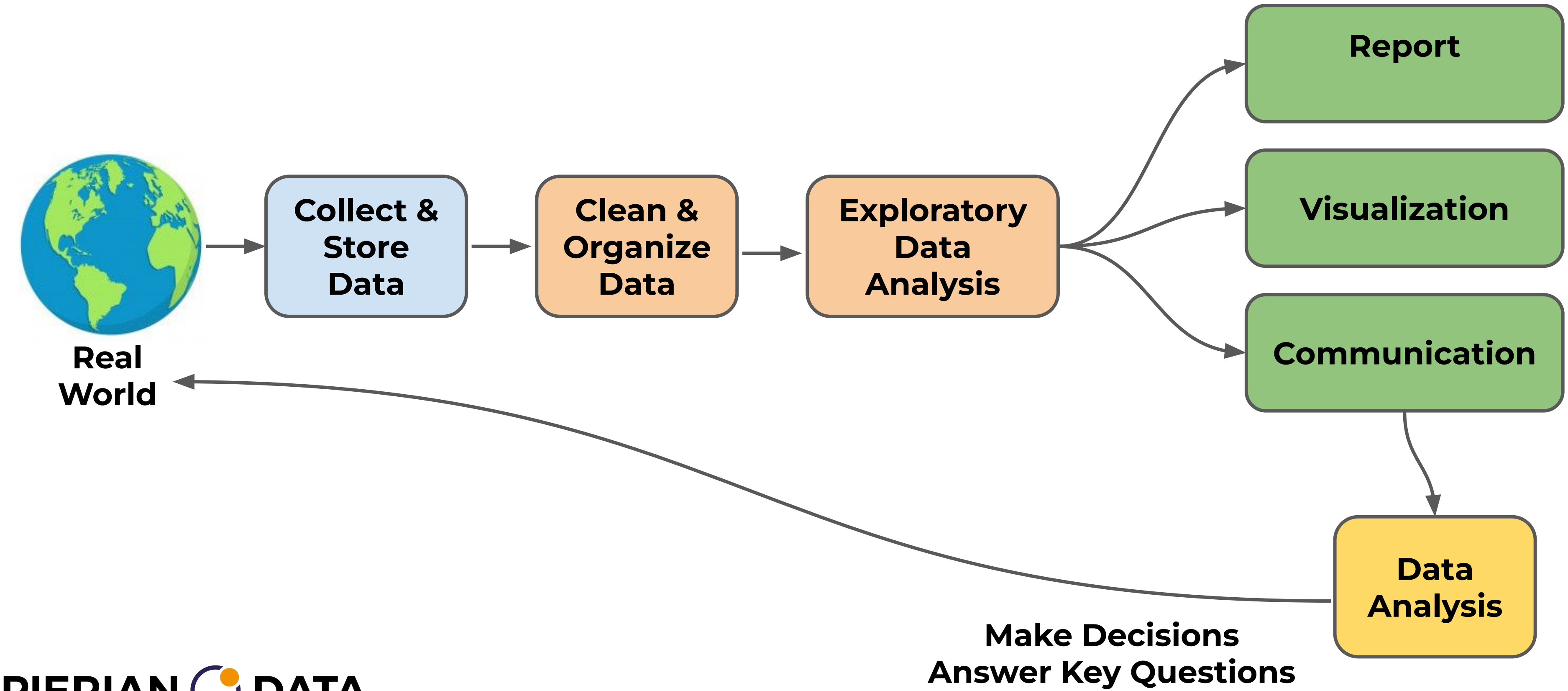
Pandas

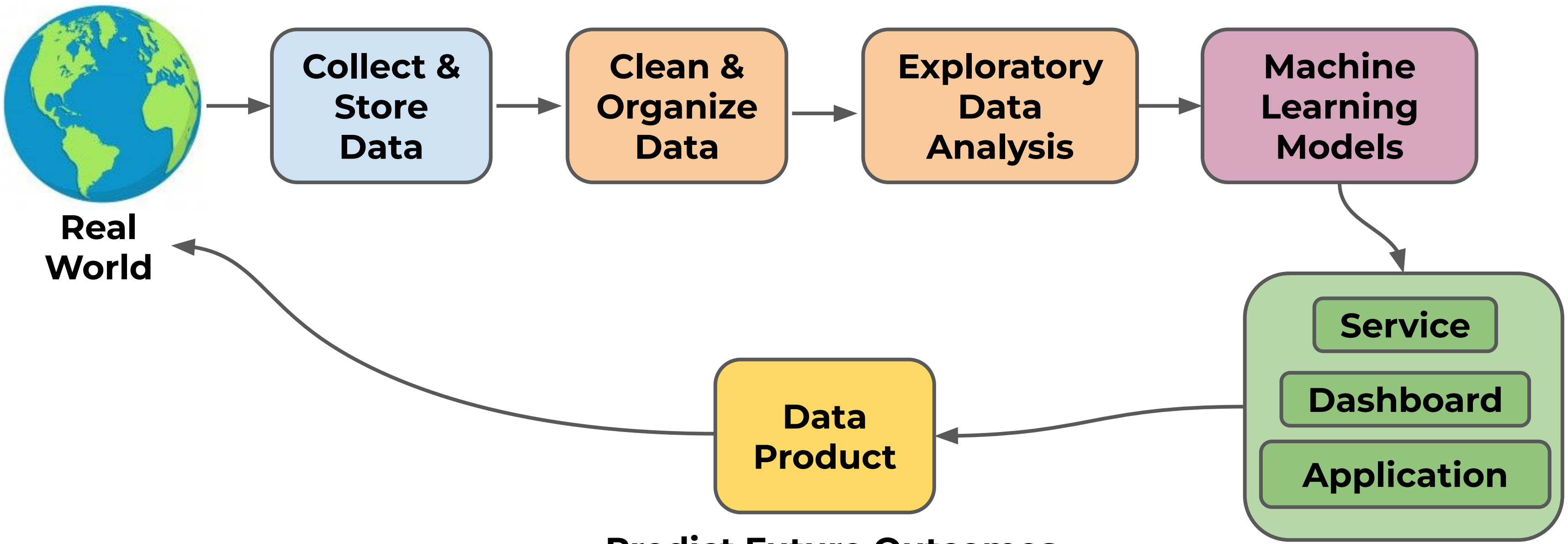
Pandas

- Let's quickly review our ML Pathway...

ML Pathway

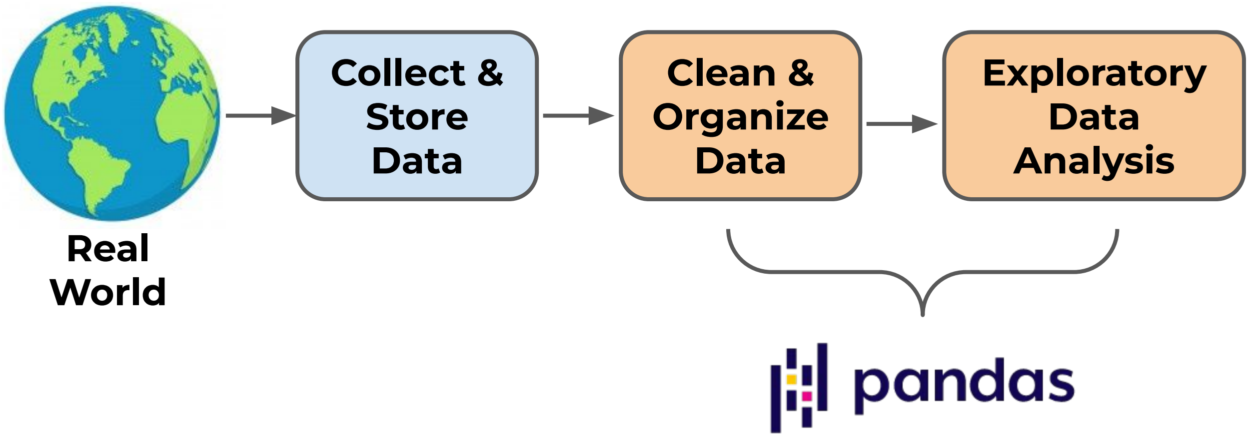

Collect & Store Data
Clean & Organize Data
Exploratory Data Analysis
Report
Visualization
Communication
Real World
Data Analysis
Make Decisions
Answer Key Questions

ML Pathway



Predict Future Outcomes
Gain Insight on Data

ML Pathway


Collect &
Store
Data
Clean &
Organize
Data
Exploratory
Data
Analysis
Real
World
pandas

Pandas

- Pandas is a library for Data Analysis.
- Extremely powerful table (DataFrame) system built off of NumPy.
- Fantastic documentation:
    - https://pandas.pydata.org/docs/

Pandas

- What can we do with Pandas?
  - Tools for reading and writing data between many formats.
  - Intelligently grab data based on indexing,logic, subsetting, and more.
  - Handle missing data.
  - Adjust and restructure data.

