MSc Data Analytics Dissertation: Visualisation of Traffic Flow Data for Glasgow City Council

Client Report

**Elijah Reid**

**202093659**

**2021**

# Organisational Context

The Glasgow City Council is responsible for running the majority of public services in and around the Glasgow area, and in order to better apply itself across so many different simultaneous projects it is split up into several different departments and teams, two of which are the instigators of this project.

The Strategic Innovation and Technology (SIT) Data team are responsible for finding better ways to use data, technology, and user engagement to create new insights that can help to inform and shape new developments across both the Council and the City of Glasgow.

Traffcom are responsible for controlling and the management of traffic in the Glasgow City area, with their responsibilities including the maintenance and operation of over 1000 sets of traffic lights across the city, each of which make use of various adaptive control techniques. The signals generate and collect a large amount of real time data, some of which is made available to other users and departments in the council.

The SIT Data Team and Traffcom have begun to recently collaborate on several projects to consider how the Council might open up new city data to the public, how they can analyse it to validate its accuracy and utility for further use, as well as how they might be able to visualise the data to generate new insights and further understanding around what the data tells them about Glasgow.

# Background to Project

Traffcom wants to explore whether the traffic flow data generated from the pre-existing Urban Traffic Control, or SCOOT system, for Glasgow can be augmented with directional information, and following this whether it can be visualised for the city traffic network.

A proof-of-concept project featuring a small subset of the data was released to the MSc in Data Analytics programme at the University of Strathclyde in October 2020 in order to judge the feasibility of the project by bringing multiple outside perspectives in to analyse the data. After receiving back several promising ideas, the Traffcom and SIT data teams proceeded to set up a more ambitious project to be undertaken over the summer by a student from one of these groups.

The project will make use of data already collected from the Glasgow SCOOT system, which collects data from road-side sensors and adjusts traffic signals in response to real-time traffic flows. There are currently over 1500 existing links in the network, with traffic flow data being sent to a central database every 15 minutes, with data points dating back as far as May 2018. Currently live and historical traffic flow data can be requested via APIs; however, they require specific queries to provide relevant data as well as missing a directional component, limiting their use for fully understanding the data and creating visualisations.

# Project Value and Expected Outcomes

The project involves coming up with a method to establish traffic direction for each of the nodes in the network of traffic sensors across the city and then using this data to create a visualisation, ideally using software tools that are already in use or readily available to the Glasgow City Council.

The visualisation will be based off the 3+ years of historic data collected by the Glasgow City Council, as this will allow for the end users to analyse and predict traffic trends based off this. It is expected to be at least somewhat configurable to allow different groups to view their areas of interests as well as allowing different time periods and different routes/areas to be selectable for more in-depth analysis.

The plan for the project is to set up an ETL pipeline which will extract each SCOOT link from the master text file maintained by Traffcom, and then transform this extracted data into a table of pairs of linked nodes. As new links are very rarely added and only manually, the script to update the links will only run on an irregular basis and might even be only activated manually by Traffcom management.

Data will also be pulled in from the Historical Traffic Flow API to provide the location co-ordinates for each link as well as the traffic flow there over the past few years. This will be scheduled, either with a simple Cron script or a more advanced solution like Apache Airflow to update the traffic flow every 15 minutes. An additional script for undocumented links or links with old positional data will calculate the bearings for entry links, as well as converting the old British National Grid co-ordinates into Latitude and Longitude to ensure all positional data is in the same format.

Once all the links and related data are processed, it will then be fed into a Geographic Information System (GIS) Mapping software (either Arc GIS which is already in use at the council, or another free software such as QGIS, Leaflet or Plotly). The map will highlight all the links, with an indication of the volume and direction of traffic flow, with additional information being potentially added if time allows. This map will be included in a visual analytics software for internal use (either a more in-depth ArcGIS setup or a simpler product such as Microsoft Power BI or Tableau) as well as being hosted on the web for public access.

# Initial Data Pipeline Mock-up

Shape

Description automatically generated with medium confidence

# Client Use Case

The visualisation produced as an end-product of the project will be of use to a range of stakeholders, including traffic engineers and managers who can use it to monitor live traffic and predict future trends based on the historical data.

