ETL

ETL historical cryptocurrency prices

> For a given coin symbol, fetch its historical daily closing prices (in USD), volumes, and market cap (in USD) and load them into a database.

```
crypto_etl("BTC")
crypto_etl("ETH")
```

0. **Design database schema**
1. **Extract data from a source**
    i. identify endpoint, path, and query parameters from API documentation
    ii. request and get response from API
2. **Transform data**
    i. select relevant values from response
    ii. transform data into a format that can be loaded into a database
3. **Load data into database**
    i. create table
    ii. insert data into table

DB schema

Table: `coins`

Column	Type
symbol	VARCHAR(10)
date	VARCHAR(10)
close	FLOAT
volume	FLOAT
market_cap	FLOAT

Extract data from a source

- identify endpoint, path, and query parameters from API documentation
- request and get response from API

Alpha Vantage Stock and Crypo Market API

https://www.alphavantage.co/documentation/

- Endpoint
- Path
- Query parameters
    - **API key**

Example API request

```
https://www.alphavantage.co/query?
function=DIGITAL_CURRENCY_DAILY&symbol=BTC&market=CNY&apikey=demo
```

Example API response

```
{
    "Meta Data": {
        "1. Information": "Daily Prices and Volumes for Digital Currency",
        "2. Digital Currency Code": "BTC",
        ...
    },
    "Time Series (Digital Currency Daily)": {
        "2023-10-30": {
            "1a. open (CNY)": "252629.31654800",
            "1b. open (USD)": "34525.88000000",
            "2a. high (CNY)": "252955.87872100",
            "2b. high (USD)": "34570.51000000",
            "3a. low (CNY)": "252081.55844200",
            "3b. low (USD)": "34451.02000000",
            "4a. close (CNY)": "252122.24151800",
            "4b. close (USD)": "34456.58000000",
            "5. volume": "702.10687000",
            "6. market cap (USD)": "702.10687000"
        },
        "2023-10-30": {...},
        ...
    }
}
```

Make a request to API

```python
url = "https://www.alphavantage.co/query"
apikey = "yourkey"

params = {
    "function": "DIGITAL_CURRENCY_DAILY",
    "symbol": "BTC",
    "market": "CAD",
    "apikey": apikey,
}

response = requests.get(url, params=params)
response = response.json()

print(response)
```

Transform data

- select relevant values from response
- transform data into a format that can be loaded into a database

Select for a single date

- symbol
- date
- closing price
- volume
- market cap

Select for a single date

```
date = "2021-10-30"

symbol = response["Meta Data"]["2. Digital Currency Code"]

price_chart = response["Time Series (Digital Currency Daily)"]
price = price_chart[date]

close = price["4b. close (USD)"]
volume = price["5. volume"]
market_cap = price["6. market cap (USD)"]

print(symbol, date, close, volume, market_cap)
```

Select for all dates

```python
symbol = response["Meta Data"]["2. Digital Currency Code"]
price_chart = response["Time Series (Digital Currency Daily)"]

for date, price in price_chart.items():
    close = price["4b. close (USD)"]
    volume = price["5. volume"]
    market_cap = price["6. market cap (USD)"]

    print(symbol, date, close, volume, market_cap)
```

Insert data manually

```
conn.execute("INSERT INTO table VALUES (1, 'Harry Potter')")
```

Insert data dynamically (single record)

Option 1. Insert tuple using f-string

```
data = (1, 'Harry Potter')
conn.execute(f"INSERT INTO customer VALUES {data}")
```

Insert data dynamically (single record)

Option 2. Insert tuple using params

```
data = (1, 'Harry Potter')
conn.execute("INSERT INTO customer VALUES (?, ?)", data)
```

Insert data dynamically (single record)

Option 3. Insert dict using params

```
data = {'id': 1, 'name': 'Harry Potter'}
conn.execute("INSERT INTO customer VALUES (:id, :name)", data)
```

Insert data dynamically (multiple records)

Option 1. Tuple with f-string

```
data = [
    (2, 'Ron Weasley', 'Gryffindor', 11),
    (3, 'Hermione Granger', 'Gryffindor', 11),
    (4, 'Draco Malfoy', 'Slytherin', 11),
    (5, 'Cedric Diggory', 'Hufflepuff', 14),
    (6, 'Cho Chang', 'Ravenclaw', 13),
]

for record in data:
    query = f"INSERT INTO students VALUES {record}"
    conn.execute(query)
```

