Elements Of Data Science - S2022

Week 12: ETL and Databases

4/26/2022

TODOs

- Quiz 12, **No need to submit, but please review and see if you will be able to do it, potentially there might be questions from this quiz in the final exam??**
- Final
    - Release Tuesday May 10th at 10AM EST
    - **Due Tuesday May 10th, 9:59pm EST**
    - Have 12 hours after starting exam to finish
    - 30-40 questions (fill in the blank/multiple choice/short answer)
    - Onine in Gitbhub
    - Open-book, open-notes, open-python
    - Questions asked/answered **privately** via email

Today

- ETL process
- Relational DBs and SQL
- Connecting to databases with sqlalchemy and pandas
- Review for the final

Questions re Logistics?

Environment Setup

In [1]:

```
import numpy
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

sns.set_style('darkgrid')
%matplotlib inline
```

Data Processing and Delivery: ETL

- **E**xtract **T**ransform **L**oad
- Extract: Reading in data
- Transform: Transforming data
- Load: Delivering data

Extract: Various Data Sources

- flatfiles (csv, excel)
- semi-structured documents (json, html)
- unstructured documents
- data + schema (dataframe,database, parquet)
- APIs (wikipedia, twitter, spotify, etc.)
- databases

- Pandas to the rescue!
- Plus other specialized libraries

Extracting Data with Pandas

- read_csv
- read_excel
- read_parquet
- read_json
- read_html
- read_sql
- read_clipboard
- ...

Extract Data: CSV

Comma Separated Values

In [57]:

```
%cat ../data/example.csv
```

```
Year,Make,Model,Description,Price

1997,Ford,E350,"ac, abs, moon",3000.00

1999,Chevy,"Venture Extended Edition","",490
0.00

1999,Chevy,"Venture Extended Edition, Very L
arge",,5000.00
```

```
1996,Jeep,Grand Cherokee,"MUST SELL! air, mo
on roof, loaded",4799.00
```

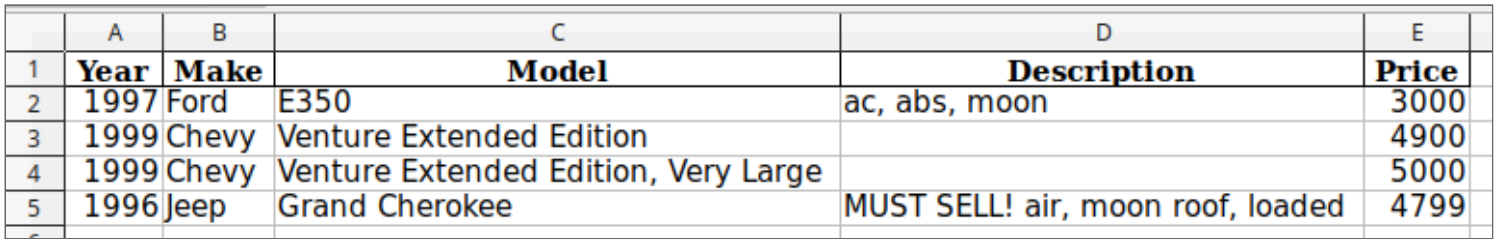
In [58]:

```
df = pd.read_csv('../data/example.csv',header=0,sep=',')
df.head()
```

Out[58]:

	Year	Make	Model	Description	Price
0	1997	Ford	E350	ac, abs, moon	3000.0
1	1999	Chevy	Venture Extended Edition	NaN	4900.0
2	1999	Chevy	Venture Extended Edition, Very Large	NaN	5000.0
3	1996	Jeep	Grand Cherokee	MUST SELL! air, moon roof, loaded	4799.0

Extract Data: Excel



	A	B	C	D	E
1	**Year**	**Make**	**Model**	**Description**	**Price**
2	1997	Ford	E350	ac, abs, moon	3000
3	1999	Chevy	Venture Extended Edition		4900
4	1999	Chevy	Venture Extended Edition, Very Large		5000
5	1996	Jeep	Grand Cherokee	MUST SELL! air, moon roof, loaded	4799

In [59]:

```
pd.read_excel('../data/example.xls')
```

Out[59]:

	Year	Make	Model	Description	Price
0	1997	Ford	E350	ac, abs, moon	3000
1	1999	Chevy	Venture Extended Edition	NaN	4900
2	1999	Chevy	Venture Extended Edition, Very Large	NaN	5000

	Year	Make	Model	Description	Price
3	1996	Jeep	Grand Cherokee	MUST SELL! air, moon roof, loaded	4799

Extract Data: Parquet

- open source column-oriented data storage
- part of the Apache Hadoop ecosystem
- often used when working with Spark
- requires additional parsing engine eg `pyarrow`
- includes both data and **schema**
- **Schema** : metadata about the dataset (column names, datatypes, etc.)

