




THE UNIVERSITY OF
CHICAGO

MASTERS IN
COMPUTATIONAL
SOCIAL SCIENCE
THE UNIVERSITY OF CHICAGO

A large, semi-transparent image of Po the panda from the movie Kung Fu Panda. He is wearing his signature conical straw hat and a red and yellow scarf. He has a confident, slightly smug expression. The background of the image shows a blurred view of a traditional Chinese village with tiled roofs.

MACS 30111

M7: Pandas

Announcements

- ▶ **Switch to accelerated track: need MPCS exam**

- ▶ <https://masters.cs.uchicago.edu/student-resources/placement-exams/>
- ▶ Register, take the exam

- ▶ **Final exam schedule**

- ▶ Tues, Dec 10th, 10-12pm

- ▶ **Review**

- ▶ Textbook
- ▶ Team tutorial
- ▶ Short exercise
- ▶ Programming assignment
- ▶ Extra exercise

Agenda / misc

Topics:

- ❑ **Introduction to Pandas**
- ❑ Creating DataFrames and Series
- ❑ Working with DataFrames and Series
- ❑ Applying functions to data
- ❑ Group, combine, and pivoting
- ❑ Visualizing DataFrames and Series (Probably Thurs)



Data analysis toolkit

Reference: <https://pandas.pydata.org>

The “*pandas*” name

pan for “Panel”

da for “Data”

Panel data: multidimensional, structured datasets that include observations of something over a number of different time periods.

Use cases for the pandas library

Tabular Data

Time-series Data

Matrix Data

Statistical Datasets

Topics:

- ❑ Introduction to Pandas
- ❑ **Creating DataFrames and Series**
- ❑ Working with DataFrames and Series
- ❑ Applying functions to data
- ❑ Group, combine, and pivoting
- ❑ Visualizing DataFrames and Series

Pandas

Series

In: s

Out:

```
Rory    90
Lorelai  95
Luke     90
dtype: int64
```

DataFrame

In: s = pd.DataFrame(data)

In: s

Out:

```
      PA1  PA2
Rory    90   88
Lorelai  95   90
Luke     90   75
```

Reference:

- <https://pandas.pydata.org/docs/reference/api/pandas.Series.html>
- <https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html>

Importing the Pandas library

Import pandas using the alias “pd”

```
import pandas as pd
```

Series Constructor

Pass in 1-d data (list/np.array), alongside an accompanying (1-d) index:

```
data = [90, 95, 90]
```

```
index = ["Rory", "Lorelai", "Luke"]
```

```
s = pd.Series(data, index)
```

```
s  
Out:  
Rory    90  
Lorelai 95  
Luke    90  
dtype: int64
```

Pass in a dictionary, where the keys are inferred to be the indices:

```
data = {"Rory":90, "Lorelai": 95, "Luke": 90}
```

```
s = pd.Series(data)
```

DataFrame Constructor

Pass in a dictionary of lists/arrays, alongside an accompanying (1-d) index:

```
data = {"PA1": [90,95,90], "PA2": [88,90,75]}
```

```
index = ["Rory", "Lorelai", "Luke"]
```

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

	PA1	PA2
Rory	90	88
Lorelai	95	90
Luke	90	75

Pass in a dictionary of dictionaries

```
data = {"PA1": {"Rory": 90, "Lorelai": 95, "Luke": 90}, "PA2": {"Rory":  
88, "Lorelai": 90, "Luke": 75}}
```

DataFrame Constructor

Data can also be passed in as an NumPy multidimensional array:

```
In: import numpy as np
```

```
In: np.random.seed(13)
```

```
In: data = np.random.uniform(0, 1, (3,2))
```

```
In: data
```

```
Out:
```

```
array([[0.77770241, 0.23754122],  
       [0.82427853, 0.9657492 ],  
       [0.97260111, 0.45344925]])
```

```
In: index = ["Rory", "Lorelai", "Luke"]
```

```
In: columns = ["pa1", "pa2"]
```

```
In: df = pd.DataFrame(data, index, columns)
```

```
In: df
```

```
Out:
```

	pa1	pa2
Rory	0.777702	0.237541
Lorelai	0.824279	0.965749
Luke	0.972601	0.453449

Read from file

CSV

JSON

HTML

Excel Spreadsheets

And much more...

```
trees = pd.read_csv("2015_Street_Tree_Census_Tree_Data_20231117.csv")
```