Pandas - Section Overview

- Series and DataFrames
- Conditional Filtering and Useful Methods
- Missing Data
- Group By Operations
- Combining DataFrames
- Text Methods and Time Methods
- Inputs and Outputs

Let's get started!

Series

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

0	1776
1	1867
2	1821

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.

Series

PART TWO

DataFrames

Part One

Pandas

- A DataFrame is a table of columns and rows in pandas that we can easily restructure and filter.
- **Formal Definition:** A group of Pandas Series objects that *share* the same index.

Pandas

- Example of a Series

Index	Year
USA	1776
CANADA	1867
MEXICO	1821

Pandas

- Example of Series with Same Index

Index	Year
USA	1776
CANADA	1867
MEXICO	1821

Index	Pop
USA	328
CANADA	38
MEXICO	126

Index	GDP
USA	20.5
CANADA	1.7
MEXICO	1.22

Pandas

- Example of Series with Same Index

Index	Year
USA	1776
CANADA	1867
MEXICO	1821

Index	Pop
USA	328
CANADA	38
MEXICO	126

Index	GDP
USA	20.5
CANADA	1.7
MEXICO	1.22

Pandas

- DataFrame

Index	Year	Pop	GDP
USA	1776	328	20.5
CANADA	1867	38	1.7
MEXICO	1821	126	1.22

Pandas

- DataFrame is the main Pandas object we will work with and it is **extremely** useful!
- This series covers first the “basics”
    - Create a DataFrame
    - Grab a column or multiple columns
    - Grab a row or multiple rows
    - Insert a new column or new row

Pandas

- ***Quick Note: Each video lecture in this DataFrames series refers to the same 01-DataFrames.ipynb notebook!***

DataFrames

Part Two

DataFrames

Part Three

DataFrames

Part Four

Conditional Filtering

Pandas

- Typically in data analysis our datasets are large enough that we don’t filter based on position, but instead based on a **condition.**
- Conditional Filtering allows us to select **rows** based a condition on a column.
- This leads to a discussion on organizing our data...

Pandas

- Organizing Data

Index	Year	Pop	GDP
USA	1776	328	20.5
CANADA	1867	38	1.7
MEXICO	1821	126	1.22

Pandas

- Columns are Features

Index	Year	Pop	GDP
USA	1776	328	20.5
CANADA	1867	38	1.7
MEXICO	1821	126	1.22

Pandas

- Rows are instances of data

Index	Year	Pop	GDP
USA	1776	328	20.5
CANADA	1867	38	1.7
MEXICO	1821	126	1.22

Pandas

* This format is required for ML later on!

Index	Year	Pop	GDP
USA	1776	328	20.5
CANADA	1867	38	1.7
MEXICO	1821	126	1.22

Pandas

- This allows to directly answer questions

Index	Year	Pop	GDP
USA	1776	328	20.5
CANADA	1867	38	1.7
MEXICO	1821	126	1.22

Pandas

- What countries have Pop greater than X?

Index	Year	Pop	GDP
USA	1776	328	20.5
CANADA	1867	38	1.7
MEXICO	1821	126	1.22

Pandas

- What countries have Pop greater than 50?

Index	Year	Pop	GDP
USA	1776	328	20.5
CANADA	1867	38	1.7
MEXICO	1821	126	1.22

Pandas

- df["Pop"]

Index	Year	Pop	GDP
USA	1776	328	20.5
CANADA	1867	38	1.7
MEXICO	1821	126	1.22

Pandas

- df["Pop"] > 50

Index	Year	Pop	GDP
USA	1776	328	20.5
CANADA	1867	38	1.7
MEXICO	1821	126	1.22

Pandas

- df["Pop"] > 50

Index	Year	Pop	GDP
USA	1776	328	20.5
CANADA	1867	38	1.7
MEXICO	1821	126	1.22

Pandas

- df["Pop"] > 50

Index	Year	Pop	GDP
USA	1776	True	20.5
CANADA	1867	False	1.7
MEXICO	1821	True	1.22

Pandas

- df[df[“Pop”] > 50]

Index	Year	Pop	GDP
USA	1776	True	20.5
CANADA	1867	False	1.7
MEXICO	1821	True	1.22

Pandas

- df[df[“Pop”] > 50]

Index	Year	Pop	GDP
USA	1776	True	20.5
MEXICO	1821	True	1.22

Pandas

* Conditional Filtering:
    * Filter by single condition
    * Filter by multiple conditions
    * Check against multiple possible values

Useful Methods

PART ONE - APPLY METHODS

Pandas

- We now understand the basics of how to grab and filter data from a Series or DataFrame in pandas.
- We are now going to cover a wide variety of method calls available in Pandas.
- This will be part of a series of lectures since there are quite a few methods to cover.

