Pandas

- A Series is a data structure in Pandas that holds an array of information along with a named index.
- The named index differentiates this from a simple NumPy array.
- **Formal Definition:** One-dimensional ndarray with axis labels

Pandas

- NumPy array has numeric index

Index	Data
0	1776
1	1867
2	1821

Pandas

* Pandas Series adds on a labeled index

Labeled Index	Data
USA	1776
CANADA	1867
MEXICO	1821

Pandas

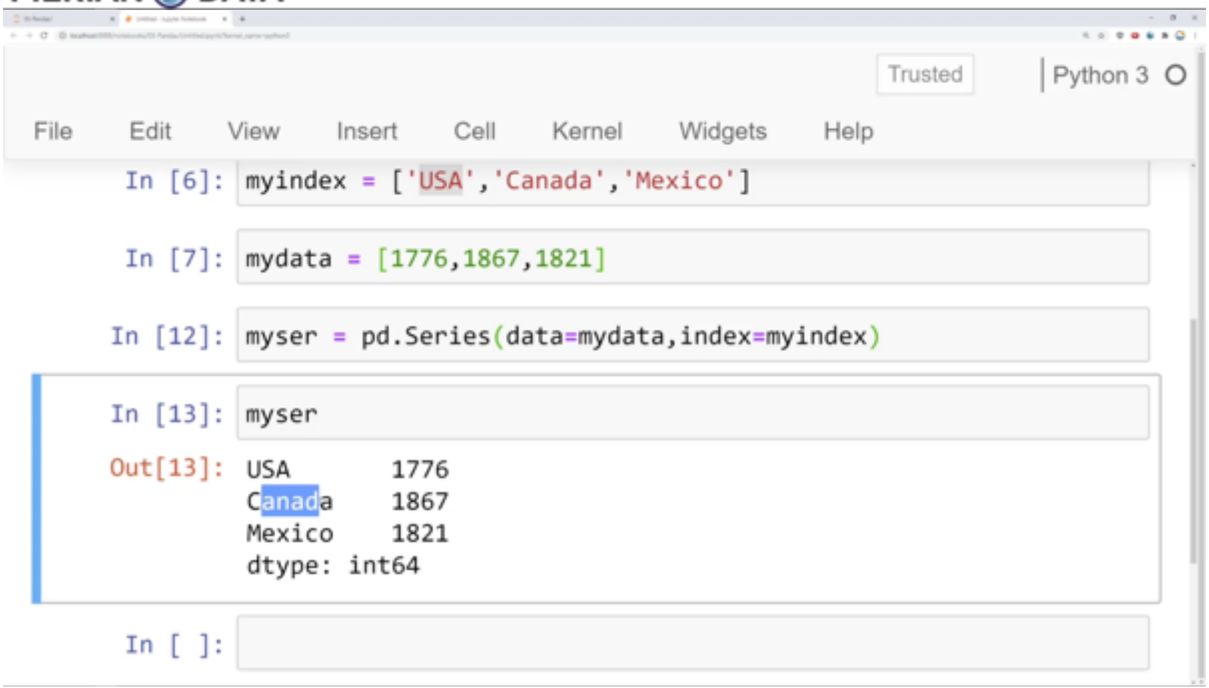
* Data is still numerically organized

Numeric Index	Labeled Index	Data
0	USA	1776
1	CANADA	1867
2	MEXICO	1821

Pandas

- Let's explore the various ways to create a Pandas Series object.
- We'll also learn about some key properties and operations.
- Later on we will learn how to combine Series with a shared index to create a tabular data structure called a DataFrame.

PIERIAN DATA



Pandas

- When reading in missing values, pandas will display them as **NaN** values.
- There are also newer specialized null pandas values such as **pd.NaT** to imply the value missing should be a timestamp.

PIERIAN DATA

Not a number , not a timestamp

Pandas

- Options for Missing Data
  - Keep it
  - Remove it
  - Replace it
  - *Note, there is never 100% correct approach that applies to all circumstances, it all depends on the exact situation you encounter!*

PIERIAN DATA

Pandas

- Keeping the missing data
  - PROS:
    - Easiest to do
    - Does not manipulate or change the true data
  - CONS:
    - Many methods do not support NaN
    - Often there are reasonable guesses

Pandas

- Keeping the missing data
  - PROS:
    - Easiest to do
    - Does not manipulate or change the true data
  - CONS:
    - Many methods do not support NaN
    - Often there are reasonable guesses

PIERIAN DATA

Pandas

- Dropping or Removing the missing data
    - PROS:
        - Easy to do.
        - Can be based on rules.
    - CONS:
        - Potential to lose a lot of data or useful information.
        - Limits trained models for future data.

