**Extract-Transform-Load**

**Introduction**

In any machine learning problem, the first questions that must be asked concern *data*. Where do they come from? How are they structured? What transformations must be applied to convert them into a form suitable for a machine learning model? How should they be stored?

The answers to these questions fall under the purview of the Extract-Transform-Load (ETL) pipeline[[1]](#footnote-1). This pipeline *extracts* data from sources, *transforms* them into a usable format, and *loads* them into storage for future operations. In brief, it takes data that is heterogenous and makes it homogeneous[[2]](#footnote-2). The importance of ETL cannot be overstated. It is the foundation upon which this dissertation is built. It is also a monumental undertaking. As Ponniah (2010, pg. 284) sadly notes, “it is not uncommon for a project team to spend as much as 50% to 70% of project effort on ETL functions”. This was certainly the case, so considerable space is dedicated to explaining each stage, and how it was realised in the context of this dissertation, in the following section.

I definitely want to bitch about how I had to become my own SME.

**Extraction**

*Define*

Extraction is the process of collecting data, often from multiple different sources, and moving it to a *staging area*.

“Data extraction presupposes a selection process” (Ponniah, pg. 282). Data must be selected based on requirements.

Problems are numerous and varied.

Data is extracted from the source system into the staging area. If corrupted data is copied directly from the source into data warehouse database, rollback will be a challenge. Validations can be done during extraction: reconcile records with source data. Make sure that no spam / unwanted data is loaded. Data type check. Removal of all types of duplicate / fragmented data. Check whether keys are in place or not.

Extracts the data from different source systems.

The collection of data from various sources.

Retrieving data from external storage or transmission sources

Most data storage projects integrate data received from various source systems. Each individual system may employ a separate data organisation or format. Each individual system may employ a separate data organisation or format. Common data source structures are relational databases and pure data files.

Someone in the organisation identifies the desired data sources and the rows, columns, and fields to be extracted from those sources. These sources likely include transactional databases hosted on-site and in the cloud.

The process of reading data from a database. In this stage, data is collected, often from multiple and different types of sources.

Extracts the data from different source systems.

Part of the planning for this stage should include estimating data volumes for each data source. Data must be extracted in a way that doesn’t have a negative impact on source systems or response times.

Retrieving data from external data storage or transmission sources.

Target sources may include ERP, CRM, streaming sources, other enterprise systems, and data from third-party sources.

The main challenge is integrating with APIs. Integration is different for every application; not every product provides a vanilla REST API. Some REST APIs are surprisingly convoluted, and some applications are still stuck in protocols like SOAP. Many APIs are not rigorously and accurately documented. An API may have a large surface area with dozens of built-in resource endpoints. APIs are constantly changing and breaking.

The first phase of ETL. Data is collected from one or more data sources and held in temporary storage where the subsequent two phases can be executed. During extraction, validation rules are applied to test whether data has expected values essential to the data warehouse. Data that fails the validation is rejected and further processed to discover why it failed validation, allowing remediation if possible.

**Extraction for this dissertation**

It is well-known in machine learning that the more data is available to a model, the better (in general) that model would perform. The extraction process therefore sought to output as much data as possible

There are, broadly, four classes of data used in this dissertation: schedule, movement, location, and weather. Their sources, and the method of extraction, are described below.

**Schedule**

Schedule data is available from Peter Hicks’ website. Both full and update extracts are downloaded as gzipped files.

**Movement**

Darwin data is also available from Peter Hicks’ website. Each day is a bzip2-compressed tar file containing 1440 files, one for each minute in the day. Each file comprises XML messages. I/O is an expensive operation, so each minute is extracted in memory and written to one CSV for each file.

**Location**

NaPTAN is available as a zipped CSV from the Department for Transport (DfT). Only the railway nodes are extracted and saved. CORPUS is available from NR as a gzipped JSON. Only the TIPLOC information is extracted and saved.

**Weather**

This data is available for download as a CSV file, via FTP, CEDA, though headers must be downloaded separately. Each row has a *src\_id*, the identifier of the station responsible for that record.

Station data is also available from CEDA as a KMZ file. A KMZ file is a zipped KML (Keyhole Markup Language) file; KML is an XML notation for expressing geographic annotations and visualisations. The KML file is parsed using pykml and station metadata using BeautifulSoup to produce a CSV. Some processing is done here: latitudes and longitudes are converted to floats, station IDs to integers, and headers to standard Pythonic form.

