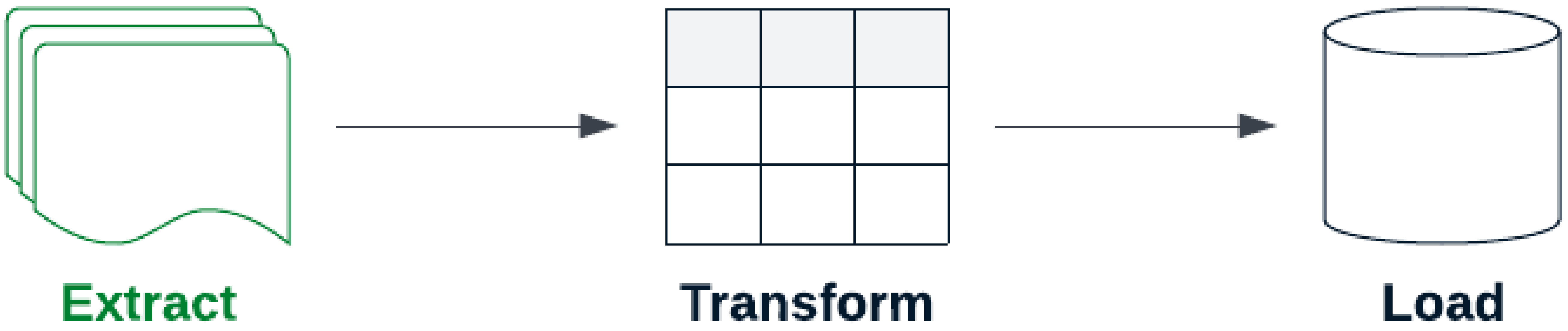
Extracting non-tabular data

ETL AND ELT IN PYTHON

Jake Roach
Data Engineer

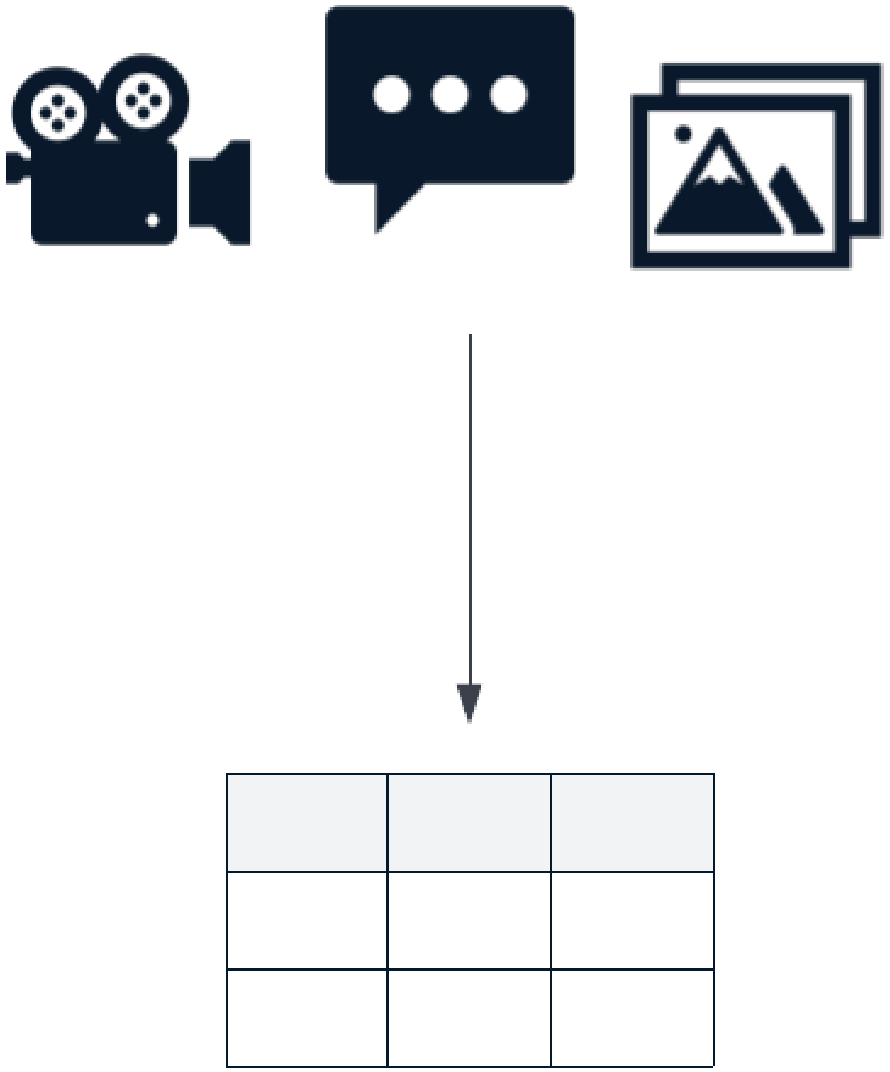
Extracting non-tabular data



Types of non-tabular data

Most data produced and consumed is unstructured data

- Text
- Audio
- Image
- Video
- Spatial
- IoT

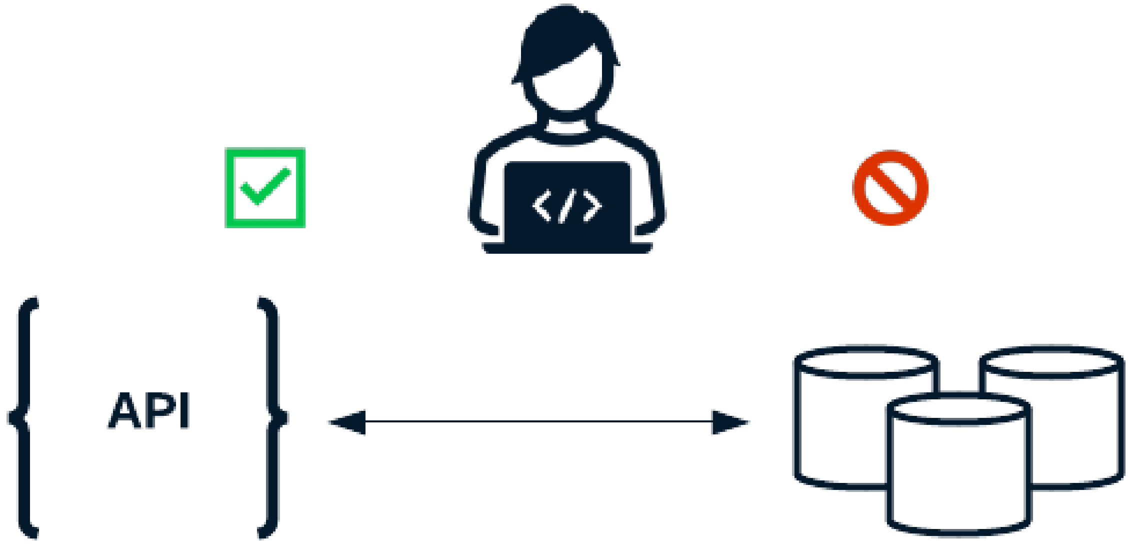


[1] https://mitsloan.mit.edu/ideas-made-to-matter/tapping-power-unstructured-data

Working with APIs and JSON data

API (Application Programming Interface)

- Software that sits on top of data sources
- Prevents direct interaction with database



JSON (JavaScript Object Notation)

- Key-value pairs
- No set schema
- Look and feel similar to `dict` ionaries

```
{
    "key": "value",
    ...
    "open": 0.121875
}
```

Reading JSON files with pandas

```
{
    "timestamps": [863703000, 863789400, ...],
    "open": [0.121875, 0.098438, ...],
    "close": [...],
    "volume": [...]
}
```