Pandas

- Removing or Dropping missing data
    - Dropping a Row
        - Makes sense when a lot of info is missing

	Year	Pop	GDP	Area
USA	1776	NAN	NAN	NAN
CANADA	1867	38	1.7	3.86
MEXICO	1821	126	1.22	0.76

Pandas

- Removing or Dropping missing data
  - Dropping a Row
    - Clearly this data point as a row should probably be dropped

	Year	Pop	GDP	Area
USA	1776	NAN	NAN	NAN
CANADA	1867	38	1.7	3.86
MEXICO	1821	126	1.22	0.76

Pandas

- Removing or Dropping missing data
  - Dropping a Column
    - Good choice if every row is missing that particular feature

	Year	Pop	GDP	Area
USA	1776	328	20.5	NAN
CANADA	1867	38	1.7	NAN
MEXICO	1821	126	1.22	0.76

Pandas

- Filling in the missing data
  - PROS:
    - Potential to save a lot of data for use in training a model
  - CONS:
    - Hardest to do and somewhat arbitrary
    - Potential to lead to false conclusions

Pandas

- Filling in missing data
  - Fill with same value
    - Good choice if NaN was a placeholder

	Year	Pop	GDP	Carriers
USA	1776	328	20.5	11
CANADA	1867	38	1.7	NAN
MEXICO	1821	126	1.22	NAN

Pandas

- Filling in missing data
  - Fill with same value
    - Here NAN can be filled in with zero

	Year	Pop	GDP	Carriers
USA	1776	328	20.5	11
CANADA	1867	38	1.7	0
MEXICO	1821	126	1.22	0

PIERIAN DATA

Pandas

- Filling in missing data
  - Fill with interpolated or estimated value
    - Much harder and requires reasonable assumptions

	Year	Pop	GDP	Perct
USA	1776	328	20.5	75%
CANADA	1867	38	1.7	NAN
MEXICO	1821	126	1.22	25%

PIERIAN DATA

Pandas

- Filling in missing data
  - Fill with interpolated or estimated value
    - Much harder and requires reasonable assumptions

	Year	Pop	GDP	Perct
USA	1776	328	20.5	75%
CANADA	1867	38	1.7	50%
MEXICO	1821	126	1.22	25%

Pandas

- Let's explore the code syntax in pandas for dealing with missing values.
- Later on in the course we will have a deeper discussion on trying to decide between keep,remove, and replace options.

PIERIAN DATA

Pandas

- Often the data you need exists in two separate sources, fortunately, Pandas makes it easy to combine these together.
- The simplest combination is if both sources are already in the same format, then a **concatenation** through the **pd.concat()** call is all that is needed.

Pandas

- Concatenation is simply “pasting” the two DataFrames together, by columns:

	Year	Pop
USA	1776	328
CANADA	1867	38
MEXICO	1821	126

	GDP	Perct
USA	20.5	75%
CANADA	1.7	NAN
MEXICO	1.22	25%

Pandas

- Concatenation is simply "pasting" the two DataFrames together, by columns:

	Year	Pop
USA	1776	328
CANADA	1867	38
MEXICO	1821	126

	GDP	Perct
USA	20.5	75%
CANADA	1.7	NAN
MEXICO	1.22	25%

Pandas

- Concatenation is simply "pasting" the two DataFrames together, by rows:

	Year	Pop	GDP
USA	1776	328	20.5
CANADA	1867	38	1.7

	Year	Pop	GDP
MEXICO	1821	126	1.22
BRAZIL	1822	209	1.86

Pandas

- Often DataFrames are not in the exact same order or format, meaning we can not simply concatenate them together.
- In this case, we need to **merge** the DataFrames.
- This is analogous to a JOIN command in SQL.

Pandas

- The .merge() method takes in a key argument labeled **how**
- There are 3 main ways of merging tables together using the **how** parameter:
  - Inner
  - Outer
  - Left or Right

Pandas

- The main idea behind the argument is to decide **how** to deal with information only present in one of the joined tables.

Pandas

- Let's imagine a simple example.
- Our company is holding a conference for people in the movie rental industry.
- We'll have people register online beforehand and then login the day of the conference.

