

# ETL pipeline

# 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")
```

## 1. Define data requirements and database schema

## 2. Extract data from a source

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

## 3. Transform data

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

## 4. Load data into database

- i. create table
- ii. insert data into table

# DB schema

Table: `coins`

Column	Type
symbol	TEXT
date	TEXT
close	NUMERIC
volume	NUMERIC

## Extract data from source

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

# Alpha Vantage Stock and Crypto 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=EUR&apikey=demo
```

# Example API response

```
{
  "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"
    },
    ...
  }
}
```



# Make a request to 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)
data = response.json()

print(data)
```

# 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

## Select for a single date

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

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

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

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

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

## Select for 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)
```

## Insert data manually

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

# Insert data dynamically (single record)

## Option 1. Insert tuple using f-string

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

# Insert data dynamically (single record)

## Option 2. Insert tuple using params

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



# Insert data dynamically (single record)

## Option 3. Insert dict using params

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

# Insert data dynamically (multiple records)

## Option 1. Tuple with f-string

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

# Insert data dynamically (multiple records)

## Option 2. Tuple with 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. Dict with 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)
```

## execute() vs. executemany()

`execute()` : execute a query once

```
for record in records:  
    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 = records)
```

## Convert response to a list of tuples (Option 1)

```
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) # tuple
    records.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
)
"""
conn.execute(query)
conn.commit()
```



## Create table with `table_name` and `schema`

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

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

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

## Load data into table (Option 1)

```
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

```
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`)

