pandas

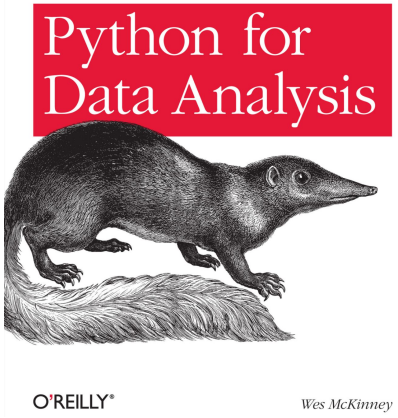
(for Python data analysis)

Frank Burkholder

Objectives

- pandas motivation, and what it is
- pandas DataFrame conceptual model
  - pandas Series
- Why learning pandas is tricky (and what to do about it)
- Making a DataFrame
- Working with the DataFrame
  - statistics
  - columns, rows, cells
  - nans (**n**ot **a** **n**umber)
- Joining DataFrames
- Aggregating in DataFrames (groupby)
- Plotting in DataFrames

pandas motivation



Wes McKinney, creator of pandas:

"...started building pandas in early 2008" while working at a quantitative investment firm.

"I had a distinct set of requirements that were not well addressed by a single tool at my disposal."

Some of those requirements:

- data structures with labeled axes that support explicit data alignment
- integrated time-series functionality
- handle non time-series and time-series data
- flexible handling of missing data
- merge and other relational operations found in popular databases (like SQL, for example)

panel **da**ta -> pandas (not the bear)

What is pandas

pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. --Wikipedia

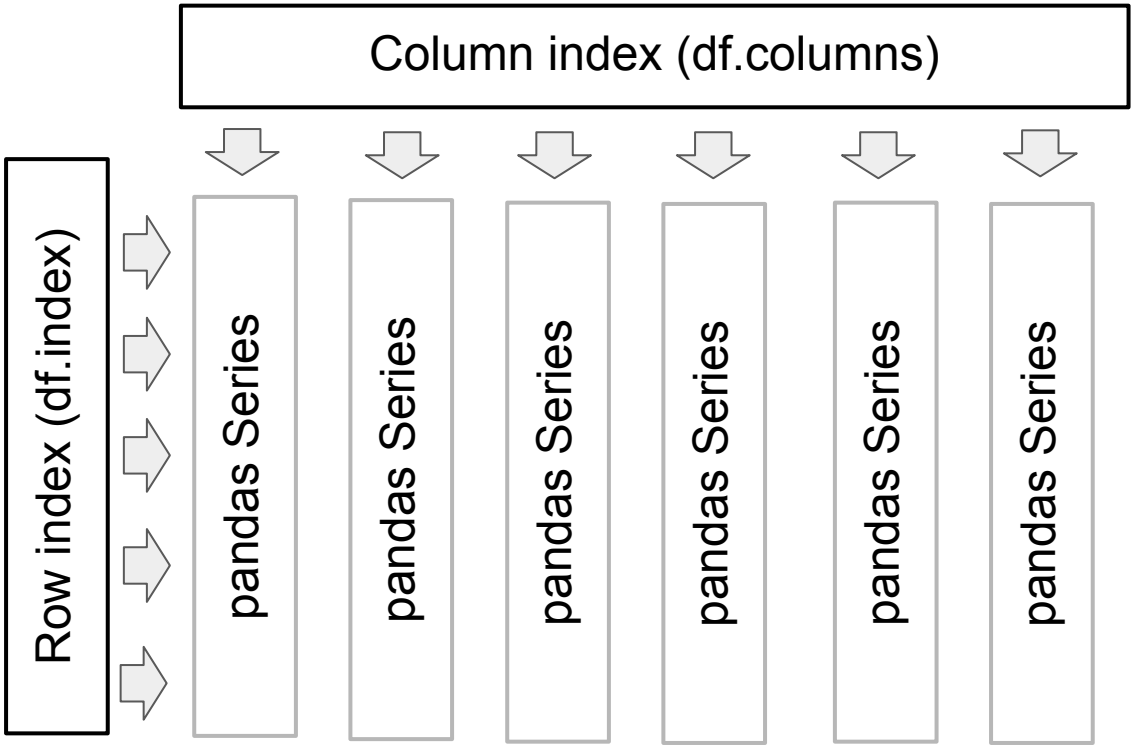
```
import pandas as pd

df = pd.read_csv('data/winequality-red.csv', delimiter=';')
df.head()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5

pandas DataFrame conceptual model

The DataFrame *object* is a two-dimensional table of data with column and row indexes (it's like a spreadsheet). The columns are pandas Series.



pandas Series

A pandas Series stores columnar data. It's an ordered, one-dimensional array of a single data type with specified indices. It's built on a one-dimensional Numpy array.

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.0	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.0	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5