Pandas

- For your convenience, the lecture notebook for this series has a list at the top with links that take you directly to the relevant section of the notebook for a topic.

Pandas

- While pandas has many built in methods, we can use the .apply() method call to apply any custom python function of our own to every row in a Series.
- We can use either one or multiple columns as input, let's explore this in the notebook!

Useful Methods

PART TWO - APPLY WITH MULTIPLE COLUMNS

Useful Methods

PART THREE - DESCRIBING AND SORTING

Useful Methods

PART THREE - METHOD CALLS

Missing Data

PART ONE - OVERVIEW

Pandas

- Real world data will often be missing data for a variety of reasons.
- Many machine learning models and statistical methods can not work with missing data points, in which case we need to decide what to do with the missing 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.

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!*

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

- 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 Row
    - Often a good idea to calculate a percentage of what data is 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
    - 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

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%

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	$\boxed{50\%}$
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.

Missing Data

PART TWO - PANDAS

Groupby Operations

Pandas

- A groupby() operation allows us to examine data on a **per category** basis.
- Let’s explore what this looks like in pandas…

Pandas

Category	Data Value
A	10
A	5
B	2
B	4
C	12
C	6

Pandas

Category	Data Value
A	10
A	5
B	2
B	4
C	12
C	6

We need to choose a **categorical** column to call with **groupby()**.

Categorical columns are non-continuous.

Keep in mind, they can still be numerical, such as cabin class categories on a ship (e.g. Class 1, Class 2, Class 3)

Pandas

Category	Data Value
A	10
A	5
B	2
B	4
C	12
C	6

Let's now see what happens with a .groupby() call combined with an aggregate function call.

Pandas

Category	Data Value
A	10
A	5
B	2
B	4
C	12
C	6

A	10
A	5

B	2
B	4

C	12
C	6

Pandas

Category	Data Value
A	10
A	5
B	2
B	4
C	12
C	6

A	10
A	5

B	2
B	4

C	12
C	6

Aggregate Function
.sum()

Category	Result
A	15
B	6
C	18

Pandas

Category	Data Value
A	10
A	5
B	2
B	4
C	12
C	6

A	10
A	5

B	2
B	4

C	12
C	6

Aggregate Function
.mean()

Category	Result
A	7.5
B	3
C	9

Pandas

Category	Data Value
A	10
A	5
B	2
B	4
C	12
C	6

A	10
A	5

B	2
B	4

C	12
C	6

Aggregate Function
.count()

Category	Result
A	2
B	2
C	2

Pandas

- Note that in pandas calling **groupby()** by itself creates a "*lazy*" groupby object waiting to be evaluated by an aggregate method call.
- Let's explore this further in pandas!

Groupby Operations

MULTI-LEVEL INDEX CONTINUED...

Combining DataFrames

Concatenation

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

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

	Year	Pop	GDP
USA	1776	328	20.5
CANADA	1867	38	1.7
BRAZIL	1822	209	1.86
MEXICO	1821	126	1.22

Pandas

- Pandas will also automatically fill NaN where necessary.
- Let's explore some examples in Pandas!

Combining DataFrames

"Inner" Merge

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

* For the sake of simplicity, we will assume the names are unique.