**Transform**

*Define*

Needs to be cleansed, mapped, and transformed. Apply a set of functions to extraction data. Correcting data integrity issues.

Validations done during this stage including filtering, character set conversion and encoding handling, conversion of units of measurements, data threshold validation check. Cleaning, splitting and merging columns. Transposing rows and data, and using lookups.

Data is processed to make values and structure consistent across all data. Typical transformations include things like date formatting, resorting rows or columns of data, joining data from two values into one, or splitting data from one value into two. The goal of transformation is to make all data conform to the a uniform schema.

The cleansing and aggregation

There is a lot of overlap between transform and *pre-processing*, the stage at which data is cleaned, unnecessary fields removed, columns encoded, and so on. The difference lies in ease of repetition. Transform would, ideally, be performed once. Operations are typically expensive, operating on vast quantities of data. Pre-processing, using heavily optimised code in NumPy and Pandas, can be iteratively improved; transform less so.

Applies calculations, concatenations.

Uses a series of rules or operations to retrieve pure data from the source and deliver the data in its final form for manipulation at the receiving end. Some data sources need very little, or even no, data processing. Sometimes one or more transformation may be critical to match the business and technical requirements of the target database.

Transform according to business rules. Usually involves operations such as filtering, sorting, aggregating, joining, cleaning, de-duplicating, and validating. Often run in parallel to save time.

The cleansing and aggregation that may need to happen to data to prepare it for analysis. Two approaches.

Shift in recent years toward transforming data within the warehouse rather than transforming it beforehand, primarily driven by two factors: the increased performance and scalability of the modern analytics database, and the ability of in-database transformation to be written in SQL, the data manipulation language of choice for most analysts.

Transformations can be divided into basic and advanced. Basic:

* Cleaning: mapping NULL to 0, “Male” to “M”, formatting dates
* Deduplication: identifying and removing duplicate records
* Format revision: character set conversion, unit of measurement conversion, data/time conversion
* Key restructuring: establishing key relationships across tables .

Advanced

* Derivation: applying business rules to your data that derive new calculated values from existing data
* Filtering: selecting only certain rows and / or columns
* Joining: linking data from multiple sources
* Splitting: splitting a single column into multiple columns
* Data validation: simple or complex data validation
* Summarisation
* Aggregation: aggregating data elements from multiple data sources and databases
* Integration: reconciling different data names and values for the same data element.

Transformation converts the raw data that has been extracted from the source server. It gets cleaned, mapped, and transformed, often to a specific data schema, so it will meet operational needs. Cleaning, format revision, data threshold validation checks, restructuring, de-duplication, filtering, merging, splitting, derivation, summarisation, integration, aggregation and complex data validation.

Data is not usually loaded directly into the target data warehouse, but it is common to have it uploaded into a staging database. This step ensures a quick roll in case something does not go as planned. Possibility to generate audit reports for regulatory compliance or diagnose and repair any data issues.

**Transform for this dissertation**

Unifying data via ID

Correctly identifying SCHEDULEs

Getting destination times for trains to match with schedules

Converting into a tabular format

**Transforming schedules**

Extracting schedules is a difficult task.

To retrieve the correct schedule for a given UID on a given day, all schedules with that UID must be retrieved. Those not active on the *day* in question – as defined by *days\_run* – can be discarded, as well as those with a start date in the future. From those remaining, the schedule with the lowest *stp\_indicator* is the correct one, in order “C”, “O”, “V”, “P”. “C” is a planned cancellation; “O” is an overly from STP; “V” is an overlay from LTP; “P” is the permanent base schedule.

As mentioned, an update CIF is released each morning at 0100. It contains deletions, amendments, and new schedules, which must be applied to the currently-held schedule database. After each update is applied, the schedules for that day are written to file by filtering those by the criteria above.

There is tremendous scope for error. Train scheduling is a very complex problem, and this is reflected in the data formats. Fortunately, full schedules are available weekly, so errors are reset every week. It is likely the bugs in this stage will lead to later problems.

CIF format. Each line is 80 characters.