In [60]:

```
# conda install -n eods-s21 pyarrow
pd.read_parquet('../data/example.parquet')
```

Out[60]:

	Year	Make	Model	Description	Price
0	1997	Ford	E350	ac, abs, moon	3000.0

	Year	Make	Model	Description	Price
1	1999	Chevy	Venture Extended Edition	None	4900.0
2	1999	Chevy	Venture Extended Edition, Very Large	None	5000.0
3	1996	Jeep	Grand Cherokee	MUST SELL! air, moon roof, loaded	4799.0

Extract Data: JSON

- **J**ava**S**cript **O**bject **N**otation
- often seen as return from api call
- looks like a dictionary or list of dictionaries
- pretty print using json.loads(json_string)

```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 27,
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    },
  ],
  "children": [],
  "spouse": null
}
```

Extract Data: JSON

In [61]:

```
json = """
{"0": {"Year": 1997,
  "Make": "Ford",
  "Model": "E350",
  "Description": "ac, abs, moon",
  "Price": 3000.0},
 "1": {"Year": 1999,
  "Make": "Chevy",
  "Model": "Venture Extended Edition",
  "Description": null,
  "Price": 4900.0},
 "2": {"Year": 1999,
  "Make": "Chevy",
  "Model": "Venture Extended Edition, Very Large",
  "Description": null,
  "Price": 5000.0},
 "3": {"Year": 1996,
  "Make": "Jeep",
  "Model": "Grand Cherokee",
  "Description": "MUST SELL! air, moon roof, loaded",
  "Price": 4799.0}}
"""
```

In [62]:

```
pd.read_json(json,orient='index')
```

Out[62]:

	Year	Make	Model	Description	Price
0	1997	Ford	E350	ac, abs, moon	3000
1	1999	Chevy	Venture Extended Edition	None	4900
2	1999	Chevy	Venture Extended Edition, Very Large	None	5000
3	1996	Jeep	Grand Cherokee	MUST SELL! air, moon roof, loaded	4799

Extract Data: HTML

- **H**yper**T**ext **M**arkup **L**anguage
- Parse with BeautifulSoup

In [63]:

```
html = """
<html>
    <head>
        <title>Example</title>
    </head>
    <body>
        <p id="first" class="example"><strong>Example text!</strong></p>
        <p id="second" class="example">And More!</p>
    </body>
</html>
"""

from bs4 import BeautifulSoup

soup = BeautifulSoup(html)
[p.text for p in soup('p')]
```

Out[63]:

```
['Example text!', 'And More!']
```

Extract Data: APIs

- **A**pplication **P**rogramming **I**nterface
- defines interactions between software components and resourses
- most datasources have an API
- some require authentication
- python libraries exist for most common APIs

- **requests**: library for making web requests and accessing the results

API Example: Wikipedia

In [64]:

```
import requests
url = 'http://en.wikipedia.org/w/api.php?action=query&prop=info&format=json&titles='
title = 'Data Science'
title = title.replace(' ','%20')
print(url+title)
```

```
http://en.wikipedia.org/w/api.php?action=que
ry&prop=info&format=json&titles=Data%20Scien
ce
```

In [65]:

```
resp = requests.get(url+title)
resp.json()
```

Out[65]:

```
{'batchcomplete': '',
 'query': {'pages': {'49495124': {'pageid':
```

```
49495124,
    'ns': 0,
    'title': 'Data Science',
    'contentmodel': 'wikitext',
    'pagelanguage': 'en',
    'pagelanguagehtmlcode': 'en',
    'pagelanguagedir': 'ltr',
    'touched': '2021-11-13T20:22:16Z',
    'lastrevid': 706007296,
    'length': 26,
    'redirect': '',
    'new': ''}}}}
```

In [66]:

```
resp.text
```

Out[66]:

```
'{"batchcomplete":"","query":{"pages":{"4949
5124":{"pageid":49495124,"ns":0,"title":"Dat
a Science","contentmodel":"wikitext","pagela
```

nguage":"en","pagelanguagehtmlcode":"en","pa
gelanguagedir":"ltr","touched":"2021-11-13T2
0:22:16Z","lastrevid":706007296,"length":2
6,"redirect":"","new":""}}}}'

API Example: Twitter

1. **Apply for Twitter developer account**
2. **Create a Twitter application to generate tokens and secrets**

In [67]:

```
with open('/home/bgibson/proj/twitter/twitter_consumer_key.txt') as f:
    consumer_key = f.read().strip()
with open('/home/bgibson/proj/twitter/twitter_consumer_secret.txt') as f:
    consumer_secret = f.read().strip()
with open('/home/bgibson/proj/twitter/twitter_access_token.txt') as f:
    access_token = f.read().strip()
with open('/home/bgibson/proj/twitter/twitter_access_token_secret.txt') as f:
    access_token_secret = f.read().strip()

from twython import Twython
twitter = Twython(consumer_key,consumer_secret,access_token,access_token_secret)
```