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

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

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

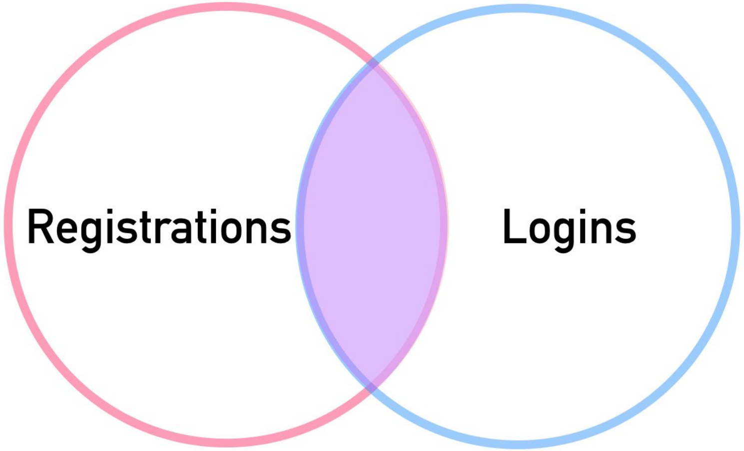
LOGINS	
log_id	name
1	Xavier
2	Andrew
3	Yolanda
4	Bob

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

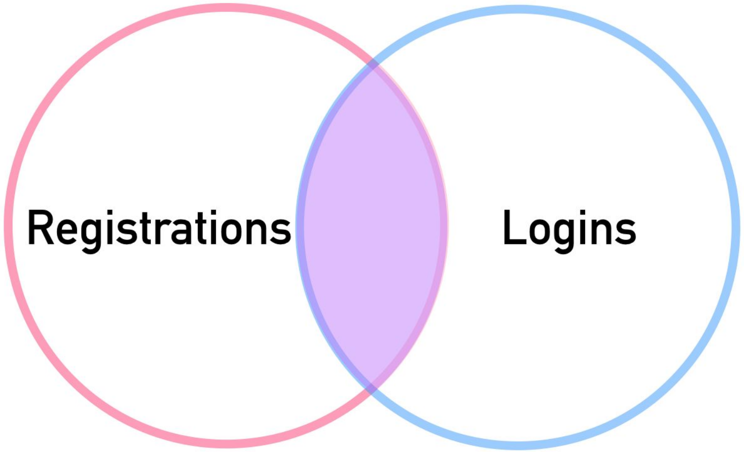


LOGINS	
log_id	name
1	Xavier
2	Andrew
3	Yolanda
4	Bob

Pandas

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

REGISTRATIONS	
reg_id	name
1	Andrew
2	Bob
3	Charlie
4	David

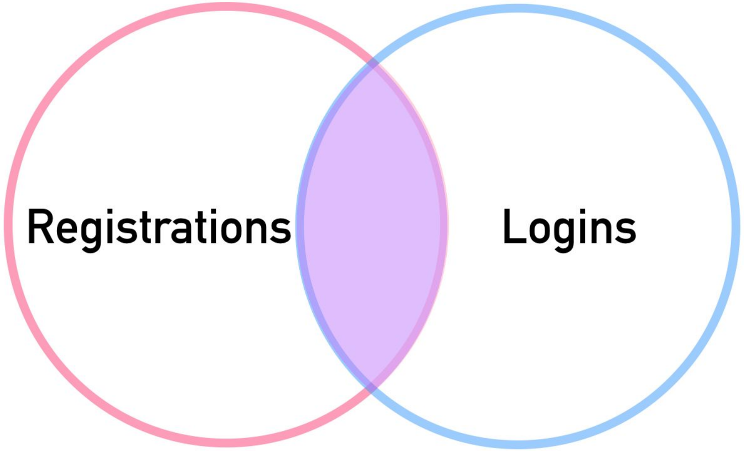


LOGINS	
log_id	name
1	Xavier
2	Andrew
3	Yolanda
4	Bob

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

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

PIERIAN DATA

Pandas

* Let’s quickly explore this in pandas!

Combining DataFrames

"left" and "right" merge

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!

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

Pandas

Note: Registrations is the left table, logins will be the right table

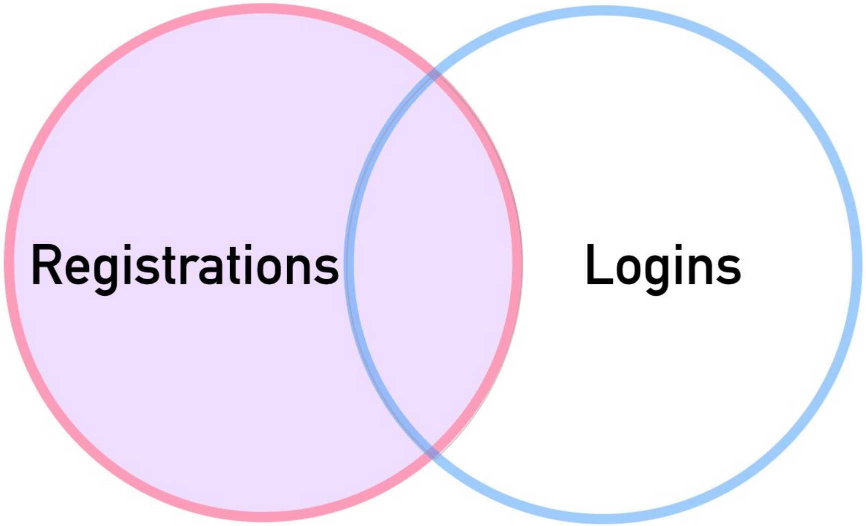
REGISTRATIONS	
reg_id	name
1	Andrew
2	Bob
3	Charlie
4	David