Each schedule is a CIF file. CIF is obsolete

Each type of point (LO, LI, and LT) is standardised into a common location format, L.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **LO** | **LI** | **LT** | **Description** |
| *type* | Y | Y | Y | Point type. Either LO (origin), LI (intermediate), or LT (terminus) |
| *location* | Y | Y | Y | Location. A TIPLOC |
| *suffix* | Y | Y | Y | The number of times the location TIPLOC has appeared |
| *sta* |  | Y | Y | Scheduled time of arrival |
| *std* | Y | Y |  | Scheduled time of departure |
| *stp* |  | Y |  | Scheduled time of pass |
| *pta* |  | Y | Y | Planned time of arrival |
| *ptd* | Y | Y |  | Planned time of departure |
| *pass* | Y | Y | Y | Whether or not the event is a pass |
| *platform* | Y | Y | Y | The platform used by the service |
| *line* | Y | Y |  | Line to be used on departure from the location |
| *path* |  | Y | Y | Line to be used on arrival at the location |
| *activity* | Y | Y | Y | The activities occurring at this location |
| *engineering\_allowance* | Y | Y |  | Time allowed for recovery from engineering activities |
| *pathing\_allowance* | Y | Y |  | Time allowed for pathing requirements |
| *performance\_allowance* | Y | Y |  |  |

Two types of record are ignored to reduce complexity: CR (change en route) and AA (associations).

Associations also have the potential to be a rich source of predictions. Perhaps some metadata could be extracted. Is, for instance, a train with more associations more likely to be delayed? It seems probable. But the problem with using AAs is fundamental. Doing so would necessitate a much more complex model – one that could handle dependencies between trains – rather than the (simpler) tabular format desired.

CRs indicate that some metadata contained with BRs changes during a train’s journey. The main motivator here was to reduce the size of the dataset.

The metadata for a day is roughly 60 MB. The *route* information for a day is roughly 800 MB. Storing the metadata once, rather than merging it onto every location record, significantly reduced space requirements. The other potential solution – checking a database of CRs for every L - would have incurred significant cost later in the pipeline.

Several fields are one-hot encoded here: activity, characteristics, and catering.

A significant reduction in complexity may be achieved by ignoring CR (change en route) records

**Transforming location**

Each day of train movements is a file roughly 1.5GB in size. This file doesn’t just contain movements – it also contains a variety of other messages: forecasts, schedules, station messages, alarms, warnings, and so on. The data must be converted into a CSV for usage.

It is assumed that no trains runs longer than 24 hours, i.e. there is no train that starts on a given date and ends two days afterward.

Two file writers are maintained: one for today, and one for *yesterday*. Initially, yesterday is null. Once today’s file has been read, yesterday is closed if it is not null. Today and yesterday are then switched. A train that originates on 2018-05-23 but terminates on 2018-05-24 will be written to 2018-05-23. This ensures consistency with the day’s schedules. Each movement message contains an *ssd* field.

Each record may have a selection of arrival, departure, and pass times: working, estimated, and public.

As Darwin is the source of all data displayed at station, there is a lot of metadata relating to whether to *show* certain fields to the public – primarily platform information – which can be safely discarded. I think that it should be here. Other fields may also be trimmed: predicted times of arrival, departure, and passing, for instance. SCHEDULE will serve as ground truth for schedules. Only the actual arrival and departure times concern us.

This is an important decision. Again, it simplifies the issue.

**Transforming weather**

Wind direction is fairly consistently south-westerly in the UK, driven by the North Atlantic Current. Missing values are filled with the mean and converted to compass directions.

As mentioned previously, there are 10 available fields for forecasts, of which only 7 could be meaningfully mapped to equivalent MIDAS fields. For several, this was a simple unit conversion. Boolean masks are used to convert *wind\_speed* and *wind\_gust­­* to mph. For visibility and weather type, predefined codes were used to establish a map between the two formats[[3]](#footnote-3).