Insert data dynamically (multiple records)

Option 2. Tuple with params

```python
data = [
    (2, 'Ron Weasley', 'Gryffindor', 11),
    (3, 'Hermione Granger', 'Gryffindor', 11),
    (4, 'Draco Malfoy', 'Slytherin', 11),
    (5, 'Cedric Diggory', 'Hufflepuff', 14),
    (6, 'Cho Chang', 'Ravenclaw', 13),
]

for record in data:
    query = "INSERT INTO students VALUES (?, ?, ?, ?)"
    conn.execute(query, params = record)
```

Insert data dynamically (multiple records)

Option 3. Dict with params

```python
data = [
    {'id': 2, 'name': 'Ron Weasley', 'house': 'Gryffindor', 'age': 11},
    {'id': 3, 'name': 'Hermione Granger', 'house': 'Gryffindor', 'age': 11},
    {'id': 4, 'name': 'Draco Malfoy', 'house': 'Slytherin', 'age': 11},
    {'id': 5, 'name': 'Cedric Diggory', 'house': 'Hufflepuff', 'age': 14},
    {'id': 6, 'name': 'Cho Chang', 'house': 'Ravenclaw', 'age': 13},
]

for record in data:
    query = "INSERT INTO students VALUES (:id, :name, :house, :age)"
    conn.execute(query, record)
```

`execute()` vs. `executemany()`

`execute()`: execute a query once

```
for record in data:
    query = "INSERT INTO students VALUES (?, ?, ?, ?)"
    conn.execute(query, params = record)
```

`executemany()`: execute a query multiple times

```
query = "INSERT INTO students VALUES (?, ?, ?, ?)"
conn.executemany(query, params = data)
```

Convert response to a list of tuples (Option 1)

```python
symbol = response["Meta Data"]["2. Digital Currency Code"]
price_chart = response["Time Series (Digital Currency Daily)"]

data = []
for date, price in price_chart.items():
    close = price["4b. close (USD)"]
    volume = price["5. volume"]
    market_cap = price["6. market cap (USD)"]

    record = (symbol, date, close, volume, market_cap) # tuple
    data.append(record)
```

Load data into database

- create table
- insert data into table

Create table

```
conn = sqlite3.connect("coins.db")
query = """
CREATE TABLE coins (
    symbol VARCHAR(10),
    date VARCHAR(10),
    close FLOAT,
    volume FLOAT,
    market_cap FLOAT
)
"""
conn.execute(query)
conn.commit()
```

Create table with `table_name` and `schema`

```python
conn = sqlite3.connect("coins.db")

table_name = "coins"
schema = """
    symbol VARCHAR(10),
    date VARCHAR(10),
    close FLOAT,
    volume FLOAT,
    market_cap FLOAT
"""

query = f"CREATE TABLE {table_name} ({schema})"
conn.execute(query)
conn.commit()
```

Load data into table (Option 1)

```python
data = [(...), (...), ...] # list of tuples

for record in data:
    query = f"INSERT INTO coins VALUES {record}"
    conn.execute(query)
conn.commit()
```

Verify data

```
conn.execute("SELECT * FROM coins limit 10").fetchall()
conn.execute("SELECT * FROM coins").fetchmany(10)
```

💻 Load data into table (Option 2 and Option 3)

Option 2:

1. Transform data into a list of tuples
2. Write a query with `?` as placeholders
3. Use `execute()` or `executemany()` with params

Option 3:

1. Transform data into a list of dicts
2. Write a query with `:key` as placeholders
3. Use `execute()` or `executemany()` with params