Pandas

- After the conference we have these tables

REGISTRATIONS	
reg_id	name
1	Andrew
2	Bob
3	Charlie
4	David

LOGINS	
log_id	name
1	Xavier
2	Andrew
3	Yolanda
4	Bob

Pandas

- The respective id columns indicate what order they registered or logged in on site.

REGISTRATIONS	
reg_id	name
1	Andrew
2	Bob
3	Charlie
4	David

LOGINS	
log_id	name
1	Xavier
2	Andrew
3	Yolanda
4	Bob

Pandas

- (e.g. There is only one person in the company named "Andrew")

REGISTRATIONS	
reg_id	name
1	Andrew
2	Bob
3	Charlie
4	David

LOGINS	
log_id	name
1	Xavier
2	Andrew
3	Yolanda
4	Bob

Pandas

- To help you keep track, Registrations names' first letters go A,B,C,D

REGISTRATIONS	
reg_id	name
1	Andrew
2	Bob
3	Charlie
4	David

LOGINS	
log_id	name
1	Xavier
2	Andrew
3	Yolanda
4	Bob

Pandas

- First we need to decide **on** what column to merge together.

REGISTRATIONS	
reg_id	name
1	Andrew
2	Bob
3	Charlie
4	David

LOGINS	
log_id	name
1	Xavier
2	Andrew
3	Yolanda
4	Bob

Pandas

- The **on** column should be a *primary* identifier, meaning unique per row.

REGISTRATIONS	
reg_id	name
1	Andrew
2	Bob
3	Charlie
4	David

LOGINS	
log_id	name
1	Xavier
2	Andrew
3	Yolanda
4	Bob

Pandas

- The **on** column should also be present in both tables being merged.

REGISTRATIONS	
reg_id	name
1	Andrew
2	Bob
3	Charlie
4	David

LOGINS	
log_id	name
1	Xavier
2	Andrew
3	Yolanda
4	Bob

Pandas

- Since we assume names are unique here, will we merge **on= "name".**

REGISTRATIONS	
reg_id	name
1	Andrew
2	Bob
3	Charlie
4	David

LOGINS	
log_id	name
1	Xavier
2	Andrew
3	Yolanda
4	Bob

PIERIAN DATA

Pandas

- Next we need to decide **how** to merge the tables **on** the **name** column.

REGISTRATIONS	
reg_id	name
1	Andrew
2	Bob
3	Charlie
4	David

LOGINS	
log_id	name
1	Xavier
2	Andrew
3	Yolanda
4	Bob

PIERIAN DATA

Pandas

- With **how="inner"** the result will be the set of records that match in both tables.

REGISTRATIONS	
reg_id	name
1	Andrew
2	Bob
3	Charlie
4	David

LOGINS	
log_id	name
1	Xavier
2	Andrew
3	Yolanda
4	Bob

Pandas

- With **how= "inner"** the result will be the set of records that **match in both** tables.

REGISTRATIONS	
reg_id	name
1	Andrew
2	Bob
3	Charlie
4	David

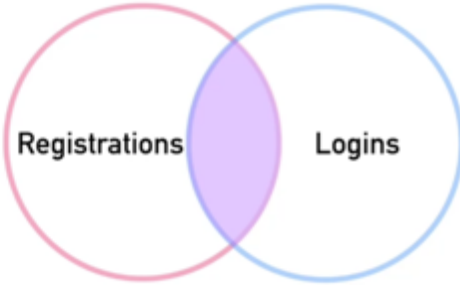
LOGINS	
log_id	name
1	Xavier
2	Andrew
3	Yolanda
4	Bob

Pandas

Merges are often shown as a Venn diagram

pd.merge(registrations,logins,how='inner',on='name')

REGISTRATIONS	
reg_id	name
1	Andrew
2	Bob
3	Charlie
4	David



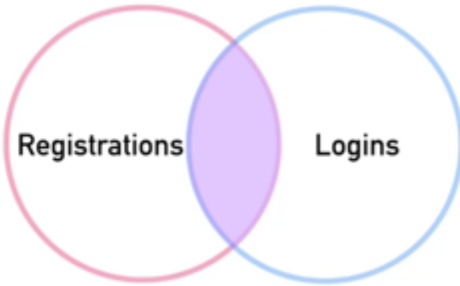
LOGINS	
log_id	name
1	Xavier
2	Andrew
3	Yolanda
4	Bob