|  |  |  |
| --- | --- | --- |
| **MIDAS** | **Datapoint** | **Notes** |
| *prst\_wx\_id* | Weather type | MIDAS code definition depends on *src\_opr\_type.* Use mapping between relevant table and Datapoint code definition |
| *rltv\_hum* (%) | Screen relative humidity (%) | No conversion necessary |
| *visibility* (decametre) | Visibility | Used code definition |
| *air­­\_temperature* (°C) | Screen temperature (°C) | No conversion necessary |
| *wind\_direction* (degrees) | Wind direction (16-point compass) | Map degrees to compass directions, with 0 as N. |
| *wind\_speed* (knots or ms-1) | Wind speed (mph) | MIDAS unit depends on *wind\_speed\_unit\_id* |
| *q10mnt\_mxgst\_spd* (knots or ms-1) | Wind gust (mph) | MIDAS unit depends on *wind\_speed\_unit\_id* |
|  | Feels-like temperature (°C) | No equivalent |
|  | Precipitation probability (%) | Clearly nonsensical |
|  | UV Index | No equivalent |

**Transforming location**

There are 2563 railway stations in the UK[[4]](#footnote-4). The discrepancy is

**Load**

Loading is the simplest stage. It is simply the storing of data in a format accessible for future usage. Sends data to the receiving end, likely to be data storage. May be very simple or very complex, depending on the needs of the application. Some data storage methods may replace old data with cumulative data. Updating of extracted data is normally done on a periodic basis.

Move the transformed data into the permanent, target database. Once loaded, the ETL process is complete. In many organisations ETL is performed regularly in order to keep the data warehouse updated with the latest data.

The process of writing converted data from a staging area to a target database.

Ensure that key field data is neither missing nor null

Test modelling views based on the target tables

Check that combined values and calculated measures

**Load for this dissertation**

Loading is a more involved process here than normal. It involves the merging of the four data strands discussed: schedule, location, movement, and weather.

Each row should be a train. We merge on UID, for each day. We identify the origin and terminus of each train, and calculate the delay using the scheduled values. We use the origin and destination locations to identify the closest weather station, and from there we add information on the weather experienced at the start and end of a train’s journey.

Without weather, location is also unnecessary, and so the task is greatly simplified. Two columns are added to the ‘schedule’ dataframe: the actual time of arrival (at the destination; ‘ata’) and the actual time of departure (from the origin; ‘atd’). The UIDs from ‘schedule’ are used to select records from ‘movement’. Those with TIPLOCs matching the ‘origin’ and ‘destination’ in ‘schedule’ are used to fill in the ‘ata’ and ‘atd’ respectively. There is plenty of scope for error here, in particular ‘schedule’ UIDs not in ‘movement’ and mismatching TIPLOCs between the ‘schedule’ and ‘movement’ dataframes. These issues cannot be resolved. They are believed to be the result of two planning processes which generate, or modify, schedules: VSTP and STP. In short, it is possible there exists trains in ‘movement’ with no corresponding metadata in ‘schedule’, corresponding to schedules created by VSTP and STP, and vice versa, corresponding to schedules cancelled by either of the two. However, there are still a significant number of rows per day

**Sources**

* <https://www.webopedia.com/TERM/E/ETL.html>
* <https://www.sas.com/en_us/insights/data-management/what-is-etl.html>
* <https://databricks.com/glossary/extract-transform-load>
* <https://www.stitchdata.com/etldatabase/etl-extract/>
* <https://www.informatica.com/services-and-training/glossary-of-terms/extract-transform-load-definition.html>
* <https://docs.microsoft.com/en-us/azure/architecture/data-guide/relational-data/etl>
* <https://www.techopedia.com/definition/24170/extract-transform-load-etl>
* <https://business.baylor.edu/gina_green/teaching/dw/spr16/Ponniah_data-warehousing-fundamentals-for-it-professionalsSecondEdition.pdf>
* <http://www.cs.uoi.gr/~pvassil/publications/2009_DB_encyclopedia/Extract-Transform-Load.pdf>

1. Sometimes a different order is used: Extract-Load-Transform (ELT). The sole difference is where the transformation stage takes place. The distinction is largely academic here, so ETL is used throughout. [↑](#footnote-ref-1)
2. <https://www.informatica.com/services-and-training/glossary-of-terms/extract-transform-load-definition.html> [↑](#footnote-ref-2)
3. https://www.metoffice.gov.uk/services/data/datapoint/code-definitions [↑](#footnote-ref-3)
4. https://web.archive.org/web/20180907144642/https://dataportal.orr.gov.uk/displayreport/report/html/640e836d-8863-4243-b794-df1abae05639 [↑](#footnote-ref-4)