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

Key commands

- ↳ Loading: `read_csv()`
- ↳ Summarizing:
 - ↳ `data.head()`
 - ↳ `data.tail()`
- ↳ Selecting:
 - ↳ By index: `iloc` (e.x. `trees.iloc[:, 1]`)
 - ↳ By name: `loc` (e.x. `trees.loc[:, "block_id"]`)

Series VS DataFrame

► Series:

- One-dimensional ndarray with axis labels

Rory	90
Lorelai	95
Luke	90

► DataFrame

- Two-dimensional tabular data with labeled row and column axes

	PA1	PA2
Rory	90	88
Lorelai	95	90
Luke	90	75

Reference:

- Series: <https://pandas.pydata.org/docs/reference/api/pandas.Series.html>
- DataFrame: <https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html>

Approaching the Series / DataFrame

Size, column names and data types: *dtype*, *shape*

```
In [34]: s
Out[34]:
Rory    90
Lorelai 95
Luke    90
dtype: int64
```

```
In [35]: s.dtype
Out[35]: dtype('int64')
```

```
In [36]: s.shape
Out[36]: (3,)
```

```
In [37]:
```

```
In [37]: df
Out[37]:
      PA1  PA2
Rory    90   88
Lorelai 95   90
Luke    90   75
```

```
In [38]: df.dtypes
Out[38]:
PA1    int64
PA2    int64
dtype: object
```

```
In [39]: df.shape
Out[39]: (3, 2)
```

Approaching the Series / DataFrame

What does the data look like? *df.head()*, *df.tail()*

```
In [40]: s.head(n=2)
```

```
Out[40]:
```

```
Rory    90  
Lorelai 95  
dtype: int64
```

```
In [41]: s.tail(n=1)
```

```
Out[41]:
```

```
Luke    90  
dtype: int64
```

Motivating example

- ▶ Data exploration for the 2015 New York City Street Tree Survey
- ▶ Data: <https://data.cityofnewyork.us/Environment/2015-Street-Tree-Census-Tree-Data/uvpi-gqnh>
 - ▶ CSV, 683,789 lines

1. How many different species are planted as street trees in New York?
2. What are the five most common street tree species in New York?
3. What is the most common street tree species in Brooklyn?
4. What percentage of the street trees in Queens are dead or in poor health?
5. How does street tree health differ by borough?

https://github.com/computer-science-with-applications/examples/tree/main/working_with_data/pandas

Coding practice: 4.3.1

Missing Data

Identifying missing data in Series and DataFrames:

```
s.isna(); s.notna()  
df.isna(); df.notna()
```

We can drop rows or columns with missing data:

```
df_drop_byrow = df.dropna()  
df_drop_bycol = df.dropna(axis = 1)
```

We can also fill missing values:

```
df.col.fillna(np.mean(df.col))
```

```
In [42]: s.isna()
```

```
Out[42]:
```

```
Rory    False  
Lorelai False  
Luke    False  
dtype: bool
```

```
In [43]: s.notna()
```

```
Out[43]:
```

```
Rory    True  
Lorelai True  
Luke    True  
dtype: bool
```

```
In [44]: df.isna()
```

```
Out[44]:
```

```
      PA1  PA2  
Rory   False False  
Lorelai False False  
Luke   False False
```

```
In [45]: df.notna()
```

```
Out[45]:
```

```
      PA1  PA2  
Rory   True  True  
Lorelai True  True  
Luke   True  True
```

Coding practice: 4.3.2

Selecting rows (indexing)

```
In [6]: trees.head()
```

```
Out[6]:
```

	tree_id	block_id	created_at	tree_dbh	...	council district	census tract	bin	bbl
0	180683	348711	08/27/2015	3	...	29.0	739.0 4052307.0	4.022210e+09	
1	200540	315986	09/03/2015	21	...	19.0	973.0 4101931.0	4.044750e+09	
2	204026	218365	09/05/2015	3	...	34.0	449.0 3338310.0	3.028870e+09	
3	204337	217969	09/05/2015	10	...	34.0	449.0 3338342.0	3.029250e+09	
4	189565	223043	08/30/2015	21	...	39.0	165.0 3025654.0	3.010850e+09	

Row label-based indexing

```
In [7]: trees.loc[180683]
```

```
Out[7]:
```

tree_id	362889
block_id	209521
created_at	10/22/2015
tree_dbh	6
...	