PIERIAN DATA

Pandas

pd.merge(registrations,logins,how='inner',on='name')

REGISTRATIONS	
reg_id	name
1	Andrew
2	Bob
3	Charlie
4	David



LOGINS	
log_id	name
1	Xavier
2	Andrew
3	Yolanda
4	Bob

PIERIAN DATA

Pandas

```
pd.merge(registrations,logins,how='inner',on='name')
```

REGISTRATIONS	
reg_id	name
1	Andrew
2	Bob
3	Charlie
4	David

RESULTS		
reg_id	name	log_id
1	Andrew	2
2	Bob	4

LOGINS	
log_id	name
1	Xavier
2	Andrew
3	Yolanda
4	Bob

Pandas

- Now that we understand an “inner” merge, let’s explore “left” versus “right” merge conditions.
- Note! Order of the tables passed in as arguments does matter here!

PIERIAN DATA

Pandas

Let’s explore an **how= “left”** condition with our two example tables.

REGISTRATIONS	
reg_id	name
1	Andrew
2	Bob
3	Charlie
4	David

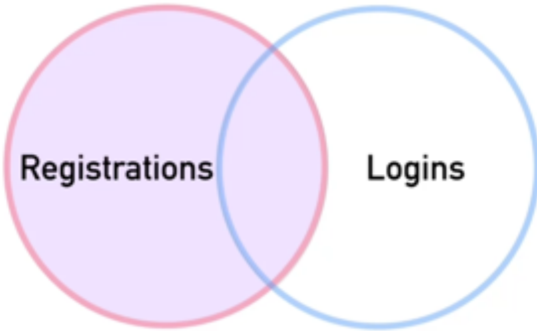
LOGINS	
log_id	name
1	Xavier
2	Andrew
3	Yolanda
4	Bob

PIERIAN DATA

Pandas

```
pd.merge(registrations,logins,how='left',on='name')
```

REGISTRATIONS	
reg_id	name
1	Andrew
2	Bob
3	Charlie
4	David



LOGINS	
log_id	name
1	Xavier
2	Andrew
3	Yolanda
4	Bob

PIERIAN DATA

Pandas

```
pd.merge(registrations,logins,how='left',on='name')
```

REGISTRATIONS	
reg_id	name
1	Andrew
2	Bob
3	Charlie
4	David

RESULTS		
reg_id	name	log_id
1	Andrew	2
2	Bob	4
3	Charlie	NaN
4	David	NaN

LOGINS	
log_id	name
1	Xavier
2	Andrew
3	Yolanda
4	Bob

PIERIAN DATA

Pandas

pd.merge(registrations,logins,how='right',on='name')

REGISTRATIONS	
reg_id	name
1	Andrew
2	Bob
3	Charlie
4	David

RESULTS		
reg_id	name	log_id
1	Andrew	2
2	Bob	4
NaN	Xavier	1
NaN	Yolanda	3

LOGINS	
log_id	name
1	Xavier
2	Andrew
3	Yolanda
4	Bob

Pandas

- Setting **how= “outer”** allows us to include everything present in both tables.

Pandas

* But we have names that only appear in one table!

REGISTRATIONS	
reg_id	name
1	Andrew
2	Bob
3	Charlie
4	David

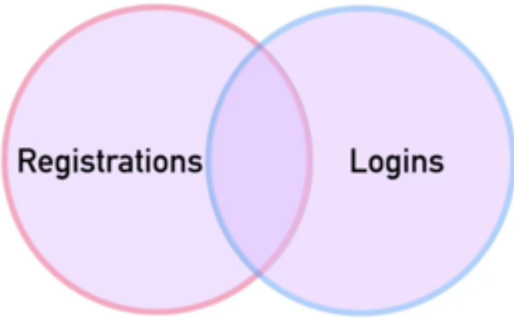
LOGINS	
log_id	name
1	Xavier
2	Andrew
3	Yolanda
4	Bob

PIERIAN DATA

Pandas

pd.merge(registrations,logins,how='outer',on='name')

REGISTRATIONS	
reg_id	name
1	Andrew
2	Bob
3	Charlie
4	David



LOGINS	
log_id	name
1	Xavier
2	Andrew
3	Yolanda
4	Bob

Pandas

```
pd.merge(registrations,logins,how='outer',on='name')
```

REGISTRATIONS	
reg_id	name
1	Andrew
2	Bob
3	Charlie
4	David

RESULTS		
reg_id	name	log_id
1	Andrew	2
2	Bob	4
3	Charlie	NaN
4	David	NaN
NaN	Xavier	1
NaN	Yolanda	3

LOGINS	
log_id	name
1	Xavier
2	Andrew
3	Yolanda
4	Bob

Pandas

- Often text data needs to be cleaned or manipulated for processing.
- While we can always use a custom apply() function for these tasks, pandas comes with many built-in string method calls.
- Let’s learn how to use them!