In [68]:

```
public_tweets = twitter.search(q='columbia')['statuses']
for status in public_tweets[:3]:
    print('-------')
    print(status["text"])
```

```
-------
James Monroe is an American politician and a
ctor who served as the most important invent
ion of the original District of Columbia, th
e (1/2)
-------
And not Columbia, TN… ya small minded gyals
```

```
-------
RT @ellisemmagarey: Workers at @Columbia are
on their SIXTH week of strike &amp;amp; face mas
s retaliatory firings. This is their FOURTH
strike in…
```

API Example: Twitter

In [69]:

```
public_tweets[0]
```

Out[69]:


```
{'created_at': 'Mon Dec 06 19:57:38 +0000 20
21',
 'id': 1467946353493872643,
 'id_str': '1467946353493872643',
 'text': 'James Monroe is an American politi
cian and actor who served as the most import
ant invention of the original District of Co
lumbia, the (1/2)',
 'truncated': False,
 'entities': {'hashtags': [], 'symbols': [],
'user_mentions': [], 'urls': []},
```

```
 'metadata': {'iso_language_code': 'en', 're
sult_type': 'recent'},
 'source': '<a href="http://www.github.com/l
dermer/" rel="nofollow">dunning kruger bot</
a>',
 'in_reply_to_status_id': None,
 'in_reply_to_status_id_str': None,
 'in_reply_to_user_id': None,
 'in_reply_to_user_id_str': None,
 'in_reply_to_screen_name': None,
 'user': {'id': 739988612243062784,
  'id_str': '739988612243062784',
  'name': 'dunningkrugerbot',
  'screen_name': 'bottingkruger',
  'location': 'Seattle, WA',
  'description': "Ask me about a person, and
I'll tell you everything I think I know. I o
nly read Wikipedia, but I think I got this.
```

```
Maintained by @lauriedermer.",
  'url': None,
  'entities': {'description': {'urls': []}},
  'protected': False,
  'followers_count': 71,
  'friends_count': 1,
  'listed_count': 8,
  'created_at': 'Tue Jun 07 01:13:28 +0000 2
016',
  'favourites_count': 5,
  'utc_offset': None,
  'time_zone': None,
  'geo_enabled': False,
  'verified': False,
  'statuses_count': 113407,
  'lang': None,
  'contributors_enabled': False,
  'is_translator': False,
```

```
  'is_translation_enabled': False,
  'profile_background_color': '000000',
  'profile_background_image_url': 'http://ab
s.twimg.com/images/themes/theme1/bg.png',
  'profile_background_image_url_https': 'htt
ps://abs.twimg.com/images/themes/theme1/bg.p
ng',
  'profile_background_tile': False,
  'profile_image_url': 'http://pbs.twimg.co
m/profile_images/776678331198472192/-wdccnf1
_normal.jpg',
  'profile_image_url_https': 'https://pbs.tw
img.com/profile_images/776678331198472192/-w
dccnf1_normal.jpg',
  'profile_link_color': '7B9095',
  'profile_sidebar_border_color': '000000',
  'profile_sidebar_fill_color': '000000',
  'profile_text_color': '000000',
```

```
 'profile_use_background_image': False,
 'has_extended_profile': True,
 'default_profile': False,
 'default_profile_image': False,
 'following': False,
 'follow_request_sent': False,
 'notifications': False,
 'translator_type': 'none',
 'withheld_in_countries': []},
'geo': None,
'coordinates': None,
'place': None,
'contributors': None,
'is_quote_status': False,
'retweet_count': 0,
'favorite_count': 0,
'favorited': False,
```

```
'retweeted': False,
'lang': 'en'}
```

Transforming Data

- Standardization
- Creating dummy variables
- Filling missing data
- One-Hot-Encoding
- Binning
- Parsing natural language
- Dimensionality reduction
- etc...
- Pipeline and ColumnTransformer

Extract and Transform Example: Titanic

In [70]:

```
titanic_url = ('https://raw.githubusercontent.com/amueller/'
               'scipy-2017-sklearn/091d371/notebooks/datasets/titanic3.csv')
df_titanic = pd.read_csv(titanic_url)[['age','fare','embarked','sex','pclass','survived']]
display(df_titanic.head(1))
df_titanic.info()
```

	age	fare	embarked	sex	pclass	survived
0	29.0	211.3375	S	female	1	1

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1309 entries, 0 to 1308
Data columns (total 6 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       1046 non-null   float64
 1   fare      1308 non-null   float64
```

