Building ETL Pipelines

ETL pipeline for 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")
```

ETL development process

1. **Design the database schema**
2. **Extract data from the source**
    - Identify endpoint and query parameters from API documentation
    - Request and get response from API
3. **Transform the data**
    - Select and clean the relevant values from the API response
    - Prepare the data for database loading
4. **Load the data into the database**
    - Create the table(s) based on the schema
    - Insert the transformed data into the table(s)

Design the database schema

Table: `coins`

Column	Type
symbol	TEXT
date	TEXT
close	NUMERIC
volume	NUMERIC

> 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.

Extract data from the source

- Identify endpoint and query parameters from API documentation
- Request and get response from API

Alpha Vantage Stock and Crypo Market API

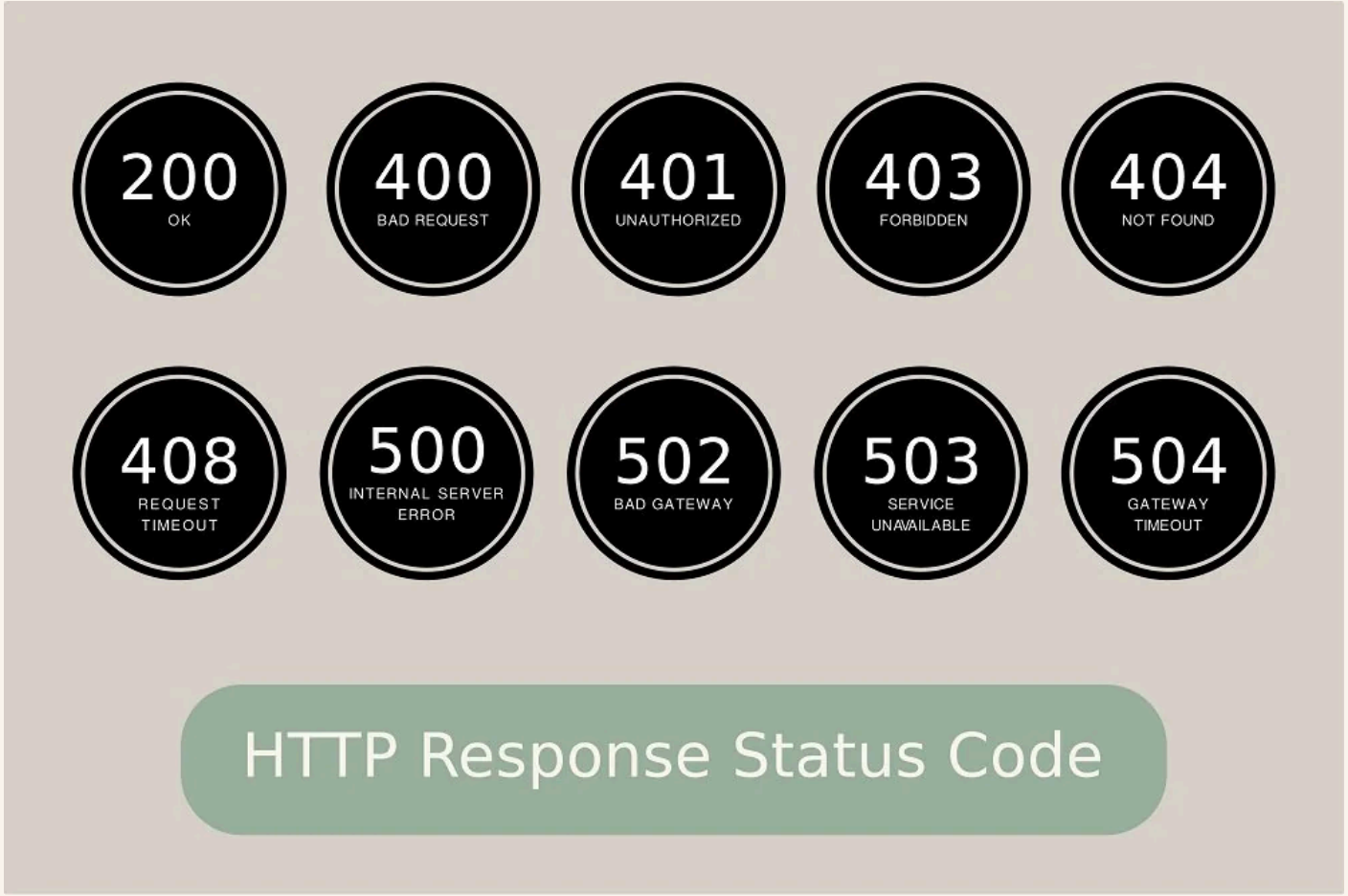
https://www.alphavantage.co/documentation/

- Endpoint?
- Query parameters?

Send a request to the API

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

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

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

# Only works if the request is successful
data = response.json()

print(data)
```