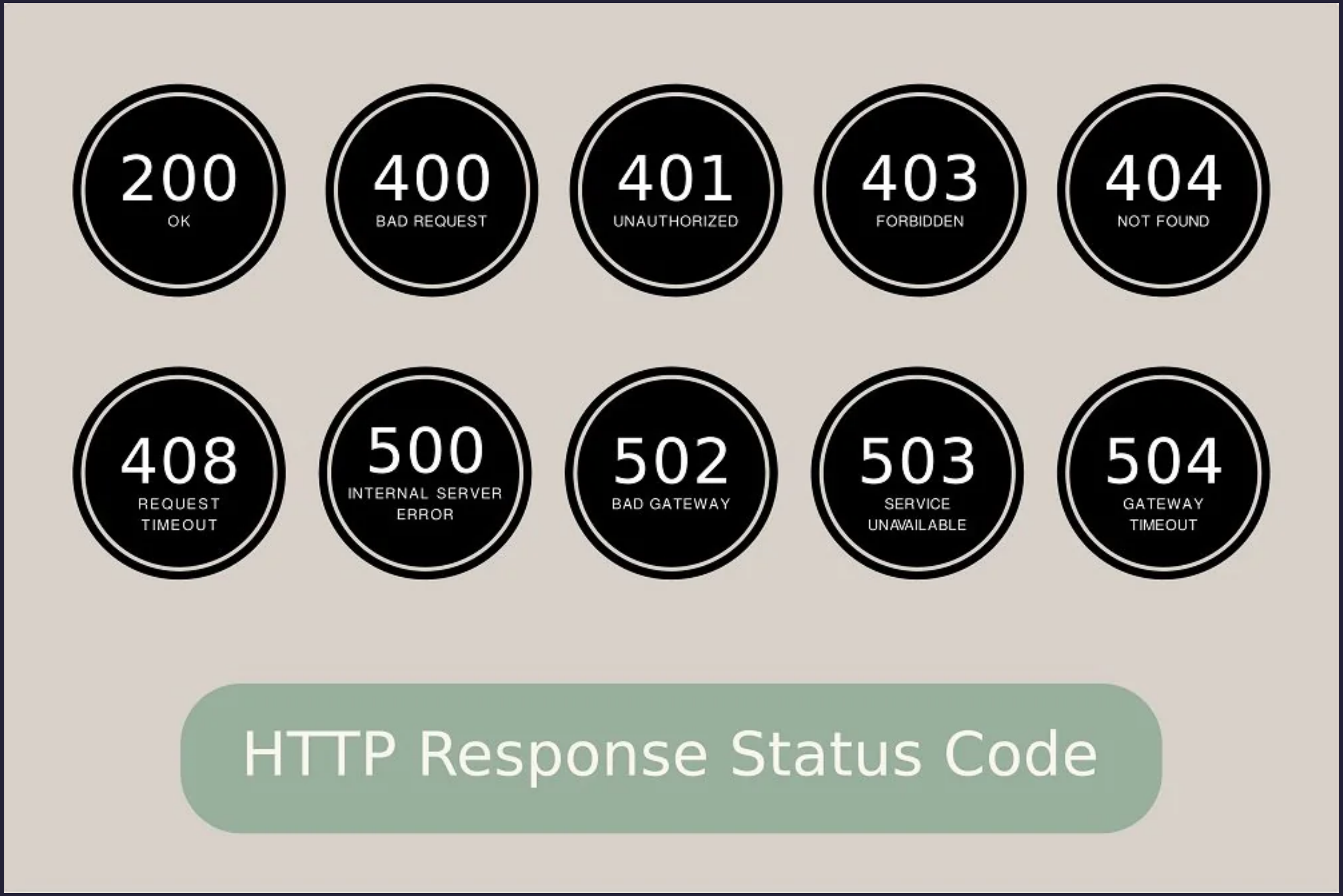
`etl()` to extract, transform, and load data

```python
def crypto_etl(conn, symbol):
    """Extract, transform, and load data"""

    response = extract_data(symbol)
    data = transform_data(response)
    load_data(conn, "coins", data)
```

`etl()` to extract, transform, and load data

```
conn = sqlite3.connect("coins.db")

create_table(conn, "coins", schema)

for symbol in ["BTC", "ETH", "DOGE"]:
    crypto_etl(conn, symbol)

conn.execute("SELECT * FROM coins").fetchmany(5)

conn.close()
```

Error handling with conditional statements (`if` or `try-except`)



https://stackoverflow.com/questions/16511337/correct-way-to-try-except-using-python-requests-module