```
 2   embarked  1307 non-null   object
 3   sex       1309 non-null   object
 4   pclass    1309 non-null   int64
 5   survived  1309 non-null   int64
dtypes: float64(2), int64(2), object(2)
memory usage: 61.5+ KB
```

In [71]:

```
X_titanic = df_titanic.drop('survived', axis=1)
y_titanic = df_titanic['survived']
X_titanic_train, X_titanic_test, y_titanic_train, y_titanic_test = train_test_split(X_titanic,
                                                                                    y_titanic,
                                                                                    test_size=0.2,
                                                                                    random_state=42)
```

Extract and Transform Example: Titanic

In [72]:

```
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder,StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression

numeric_features = ['age', 'fare']
numeric_transformer = Pipeline(steps=[('imputer', SimpleImputer(strategy='median')),
                                      ('scaler', StandardScaler())])

categorical_features = ['embarked', 'sex', 'pclass']
categorical_transformer = Pipeline(steps=[('imputer', SimpleImputer(strategy='constant', fill_value='missing')),
                                          ('onehot', OneHotEncoder(handle_unknown='ignore'))])

preprocessor = ColumnTransformer(transformers=[('num', numeric_transformer, numeric_features),
                                               ('cat', categorical_transformer, categorical_features)])

pipe = Pipeline(steps=[('preprocessor', preprocessor),
                       ('classifier', LogisticRegression(solver='lbfgs', random_state=42))])

param_grid = {
    'preprocessor__num__imputer__strategy': ['mean', 'median'],
    'classifier__C': [0.1, 1.0, 10, 100],
}
gs_pipeline = GridSearchCV(pipe, param_grid, cv=3)
gs_pipeline.fit(X_titanic_train, y_titanic_train)
print("best test set score from grid search: {:.3f}".format(gs_pipeline.score(X_titanic_test, y_titanic_test)))
print("best parameter settings: {}".format(gs_pipeline.best_params_))
```

```
best test set score from grid search: 0.771
best parameter settings: {'classifier__C': 1
00, 'preprocessor__num__imputer__strategy':
'median'}
```

Loading Data with pandas

- to_csv
- to excel
- to_json
- to_html
- to_parquet
- to_sql
- to_clipboard
- to_pickle

Data Processing Summary

- ETL
- reading datafiles using pandas
- website scraping (requests,BeautifulSoup)
- accessing data via API
- Tranforming data with Pipelines
- Loading data with pandas

Questions re Data Processing and Delivery?

Environment Setup

In [1]:

```
import numpy
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from mlxtend.plotting import plot_decision_regions