City stakeholders will be able to use it to anticipate the building or removal of connecting roads as well as the effect on traffic from closing roads for works at different times of the day and year.

Finally, residents of Glasgow will also be able to make use of the data and visualisation as it will be hosted as an app as part of the Glasgow Open Data initiative, allowing them to keep track of traffic themselves whilst also allowing local entrepreneurs to make use of the data for their own future projects.

# Project Plan

## Initial Plan

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Week 1 | Week 2 | Week 3 | Week 4 | Week 5 | Week 6 | Week 7 | Week 8 | Week 9 | Week 10 | Week 11 | Week 12 |
| 31-May | 07-Jun | 14-Jun | 21-Jun | 28-Jun | 05-Jul | 12-Jul | 19-Jul | 26-Jul | 02-Aug | 09-Aug | 16-Aug |
| Intro to project, datasets and currently used tools  Meeting with traffic, data and visualisation departments  Extract relevant link data from Network file  Ethics Form completed | Gained access to ArcGIS and FME  Complete Overview Document  Begin research into traffic visualisation techniques  Extract all co-ordinate data for each link from API  Team Meeting Wednesday | Get each SCOOT link plotted on a map (without directions)  Create a table containing every junction connection pair with latitude and longitude for each  Continue visualisation research  Team Meeting Wednesday | Write a function to calculate the bearings for each of these pairs and add to table  Begin Literature Review and Methodology section  Further visualisation research if necessary  Team Meeting Wednesday | Begin trying to visualise connections between the junctions as lines  Team Meeting Wednesday  Check in with SLP (Wed) and Pierro and Liz (Fri) | Continue visualisation process as per week 5  Redraft Literature Review and Methodology  Team Meeting Wednesday | If visualisation has worked attempt to automate process from weeks 1 to 7  Gain access to data for traffic flow and other information to be overlayed  Team Meeting Wednesday | Add the flow information to the visualisation, colour coding by set criteria (to be agreed upon)  Begin writing client report for GCC internal use  Discuss system for internal use- using Python and R, creating a dashboard (PowerBI etc) or online  Team Meeting Wednesday | Begin to implement the internal system  Begin research in ways to host visualisation online for public access  Create aggregates of a few different time periods to speed up user requests  Team Meeting Wednesday  Check in with SLP (Wed) and Pierro and Liz (Fri) | Continue working on the internal system  Begin creation of web version of the visualisation  Potentially start adding other data sources if time allows  Team Meeting Wednesday | Continue working on the public version  Review and redraft client report and technical manual  Team Meeting Wednesday | Handover of project and mop-up any remaining issues  Write conclusions and recommendations for dissertation and spend next few weeks reviewing  Team Meeting Wednesday |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Week 1 | Week 2 | Week 3 | Week 4 | Week 5 | Week 6 | Week 7 | Week 8 | Week 9 | Week 10 | Week 11 | Week 12 | Week 13 | Week 14 |
| 31-May | 07-Jun | 14-Jun | 21-Jun | 28-Jun | 05-Jul | 12-Jul | 19-Jul | 26-Jul | 02-Aug | 09-Aug | 16-Aug | 23-Aug | 30-Aug |
| Intro to project, datasets and currently used tools  Meeting with traffic, data and visualisation departments  Extract relevant link data from Network file  Ethics Form completed | Gained access to ArcGIS and FME  Complete Overview Document  Begin research into traffic visualisation techniques  Extract all co-ordinate data for each link from API  Team Meeting Wednesday | Get each SCOOT link plotted on a map (without directions)  Extracted location data from Traffcom database instead of API  Continue visualisation research  Team Meeting Wednesday | Begin Literature Review and Methodology section  Further visualisation research  Create a table containing every junction connection pair with latitude and longitude for each  Team Meeting Wednesday | Ill with Covid and unable to work | Ill with Covid and unable to work | Team Meeting Wednesday  Check in with SLP (Wed) and Pierro and Liz (Fri)  Construct FME pipelines based on meeting with Liz | Complete FME pipelines and convert data to ArcGIS format  Team Meeting Wednesday | Begin trying to visualise connections between the junctions as lines  Gain access to data for traffic flow and other information to be overlayed  Redraft Literature Review and Methodology  Team Meeting Wednesday | Add the flow information to the visualisation, colour coding by set criteria (provided by Traffcom)  Begin writing client report for GCC internal use  Discuss system for internal use- using Python and R  Team Meeting Wednesday | Automate previous Python and FME process  Begin to implement the internal system  Team Meeting Wednesday  Check in with SLP (Wed) and Pierro and Liz (Fri) | Continue working on the internal system  Potentially start adding other data sources if time allows  Team Meeting Wednesday | Present findings to Council Leadership  Review and redraft client report and technical manual  Team Meeting Wednesday | Handover of project and mop-up any remaining issues  Write conclusions and recommendations for dissertation and spend next few weeks reviewing  Team Meeting Wednesday |