```
type(df['fixed acidity'])
```

```
pandas.core.series.Series
```

```
df['fixed acidity'].iloc[0:2]
```

```
0    7.4
1    7.8
Name: fixed acidity, dtype: float64
```

```
df['fixed acidity'].values
```

```
array([7.4, 7.8, 7.8, ..., 6.3, 5.9, 6. ])
```

```
type(df['fixed acidity'].values)
```

```
numpy.ndarray
```

Why learning pandas (syntax) is tricky

Paraphrased from “Minimally Sufficient Pandas” by Ted Petrou:

- There are often multiple ways to do the same thing
- There are over 240 DataFrame attributes and methods
- Some methods are aliases (reference the same exact underlying code) of each other
- Some methods have nearly identical functionality
- There are many tutorials written by different people that show different ways to do the same thing
- There is no official document with guidelines on “the right way” to complete common tasks
- The official documentation contains some non-idiomatic code

So what we're going to do about it

- This lecture!
  - Heavily influenced by Minimally Sufficient Pandas (MSP) and your pandas cheatsheet by Mark Graph
- If you have a question about how to do something in pandas, suggest the following steps
  - 1) Look in your cheatsheet, if you can't find it there then…
  - 2) Look in the *current* Pandas documentation, if you can't find it there then...
  - 3) Google it (and be prepared for chaos)

Making a (toy) DataFrame

from pandas Series:

```
s1 = pd.Series({'Tom': 2012, 'Sara': 2008})
s2 = pd.Series({'Tom': 'Bronze', 'Mary': 'Gold'})
df = pd.DataFrame({'Year': s1, 'Medal': s2 })
df
```

	Year	Medal
Mary	NaN	Gold
Sara	2008.0	NaN
Tom	2012.0	Bronze

```
df = df.sort_index(axis=0, ascending=False)
df
```

	Year	Medal
Tom	2012.0	Bronze
Sara	2008.0	NaN
Mary	NaN	Gold

from specifying data/columns/index:

```
vals = np.random.rand(4,3).round(2) #  4 rows, 3 columns
cols = ['col 1', 'col 2', 'col 3']
idx = ['a', 'b', 'c', 'd']

df = pd.DataFrame(data=vals, columns=cols, index=idx)
df
```

	col 1	col 2	col 3
a	0.17	0.97	0.23
b	0.34	0.03	0.29
c	0.15	0.18	0.74
d	0.40	0.77	0.06

toy DataFrames are useful subjects for code exploration before you try it on your full “real” DataFrame

How you'll normally make a DataFrame

```
df = pd.read_csv('data/winequality-red.csv')
df.head(2)
```

	fixed acidity;"volatile acidity";"citric acid";"residual sugar";"chlorides";"free sulfur dioxide";"total sulfur dioxide";"density";"pH";"sulphates";"alcohol";"quality"
0	7.4;0.7;0;1.9;0.076;11;34;0.9978;3.51;0.56;9.4;5
1	7.8;0.88;0;2.6;0.098;25;67;0.9968;3.2;0.68;9.8;5

whoops!

```
df = pd.read_csv('data/winequality-red.csv', delimiter=';')
df.head(2)
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.0	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.0	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5

let's check out the pandas read_csv documentation.