sns.set_style('darkgrid')
%matplotlib inline
```

Accessing Databases with Python

- databases vs flat-files
- Relational Databases and SQL
- NoSQL databases

Flat Files

Company Details

E_ID	Name	Department	Dept_ID	Manager_Name
101	Anoop	Accounts	AC-10	Mr Gagan Thakral
201	Anurag	Accounts	AC-10	Mr Gagan Thakral
301	Rakesh	Accounts	AC-10	Mr Gagan Thakral
401	Saurav	Accounts	AC-10	Mr Gagan Thakral

- eg: csv, json, etc
- Pros
    - Ease of access
    - Simple to transport
- Cons
    - May include redundant information
    - Slow to search

- No integity checks

Relational Databases

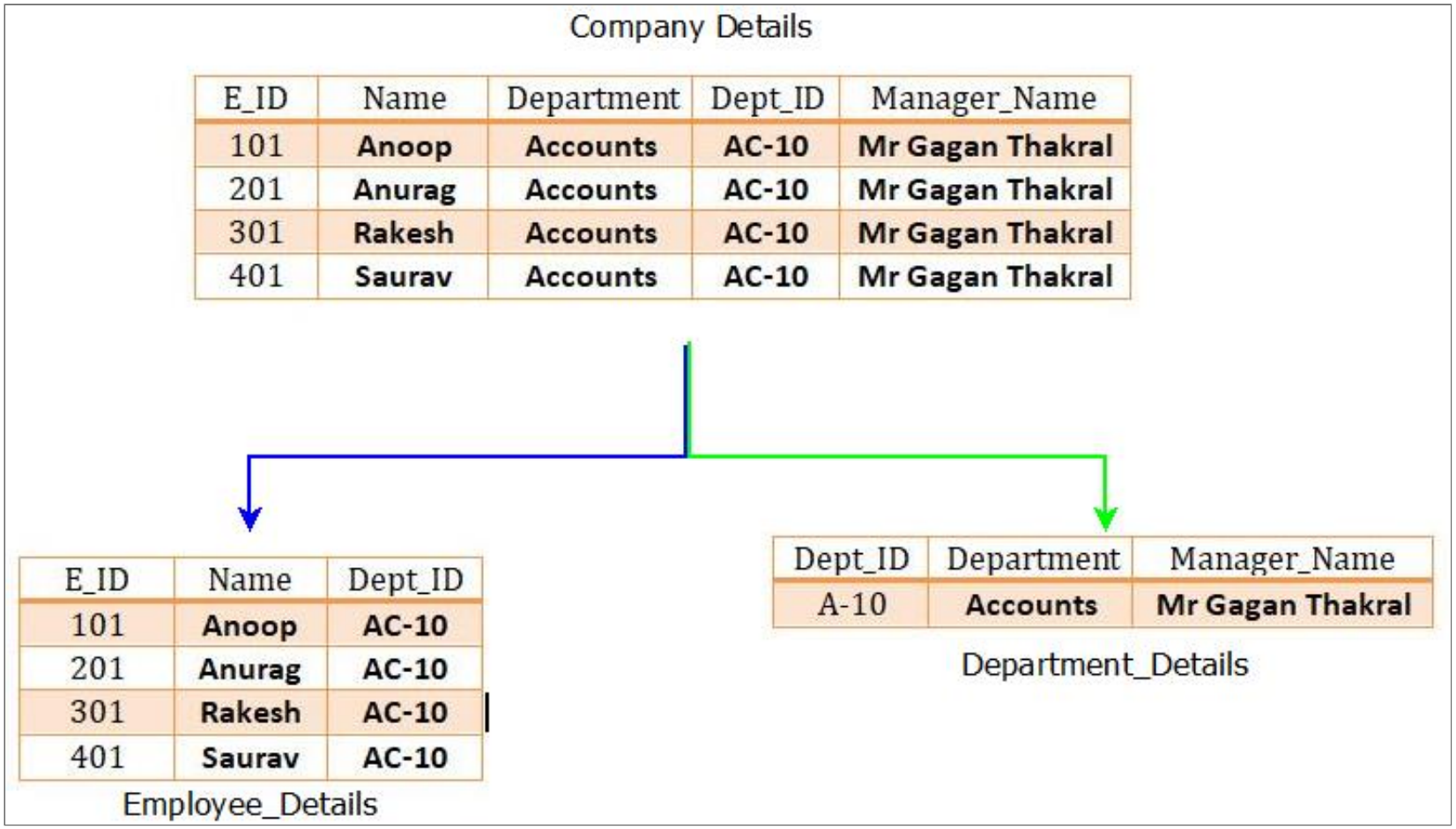
- Data stored in **tables** (rows/columns)
- Table columns have well defined datatype requirements
- Complex **indexes** can be set up over often used data/searches
- Row level security, separate from the operating system
- Related data is stored in separate tables, referenced by **keys**
- Many commonly used Relational Databases
    - sqlite (small footprint db, might already have it installed)
    - Mysql
    - PostgreSQL
    - Microsoft SQL Server
    - Oracle

Database Normalization

- Organize data in accordance with **normal forms**
- Rules designed to:
    - reduce data redundancy
    - improve data integrity

- Rules like:
    - Has Primary Key
    - No repeating groups
    - Cells have single values
    - No partial dependencies on keys (use whole key)
    - ...

Database Normalization



From **https://www.minigranth.com/dbms-tutorial/database-normalization-dbms/**

De-Normalization

- But we want a single table/dataframe!
- Very often need to **denormalize**
- .. using joins! (see more later)

Structured Query Language (SQL)

- (Semi) standard language for querying, transforming and returning data
- Notable characteristics:
    - generally case independent
    - white-space is ignored
    - strings denoted with single quotes
    - comments start with double-dash "--"

```
SELECT
    client_id
    ,lastname
FROM
    company_db.bi.clients --usually database.schema.table
WHERE
    lastname LIKE 'Gi%'   --only include rows with lastname starting with Gi
LIMIT 10
```

Small but Powerful DB: SQLite3

- likely already have it installed
- many programs use it to store configurations, history, etc
- good place to play around with sql

```
bgibson@civet:~$ sqlite3
SQLite version 3.22.0 2018-01-22 18:45:57
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite>
```

Accessing Relational DBs: `sqlalchemy`

- flexible library for accessing a variety of sql dbs
- can use to query through pandas itself to retrieve a dataframe

In [2]:

```
import sqlalchemy

# sqlite sqlalchemy relative path syntax: 'sqlite:///[path to database file]'
engine = sqlalchemy.create_engine('sqlite:///../data/example_business.sqlite')