Position-based indexing

```
In [8]: trees.iloc[0]
```

```
Out[8]:
```

tree_id	180683
block_id	348711
created_at	08/27/2015
tree_dbh	3
stump_diam	0
curb_loc	OnCurb
status	Alive
health	Fair
spc_latn	Acer rubrum
...	

Naming columns

↓ If you want to explore the column names that were imported, you can use: `trees.columns`

```
Index(['tree_id', 'block_id', 'created_at', 'tree_dbh', 'stump_diam',  
      'curb_loc', 'status', 'health', 'spc_latin', 'spc_common', 'steward',  
      'guards', 'sidewalk', 'user_type', 'problems', 'root_stone',  
      'root_grate', 'root_other', 'trunk_wire', 'trnk_light', 'trnk_other',  
      'brch_light', 'brch_shoe', 'brch_other', 'address', 'postcode',  
      'zip_city', 'community board', 'borocode', 'borough', 'cnclldist',  
      'st_assem', 'st_senate', 'nta', 'nta_name', 'boro_ct', 'state',  
      'latitude', 'longitude', 'x_sp', 'y_sp', 'council district',  
      'census tract', 'bin', 'bbl'],  
      dtype='object')
```

Selecting columns

```
1 type(trees['health'])
```

pandas.core.series.Series

```
In [16]: trees["health"]
```

```
Out[16]:
```

0	Fair
1	Fair
2	Good
3	Good
4	Good

...

683783	Good
683784	Good
683785	Good
683786	Good
683787	Fair

Name: health, Length: 683788, dtype: object

```
1 type(trees[['health']])
```

pandas.core.frame.DataFrame

```
In [17]: trees[["health"]]
```

```
Out[17]:
```

	health
0	Fair
1	Fair
2	Good
3	Good
4	Good

...

683783	Good
683784	Good
683785	Good
683786	Good
683787	Fair

[683788 rows x 1 columns]

Note: depending on data, might have 'borough'

```
1 type(trees[['health', 'borough']])
```

pandas.core.frame.DataFrame

```
In [18]: trees[["health", "borough"]]
```

```
Out[18]:
```

	health	borough
0	Fair	Queens
1	Fair	Queens
2	Good	Brooklyn
3	Good	Brooklyn
4	Good	Brooklyn
...
683783	Good	Brooklyn
683784	Good	Queens
683785	Good	Staten Island
683786	Good	Bronx
683787	Fair	Queens

[683788 rows x 2 columns]

Selecting rows and columns

Note: depending on data, might have 'boroname'

```
In [19]: trees.iloc[:5][["health", "borough"]]
```

```
Out[19]:
```

	health	borough
0	Fair	Queens
1	Fair	Queens
2	Good	Brooklyn
3	Good	Brooklyn
4	Good	Brooklyn

```
In [20]: trees.loc[[180683, 200540, 204026], ["health",  
...: "borough"]]
```

```
Out[20]:
```

	health	borough
180683	Good	Brooklyn
200540	NaN	Staten Island
204026	Good	Staten Island

Filtering

- Boolean masks
- Logical operations

```
filter = (trees.borough == "Queens") & ((trees.status == "Dead") | (trees.health == "Poor"))  
trees[filter]
```

- Remove rows where trees.status is Dead
- Select rows where trees.status is not Dead

```
trees[trees.status != "Dead"]
```

```
0    False  
1    False  
2    False  
3    False  
4    False
```

```
...  
683783  False  
683784  False  
683785  False  
683786  False  
683787  False
```

```
Name: status, Length: 683788, dtype:  
bool
```

Agenda / misc

- ▶ Applied practice
- ▶ Room reservation: final exam
- ▶ Deadlines/grading/etc.
- ▶ Anything else?