Working with your entire DataFrame

```
df = pd.read_csv('data/winequality-red.csv', delimiter=';')
```

```
n = 2
df.head(2) # top n rows
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.0	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.0	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5

```
df.tail(2) # bottom n rows
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
1597	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	10.2	5
1598	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	11.0	6

```
df.shape # Gives you the number of rows and number of columns
```

```
(1599, 12)
```

Working with your entire DataFrame

```
df.columns # Gives you back a list of all of the column names.
```

```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'quality'],
      dtype='object')
```

```
df.info() # Allows you to look at the data type for each column,
          # and the number of null values.
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
fixed acidity           1599 non-null float64
volatile acidity        1599 non-null float64
citric acid             1599 non-null float64
residual sugar          1599 non-null float64
chlorides               1599 non-null float64
free sulfur dioxide     1599 non-null float64
total sulfur dioxide    1599 non-null float64
density                 1599 non-null float64
pH                      1599 non-null float64
sulphates               1599 non-null float64
alcohol                 1599 non-null float64
quality                 1599 non-null int64
dtypes: float64(11), int64(1)
memory usage: 150.0 KB
```

Working with your entire DataFrame

```
df.describe() # Gives you summary statistics for all of your numeric columns.
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	46.467792	0.996747	3.311113	0.658149	10.422983
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.895324	0.001887	0.154386	0.169507	1.065668
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.000000	0.990070	2.740000	0.330000	8.400000
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.000000	0.995600	3.210000	0.550000	9.500000
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.000000	0.996750	3.310000	0.620000	10.200000
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.000000	0.997835	3.400000	0.730000	11.100000
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.000000	1.003690	4.010000	2.000000	14.900000

Working with your entire DataFrame

```
df.corr(method='pearson').round(3)
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
fixed acidity	1.000	-0.256	0.672	0.115	0.094	-0.154	-0.113	0.668	-0.683	0.183	-0.062	0.124
volatile acidity	-0.256	1.000	-0.552	0.002	0.061	-0.011	0.076	0.022	0.235	-0.261	-0.202	-0.391
citric acid	0.672	-0.552	1.000	0.144	0.204	-0.061	0.036	0.365	-0.542	0.313	0.110	0.226
residual sugar	0.115	0.002	0.144	1.000	0.056	0.187	0.203	0.355	-0.086	0.006	0.042	0.014
chlorides	0.094	0.061	0.204	0.056	1.000	0.006	0.047	0.201	-0.265	0.371	-0.221	-0.129
free sulfur dioxide	-0.154	-0.011	-0.061	0.187	0.006	1.000	0.668	-0.022	0.070	0.052	-0.069	-0.051
total sulfur dioxide	-0.113	0.076	0.036	0.203	0.047	0.668	1.000	0.071	-0.066	0.043	-0.206	-0.185
density	0.668	0.022	0.365	0.355	0.201	-0.022	0.071	1.000	-0.342	0.149	-0.496	-0.175
pH	-0.683	0.235	-0.542	-0.086	-0.265	0.070	-0.066	-0.342	1.000	-0.197	0.206	-0.058
sulphates	0.183	-0.261	0.313	0.006	0.371	0.052	0.043	0.149	-0.197	1.000	0.094	0.251
alcohol	-0.062	-0.202	0.110	0.042	-0.221	-0.069	-0.206	-0.496	0.206	0.094	1.000	0.476
quality	0.124	-0.391	0.226	0.014	-0.129	-0.051	-0.185	-0.175	-0.058	0.251	0.476	1.000

Pearson (and Spearman and other) correlations

Working with your entire DataFrame

```
df.var()
```

```
fixed acidity              3.031416
volatile acidity           0.032062
citric acid                0.037947
residual sugar             1.987897
chlorides                  0.002215
free sulfur dioxide      109.414884
total sulfur dioxide    1082.102373
density                    0.000004
pH                         0.023835
sulphates                  0.028733
alcohol                    1.135647
quality                    0.652168
dtype: float64
```

```
np.sqrt(df.var())
```

```
fixed acidity             1.741096
volatile acidity          0.179060
citric acid               0.194801
residual sugar            1.409928
chlorides                 0.047065
free sulfur dioxide      10.460157
total sulfur dioxide     32.895324
density                   0.001887
pH                        0.154386
sulphates                 0.169507
alcohol                   1.065668
quality                   0.807569
dtype: float64
```

Easier to interpret (in same units as data)

Breakout

In the data folder accompanying this lecture, there are 3 csv (comma separated value) data files:

```
winequality-red.csv
winequality-white.csv
mass_shootings_2015.csv
```

Read the `mass_shootings_2015.csv` into a DataFrame. Then explore it using the commands demonstrated above.