Use the `.read_json()` function

```
# Read in a JSON file in the format above
raw_stock_data = pd.read_json("raw_stock_data.json", orient="columns")
```

[1] https://pandas.pydata.org/docs/reference/api/pandas.read_json.html

Nested or unstructured JSON data

Data is not always DataFrame-ready

- Nested objects
- Varying "schema"

```
{
    "863703000": {
        "volume": 1443120000,
        "price": {
            "close": 0.09791,
            "open": 0.12187
        }
    },
    "863789400": {
        ...
    }, ...
}
```

Reading JSON files with json

```
import json

with open("raw_stock_data.json", "r") as file:
    # Load the file into a dictionary
    raw_stock_data = json.load(file)

# Confirm the type of the raw_stock_data variable
print(type(raw_stock_data))
```

```
<class 'dict'>
```

Let's practice!

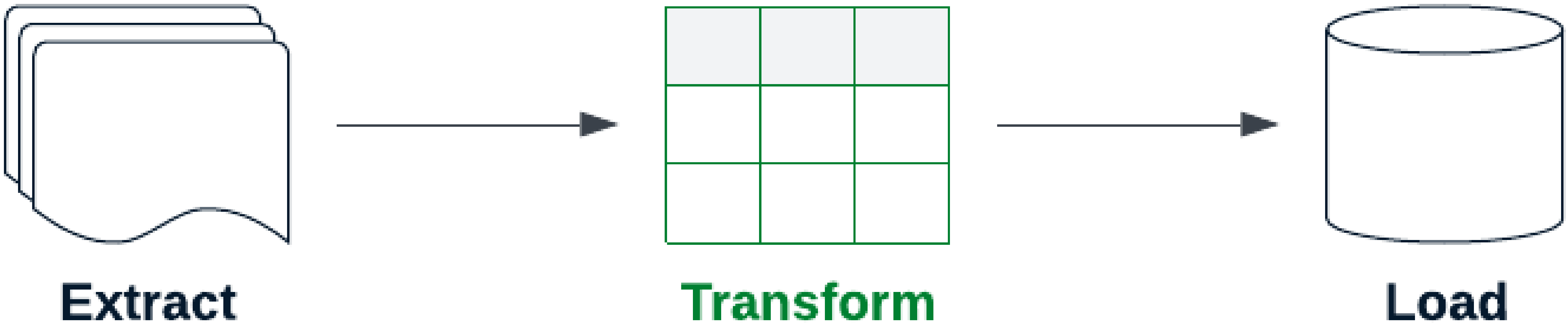
ETL AND ELT IN PYTHON

Transforming non-tabular data

ETL AND ELT IN PYTHON

Jake Roach
Data Engineer

Transforming non-tabular data



Storing data in dictionaries

Nested JSON

```
{
    "863703000": {
        "price": {
            "open": 0.12187,
            "close": 0.09791
        },
        "volume": 1443120000
    },
    "863789400": {
    }, ...
}
```

Goal:

- Convert dictionary into a DataFrame-ready format

```
[
    [863703000, 0.12187, 0.09791, 1443120000],
    [863789400, 0.09843, ...]
]
```

Iterating over dictionary components

```python
# Loop over keys
for key in raw_data.keys():
    ...
```

```python
# Loop over values
for value in raw_data.values():
    ...
```

```python
# Loop over keys and values
for key, value in raw_data.items():
    ...
```

`.keys()`

- Creates a list of keys stored in a dictionary

`.values()`

- Creates a list of values stored in a dictionary

`.items()`

- Generates a list of tuples, made up of the key-value pairs

Parsing data from dictionaries

```
entry = {
    "volume": 1443120000,
    "price": {
        "open": 0.12187,
        "close": 0.09791,
    }
}
```

```
# Parse data from dictionary using .get()
volume = entry.get("volume")
```

```
ticker = entry.get("ticker", "DCMP")
```

```
# Call .get() twice to return the nested "open" value
open_price = entry.get("price").get("open", 0)
```