# read all records from the table sales
sql = """
SELECT
    *
FROM
    clients
"""

pd.read_sql(sql,engine)
```

Out[2]:

	client_id	firstname	lastname	home_address_id
0	102	Mikel	Rouse	1002
1	103	Laura	Gibson	1003

	client_id	firstname	lastname	home_address_id
2	104	None	Reeves	1003
3	105	Scott	Payseur	1004

SQL: SELECT

In [3]:

```
sql="""
SELECT
    client_id
    ,lastname
FROM
    clients
"""

pd.read_sql(sql,engine)
```

Out[3]:

	client_id	lastname
0	102	Rouse
1	103	Gibson
2	104	Reeves
3	105	Payseur

SQL: * (wildcard)

In [4]:

```
sql="""
SELECT
    *
FROM
    clients
"""
clients = pd.read_sql(sql,engine)
clients
```

Out[4]:

	client_id	firstname	lastname	home_address_id
0	102	Mikel	Rouse	1002
1	103	Laura	Gibson	1003
2	104	None	Reeves	1003
3	105	Scott	Payseur	1004

In [5]:

```
sql="""
SELECT
    *
FROM
```

```
    addresses
"""
addresses = pd.read_sql(sql,engine)
addresses
```

Out[5]:

	address_id	address
0	1002	1 First Ave.
1	1003	2 Second Ave.
2	1005	3 Third Ave.

SQL: LIMIT

In [6]:

```
sql="""
SELECT
    *
FROM
    clients
LIMIT 2
"""
pd.read_sql(sql,engine)
```

Out[6]:

	client_id	firstname	lastname	home_address_id
0	102	Mikel	Rouse	1002
1	103	Laura	Gibson	1003

SQL: WHERE

In [7]:

```
sql = """
SELECT
    *
FROM
    clients
WHERE home_address_id = 1003
"""

pd.read_sql(sql,engine)
```

Out[7]:

	client_id	firstname	lastname	home_address_id
0	103	Laura	Gibson	1003
1	104	None	Reeves	1003

SQL: LIKE and %

In [21]:

```
sql = """
SELECT
    *
FROM
    clients
WHERE (home_address_id = 1003) AND (lastname LIKE 'Gi%')
"""

pd.read_sql(sql,engine)
```

Out[21]:

	client_id	firstname	lastname	home_address_id
0	103	Laura	Gibson	1003

SQL: AS alias

In [9]:

```
sql="""
SELECT
    client_id AS CID
    ,lastname AS Lastname
FROM
    clients AS ca
"""

pd.read_sql(sql,engine)
```

Out[9]:

	CID	Lastname
0	102	Rouse
1	103	Gibson
2	104	Reeves
3	105	Payseur

SQL: (INNER) JOIN

In [10]:

```
sql="""
SELECT
    c.firstname
    ,a.address
FROM clients AS c
JOIN addresses AS a ON c.home_address_id = a.address_id
WHERE c.firstname IS NOT NULL
"""

pd.read_sql(sql,engine)
```

Out[10]:

	firstname	address
0	Mikel	1 First Ave.
1	Laura	2 Second Ave.

SQL: LEFT JOIN

In [11]:

```
sql="""
SELECT
    c.firstname,a.address
FROM clients AS c
LEFT JOIN addresses AS a ON c.home_address_id = a.address_id
WHERE c.firstname IS NOT NULL
"""

pd.read_sql(sql,engine)
```

Out[11]:

	firstname	address
0	Mikel	1 First Ave.
1	Laura	2 Second Ave.
2	Scott	None

SQL: RIGHT JOIN

In [12]:

```
# this will cause an error in pandas, right join not supported in sqlalchemy + sqlite3
sql="""
SELECT
    c.firstname,a.address
FROM clients AS c
RIGHT JOIN addresses AS a ON c.home_address_id = a.address_id
"""
#pd.read_sql(sql,engine)
```

In [13]:

```
pd.merge(clients,addresses,left_on='home_address_id',right_on='address_id',how='right')[['firstname','address']]
```

Out[13]:

	firstname	address
0	Mikel	1 First Ave.
1	Laura	2 Second Ave.
2	None	2 Second Ave.
3	NaN	3 Third Ave.

SQL: FULL OUTER JOIN

In [14]:

```
# this will cause an error in pandas, outer join not supported in sqlalchemy + sqlite3
sql="""
SELECT
    c.firstname,a.address
FROM clients AS c
OUTER JOIN addresses AS a ON c.home_address_id = a.address_id
"""
#pd.read_sql(sql,engine)
```

In [15]:

