Introduction to Data Engineering: Basics and Tools

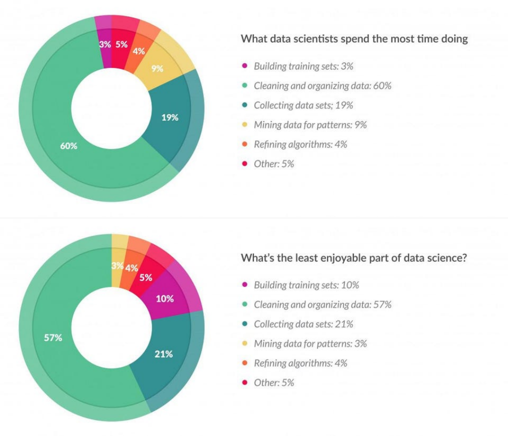
Sebastian Ernst, PhD

Course: Data Engineering, EAIiIBISIS.Ii8K.5dfa09851a120.22

Introduction

What is this course about?

- You know how to **design databases** and make them work.
- What to do if **data already exists**?
- Cool, we have a **schema**. But is it any **good**?
- **Data preparation** accounts for about 80% of the work of data scientists (Press 2016)
- Data engineers enable better **data science** (Gavin 2021)



Source: (Press 2016)

The problems with datasets

- **Getting data in** (at all) – design **schemes** or use tools like pgfutter or sqlitebiter
- Getting data in **too early** – e.g. **type outliers** leading to **wrong column types** or **omitted records**
- **Non-repeatable** datasets – our **case** involves datasets from **various origins** rather than a stable **data pipeline**

By PIX1861 - Pixabay, CC BY-SA 4.0,
https://commons.wikimedia.org/w/index.php?curid=112732031

What to expect in this course

- Coming up: **tools** we use **before** data goes in a **production database**
- How to **clean** and **visualise data** – practical **examples** and things to look out for
- Dealing with **spatial data**: how to (and how not to) include location data in the analytic pipeline
- Data in **series**: both **time series** and other examples of **data sequences**
- What to do if data no longer **fits in memory**
- Advanced **SQL** – because what you'll learn here is meant to **supplement**, not replace it
- More tools: managing **data pipelines**, data processing in **shell scripts**, using **literate programming** to document your process

Pandas

What is pandas?

- Python **library** for **data manipulation** and **analysis**
- Provides **data structures** with integrated **indexing**
- Can perform **joins**, **grouping** and data **alignment**
- Flexible **input/output** functions with many supported **file formats**
- High **performance** thanks to critical code written in Cython
- **Easy** to use:

```python
import pandas as pd
```

The Role of NumPy

- NumPy adds Python support for multi-dimensional **arrays** and **matrices**
- Pandas uses NumPy **data types** to store values and to create objects
- Pandas **data types** can often **act** similarly to NumPy structures

Data structures: Series

- One-dimensional labelled **array**, can hold **any data type**
- **Axis labels** constitute the **index**
- **Creating** a Series:

  ```
  s = pd.Series(data, index=index)
  ```

  where `data` is a `dict`, an `ndarray` or a scalar
- Acts similarly to an **ndarray** – e.g. supports **slicing** and can act as **input** to NumPy functions
- Also similar to a `dict` – can get and set values by **index label**
- Has a `dtype` and a `name`

Data structures: DataFrame

- **Two-dimensional**, labelled data structure
- Usually **created** from:
  - a `dict` of `Series` or (identical) `dicts`
  - a `dict` of `ndarrays`/`lists`
  - a `list` of `dicts`
- Has two *indexes*:
  - `index` – equivalent of `index` in a `Series`
  - `columns` – holds the names of columns

DataFrames: working with columns

- A DataFrame can be treated as a `dict` of `Series` objects
- **Columns** can be **created** by "adding" a new element to the `dict`:

```
df["three"] = df["one"] * df["two"]
```

- Or, if you want the column to appear **somewhere else** than at the end:

```
df.insert(1, "three", df["one"] * df["two"])
```

- A column can also be **accessed directly** as an **attribute** (e.g. `df.three`), but only if:
    - its name is a **valid Python identifier** (e.g. `df.1` is not allowed),
    - it doesn't **conflict with existing method** names (e.g. `df.min` is already taken),
    - it **already exists** (e.g. this notation cannot be used to create new columns)

DataFrames: viewing

- A DataFrame is often **too large** to be viewed as-is; Pandas will **omit** the rows and columns in the middle by default
- If you only want to see **a subset of rows**:
  - `df.head()` and `df.tail()` will print the `n` **first/last** rows (5 by default)
  - `df.sample()` displays a **random sample** of `n` rows (1 by default), or a given fraction (`frac`) of all rows
- **Summaries** are also available:
  - `df.describe()` returns a DataFrame with a **statistical** summary of the **numeric data** in a DataFrame
  - `df.info()` prints a **technical** summary of all columns

DataFrames: indexes and columns

- As mentioned, a DataFrame has **two index** structures, holding the row labels (`index`) and column names (`columns`).
- DataFrames can be **transposed**, which reverses their roles:

```
df.T
```

- The `index` can be **converted to a regular column** and replaced with the default one (0, 1, 2, 3, . . .):

```
df.reset_index()
```

- Also, any **column** can be designated as the `index`:

```
df.set_index('foobar')
```

DataFrames: selection and indexing

Operation	Syntax	Result
Select column	`df[col]`	Series
Select row by label	`df.loc[label]`	Series
Select row by integer location	`df.iloc[loc]`	Series
Slice rows	`df[5:10]`	DataFrame
Select rows by boolean vector	`df[bool_vec]`	DataFrame
Get scalar value	`df.loc[label, col]`	scalar

DataFrames: boolean indexing

- Used to **filter rows** based on some **properties** – can be viewed as an equivalent of the `WHERE` clause in SQL:

```
df[df["three"] > 3.14]
```

- **Conditions** can be joined using operators `|`, `&` and `~`, and must be grouped with **parentheses**:

```
df[(df["three"] > 3.14) & ~(df["two"] < 0)]
```

- For clarity, the **boolean vectors** (which are `Series`) can be stored as variables:

```
three_more_than_pi = df["three"] > 3.14
two_negative = df["two"] < 0
df[three_more_than_pi & ~two_negative]
```

Input/Output

- DataFrames can be **read** from and **written** to:
  - **file formats**: CSV, FWF, Excel, JSON, HTML (tables), XML, HDF, Feather, Apache Parquet, ORC, SAS, SPSS, Stata
  - **SQL databases** and **Google BigQuery**
  - **Python** and NumPy structures (see `to_dict`, `to_numpy`)
- Generally, to **create a DataFrame** from a data source, use `pd.read_FORMAT()` where `FORMAT` is the file format (e.g. `read_excel()` or `read_csv()`)...
- ...and to **save a DataFrame**, run the `df.to_FORMAT()` method of the DataFrame itself
- **Structured JSON** can be flattened using `json_normalize`

Bibliography

Bibliography

Gavin, Lewis. 2021. "What Is a Data Engineer? Clue: We're Data Science Enablers." In *97 Things Every Data Engineer Should Know: Collective Wisdom from the Experts*, 1st edition. O'Reilly Media.

Press, Gil. 2016. "Cleaning Big Data: Most Time-Consuming, Least Enjoyable Data Science Task, Survey Says." *Forbes*. https://www.forbes.com/sites/gilpress/2016/03/23/data-preparation-most-time-consuming-least-enjoyable-data-science-task-survey-says/.