Topics:

- ❑ Introduction to Pandas
- ❑ Creating DataFrames and Series
- ❑ **Working with DataFrames and Series**
- ❑ Applying functions to data
- ❑ Group, combine, and pivoting
- ❑ Visualizing DataFrames and Series (Probably Thurs)

Review: load in data + add row names

- ↳ How to load data?
- ↳ How to set row names? (and why)
 - ↳ Hint: `index_col`
- ↳ **Application:** find tree at index 9 and tree_id of 192755

Practice Task: answer a research question

- ↓ **Suppose we have the following question:** how do tree health indicators vary by borough?
GIVEN THE CURRENT SKILLS / TOOLS WE HAVE DISCUSSED SO FAR
- ↓ **In groups:**
 - ↩ Determine how you will measure this. (which column)
 - ↩ What tools do you currently have to do this?
 - ↩ What do you want to be able to do?

Topics:

- ❑ Introduction to Pandas
- ❑ Creating DataFrames and Series
- ❑ Working with DataFrames and Series
- ❑ **Applying functions to data**
- ❑ Group, combine, and pivoting
- ❑ Visualizing DataFrames and Series

Differences: help me understand

```
In : trees.count()
```

```
Out :
```

tree_id	683788
block_id	683788
created_at	683788
tree_dbh	683788
stump_diam	683788
curb_loc	683788
status	683788
health	652172
spc_latin	652169
spc_common	652169

```
In : trees.status.value_counts()
```

```
Out :
```

status	
Alive	652173
Stump	17654
Dead	13961

Name: count, dtype: int64

```
In : trees.count()
```

```
Out :
```

block_id	683788
created_at	683788
tree_dbh	683788
stump_diam	683788
curb_loc	683788
status	683788
health	652172
spc_latin	652169
spc_common	652169

```
In : trees.status.value_counts()
```

```
Out:
```

status	
Alive	652173
Stump	17654
Dead	13961

Name: count, dtype: int64

Reference: <https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.describe.html>

Common Summary Statistics

`DataFrame.count`

Count number of non-NA/null observations.

`DataFrame.max`

Maximum of the values in the object.

`DataFrame.min`

Minimum of the values in the object.

`DataFrame.mean`

Mean of the values.

`DataFrame.std`

Standard deviation of the observations.

`DataFrame.select_dtypes`

Subset of a DataFrame including/excluding columns based on their dtype.

Summary statistics for each column: `df.describe()`

Summary of categorical values: `df.value_counts()`

Pairwise correlation of columns: `df.corr()`

```
In : trees.count()
```

```
Out :
```

block_id	683788
created_at	683788
tree_dbh	683788
stump_diam	683788
curb_loc	683788
status	683788
health	652172
spc_latin	652169
spc_common	652169

```
In : trees.status.value_counts()
```

```
Out :
```

status	
Alive	652173
Stump	17654
Dead	13961

Name: count, dtype: int64

Reference: <https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.describe.html>

Good exam question: Programming practice

- ▶ CLOSED NOTE
- ▶ Suppose you have a dataframe with 20 rows and 10 columns.
 - ▶ DF name: albums
 - ▶ Column names: title, num_songs, year, genre, length, billboard, weeks_on_charts, label, avg_bpm, number_sold
- ▶ From the above dataframe, write the following code on your paper:
 - ▶ Get the shape of the dataframe
 - ▶ Print the information for the album in the 10th row
 - ▶ Print the information for the “genre” column
 - ▶ (bonus) Select albums that have have been in the top 100 on billboard (e.g. have a value less than or equal to 100)

Applying Functions to Data

`Series.apply`

For applying more complex functions on a Series.

`DataFrame.apply`

Apply a function row-/column-wise.

`DataFrame.applymap`

Apply a function elementwise on a whole DataFrame.

