DS4300: Redis + Python

Redis-py

- The standard client for Python.
- Maintained by the Redis Company itself.
- ***GitHub repo***: redis/redis-py

Connecting to the Server

Sample py code

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

- Docker deployment: *localhost* or *127.0.0.1*
- Port is the port mapping given when you created the container (most likely 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 then (decodes) to strings.

String Commands Code:

```
# r represents the Redis client object

r.set('clickCount:/abc', 0)
val = r.get('clickCount:/abc')
r.incr('clickCount:/abc')
ret_val = r.get('clickCount:/abc')
print(f'click count = {ret_val}')

redis_client.mset({'key1': 'val1',
                   'key2': 'val2',
                   'key3': 'val3'})
print(redis_client.mget('key1',
                        'key2',
                        'key3'))
# returns as list ['val1', 'val2', 'val3']
```

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

List Commands

```
# create list: key = 'names'
# values = ['mark', 'sam', 'nick']
redis_client.rpush('names',
                   'mark', 'sam', 'nick')
# prints ['mark', 'sam', 'nick']
print(redis_client.lrange('names', 0, -1))
```

- 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

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

# prints:
#{'name': 'Sam', 'surname': 'Uelle', 'company': 'Redis', 'age': '30'}
print(redis_client.hgetall('user-session:123'))
```

- hset(), hget(), hgetall()
- hkeys()
- hdel(), hexists(), hlen(), hstrlen()

Redis Pipelines

- Helps avoid multiple related calls to the server → less network overhead

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

for i in range(5):
  pipe.set(f"seat:{i}", f"#{i}")

set_5_result = pipe.execute()
print(set_5_result)  # >>> [True, True, True, True, True]

pipe = r.pipeline()

# "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
    - Data Source -> Transformation -> Inference Store/ Training Store

Redis in DS/ML

- App Data Ingestion:
    - Data is processed using Pandas, Spark, and dbt.
    - Batch data is stored in a data warehouse (DWH), blob storage, or database.
    - Example: Snowflake.
- Offline Feature Store:
    - Features are stored in an offline store (e.g., Google BigQuery).

        - get_historical_features() retrieves training data for the model.
    - Model Training & Predictions:
        - The retrieved training data is used to train the model.
        - The model makes predictions based on feature inputs.
    - Materialization to Online Store:
        - Features are materialized from the offline store to an online store.
        - Example databases: SQLite, Redis.
    - Online Feature Retrieval:
        - get_online_features() retrieves serving data for real-time inference.
        - Optional: Low latency serving (<10ms)