Elements Of Data Science - F2022

Week 13: Data Processing, Databases and SQL

12/7/2022

TODOs

- Quiz 13, **Due Tuesday December 13th, 11:59pm ET**
- Final
    - Online via Gradescope, open-book, open-note, open-python
    - Released *Wednesday December 7th 11:59pm ET*
    - **Due Friday December 9th 11:59pm ET**
    - Have maximum of 24hrs after starting to finish
    - 30-40 questions (fill in the blank/multiple choice/short answer)
    - Questions asked/answered **privately** via Ed

Today

- Data processing with Pandas
- Relational DBs and SQL
- Connecting to databases with sqlalchemy and pandas

Questions?

Environment Setup

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
```

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

Extract Data: CSV

Comma Separated Values

In [2]:
```
%cat ../data/example.csv
```

```
Year,Make,Model,Description,Price
1997,Ford,E350,"ac, abs, moon",3000.00
1999,Chevy,"Venture Extended Edition","",4900.00
1999,Chevy,"Venture Extended Edition, Very Large",,5000.00
1996,Jeep,Grand Cherokee,"MUST SELL! air, moon roof, loaded",4799.00
```

Extract Data: CSV

Comma Separated Values

In [2]:
```
%cat ../data/example.csv
```

```
Year,Make,Model,Description,Price
1997,Ford,E350,"ac, abs, moon",3000.00
1999,Chevy,"Venture Extended Edition","",4900.00
1999,Chevy,"Venture Extended Edition, Very Large",,5000.00
1996,Jeep,Grand Cherokee,"MUST SELL! air, moon roof, loaded",4799.00
```

In [3]:
```
df = pd.read_csv('../data/example.csv',header=0,sep=',')
df.head()
```

Out[3]:

	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

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 [4]:
```
pd.read_excel('../data/example.xls')
```

Out[4]:

	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
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.)

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 [5]:
```
pd.read_parquet('../data/example.parquet')
```

Out[5]:

	Year	Make	Model	Description	Price
0	1997	Ford	E350	ac, abs, moon	3000.0
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"
    },
    {
```

Extract Data: JSON

Extract Data: JSON

In [6]:
```
json_example = """
{"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}}
"""
```

Extract Data: JSON

In [6]:
```
json_example = """
{"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 [7]:
```
pd.read_json(json_example,orient='index')
```

Out[7]:

	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

Extract Data: HTML

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

In [8]:
```
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[8]:
```
['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

API Example: Wikipedia

In [9]:
```
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=query&prop=info&format=json&titles=Data%20Science
```

API Example: Wikipedia

In [9]:
```
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=query&prop=info&format=json&titles=Data%20Science
```

In [10]:
```
resp = requests.get(url+title)
resp.json()
```

Out[10]:
```
{'batchcomplete': '',
 'query': {'pages': {'49495124': {'pageid': 49495124,
    'ns': 0,
    'title': 'Data Science',
    'contentmodel': 'wikitext',
    'pagelanguage': 'en',
    'pagelanguagehtmlcode': 'en',
    'pagelanguagedir': 'ltr',
    'touched': '2022-11-07T18:39:30Z',
    'lastrevid': 706007296,
    'length': 26,
    'redirect': '',
    'new': ''}}}}
```

API Example: Wikipedia

In [9]:
```
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=query&prop=info&format=json&titles=Data%20Science
```

In [10]:
```
resp = requests.get(url+title)
resp.json()
```

Out[10]:
```
{'batchcomplete': '',
 'query': {'pages': {'49495124': {'pageid': 49495124,
    'ns': 0,
    'title': 'Data Science',
    'contentmodel': 'wikitext',
    'pagelanguage': 'en',
    'pagelanguagehtmlcode': 'en',
    'pagelanguagedir': 'ltr',
    'touched': '2022-11-07T18:39:30Z',
    'lastrevid': 706007296,
    'length': 26,
    'redirect': '',
    'new': ''}}}}
```

In [11]:
```
resp.text
```

Out[11]:
```
'{"batchcomplete":"","query":{"pages":{"49495124":{"pageid":49495124,"ns":0,"title":"Data Science","contentmodel":"wikitext","p
agelanguage":"en","pagelanguagehtmlcode":"en","pagelanguagedir":"ltr","touched":"2022-11-07T18:39:30Z","lastrevid":706007296,"l
ength":26,"redirect":"","new":""}}}}'
```

API Example: Twitter

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

API Example: Twitter

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

In [12]:
```
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)
```

API Example: Twitter

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

In [12]:
```
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 [13]:
```
public_tweets = twitter.search(q='data science institute')['statuses']
for status in public_tweets[:3]:
    print('-------')
    print(status["text"])
```

```
-------
RT @jwoodgett: New position in data science and health research. Come join us at the Lunenfeld-Tanenbaum Research Institute @SinaiHealth, T…
-------
Are you Business owner and want to advetise your busienss like this then google this keyword - khojinINDIA and regi… https://t.co/wmNhogkfVC
-------
Come join our team at the Lunenfeld-Tanenbaum Research Institute as a Principal Investigator in Data Science and He… https://t.co/ZfbKm0wMr1
```