Topics:

- ❑ Introduction to Pandas
- ❑ Creating DataFrames and Series
- ❑ Working with DataFrames and Series
- ❑ Applying functions to data
- ❑ **Group, combine, and pivoting**
- ❑ Visualizing DataFrames and Series

Grouping

```
In: trees.groupby("borough")[["tree_dbh"]].mean()
```

Out:

	tree_dbh
borough	
Bronx	9.693649
Brooklyn	11.738884
Manhattan	8.473641
Queens	12.557870
Staten Island	10.492746

Grouping

```
In : trees.groupby(["borough", "status"]).size()
```

Out :

borough	status	
Bronx	Alive	80585
	Dead	2530
	Stump	2088
Brooklyn	Alive	169744
	Dead	3319
	Stump	4230
Manhattan	Alive	62427
	Dead	1802
	Stump	1194
Queens	Alive	237974
	Dead	4440
	Stump	8137
Staten Island	Alive	101443
	Dead	1870
	Stump	2005

dtype: int64

Reshaping: Stacking / Unstacking

- ▶ Think about maneuvering your data around
 - what do you want your rows and columns to be - what is the observation?
- ▶ Do you want an indicator variable, a count variable, etc?

Unstack: converts series with hierarchical index into a DataFrame

Unstacking

By default, unstack will order the columns by value

```
In : trees.groupby(["borough", "status"]).size()
```

Out :

borough	status	
Bronx	Alive	80585
	Dead	2530
	Stump	2088
Brooklyn	Alive	169744
...		

```
In : trees.groupby(["borough", "status"]).size().unstack()
```

Out:

status	Alive	Dead	Stump
borough			
Bronx	80585	2530	2088
Brooklyn	169744	3319	4230
Manhattan	62427	1802	1194
Queens	237974	4440	8137
Staten Island	101443	1870	2005

TASK: develop a research question

- ↓ In your groups, develop a research question you can answer with the trees dataset.
- ↓ Your question needs to use ALL of the following:
 - ↪ Filtering
 - ↪ Applying a Boolean criterion
 - ↪ Label-based indexing
 - ↪ Calculating a numerical value (e.g. mean/min/max/std/corr)
 - ↪ Exporting the text file (documentation!)

Using functions

What does this do/mean? Why and how would we do this?

```
for col_name in ["boroname", "health", "spc_common", "status"]:  
    trees[col_name] = trees[col_name].astype("category")
```

Using functions

↳ What does this do/mean? Why and how would we do this?

```
def tree_health_by_boro(trees):  
    combined_status = trees.status.where(trees.status != "Alive", trees.health)  
    num_per_boro = trees.groupby("borough").size()  
    combined_per_boro = trees.groupby(["borough", combined_status]).size()  
    pct_per_boro = combined_per_boro/num_per_boro*100.0  
    pct_per_boro_df = pct_per_boro.unstack()  
  
    return pct_per_boro_df[["Good", "Fair", "Poor", "Dead", "Stump"]]
```

```
tree_health_by_boro(trees)
```

Unstacking

By default, unstack will order the columns by value

```
In : trees.groupby(["borough", "status"]).size()
```

Out :

borough	status	
Bronx	Alive	80585
	Dead	2530
	Stump	2088
Brooklyn	Alive	169744
...		

```
In : trees.groupby(["borough", "status"]).size().unstack()
```