## Updated Plan

# References

* Glasgow City Council (2018) *Digital Glasgow Strategy*, Glasgow: Scottish Government.
* Director of Governance and Solicitor to the Council (March 2018) *CORPORATE ASSET MANAGEMENT PLAN - UPDATE*, Glasgow: Glasgow City Council.

A picture containing graphical user interface

Description automatically generated

MSc Data Analytics Dissertation: Visualisation of Traffic Flow Data for Glasgow City Council

Client Report

**Elijah Reid**

**202093659**

**2021**

Graphical user interface, text, application

Description automatically generated

Signed statement

Except where explicitly stated, all the work in this dissertation – including any appendices – is my own and was carried out by me during my MSc course. It has not been submitted for assessment in any other context.

Signed: ……………………………………………………………………….

Contents

## Project Context

[Organisational Context 1](#_Toc82185696)

[Background to Project 1](#_Toc82185697)

[Project Value and Expected Outcomes 2](#_Toc82185698)

[Initial Data Pipeline Mock-up 3](#_Toc82185699)

[Client Use Case 4](#_Toc82185700)

[Project Plan 5](#_Toc82185701)

[Initial Plan 5](#_Toc82185702)

[Updated Plan 6](#_Toc82185703)

[References 7](#_Toc82185704)

## Client Report

[1.0 Executive Summary 4](#_Toc82185705)

[2.0 Acknowledgements 5](#_Toc82185706)

[3.0 Introduction/Problem Statement 6](#_Toc82185707)

[4.0 Literature Review 7](#_Toc82185708)

[4.1 Data in Traffic Management 7](#_Toc82185709)

[4.2 Working with Unstructured Data 8](#_Toc82185710)

[4.2.1 Unstructured Data and it’s Issues 8](#_Toc82185711)

[4.2.2 Regex 9](#_Toc82185712)

[4.3 Creating Generic and Reusable Data Pipelines 9](#_Toc82185713)

[4.4 Visualising Geospatial Data 11](#_Toc82185714)

[5.0 Methodology 13](#_Toc82185715)

[5.1 Introduction 13](#_Toc82185716)

[5.2 Methodology 13](#_Toc82185717)

[5.2.1 Overview of the Master Text File 13](#_Toc82185718)

[5.2.2 Bash Script with Regex 14](#_Toc82185719)

[5.2.3 Cleaning and Splitting Text File with Python 15](#_Toc82185720)

[5.2.4 Extracting Location Data from Traffic API 17](#_Toc82185721)

[5.2.5 Converting the text file into a csv table 17](#_Toc82185722)

[5.2.6 Combining Connection and Location Datasets 19](#_Toc82185723)

[5.2.7 Generating and Collecting Flow Dataset from the Traffic API 20](#_Toc82185724)

[5.2.8 Preparing Data for Visualisation 22](#_Toc82185725)

[5.2.9 Visualisation and Final Amendments 23](#_Toc82185726)

[6.0 Conclusion and Future Work 26](#_Toc82185727)

[6.1 Results to Date 26](#_Toc82185728)

[6.2 Application of Results 28](#_Toc82185729)

[6.2.1 Traffic Engineers 28](#_Toc82185730)

[6.2.2 City Stakeholders and Decision Makers 29](#_Toc82185731)