Creating a DataFrame from a list of lists

Pass a list of lists to `pd.DataFrame()`

```
# Pass a list of lists to pd.DataFrame
raw_data = pd.DataFrame(flattened_rows)
```

Set column headers using `.columns`

```
# Create columns
raw_data.columns = ["timestamps", "open", "close", "volume"]
```

Set an index using `.set_index()`

```
# Set the index column to be "timestamps"
raw_data.set_index("timestamps")
```

Transforming stock data

```python
parsed_stock_data = []

# Loop through each key-value pair of the raw_stock_data dictionary
for timestamp, ticker_info in raw_stock_data.items():
    parsed_stock_data.append([
        timestamp,
        ticker_info.get("price", {}).get("open", 0),  # Parse the opening price
        ticker_info.get("price", {}).get("close", 0),  # Parse the closing price
        ticker_info.get("volume", 0)  # Parse the volume
    ])
```

```python
# Create a DataFrame, assign column names, and set an index
transformed_stock_data = pd.DataFrame(parsed_stock_data)
transformed_stock_data.columns = ["timestamps", "open", "close", "volume"]
transformed_stock_data = transformed_stock_data.set_index("timestamps")
```

Let's practice!

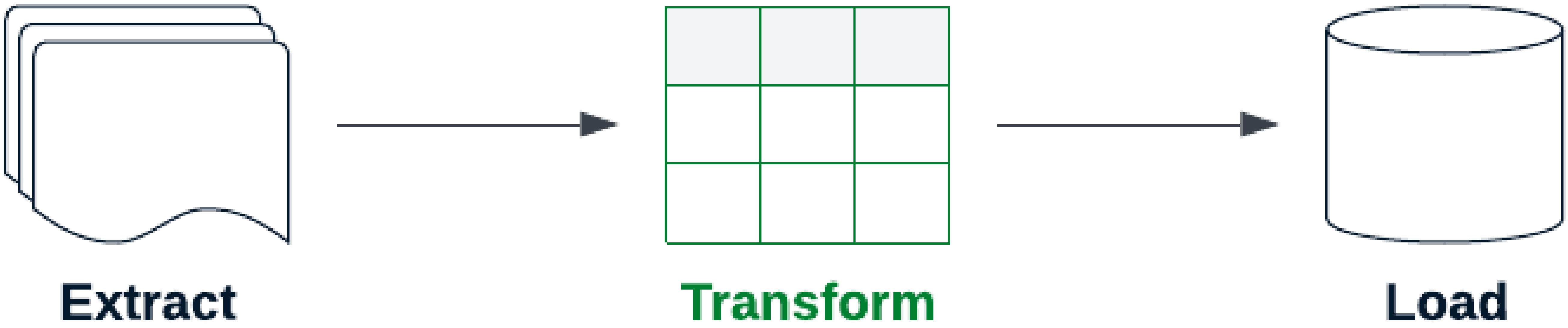
ETL AND ELT IN PYTHON

Advanced data transformation with pandas

ETL AND ELT IN PYTHON

Jake Roach
Data Engineer

Advanced data transformation with pandas



Filling missing values with pandas

```
timestamps              volume      open      close
1997-05-15 13:30:00  1443120000  0.121875  0.097917
1997-05-16 13:30:00   294000000  NaN       0.086458
1997-05-19 13:30:00   122136000  0.088021  NaN
```

```
# Fill all NaN with value 0
clean_stock_data = raw_stock_data.fillna(value=0)
```

```
timestamps              volume      open      close
1997-05-15 13:30:00  1443120000  0.121875  0.097917
1997-05-16 13:30:00   294000000  0.000000  0.086458
1997-05-19 13:30:00   122136000  0.088021  0.000000
```