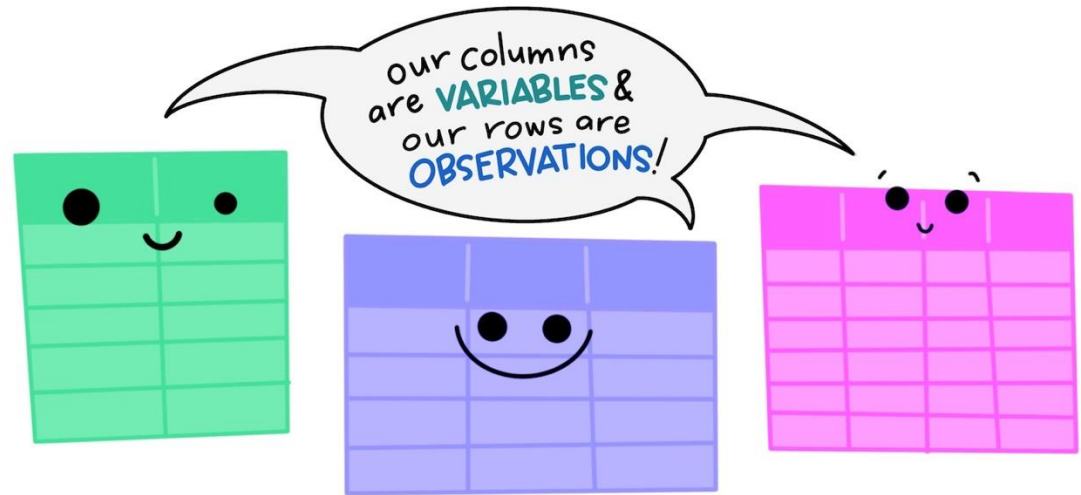
Out:

status	Alive	Dead	Stump
borough			
Bronx	80585	2530	2088
Brooklyn	169744	3319	4230
Manhattan	62427	1802	1194
Queens	237974	4440	8137
Staten Island	101443	1870	2005

“Happy families are all alike; every unhappy family is unhappy in its own way.” -- Leo Tolstoy

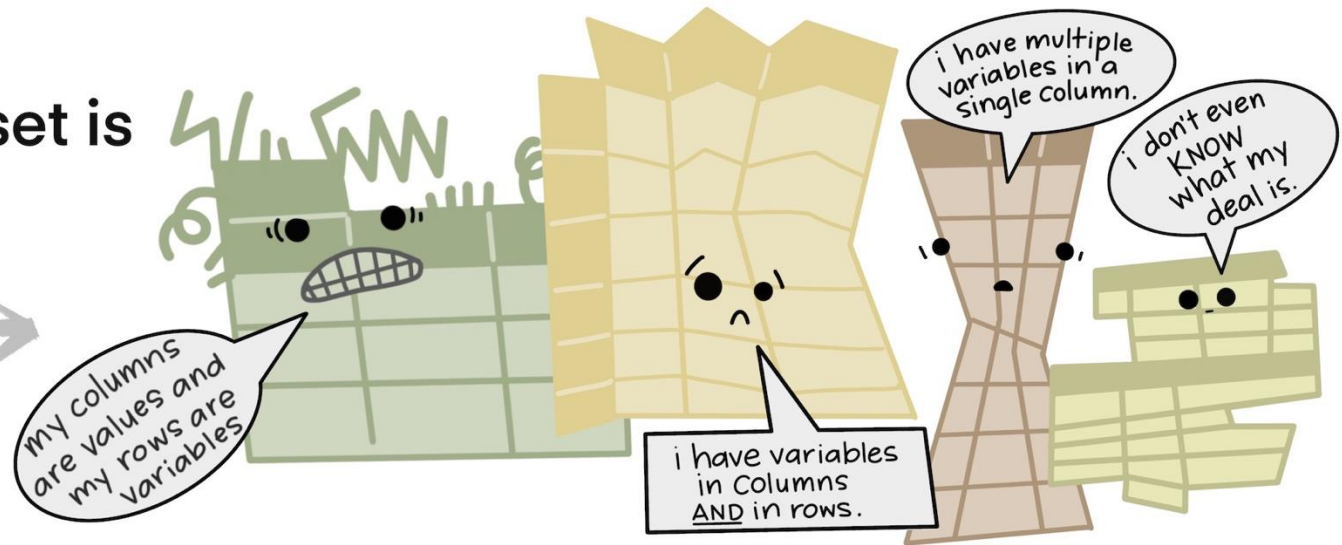
“Tidy datasets are all alike, but every messy dataset is messy in its own way.” -- Hadley Wickham

The standard structure of tidy data means that
"tidy datasets are all alike..."



"...but every messy dataset is
messy in its own way."

—HADLEY WICKHAM



“**TIDY DATA** is a standard way of mapping the meaning of a dataset to its structure.”

