DS 4300

Redis + Python

Mark Fontenot, PhD
Northeastern University

Redis-py

- Redis-py is the standard client for Python.
- Maintained by the Redis Company itself
- GitHub Repo: redis/redis-py
- In your 4300 Conda Environment:

```
pip install redis
```

Connecting to the Server

```
import redis
redis_client = redis.Redis(host='localhost',
                           port=6379,
                           db=2,
                           decode_responses=True)
```

- For your Docker deployment, host could be *localhost* or *127.0.0.1*
- Port is the port mapping given when you created the container (probably the default 6379)
- db is the database 0-15 you want to connect to
- decode_responses → data comes back from server as bytes. Setting this true converter them (decodes) to strings.

Redis Command List

- Full List > here <
- Use Filter to get to command for the particular data structure you're targeting (list, hash, set, etc.)
- Redis.py Documentation > here <
- The next slides are not meant to be an exhaustive list of commands, only some highlights. Check the documentation for a complete list.

String Commands

```
# r represents the Redis client object
r.set(‘clickCount:/abc’, 0) *(setting (k, v) pair
val = r.get(‘clickCount:/abc’) *(get value
associated with a key)
r.incr(‘clickCount:/abc’) *(increment value)
ret_val = r.get(‘clickCount:/abc’)
print(f’click count = {ret_val}’)
```

String Commands - 2

```
# r represents the Redis client object
redis_client.mset({'key1': 'val1',
                   'key2': 'val2',
                   'key3': 'val3'})
print(redis_client.mget('key1',
                        'key2',
                  'key3'))
# returns as list [‘val1’, ‘val2’, ‘val3’]
```

*.mset sets multiple (k, v) pairs at once

*(need to parse through individual values)

String Commands - 3

- `set(), mset(), setex(), msetnx(), setnx()`
- `get(), mget(), getex(), getdel()`
- `incr(), decr(), incrby(), decrby()`
- `strlen(), append()`

List Commands - 1

```
# create list: key = ‘names’
# values = [‘mark’, ‘sam’, ‘nick’]
redis_client.rpush('names',
                   'mark', 'sam', 'nick')
```

*(push these three values as the value for 'names')

.rpush can operate as stack or queue

```
# prints [‘mark’, ‘sam’, ‘nick’]
print(redis_client.lrange('names', 0, -1))
```

* (return first value to last value in 'names')

List Commands - 2

- `lpush()`, `lpop()`, `lset()`, `lrem()`
- `rpush()`, `rpop()`
- `lrange()`, `llen()`, `lpos()`
- Other commands include moving elements between lists, popping from multiple lists at the same time, etc.

Hash Commands - 1

```
redis_client.hset('user-session:123',
      mapping={'first': 'Sam',
               'last': 'Uelle',
               'company': 'Redis',
               'age': 30
}) *(set the hash)

# prints:
#{'name': 'Sam', 'surname': 'Uelle', 'company': 'Redis', 'age': '30'}
print(redis_client.hgetall('user-session:123'))
* (get all the (k, v) pairs in given hash)
```

Hash Commands - 2

- `hset()`, `hget()`, `hgetall()`
- `hkeys()`
- `hdel()` *(delete a key), `hexists()` *(check if key exists), `hlen()`, `hstrlen()`