Do your exploration in a Jupyter Notebook. When first looking at data, the rapid visualization and iterative capabilities of Jupyter Notebooks make them very useful.

Working with your DataFrame - selecting columns

```
df = pd.read_csv('data/winequality-red.csv', delimiter=';')
```

```
df['alcohol'] # Grabs the 'alcohol' column.
```

```
0        9.4
1        9.8
2        9.8
3        9.8
4        9.4
5        9.4
6        9.4
7       10.0
8        9.5
9       10.5
10       9.2
11      10.5
12       9.9
13       9.1
14       9.2
15       9.2
16      10.5
17       9.3
18       9.0
```

```
df['alcohol'].value_counts()
```

```
9.500000     139
9.400000     103
9.800000      78
9.200000      72
10.500000     67
```

dot notation also works in this case:

```
df.alcohol
```

```
0        9.4
1        9.8
2        9.8
3        9.8
4        9.4
5        9.4
```

but:

```
df['volatile acidity']  # Works!
df.volatile acidity     # Nope (has spaces)
```

MSP recommends the bracket method (because it always works). Dot notation also runs into issues when the column is named something like count, mean, or sum.

Working with your DF - selecting subset of columns

```
df.columns
```

```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'quality'],
      dtype='object')
```

```
columns_subset = ['fixed acidity', 'residual sugar', 'alcohol']
df_ss = df[columns_subset]
df_ss.head()
```

	fixed acidity	residual sugar	alcohol
0	7.4	1.9	9.4
1	7.8	2.6	9.8
2	7.8	2.3	9.8
3	11.2	1.9	9.8
4	7.4	1.9	9.4

Working with your DF - calculations with columns

```
df_ss['acid/sugar ratio'] = df_ss['fixed acidity'] / df_ss['residual sugar']
```

```
/home/frank/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:1: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexi
ng.html#indexing-view-versus-copy
  """Entry point for launching an IPython kernel.
```

```
df_ss.head()
```

	fixed acidity	residual sugar	alcohol	acid/sugar ratio
0	7.4	1.9	9.4	3.894737
1	7.8	2.6	9.8	3.000000
2	7.8	2.3	9.8	3.391304
3	11.2	1.9	9.8	5.894737
4	7.4	1.9	9.4	3.894737

df_ss is a view of df; it's not its own dataframe. Here's more on SettingWithCopy warning.

Working with your DF - calculations with columns

```
columns_subset = ['fixed acidity', 'residual sugar', 'alcohol']
df_ss = df[columns_subset].copy() # Note copy here
df_ss.head()
```

	fixed acidity	residual sugar	alcohol
0	7.4	1.9	9.4
1	7.8	2.6	9.8
2	7.8	2.3	9.8
3	11.2	1.9	9.8
4	7.4	1.9	9.4

```
df_ss['acid/sugar ratio'] = df_ss['fixed acidity'] / df_ss['residual sugar']
df_ss.head()
```

Fixed (no warning)

	fixed acidity	residual sugar	alcohol	acid/sugar ratio
0	7.4	1.9	9.4	3.894737
1	7.8	2.6	9.8	3.000000
2	7.8	2.3	9.8	3.391304
3	11.2	1.9	9.8	5.894737
4	7.4	1.9	9.4	3.894737

Working with your DF - apply functions to columns

```
def categorize_taste(val):
    if val >= 5:
        return 'acidic'
    else:
        return 'sweet'
```

```
df_ss['taste'] = df_ss['acid/sugar ratio'].apply(categorize_taste)
df_ss.head()
```

	fixed acidity	residual sugar	alcohol	acid/sugar ratio	taste
0	7.4	1.9	9.4	3.894737	sweet
1	7.8	2.6	9.8	3.000000	sweet
2	7.8	2.3	9.8	3.391304	sweet
3	11.2	1.9	9.8	5.894737	acidic
4	7.4	1.9	9.4	3.894737	sweet