—HADLEY WICKHAM

In tidy data:

- each variable forms a column
- each observation forms a row
- each cell is a single measurement

each column a variable



id	name	color
1	floof	gray
2	max	black
3	cat	orange
4	donut	gray
5	merlin	black
6	panda	calico

each row
an
observation



Long vs wide

wide

id	x	y	z
1	a	c	e
2	b	d	f

long

id	key	val
1	x	a
2	x	b
1	y	c
2	y	d
1	z	e
2	z	f

Benefits: long vs wide

‘Long’ data: better for analysis

‘Wide’ data: can be helpful for visualization

Long vs wide

Common Name	lifespan_cat_eq_Short	lifespan_cat_eq_Medium	lifespan_cat_eq_Long	lifespan_cat_eq_ExtraLong	Common Name	lifespan_cat	value
Ash, Green	0	1	0	0	Ash, Green	lifespan_cat_eq_Short	0
Ash, White	0	0	0	1	Ash, Green	lifespan_cat_eq_Medium	1
Basswood	1	0	0	0	Ash, Green	lifespan_cat_eq_ExtraLong	0
Beech, American	0	0	0	1	Ash, Green	lifespan_cat_eq_Long	0
Birch, Gray	1	0	0	0	Ash, White	lifespan_cat_eq_ExtraLong	1

Pivoting

Convert a long and thin DataFrame into a short and wide DataFrame

- A column for index
- A column to supply column names for the new DataFrame
- A column to fill the new DataFrame

Options: pivot, melt, wide_to_long

Coding practice: 4.3.6

Pivot example

```
pd.melt(trees_life, value_vars=filter_col,  
id_vars= ["Common Name", "Scientific Name"],  
var_name='lifespan_cat')
```

```
pd.wide_to_long(trees_life, stubnames="lifespan_cat_eq",  
i="Common Name",  
j="lifespan_cat", suffix='\\w+')
```

```
trees_life.pivot(index='Common Name', columns='lifespan_cat',  
values='Average_lifespan')
```

(bonus) pd.get_dummies(trees_life, columns=['lifespan_cat_eq'], dtype=int)

Combining DataFrames

- **concat()**
 - perform concatenation along an axis
 - while performing set logic of the indexes on other axes
 - make a full copy of the data
- **merge()**
 - Standard database join operations between DataFrame or Series objects
- **join()**
 - join on index
 - combine columns of two differently indexed DataFrames into a single one

Reference: https://pandas.pydata.org/docs/user_guide/merging.html

Join vs merge

- ↳ Merge is the ‘big picture’ for things
 - ↳ Can have more freedom with indices / how you merge
- ↳ ‘join’ is like a subset of merge --

Merge: syntax

↳ **`DataFrame.merge(right, how='inner', on=None, left_on=None, right_on=None, left_index=False, right_index=False, sort=False, suffixes=('_x', '_y'), copy=None, indicator=False, validate=None)`**

Parameters:

- **how**{'left', 'right', 'outer', 'inner', 'cross'}, default 'inner' Type of merge to be performed.
 - left: use only keys from left frame, similar to a SQL left outer join; preserve key order.
 - right: use only keys from right frame, similar to a SQL right outer join; preserve key order.
 - outer: use union of keys from both frames, similar to a SQL full outer join; sort keys lexicographically.
 - inner: use intersection of keys from both frames, similar to a SQL inner join; preserve the order of the left keys.
 - cross: creates the cartesian product from both frames, preserves the order of the left keys.
- **On** *label or list* Column or index level names to join on. These must be found in both DataFrames. If *on* is None and not merging on indexes then this defaults to the intersection of the columns in both DataFrames.

Syntax: is it `df.merge` or `pd.merge`?

Similar across circumstances but just depends on how you want to call it...when you look at the documentation, you'll see this trend consistently.

If you call `pd.function`, you will need to specify the data frame(s).

If you `df.function`, you no longer need to specify the data frame.

TASK: ADVENTURE TIME (cont'd)

1. Get to know your second dataset
2. Merge the dataframes
3. Think about a question you could ask and give it a go