Filling missing values with pandas

```
timestamps           volume      open      close
1997-05-15 13:30:00  1443120000  0.121875  0.097917
1997-05-16 13:30:00   294000000  NaN       0.086458
1997-05-19 13:30:00   122136000  0.088021  NaN
```

```python
# Fill NaN values with specific value for each column
clean_stock_data = raw_stock_data.fillna(value={"open": 0, "close": .5}, axis=1)
```

```
timestamps           volume      open      close
1997-05-15 13:30:00  1443120000  0.121875  0.097917
1997-05-16 13:30:00   294000000  0.000000  0.086458
1997-05-19 13:30:00   122136000  0.088021  0.500000
```

Filling missing values with pandas

```
timestamps           volume      open      close
1997-05-15 13:30:00  1443120000  0.121875  0.097917
1997-05-16 13:30:00   294000000  NaN       0.086458
1997-05-19 13:30:00   122136000  0.088021  NaN
```

```python
# Fill NaN value using other columns
raw_stock_data["open"].fillna(raw_stock_data["close"], inplace=True)
```

```
timestamps           volume      open      close
1997-05-15 13:30:00  1443120000  0.121875  0.097917
1997-05-16 13:30:00   294000000  0.086458  0.086458
1997-05-19 13:30:00   122136000  0.088021  NaN
```

Grouping data

```sql
SELECT
    ticker,
    AVG(volume),
    AVG(open),
    AVG(close)
FROM raw_stock_data
GROUP BY ticker;
```

The `.groupby()` method can recreate the query above, using `pandas`

Grouping data with pandas

```
ticker  volume      open      close
AAPL    1443120000  0.121875  0.097917
AAPL     297000000  0.098146  0.086458
AMZN     124186000  0.247511  0.251290
```

```python
# Use Python to group data by ticker, find the mean of the reamining columns
grouped_stock_data = raw_stock_data.groupby(by=["ticker"], axis=0).mean()
```

```
          volume        open       close
ticker
AAPL      1.149287e+08  34.998377  34.986851
AMZN      1.434213e+08  30.844692  30.830233
```

Can use `.min()`, `.max()` and `.sum()` to aggregate data

Applying advanced transformations to DataFrames

The `.apply()` method can handle more advanced transformations

```python
def classify_change(row):
    change = row["close"] - row["open"]
    if change > 0:
        return "Increase"
    else:
        return "Decrease"
```

```python
# Apply transformation to DataFrame
raw_stock_data["change"] = raw_stock_data.apply(
    classify_change,
    axis=1
)
```

Before transformation

ticker	...	open	close
AAPL		0.121875	0.097917
AAPL		0.098146	0.086458
AMZN		0.247511	0.251290

After transformation

ticker	...	open	close	change
AAPL		0.121875	0.097917	Decrease
AAPL		0.098146	0.086458	Decrease
AMZN		0.247511	0.251290	Increase

Let's practice!