Working with your DF - apply complicated functions

```
def decide_to_buy(alcohol, taste):
    if (alcohol > 9.5 and taste == 'sweet'):
        return 'buy'
    return "don't buy"
```

```
df_ss['buy'] = df_ss.apply(lambda row: decide_to_buy(row[2], row[4]),
                           axis=1)
df_ss.head()
```

	fixed acidity	residual sugar	alcohol	acid/sugar ratio	taste	buy
0	7.4	1.9	9.4	3.894737	sweet	don't buy
1	7.8	2.6	9.8	3.000000	sweet	buy
2	7.8	2.3	9.8	3.391304	sweet	buy
3	11.2	1.9	9.8	5.894737	acidic	don't buy
4	7.4	1.9	9.4	3.894737	sweet	don't buy

Working with your DF - rows

```
df_ss = df_ss.sort_values(by='alcohol', ascending=False)
df_ss.head()
```

	fixed acidity	residual sugar	alcohol	acid/sugar ratio	taste	buy
652	15.9	7.5	14.9	2.120000	sweet	buy
588	5.0	2.0	14.0	2.500000	sweet	buy
142	5.2	1.8	14.0	2.888889	sweet	buy
144	5.2	1.8	14.0	2.888889	sweet	buy
1270	5.0	1.6	14.0	3.125000	sweet	buy

```
df_top3 = df_ss.iloc[0:3].copy()
df_top3
```

	fixed acidity	residual sugar	alcohol	acid/sugar ratio	taste	buy
652	15.9	7.5	14.9	2.120000	sweet	buy
588	5.0	2.0	14.0	2.500000	sweet	buy
142	5.2	1.8	14.0	2.888889	sweet	buy

Working with your DF - rows

```
df_top3 = df_ss.iloc[0:3].copy()
df_top3
```

.iloc - integer indexing

	fixed acidity	residual sugar	alcohol	acid/sugar ratio	taste	buy
652	15.9	7.5	14.9	2.120000	sweet	buy
588	5.0	2.0	14.0	2.500000	sweet	buy
142	5.2	1.8	14.0	2.888889	sweet	buy

```
df_top3.loc[652,:]
```

.loc - label indexing

```
fixed acidity        15.9
residual sugar        7.5
alcohol              14.9
acid/sugar ratio     2.12
taste               sweet
buy                   buy
Name: 652, dtype: object
```

```
df_top3.loc[0, :]
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
~/anaconda3/lib/python3.7/site-packages/pandas/core/indexes/base.py in get_loc(self, ke
y, method, tolerance)
```

Working with your DF - row masking

```
df_ss.head()
```

	fixed acidity	residual sugar	alcohol	acid/sugar ratio	taste	buy
652	15.9	7.5	14.9	2.120000	sweet	buy
588	5.0	2.0	14.0	2.500000	sweet	buy
142	5.2	1.8	14.0	2.888889	sweet	buy
144	5.2	1.8	14.0	2.888889	sweet	buy
1270	5.0	1.6	14.0	3.125000	sweet	buy

```
df_ss.shape
```

```
(1599, 6)
```

```
threshold_alcohol = 10
mask_threshold = df_ss['alcohol'] >= threshold_alcohol
df_thresh = df_ss[mask_threshold]
df_thresh.shape
```

```
(919, 6)
```

Working with your DF - more row masking

```
df_ss.shape
```

```
(1599, 6)
```

```
m1 = df_ss['buy'] == 'buy'
m2 = df_ss['residual sugar'] > 3
m3 = df_ss['fixed acidity'] <= 10

mask = m1 | m2 | m3

df_mask = df_ss[mask]

df_mask.shape
```

```
(1485, 6)
```

Breakout

Continue working with your dataframe holding the mass shootings data.