Pandas

- Basic Python has a **datetime** object containing date and time information.
- Pandas allows us to easily extract information from a datetime object to use feature engineering.

Pandas

- For example, we may have recent timestamped sales data.
- Pandas will allow us to extract information from the timestamp, such as:
  - Day of the Week
  - Weekend vs Weekday
  - AM vs PM

PIERIAN DATA

Pandas

- Pandas can read in data from a wide variety of sources and has excellent online documentation!
- In this series of lectures we will cover some of the most popular ways to read in datasets.

PIERIAN DATA

Pandas

- Note!
    - You need to know the **exact** directory location and correct file name.
    - You may need passwords or permissions for certain data inputs (e.g. a SQL database password).

Pandas

* Final Note:
    * It's almost impossible for us to help with datasets outside the course, since they could be incorrectly formatted, in the wrong location, or have a different name.

Pandas

* Video Lectures:
    * CSV Files
    * HTML Tables
    * Excel Files
    * SQL Databases

Pandas

- Websites display tabular information through the use of HTML tables tags:
    - **<table>**
- Pandas has the ability to automatically convert these HTML tables into a DataFrame.

Pandas

- *Important Notes!*
    - Not every table in a website is available through HTML tables.
    - Some websites may block your computer from scraping the HTML of the site through pandas.
    - It may be more efficient to use an API.

PIERIAN DATA

Pandas

- Let's work through an example of grabbing all the tables from a Wikipedia Article and then cleaning and organizing the information to get a DataFrame.
- Output to an HTML table is also very useful to display tables on a website!

Pandas

- Pandas treats an Excel Workbook as a dictionary, with the key being the sheet name and the value being the DataFrame representing the sheet itself.
- *Note! Using pandas with Excel requires additional libraries!*
- Let's explore how this works!

Pandas

- Pandas can read and write to various SQL engines through the use of a driver and the **sqlalchemy** python library.
- So how does this work?

Pandas

- Step 1:
  - Figure out what SQL Engine you are connecting to, for just a few examples:
    - PostgreSQL
    - MySQL
    - MS SQL Server

Pandas

- Step 2:
  - Install the appropriate Python driver library *(Most likely requires a Google Search)*:
    - PostgreSQL - *psycopg2*
    - MySQL - *pymysql*
    - MS SQL Server - *pyodbc*

Pandas

* Step 3:
    * Use the sqlalchemy library to connect to your SQL database with the driver:
        * docs.sqlalchemy.org/en/13/dialects/index.html

Pandas

* Step 4:
    * Use the sqlalchemy driver connection with pandas read_sql method
    * Pandas can read in entire tables as a DataFrame or actual parse a SQL query through the connection:
        * SELECT * FROM table;

Pandas

- *Important Note!*
  - Use your skills in information lookup to easily find many online resources regarding examples for all of the major SQL engines, for example:
    - Google Search: *Oracle SQL + pandas*

PIERIAN DATA

Pandas

- For our example, we'll use SQLite since it comes with Python and we can easily create a temporary database inside of your RAM.

PIERIAN DATA

Pandas

- Pivot tables allow you to reorganize data, refactoring cells based on columns and a new index.
- This is best shown visually...

Pandas

- A DataFrame with repeated values can be pivoted for a reorganization and clarity

df

	foo	bar	baz	zoo
0	one	A	1	x
1	one	B	2	y
2	one	C	3	z
3	two	A	4	q
4	two	B	5	w
5	two	C	6	t

```
df.pivot(index='foo',
         columns='bar',
         values='baz')
```

bar	A	B	C
foo			
one	1	2	3
two	4	5	6

Pandas

- You should first go through this checklist **before** running a pivot():
  - What question are you trying to answer?
  - What would a dataframe that answers the question look like? Does it need a pivot()
  - What do you want the resulting pivot to look like?

PIERIAN DATA