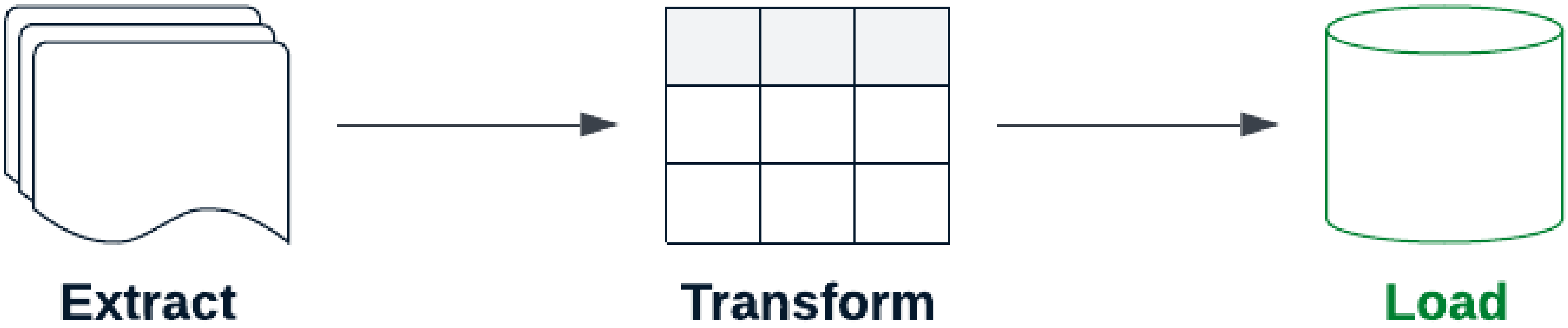
ETL AND ELT IN PYTHON

Loading data to a SQL database with pandas

ETL AND ELT IN PYTHON

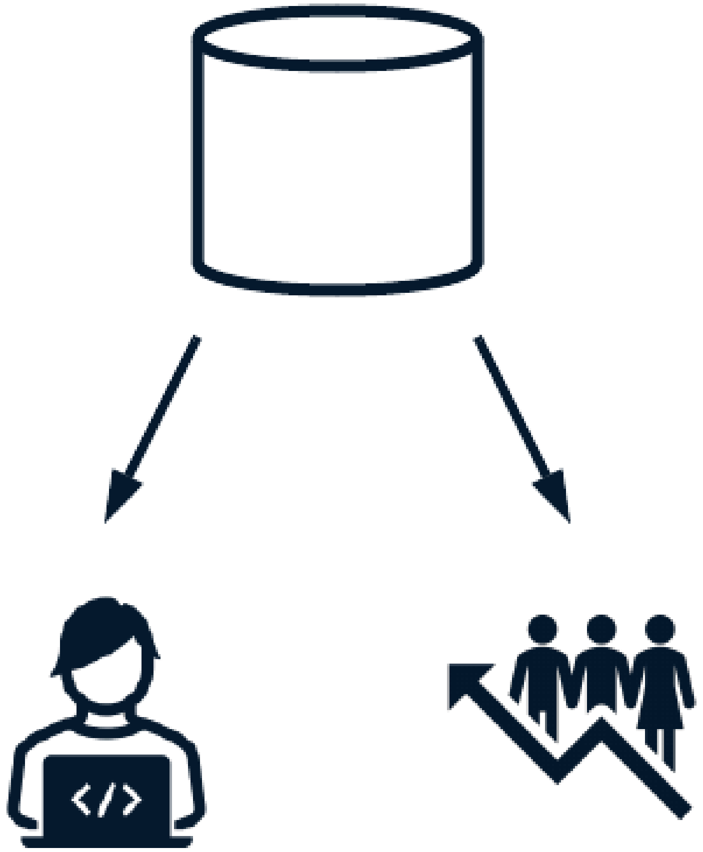
Jake Roach
Data Engineer

Load data to a SQL database with pandas



Loading data into a SQL database with pandas

...



`pandas` provides `.to_sql()` to persist data to SQL

- `name`
- `con`
- `if_exists`
- `index`
- `index_label`

Persisting data to Postgres with pandas

```python
# Create a connection object
connection_uri = "postgresql+psycopg2://repl:password@localhost:5432/market"
db_engine = sqlalchemy.create_engine(connection_uri)
```

```python
# Use the .to_sql() method to persist data to SQL
clean_stock_data.to_sql(
    name="filtered_stock_data",
    con=db_engine,
    if_exists="append",
    index=True,
    index_label="timestamps"
)
```

Validating data persistence with pandas

It's important to validate that data is persisted as expected.

- Ensure data can be queried
- Make sure counts match
- Validate that each row is present

```
# Pull data written to SQL table
to_validate = pd.read_sql("SELECT * FROM cleaned_stock_data", db_engine)
```

```
# Validate counts, record equality, etc
...
```

Let's practice!

ETL AND ELT IN PYTHON