1. Make a new column called ‘Date’ that is a datetime object based on ‘Incident Date’ (use your cheatsheet!)
2. Set the ‘Date’ column to be the row index for the dataframe. You’ll need to set ‘inplace=True’ for the .reset_index method to apply the change permanently.
3. Make a new dataframe that only contains the ‘State’, ‘Killed’, and ‘Injured’ columns (with ‘Date’ as the row index).
4. Make a new column ‘# Killed or Injured’ that is the sum of the ‘Killed’ and ‘Injured’ columns.
5. Make another new column ‘Deep South’ which will be True if the ‘State’ is Louisiana, Alabama, Mississippi, Georgia, or South Carolina.
6. Only keep the rows where there has been more that 5 people Killed or Injured. How many rows are left?
7. In this dataframe, sort the rows from the most Killed or Injured to least.

Working with your DF - selecting cells

```
df_ss.head()
```

	fixed acidity	residual sugar	alcohol	acid/sugar ratio	taste	buy
652	15.9	7.5	14.9	2.120000	sweet	buy
588	5.0	2.0	14.0	2.500000	sweet	buy
142	5.2	1.8	14.0	2.888889	sweet	buy
144	5.2	1.8	14.0	2.888889	sweet	buy
1270	5.0	1.6	14.0	3.125000	sweet	buy

```
df_ss.iloc[0,2] # integer indexing
```

```
14.9
```

```
df_ss.loc[652, 'alcohol'] # label indexing
```

```
14.9
```

Working with your DF - selecting cross-section of cells

```
df_ss.head()
```

	fixed acidity	residual sugar	alcohol	acid/sugar ratio	taste	buy
652	15.9	7.5	14.9	2.120000	sweet	buy
588	5.0	2.0	14.0	2.500000	sweet	buy
142	5.2	1.8	14.0	2.888889	sweet	buy
144	5.2	1.8	14.0	2.888889	sweet	buy
1270	5.0	1.6	14.0	3.125000	sweet	buy

```
df_ss.iloc[0:2,1:3] # integer indexing
```

	residual sugar	alcohol
652	7.5	14.9
588	2.0	14.0

```
df_ss.loc[652:588, 'residual sugar':'alcohol']  # label indexing
```

	residual sugar	alcohol
652	7.5	14.9
588	2.0	14.0

Working with your DF - NaNs (not a number)

MSP recommends using dropna, fillna, isna, notna

```
df = pd.read_csv('data/mass_shootings_2015.csv')
df.drop(columns=['Operations'], inplace=True)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 330 entries, 0 to 329
Data columns (total 6 columns):
Incident Date     330 non-null object
State             330 non-null object
City Or County    330 non-null object
Address           327 non-null object
# Killed          330 non-null int64
# Injured         330 non-null int64
dtypes: int64(2), object(4)
memory usage: 15.5+ KB
```

```
mask = pd.isna(df['Address'])
df_nans = df[mask]
df_nans
```

	Incident Date	State	City Or County	Address	# Killed	# Injured
93	September 8, 2015	Georgia	Berlin	NaN	0	4
183	June 28, 2015	Illinois	Venice	NaN	0	6
192	June 20, 2015	North Carolina	Morven	NaN	0	4

Joining DataFrames

```
red_wines_df = pd.read_csv('data/winequality-red.csv', delimiter=';')
white_wines_df = pd.read_csv('data/winequality-white.csv', delimiter=';')
```

```
red_wines_df.columns
```

```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'quality'],
      dtype='object')
```

```
white_wines_df.columns
```

```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'quality'],
      dtype='object')
```

```
red_wines_df.columns == white_wines_df.columns
```

```
array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True])
```

```
red_wines_df.shape
```

```
(1599, 12)
```

```
white_wines_df.shape
```

```
(4898, 12)
```

Joining DataFrames

```
wines_df = pd.concat([red_wines_df, white_wines_df], axis=0) # top/bottom
```

```
wines_df.shape
```

```
(6497, 12)
```

```
wines_df.iloc[1595:1605, :]
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
1595	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76	11.2	6
1596	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	11.0	6
1597	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	10.2	5
1598	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	11.0	6
0	7.0	0.270	0.36	20.7	0.045	45.0	170.0	1.00100	3.00	0.45	8.8	6
1	6.3	0.300	0.34	1.6	0.049	14.0	132.0	0.99400	3.30	0.49	9.5	6
2	8.1	0.280	0.40	6.9	0.050	30.0	97.0	0.99510	3.26	0.44	10.1	6
3	7.2	0.230	0.32	8.5	0.058	47.0	186.0	0.99560	3.19	0.40	9.9	6
4	7.2	0.230	0.32	8.5	0.058	47.0	186.0	0.99560	3.19	0.40	9.9	6
5	8.1	0.280	0.40	6.9	0.050	30.0	97.0	0.99510	3.26	0.44	10.1	6

Aggregating in DataFrames (groupby)