Error handling with HTTP response status codes



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

Handle HTTP errors

```
# Basic conditional check
response = requests.get(url, params=params)
if response.status_code == 200:
    data = response.json()
else:
    print(f"Error: {response.status_code}")
    data = None
```

```
# Try-except block
try:
    response = requests.get(url, params=params)
    response.raise_for_status()  # Raise an error for bad status codes
    data = response.json()
except requests.exceptions.HTTPError as http_err:
    print(f"HTTP error occurred: {http_err}")
    data = None
```

Transform the data

- Select and clean the relevant values from the API response
- Prepare the data for database loading

Select relevant values for the table

Table: `coins`

Column	Type
symbol	TEXT
date	TEXT
close	NUMERIC
volume	NUMERIC

Select relevant values for the table (symbol, date, close, volume)


```
{
    "Meta Data": {
        "1. Information": "Daily Prices and Volumes for Digital Currency",
        "2. Digital Currency Code": "BTC",
        "3. Digital Currency Name": "Bitcoin",
        "4. Market Code": "EUR",
        "5. Market Name": "Euro",
        "6. Last Refreshed": "2024-11-11 00:00:00",
        "7. Time Zone": "UTC"
    },
    "Time Series (Digital Currency Daily)": {
        "2024-11-11": {
            "1. open": "74780.73000000",
            "2. high": "76000.00000000",
            "3. low": "74639.10000000",
            "4. close": "75653.75000000",
            "5. volume": "20.70191291"
        },
        "2024-11-10": {
            "1. open": "71562.42000000",
            "2. high": "75699.99000000",
            "3. low": "71388.80000000",
            "4. close": "74757.19000000",
            "5. volume": "760.24717970"
        },
        ...
    }
}
```

Select relevant values (a single date)

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

# Select a specific date
date = "2021-10-30"
price = price_chart[date]

close = price["4. close"]
volume = price["5. volume"]

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

Select relevant values (all dates)

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

for date, price in price_chart.items():
    close = price["4. close"]
    volume = price["5. volume"]

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

Convert response to a list of tuples

`[(...), (...), (...)]`

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

records = []
for date, price in price_chart.items():
    close = price["4. close"]
    volume = price["5. volume"]

    record = (symbol, date, close, volume) # create tuple
    records.append(record)                 # add tuple to list
```

Load the data into the database

- Create the table(s) based on the schema
- Insert the transformed data into the table(s)

Create table `coins`

```
conn = sqlite3.connect("coins.db")
query = """
CREATE TABLE coins (
    symbol TEXT,
    date TEXT,
    close NUMERIC,
    volume NUMERIC
)
"""
# Create the table if it doesn't exist
conn.execute(query)

# Save the changes
conn.commit()
```

Create table `coins` dynamically

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

table_name = "coins"
schema = """
(
    symbol TEXT,
    date TEXT,
    close NUMERIC,
    volume NUMERIC
)
"""

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

Load the data into the table

Option 1. Tuple with f-string

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

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

# Save the changes
conn.commit()
```

Verify data

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

Insert data dynamically (multiple records)

Option 2. Insert tuple using params

```
records = [
    (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 records:
    query = "INSERT INTO students VALUES (?, ?, ?, ?)"
    conn.execute(query, params = record)
```

Insert data dynamically (multiple records)

Option 3. Insert dict using params

```
records = [
    {'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 records:
    query = "INSERT INTO students VALUES (:id, :name, :house, :age)"
    conn.execute(query, record)
```

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

Option 2: Tuple with placeholders

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

Option 3: Dict with placeholders

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

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

```
def crypto_etl(conn, symbol):
    """
    Function to extract, transform, and load crypto data for a given symbol.

    Args:
        conn: SQLite3 connection object
        symbol: Cryptocurrency symbol (e.g., 'BTC', 'ETH')
    Returns:
        None

    """

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

Putting it all together

> 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.

```
# Connect to the database
conn = sqlite3.connect("coins.db")

# Create table coins
create_table(conn, "coins", schema)

# ETL for multiple coins
crypto_etl(conn, "BTC")
crypto_etl(conn, "ETH")

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

conn.close()
```