API Example: Twitter

API Example: Twitter

In [14]:
```
public_tweets[0]
```

Out[14]:
```
{'created_at': 'Wed Dec 07 17:24:21 +0000 2022',
 'id': 1600541735519764482,
 'id_str': '1600541735519764482',
 'text': 'RT @jwoodgett: New position in data science and health research. Come join us at the Lunenfeld-Tanenbaum Research Ins
titute @SinaiHealth, T…',
 'truncated': False,
 'entities': {'hashtags': [],
  'symbols': [],
  'user_mentions': [{'screen_name': 'jwoodgett',
    'name': 'Jim Woodgett',
    'id': 134941111,
    'id_str': '134941111',
    'indices': [3, 13]},
   {'screen_name': 'SinaiHealth',
    'name': 'Sinai Health',
    'id': 17158780,
    'id_str': '17158780',
    'indices': [124, 136]}],
  'urls': []},
 'metadata': {'iso_language_code': 'en', 'result_type': 'recent'},
 'source': '<a href="https://mobile.twitter.com" rel="nofollow">Twitter Web App</a>',
 'in_reply_to_status_id': None,
```

Loading Data with pandas

- to_csv
- to excel
- to_json
- to_html
- to_parquet

- to_sql
- to_clipboard

- to_pickle

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

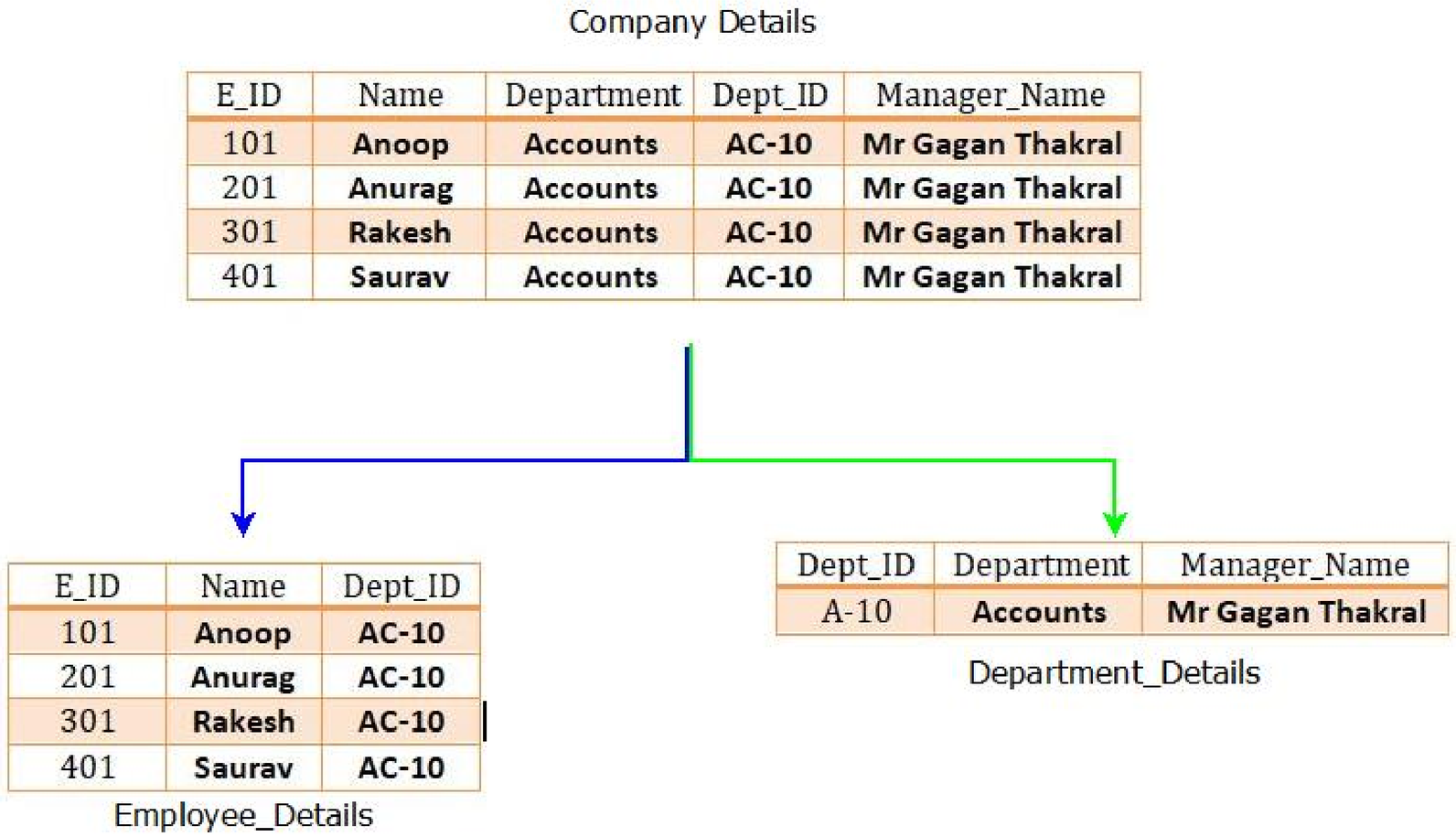
Database Normalization

- 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! (like we've seen before)