```
df = pd.read_csv('data/winequality-red.csv', delimiter=';')
df.head()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5

```
# Quality looks like something we might want to group by. Let's check it out in a little
# more detail first, though.
df['quality'].unique()
```

```
array([5, 6, 7, 4, 8, 3])
```

```
df['quality'].value_counts()
```

```
5    681
6    638
7    199
4     53
8     18
3     10
Name: quality, dtype: int64
```

Aggregating in DataFrames (groupby)

Minimally Sufficient Pandas syntax suggestion:

```
df.groupby('grouping column').agg({'aggregating column': 'aggregating function'})
```

```
grouped = df.groupby('quality').agg({'chlorides': 'sum', 'fixed acidity': 'mean', 'alcohol': 'max'})
grouped
```

quality	chlorides	fixed acidity	alcohol
3	1.225	8.360000	11.0
4	4.806	7.779245	13.1
5	63.153	8.167254	14.9
6	54.202	8.347179	14.0
7	15.241	8.872362	14.0
8	1.232	8.566667	14.0

```
grouped.index
```

```
Int64Index([3, 4, 5, 6, 7, 8], dtype='int64', name='quality')
```

```
grouped.columns
```

```
Index(['chlorides', 'fixed acidity', 'alcohol'], dtype='object')
```

Plotting the data in DataFrames

```
df_mass_shooting = df3
df_mass_shooting.head()
```

Date	State	# Killed	# Injured	# Killed or Injured	Deep South
2015-12-02	California	16	19	35	False
2015-05-17	Texas	9	18	27	False
2015-10-01	Oregon	10	9	19	False
2015-11-22	Louisiana	0	17	17	True
2015-11-27	Colorado	3	9	12	False

```
import matplotlib.pyplot as plt
print(plt.style.available)
plt.style.use('ggplot')
```

```
['seaborn-notebook', 'seaborn-colorblind', 'seaborn-dark-palette', 'seaborn-pastel', 'seaborn-talk', 'seaborn-dee
p', 'Solarize_Light2', 'tableau-colorblind10', 'grayscale', 'seaborn-dark', 'seaborn-darkgrid', 'fivethirtyeight
', 'bmh', 'fast', 'seaborn-poster', 'seaborn-paper', 'seaborn-white', 'seaborn-muted', 'classic', '_classic_test
', 'ggplot', 'seaborn-whitegrid', 'dark_background', 'seaborn-ticks', 'seaborn', 'seaborn-bright']
```

