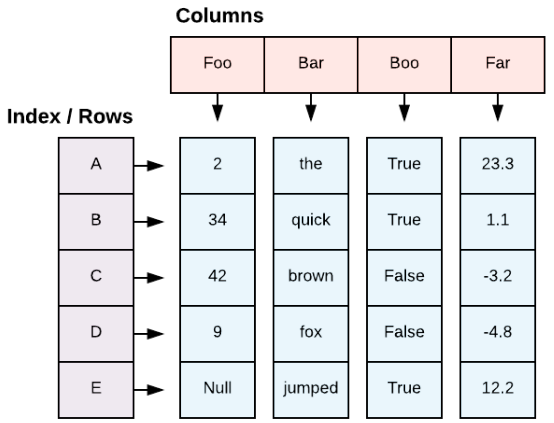
```
B    quick
C    brown
D      fox
```

```
df.iloc['B']
```

```
df.loc[1]
```

```
----------
TypeError
```

The DataFrame : ix



ix is for accessing **by integer and label**

integers are **exclusive** of the end value

labels are **inclusive** of the end value

```
df.ix[2]
```

```
df.ix['C']
```

```
Foo      42
Bar   brown
Boo   False
Far    -3.2
```

```
df.ix[1:4]
```

```
df.ix['B':'D']
```

	Foo	Bar	Boo	Far
B	34.0	quick	True	1.1
C	42.0	brown	False	-3.2
D	9.0	fox	False	-4.8

```
df.ix[1:4, 'Bar']
```

```
df.ix['B':'D', 1]
```

```
B    quick
C    brown
D      fox
```

```
df.ix[1:4, ['Bar', 'Far']]
```

	Bar	Far
B	quick	1.1
C	brown	-3.2
D	fox	-4.8

Other

Data Science

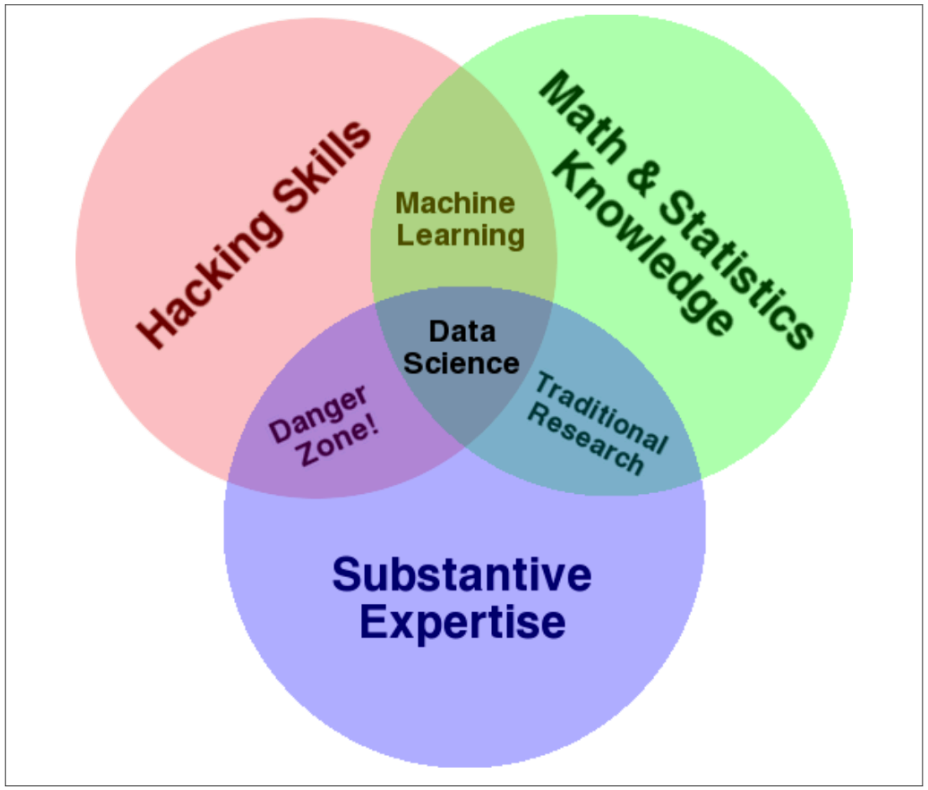


Image Credit: Drew Conway via Doing Data Science: Straight Talk From the Frontline

Library Compatibility