LOGINS	
log_id	name
1	Xavier
2	Andrew
3	Yolanda
4	Bob

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

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

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

Pandas

* Now let's see what happens in a how="right" situation.

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

* Let’s explore this further in pandas!

Combining DataFrames

"outer" merge

Pandas

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

Pandas

- Recall we match Andrew and Bob 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

* 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

Pandas

- We can use **how= “outer”** to make sure we grab all names from both tables.

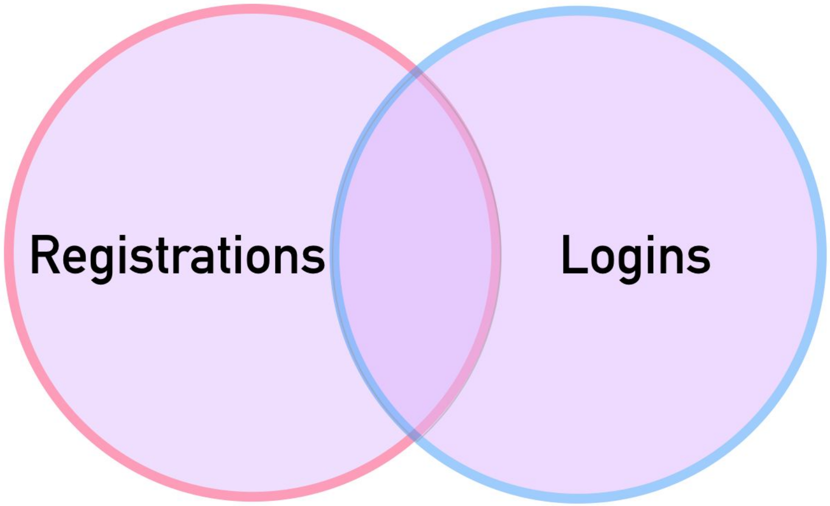
REGISTRATIONS	
reg_id	name
1	Andrew
2	Bob
3	Charlie
4	David

LOGINS	
log_id	name
1	Xavier
2	Andrew
3	Yolanda
4	Bob

Pandas

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

- Let's quickly explore this result in pandas!

Combining DataFrames

Joining on Index and Different Key Names

Text Methods

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!

Time Methods

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

Data Input and Output

CSV Files

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.

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

Data Input and Output

HTML Tables

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.

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!

Data Input and Output

Excel Files

Pandas

- Pandas can read and write to Excel files.
- *Important Note!*
    - Pandas can only read and write in raw data, it is not able to read in macros,visualizations, or formulas created inside of spreadsheets.

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!

Data Input and Output

SQL

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!*
    * It’s almost impossible for us to help with your specific work databases outside of the course material, since it requires knowledge of your permissions, database names and locations, and password information!

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*

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.

Pivot Tables

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

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

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

Pandas

- We choose columns to define the new index,columns, and values.

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

- Notice how the choices for index and column should have repeated values.

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

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

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

Pandas

- Also notice how all the information from the **zoo** column is now discarded.

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

- No new information is shown, it is merely reorganized.

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

- Note!
    - It does not make sense to pivot every DataFrame, all of the datasets used in this course will have no need for a pivot table operation to use with machine learning.

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?

Pandas

- Pandas also comes with a pivot_table method that allows for an additional aggregation function to be called.
- This could alternatively be done with a groupby() method call as well.
- Let's explore both .pivot() and pivot_table() methods in pandas!

Pivot Tables

Pandas Section Exercise - Overview

Pandas

- Let’s test all your new pandas skills!
- Keep in mind:
    - Most questions can be solved in one or two lines of pandas code.
    - There could be multiple correct solutions.
    - Be careful not to run the cell above the expected output.

Pandas Section Exercise - Solutions