Data Wrangling

Simple | Smart | Speed
CHOOLS

Tidy Data –A foundation for wrangling in pandas

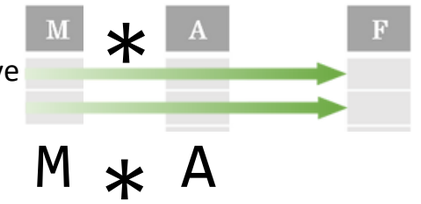
In a tidy data set:

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

&

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

Each **observation** is saved in its own **row**



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	1
	2	5	8	0
e	2	6	9	1

```
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 DataFramewith a MultiIndex

Method Chaining

Most pandasmethods return a DataFrameso that another pandas method can be applied to the result. This improves readability of code.

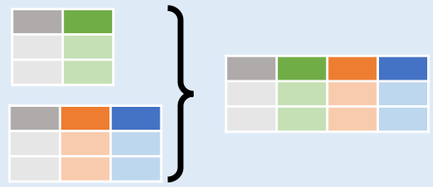
```
df = (pd.melt(df)
.rename(columns={
'variable':'var',
'value':'val'})
       .query('val>= 200')
)
```

Reshaping Data –Change layout, sorting, reindexing, renaming

```
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).

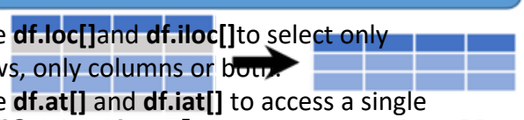
```
df.rename(columns = {'y':'year'})
```
Rename the columns of a DataFrame

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

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

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

Subset Observations-rows



Use **df.loc[]**and **df.iloc[]**to select only rows, only columns or both.

Use **df.at[]** and **df.iat[]** to access a single value by row and column.

```
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 rows.

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

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

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

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

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*.

Subsets-rows and columns

Use **df.loc[]**and **df.iloc[]**to select only rows, only columns or both.

Use **df.at[]** and **df.iat[]** to access a single value by row and column.

```
df.loc[df['width'] > 7, 'length':'species']
```
Extract rows that meet logical criteria.Select multiple columns with specific names.First index selects rows, second index columns.

```
df.iloc[10:20]
```
Select rows 10-20.

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

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

```
df.query('Length > 7 and Width < 8')
```
Select rows.Select rows meeting logical condition, and only the specific columns .

```
df.query('Name.str.startswith("abc")', engine="python")
```

```
df.iat[1, 2]
```
Access single value by index

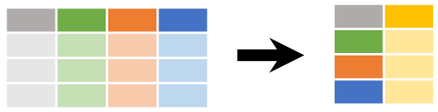
```
df.at[4, 'A']
```
Access single value by label

<	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()	Logicaland, or, not, xor, any, all

'\.'	Matches strings containinga 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 excepthe string 'Species'

Summarize Data

`df['w'].value_counts()`
Count number of rows with each unique value of variable
`len(df)`
# of rows in DataFrame.
`df.shape`
Tuple of # of rows, # of columns in DataFrame.
`df['w'].nunique()`
# of distinct values in a column.
`df.describe()`
Basic descriptive and statistics for each column (or GroupBy).



pandas provides a large set of summary functionsthat operate on different kinds of pandasobjects (DataFramecolumns, 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:

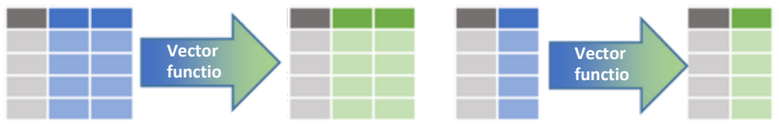
`sum()` `min()`
Sum values of each object.Minimum value in each object.
`count()` `max()`
Count non-NA/null values of Maximum value in each object.
each object.`mean()`
`median()`Mean value of each object.
Median value of each object.`var()`
`quantile([0.25,0.75])`Variance of each object.
Quantiles of each object.`std()`
`apply(function)`Standard deviation of each
Apply function to each object.object.

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

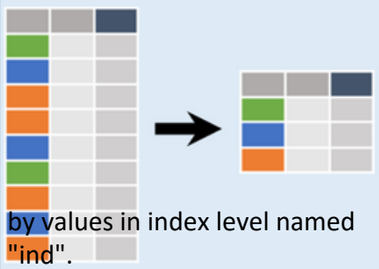
`df.assign(Area=lambda df: df.Length*df.Height)`
Compute and append one or more new columns.
`df['Volume'] = df.Length*df.Height*df.Depth`
Add single column.
`pd.qcut(df.col, n, labels=False)`
Bin column into n buckets.



pandas provides a large set of **vector functions** that operate on all columns of a DataFrameor 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)` `min(axis=1)`
Element-wise max.Element-wise min.
`clip(lower=-10,upper=10)` `abs()`
Trim values at input thresholdsAbsolute value.

Group Data



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

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

All ofthe summary functions listed above can be applied to a group.

Additional GroupByfunctions:
`size()` `agg(function)`
Size of each group.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.

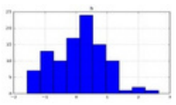
`shift(1)` `shift(-1)`
Copy with values shifted by 1.Copy with values lagged by 1.
`rank(method='dense')` `cumsum()`
Ranks with no gaps.Cumulative sum.
`rank(method='min')` `cummax()`
Ranks. Ties get min rank.Cumulative max.
`rank(pct=True)` `cummin()`
Ranks rescaled to interval [0, 1].Cumulative min.
`rank(method='first')` `cumprod()`
Ranks. Ties go to first value.Cumulative product.

Windows

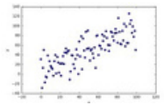
`df.expanding()`
Return an Expanding object allowing summary functions to be applied cumulatively.
`df.rolling(n)`
Return a Rolling object allowing summary functions to be applied to windows of length n.

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 bdf

Standard Joins

```
x1 x2 pd.merge(adf, bdf,
A  1  how='left', on='x1')
B  2
C  3 NaN
```
Join matching rows from bdfto 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 adfto 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.

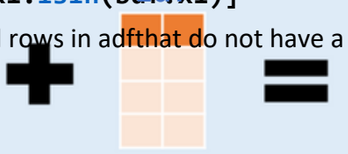
```
x1 x2 x3
A  1  T
B  2  F
C  3 NaN
D  NaN T
pd.merge(adf, bdf,
how='outer', on='x1')
```
Join data. Retain all values, all rows.

Filtering Joins

```
x1 x2 adf[adf.x1.isin(bdf.x1)]
A  1
B  2
```
All rows in adfthat have a match in bdf.

```
x1 x2 adf[~adf.x1.isin(bdf.x1)]
C  3
```
All rows in adfthat do not have a match in bdf.

ydf zdf



Set-like Operations

```
x1 x2 pd.merge(ydf, zdf)
B  2
C  3
```
Rows that appear in both ydfand 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 ydfand zdf (Union).

```
x1 x2 pd.merge(ydf, zdf, how='outer',
      indicator=True)
A1.query('_merge == "left_only"')
      .drop(columns=['_merge'])
```