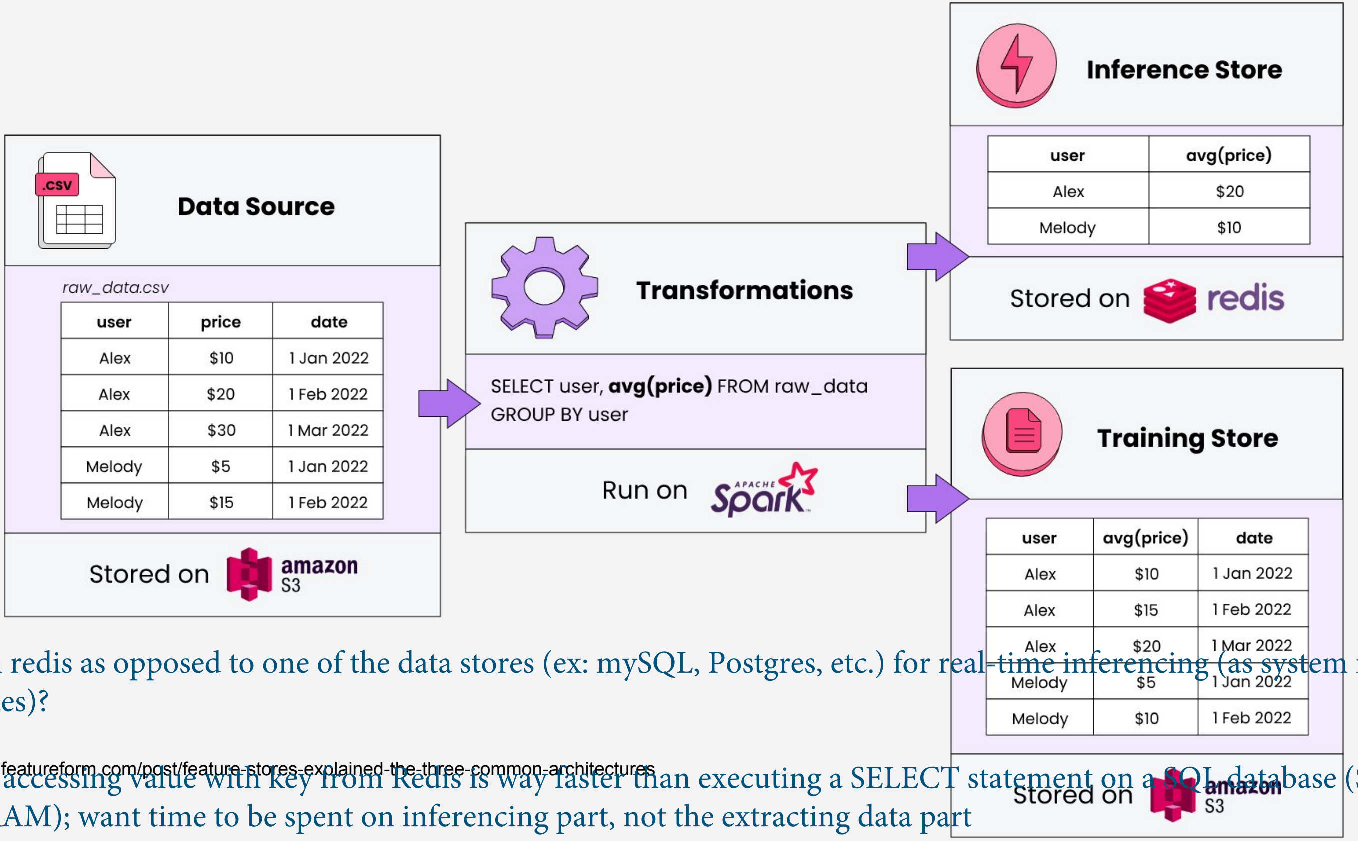
Redis Pipelines

- Helps avoid multiple related calls to the server → less network overhead *(sends all commands at one time)

```
r = redis.Redis(decode_responses=True)
pipe = r.pipeline()

for i in range(5):
   pipe.set(f"seat:{i}", f"#{i}") *(can string together the pipeline by setting different
things)

set_5_result = pipe.execute() *(not the same as a transaction, just used to reduce network
traffic)
print(set_5_result)  # >>> [True, True, True, True, True]

pipe = r.pipeline()

# "Chain" pipeline commands together. *(or you can chain pipeline commands together)
get_3_result = pipe.get("seat:0").get("seat:3").get("seat:4").execute()
print(get_3_result)  # >>> ['#0', '#3', '#4']
```

