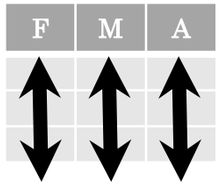
Data Wrangling

with pandas Cheat Sheet

http://pandas.pydata.org

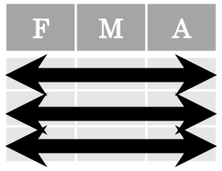
Tidy Data – A foundation for wrangling in pandas

In a tidy data set:



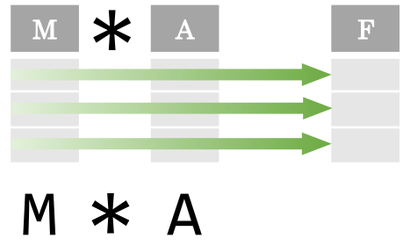
Each **variable** is saved in its own **column**

&



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.

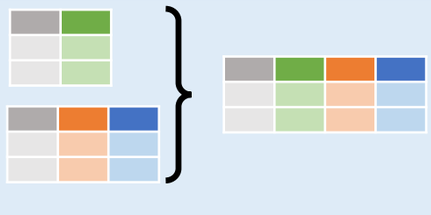
```
df = (pd.melt(df)
        .rename(columns={
                'variable' : 'var',
                'value' : 'val'})
        .query('val >= 200')
    )
```

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=df.assign(a=[1:3], b=[4:6])
Add new columns to DataFrame.

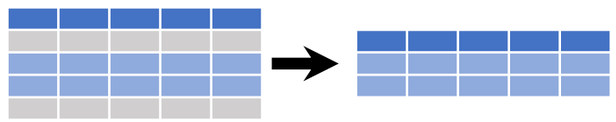
df=df.sort_values('mpg')
Order rows by values of a column (low to high).

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

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

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

Subset Observations (Rows)



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

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

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(10, 'value')
Select and order top 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 .

Summarize Data

`df.describe()`
Get basic descriptive statistics for each column.

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

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

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

pandas provides a large set of summary functions that operate on DataFrame columns and produce single values for each of the columns, returned as a pandas Series. Here are some examples:

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

`df.sum()`
Sum values of each column.

`df.prod()`
Multiply values of each column.

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

`df.median()`
Median value of each column.

`df.min()`
Minimum value in each column.

`df.max()`
Maximum value in each column.

`df.mean()`
Mean value of each column.

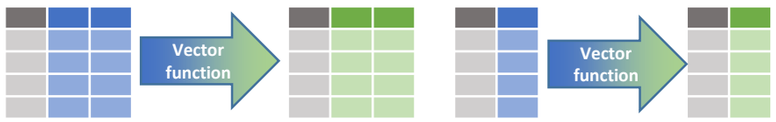
`df.var()`
Variance of each column.

`df.std()`
Standard deviation of each column.

Make New Variables

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

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



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

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

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

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

`df.col.between(a,b)`
Are column values between a and b?

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

`(df>5).expanding().apply(all)>=1`
Cumulative all.

`(df>5).expanding().apply(any)>=1`
Cumulative any.

`expanding().mean()`
Cumulative mean.

`cumsum()`
Cumulative sum.

`cummax()`
Cumulative max.

`cummin()`
Cumulative min.

`cumprod()`
Cumulative product.

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

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

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='inner', 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	x3
A	T
B	F
D	T

=

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(['_merge'],axis=1)`
Rows that appear in ydf but not zdf (Setdiff).