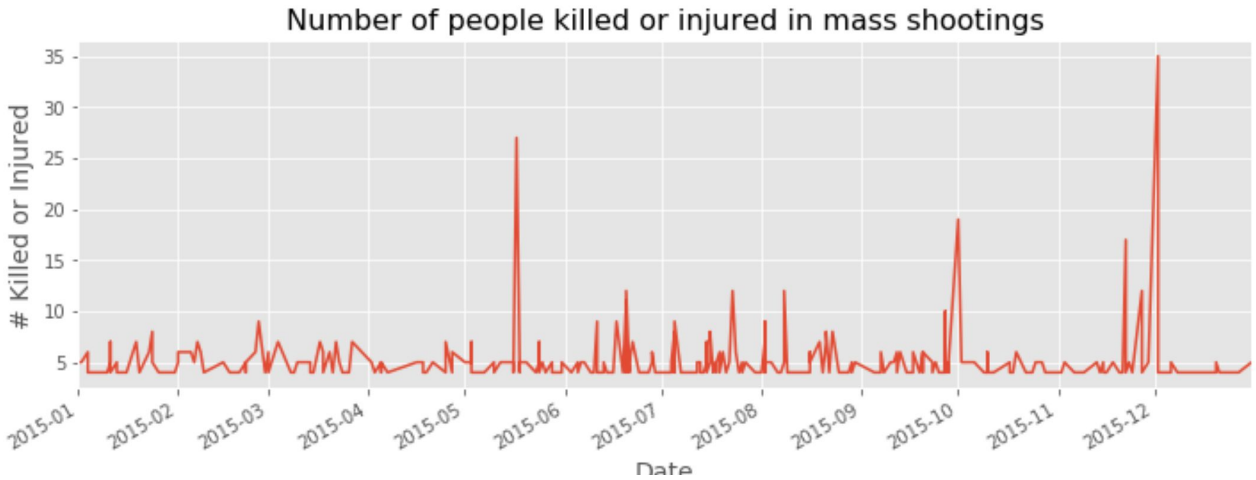
Plotting the data in DataFrames

```
ax = df_mass_shooting['# Killed or Injured'].plot()

ax.set_title('Number of people killed or injured in mass shootings', fontsize=16)
ax.set_xlabel('Date', fontsize=14)
ax.set_ylabel('# Killed or Injured', fontsize=14)

fig = ax.figure
fig.set_size_inches(10, 4)

fig.tight_layout(pad=1)
fig.savefig('number_killed_or_injured.png', dpi=125)
plt.show()
```



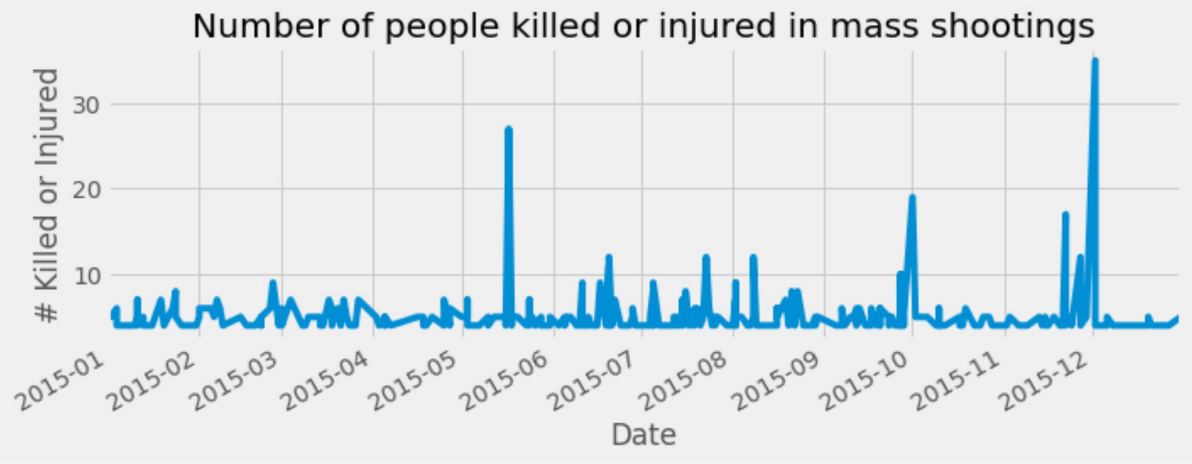
Plotting the data in DataFrames

```
plt.style.use('fivethirtyeight')
ax = df_mass_shooting['# Killed or Injured'].plot()

ax.set_title('Number of people killed or injured in mass shootings')
ax.set_xlabel('Date')
ax.set_ylabel('# Killed or Injured')

fig = ax.figure
fig.set_size_inches(10, 4)

fig.tight_layout(pad=1)
fig.savefig('number_killed_or_injured.png', dpi=125)
plt.show()
```



Objectives

- pandas motivation, and what it is
- pandas DataFrame conceptual model
  - pandas Series
- Why learning pandas is tricky (and what to do about it)
- Making a DataFrame
- Working with the DataFrame
  - statistics
  - columns, rows, cells
  - nans (**n**ot **a** **n**umber)
- Joining DataFrames
- Aggregating in DataFrames (groupby)
- Plotting in DataFrames