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

- may 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 to retrieve a dataframe

Accessing Relational DBs: `sqlalchemy`

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

In [15]:
```
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 "clients"
sql = """
SELECT
    *
FROM
    clients
"""

pd.read_sql(sql,engine)
```

Out[15]:

	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

SQL: SELECT

SQL: SELECT

In [16]:
```
sql="""
SELECT
    client_id
    ,lastname
FROM
    clients
"""

pd.read_sql(sql,engine)
```

Out[16]:

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

SQL: * (wildcard)

SQL: * (wildcard)

In [17]:
```
sql="""
SELECT
    *
FROM
    clients
"""
clients = pd.read_sql(sql,engine)
clients
```

Out[17]:

	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

SQL: * (wildcard)

In [17]:
```
sql="""
SELECT
    *
FROM
    clients
"""
clients = pd.read_sql(sql,engine)
clients
```

Out[17]:

	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 [18]:
```
sql="""
SELECT
    *
FROM
    addresses
"""
addresses = pd.read_sql(sql,engine)
addresses
```

Out[18]:

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

SQL: LIMIT

SQL: LIMIT

In [19]:
```
sql="""
SELECT
    *
FROM
    clients
LIMIT 2
"""
pd.read_sql(sql,engine)
```

Out[19]:

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

SQL: WHERE

SQL: WHERE

In [20]:
```
sql = """
SELECT
    *
FROM
    clients
WHERE home_address_id = 1003
"""

pd.read_sql(sql,engine)
```

Out[20]:

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

SQL: LIKE and %

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

SQL: AS alias

In [22]:
```
sql="""
SELECT
    client_id AS CID
    ,lastname AS Lastname
FROM
    clients AS ca
"""

pd.read_sql(sql,engine)
```

Out[22]:

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

SQL: (INNER) JOIN

SQL: (INNER) JOIN

In [23]:
```
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[23]:

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

SQL: LEFT JOIN

SQL: LEFT JOIN

In [24]:
```
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[24]:

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

SQL: RIGHT JOIN

SQL: RIGHT JOIN

In [25]:
```
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)
```

Out[25]:

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

SQL: FULL OUTER JOIN

SQL: FULL OUTER JOIN

In [26]:
```
sql="""
SELECT
    c.firstname,a.address
FROM clients AS c
FULL OUTER JOIN addresses AS a ON c.home_address_id = a.address_id
"""
pd.read_sql(sql,engine)
```

Out[26]:

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

SQL: Listing Tables

SQL: Listing Tables

In [27]:
```
sql = """
SELECT
    name
FROM sqlite_schema
WHERE type='table'
ORDER BY name
"""
pd.read_sql(sql,engine)
```

Out[27]:

	name
0	addresses
1	clients

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

pandasql

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

In [28]:
```
from pandasql import PandaSQL

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

pandasql

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

In [28]:
```
from pandasql import PandaSQL

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

In [29]:
```
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[29]:

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

For More SQL practice

- SQL Murder Mystery
- SQLZoo

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

Example: Mongo

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

In [30]:
```
import pymongo

# start up our client, defaults to the local machine
client = pymongo.MongoClient() # host="mongodb://localhost:27017"

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

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

Example: Mongo

Example: Mongo

In [31]:
```
# get one record
coll.find_one()
```

Out[31]:
```
{'_id': ObjectId('638e40d4355fc7918bf682b1'),
 'created_at': 'Mon Dec 05 17:17:01 +0000 2022',
 'id': 1599815115872038912,
 'id_str': '1599815115872038912',
 'text': "RT @anifel20: You don't need to spend $1000s to learn Data Science.❌\n\nStanford University, Harvard University &amp; Massachusetts Institute of…",
 'truncated': False,
 'entities': {'hashtags': [],
  'symbols': [],
  'user_mentions': [{'screen_name': 'anifel20',
    'name': 'Felo Anifel',
    'id': 212922754,
    'id_str': '212922754',
    'indices': [3, 12]}],
  'urls': []},
 'metadata': {'iso_language_code': 'en', 'result_type': 'recent'},
 'source': '<a href="http://twitter.com/download/iphone" rel="nofollow">Twitter for iPhone</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,
```

Example: Mongo Cont.

Example: Mongo Cont.

In [32]:
```
[x for x in coll.find(filter={'retweeted':False},projection={'user.screen_name'},limit=3)]
```

Out[32]:
```
[{'_id': ObjectId('638e40d4355fc7918bf682b1'),
  'user': {'screen_name': 'yomi_mabayoje'}},
 {'_id': ObjectId('638e40d4355fc7918bf682b2'),
  'user': {'screen_name': 'Pepperdine_RGFR'}},
 {'_id': ObjectId('638e40d4355fc7918bf682b3'),
  'user': {'screen_name': 'milzmuzing'}}]
```

Questions re Databases?

Question re Final?