```
pd.merge(clients,addresses,left_on='home_address_id',right_on='address_id',how='outer')[['firstname','address']]
```

Out[15]:

	firstname	address
0	Mikel	1 First Ave.
1	Laura	2 Second Ave.
2	None	2 Second Ave.
3	Scott	NaN
4	NaN	3 Third Ave.

SQL: And Much More!

- Multiple Joins
- DISTINCT
- COUNT
- ORDER BY
- GROUP BY
- Operators (string concatenate operator is '||' in sqlite)
- Subqueries
- HAVING
- see **Data Science From Scratch Ch. 23**

pandasql

- allows for querying of pandas DataFrames using SQLite syntax
- good way to practice SQL without a database

In [16]:

```
from pandasql import PandaSQL

# set up an instance of PandaSQL to pass SQL commands to
pysqldf = PandaSQL()
```

In [17]:

```
sql = """
SELECT
    c.firstname,a.address
FROM clients AS c
JOIN addresses AS a ON c.home_address_id = a.address_id
"""
pysqldf(sql)
```

Out[17]:

	firstname	address
0	Mikel	1 First Ave.
1	Laura	2 Second Ave.

	firstname	address
2	None	2 Second Ave.

NoSQL

- Anything that isn't traditional SQL/RDBMS
    - key-value (Redis,Berkely DB)
    - document store (MongoDB, DocumentDB)
    - wide column (Cassandra, HBase, DynamoDB)
    - graph (Neo4j)

- Rapidly growing field to fit needs
- Probably more as we speak

Example: Mongo

- records represented as documents (think json)
- very flexible structure
- great way to store semi-structure data
- a lot of processing needed to turn into feature vectors
- contains databases (db)
    - which contain collections (like tables)
        - which you then do finds on

Example: Mongo

- Need to have Mongo running on your local machine with a 'twitter_db' database

In [18]:

```
import pymongo

# start up our client, defaults to the local machine
mdb = pymongo.MongoClient()

# get a connection to a database
db = mdb.twitter_db

# get a connection to a collection in that database
coll = db.twitter_collection
```

Example: Mongo

In [19]:

```
# get one record
coll.find_one()
```

Out[19]:

```
{'_id': ObjectId('6073547ff41410932828e3c
d'),
 'created_at': 'Sun Apr 11 19:56:25 +0000 20
21',
 'id': 1381335345875279873,
 'id_str': '1381335345875279873',
 'text': 'RT @IainLJBrown: Artificial Intell
igence and the Art of Culinary Presentation
- Columbia University\n\nRead more here: htt
ps://t.co/ZCv6zcPBe…',
```

```
 'truncated': False,
 'entities': {'hashtags': [],
  'symbols': [],
  'user_mentions': [{'screen_name': 'IainLJB
rown',
    'name': 'Iain Brown, PhD',
    'id': 467513287,
    'id_str': '467513287',
    'indices': [3, 15]}],
  'urls': []},
 'metadata': {'iso_language_code': 'en', 're
sult_type': 'recent'},
 'source': '',
 'in_reply_to_status_id': None,
 'in_reply_to_status_id_str': None,
 'in_reply_to_user_id': None,
 'in_reply_to_user_id_str': None,
 'in_reply_to_screen_name': None,
```

```
 'user': {'id': 1330350611796209666,
  'id_str': '1330350611796209666',
  'name': 'Another Programmer Bot',
  'screen_name': 'aProgrammerBot',
  'location': '',
  'description': 'Created by @christianecg
_',
  'url': None,
  'entities': {'description': {'urls': []}},
  'protected': False,
  'followers_count': 415,
  'friends_count': 5,
  'listed_count': 19,
  'created_at': 'Sun Nov 22 03:22:36 +0000 2
020',
  'favourites_count': 11852,
  'utc_offset': None,
  'time_zone': None,
```

```
  'geo_enabled': False,
  'verified': False,
  'statuses_count': 93858,
  'lang': None,
  'contributors_enabled': False,
  'is_translator': False,
  'is_translation_enabled': False,
  'profile_background_color': 'F5F8FA',
  'profile_background_image_url': None,
  'profile_background_image_url_https': Non
e,
  'profile_background_tile': False,
  'profile_image_url': 'http://pbs.twimg.co
m/profile_images/1364039854682558467/Z98f0cU
L_normal.jpg',
  'profile_image_url_https': 'https://pbs.tw
img.com/profile_images/1364039854682558467/Z
98f0cUL_normal.jpg',
```

```
  'profile_banner_url': 'https://pbs.twimg.c
om/profile_banners/1330350611796209666/16140
47420',
  'profile_link_color': '1DA1F2',
  'profile_sidebar_border_color': 'C0DEED',
  'profile_sidebar_fill_color': 'DDEEF6',
  'profile_text_color': '333333',
  'profile_use_background_image': True,
  'has_extended_profile': True,
  'default_profile': True,
  'default_profile_image': False,
  'following': False,
  'follow_request_sent': False,
  'notifications': False,
  'translator_type': 'none'},
 'geo': None,
 'coordinates': None,
 'place': None,
```