Redis in Context

Redis in ML - Simplified Example



user	price	date
Alex	$10	1 Jan 2022
Alex	$20	1 Feb 2022
Alex	$30	1 Mar 2022
Melody	$5	1 Jan 2022
Melody	$15	1 Feb 2022

```
SELECT user, avg(price) FROM raw_data
GROUP BY user
```

user	avg(price)
Alex	$20
Melody	$10

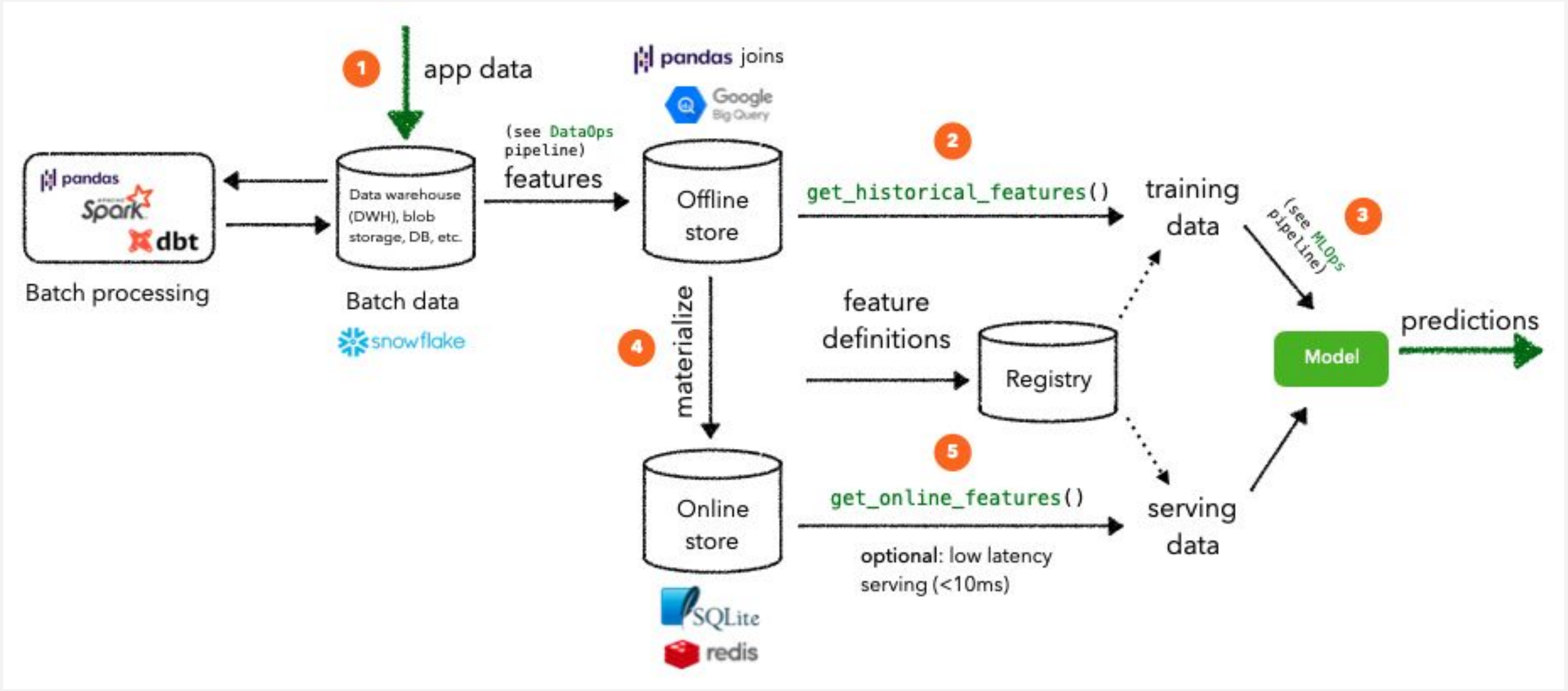
user	avg(price)	date
Alex	$10	1 Jan 2022
Alex	$15	1 Feb 2022
Alex	$20	1 Mar 2022
Melody	$5	1 Jan 2022
Melody	$10	1 Feb 2022

*why store data in redis as opposed to one of the data stores (ex: mySQL, Postgres, etc.) for real-time inferencing (as system is running, pop out inference values)?

Source: https://www.featureform.com/post/feature-stores-explained-the-three-common-architectures

for latency issues; accessing value with key from Redis is way faster than executing a SELECT statement on a SQL database (SQL stored on disk rather than RAM); want time to be spent on inferencing part, not the extracting data part

Redis in DS/ML



Source: https://madewithml.com/courses/mlops/feature-store/