Data Sources

Flat Files and Tables

Common sources:

- Excel, CSV, Parquet, etc.
- Tables in RDBMS (Relational Database Management Systems)

From RDBMS Table to DataFrame

Ask your DBA for the database hierarchy:

1. Database server
2. Database
3. Schema
4. Table

After creating a connection object in Python, you can load data directly into a pandas DataFrame.

Hierarchical Data Formats

Most common format: **HDF5**

- Each element of an n-dimensional dataset can be arbitrarily complex
- Allows storage of metadata alongside the data
- Popular for storing annotated image data

Exercise: HDF5

- Install the `h5py` package
- Read the file `mnist.hdf5` in Python
- Extract the first image as an array and identify its label
- Visualize the array as an image and verify the label

What is a REST API?

- Client-server architecture over HTTP
- Stateless communication: each request is independent
- Data can be cached to improve performance
- Uniform interface: consistent structure for communication
- Resources are identifiable and independent
- Clients can modify resources based on received representations
- Messages are self-descriptive
- Hypermedia allows discoverability of related actions
- Layered system separates concerns like security and load-balancing

Source: Red Hat - What is a REST API

OpenAPI: A REST Specification

> “The OpenAPI Specification (OAS) defines a standard, language-agnostic interface to HTTP APIs. It enables both humans and computers to understand the capabilities of the service without access to the source code.”

Source: Swagger OpenAPI Specification

Exercise: OpenAPI

- Go to https://petstore.swagger.io/
- Find the ID of a sold pet
- Retrieve the details of that pet using its ID

ODATA: A Specialized REST Implementation

Slogan: "The best way to REST"

Highlights:

- Focus on business logic
- Easy to consume RESTful APIs
- Machine-readable metadata via `$metadata`

Widely used in **Business Intelligence** contexts.

Exercise: ODATA

- Load data from the **People** endpoint of the Trip Pin service using Python
- Use `pandas` to convert it into a DataFrame
- Are the data tidy?

Working with JSON Files

- Use `pd.read_json()` to load JSON into a DataFrame with nested structure
- Use `pd.json_normalize()` to flatten simple JSON structures

Processing Complex JSON Files

- Deeply nested structures require step-by-step flattening
- Arrays can be tricky due to variable lengths

Exercise: JSON

- Reproduce the examples in `read_json_starter.ipynb`
- Load data from the **People** endpoint of the Trip Pin service
- Use `pandas` to create a tidy DataFrame from the JSON response

TensorFlow and Datasets

TensorFlow uses the `tf.data.Dataset` API for efficient data input pipelines.

- Dataset = stream of feature-label pairs (tensors)
- Can be created from:
  - NumPy arrays, TFRecord files
  - CSV or image directories
  - Text files or custom generators

Why `tf.data`?

- Handles large datasets efficiently (e.g. streaming, batching, parallel loading)
- Integrates seamlessly with `model.fit()`
- Supports transformations: `.map()`, `.batch()`, `.shuffle()`, `.prefetch()`

Example:

```
dataset = tf.data.Dataset.from_tensor_slices((x, y))
dataset = dataset.shuffle(100).batch(32).prefetch(tf.data.AUTOTUNE)
```

Example: Load Images with TensorFlow

TensorFlow provides a convenient method to load images:

```
train_ds = tf.keras.utils.image_dataset_from_directory(
    "path/to/data",
    image_size=(180, 180),
    batch_size=32
)
```

Each batch is a tuple: `(images, labels)`
This dataset can be passed directly into `model.fit()`.

Exercise: Load and Train with Images

- Download an image classification dataset (e.g. horses vs humans)
- Use `image_dataset_from_directory()` to load it
- Visualize a few samples
- Train a model using `model.fit(train_ds)`

Tutorial: TensorFlow - Load images

What is a TFRecord File?

- A binary file format used by TensorFlow for storing data.
- Stores `tf.train.Example` objects (like rows), serialized as protocol buffers.
- Efficient for large datasets and production pipelines.

Advantages of TFRecord

Feature	TFRecord	NumPy / CSV
Large-scale data support	✅ Yes	❌ Limited (in-memory)
Compact binary format	✅ Yes	❌ CSV is text-based
Heterogeneous data support	✅ (image + text)	❌ NumPy = numeric only
Streaming + parallel access	✅	❌
Built for TF	✅ Native format	❌ Needs conversion

Example: Write and Read TFRecord

```
# Writing
with tf.io.TFRecordWriter("data.tfrecord") as writer:
    example = tf.train.Example(features=tf.train.Features(feature={
        'label': tf.train.Feature(int64_list=tf.train.Int64List(value=[1])),
        'feature': tf.train.Feature(float_list=tf.train.FloatList(value=[0.1, 0.2]))
    }))
    writer.write(example.SerializeToString())

# Reading
raw_dataset = tf.data.TFRecordDataset("data.tfrecord")
```

You can then parse the records using `tf.io.parse_single_example()`.