```
 'contributors': None,
 'retweeted_status': {'created_at': 'Mon Mar
15 13:53:34 +0000 2021',
  'id': 1371459560788086788,
  'id_str': '1371459560788086788',
  'text': 'Artificial Intelligence and the A
rt of Culinary Presentation - Columbia Unive
rsity\n\nRead more here:… https://t.co/ExdNe
2iMgF',
  'truncated': True,
  'entities': {'hashtags': [],
   'symbols': [],
   'user_mentions': [],
   'urls': [{'url': 'https://t.co/ExdNe2iMg
F',
     'expanded_url': 'https://twitter.com/i/
web/status/1371459560788086788',
     'display_url': 'twitter.com/i/web/statu
```

```
s/1…',
     'indices': [101, 124]}]},
  'metadata': {'iso_language_code': 'en', 'r
esult_type': 'recent'},
  'source': '<a href="https://ifttt.com" rel
="nofollow">IFTTT</a>',
  'in_reply_to_status_id': None,
  'in_reply_to_status_id_str': None,
  'in_reply_to_user_id': None,
  'in_reply_to_user_id_str': None,
  'in_reply_to_screen_name': None,
  'user': {'id': 467513287,
   'id_str': '467513287',
   'name': 'Iain Brown, PhD',
   'screen_name': 'IainLJBrown',
   'location': 'Marlow',
   'description': 'Head of #DataScience @SAS
Software | Adjunct Prof @UniSouthampton | Au
```

```
thor | #DataScientist | Interests #AI #ML #D
L #NLP #IoT #BigData | Opinions my own',
   'url': 'https://t.co/wo8jxLIetd',
   'entities': {'url': {'urls': [{'url': 'ht
tps://t.co/wo8jxLIetd',
       'expanded_url': 'https://www.linkedi
n.com/in/iainljbrown/',
       'display_url': 'linkedin.com/in/iainl
jbrown/',
       'indices': [0, 23]}]},
    'description': {'urls': []}},
   'protected': False,
   'followers_count': 117163,
   'friends_count': 100291,
   'listed_count': 430,
   'created_at': 'Wed Jan 18 15:10:25 +0000
2012',
   'favourites_count': 2081,
```

```
    'utc_offset': None,
    'time_zone': None,
    'geo_enabled': False,
    'verified': False,
    'statuses_count': 50114,
    'lang': None,
    'contributors_enabled': False,
    'is_translator': False,
    'is_translation_enabled': False,
    'profile_background_color': '000000',
    'profile_background_image_url': 'http://a
bs.twimg.com/images/themes/theme1/bg.png',
    'profile_background_image_url_https': 'ht
tps://abs.twimg.com/images/themes/theme1/bg.
png',
    'profile_background_tile': False,
    'profile_image_url': 'http://pbs.twimg.co
m/profile_images/1352606177352364032/jQ7AI0v
```

```
G_normal.jpg',
   'profile_image_url_https': 'https://pbs.t
wimg.com/profile_images/1352606177352364032/
jQ7AI0vG_normal.jpg',
   'profile_banner_url': 'https://pbs.twimg.
com/profile_banners/467513287/1611321703',
   'profile_link_color': '3B7CBF',
   'profile_sidebar_border_color': '000000',
   'profile_sidebar_fill_color': '000000',
   'profile_text_color': '000000',
   'profile_use_background_image': False,
   'has_extended_profile': True,
   'default_profile': False,
   'default_profile_image': False,
   'following': False,
   'follow_request_sent': False,
   'notifications': False,
   'translator_type': 'none'},
```

```
 'geo': None,
 'coordinates': None,
 'place': None,
 'contributors': None,
 'is_quote_status': False,
 'retweet_count': 535,
 'favorite_count': 61,
 'favorited': False,
 'retweeted': False,
 'possibly_sensitive': False,
 'lang': 'en'},
'is_quote_status': False,
'retweet_count': 535,
'favorite_count': 0,
'favorited': False,
'retweeted': False,
'lang': 'en'}
```

Example: Mongo Cont.

In [20]:

```
[x for x in coll.find(filter={'retweeted':False},projection={'user.screen_name'},limit=3)]
```

Out[20]:

```
[{'_id': ObjectId('6073547ff41410932828e3c
d'),
  'user': {'screen_name': 'aProgrammerBo
t'}},
 {'_id': ObjectId('6073547ff41410932828e3c
e'),
  'user': {'screen_name': 'RobynPope83'}},
 {'_id': ObjectId('6073547ff41410932828e3c
f'),
  'user': {'screen_name': 'ChukaEjeckam'}}]
```

Questions re Databases?

For SQL practice, check out SQL Murder Mystery (**https://mystery.knightlab.com/**)

Final Review