[6.2.3 Glasgow City Residents 30](#_Toc82185732)

[6.3 Limitations 30](#_Toc82185733)

[6.4 Suggested Further Work 31](#_Toc82185734)

[6.4.1 Data Gaps and Data Augmentation 31](#_Toc82185735)

[6.4.2 App Improvements and Data Sharing 32](#_Toc82185736)

[7.0 Appendix 34](#_Toc82185737)

[8.0 References 51](#_Toc82185738)

# Executive Summary

This report includes details about the steps undertaken in the creation of a data pipeline and traffic visualisation app on behalf of the Glasgow City Council. This app would allow Traffcom to better understand and control traffic patterns, whilst also acting as a proof of concept for future data visualisation projects the Council may wish to carry out in the future.

The goal of this project was to find a way to establish the traffic direction for the network of traffic sensors across the city and then subsequently create an informative visualisation making use of pre-existing software tools available to the Council.

Some of the key facts that the Council wished to be included in the design of the visualisation are the following:

* The visualisation must make use of software tools that are either already available to GCC (such as ArcGIS spatial mapping applications) or those which can be freely licenced by the Council.
* The visualisations should be configurable so that different groups within in the council are able to view only the information that is relevant to them.
* The visualisation should be based off historical traffic flow data rather than real time data.
* Following this should be possible to select different time periods when viewing the traffic flow visualisation
* It should also be possible to view and select different routes and areas

The central challenge comes from the fact that the geo-position and the direction of travel for the SCOOT detectors are not included in the master dataset of SCOOT links and will therefore have to be extracted or interpolated from other sources.

To tackle the challenges, research was undertaken into the usage and benefits of using data in local government for city management, specifically in traffic management. More specific research was undertaken into the best methods for extracting data from unstructured text files, as the files used by the council contain inconsistent amounts of data for each column.

Research was also undertaken into the best programming languages and software packages to use in order to create a generic and reusable data pipeline, that could be scheduled to run at regular intervals without the need for direct intervention from the end user. Time was also spent in understanding the various methods by which geospatial data is recorded by the Council and in other sources, and following on from this, the best way to prepare and display this data to demonstrate the data’s utility to decision makers with the Council. A link to the source code of the final app can be found here: <https://github.com/Wizzzzzzard/Glasgow-City-Council-Project>

The ArcGIS app allowed Traffcom to visualise the direction and volume of traffic flow on a map of the Glasgow City Council area. Over 1000 junctions and their directions were successfully displayed, and the app allowed the flow of traffic to be easily followed from junction to junction. While only two time periods were included in the current version of the app, this can be easily expanded to as large or as small a range as needed using the pre-existing data pipeline and app as a jumping off point. Other departments of the council were also impressed by the potential utility of the app, and future plans for additional app functionality have already been suggested, which will ultimately lead to the Data Team’s goal of improving data access and transparency within Glasgow.

# 2.0 Acknowledgements

I would like to thank the Glasgow City Council for giving me the opportunity to work with them on this ground-breaking project, as well as providing me with access to various data sources and for their guidance and assistance over the course of the project. I’d like to particularly thank Stephen Sprott, Brian Davidson and Liz Irvine for supporting me through the entire project and providing a few moments of levity when progress was slow. Sparky the cat in particular deserves an honourable mention for ensuring time spent waiting for programs to load wasn’t wasted. I would also like to thank my advisor, Kerem Akartunali, for his supervision and advice throughout this project.

# Introduction/Problem Statement

This project was carried out at the behest of the Glasgow City Council Strategic Innovation and Technology (SIT) data team and Traffcom. The client has several separate and unconnected data sources: one a large master text file of each Split Cycle Offset Optimisation Technique (SCOOT) junction in Glasgow with each of its connecting links along with other relevant and irrelevant data. There were also two APIs which were connected to a larger data warehouse, one which output a list of SCOOT links along with their geographical location and a second API which outputs the traffic concentration at each SCOOT link that is updated on a 15-minute basis.

The client wanted to find a way to use these disparate sources of information to create an application that would allow them to see the direction traffic was flowing in at each junction, as well as monitoring the concentration of traffic at each point. The client wanted the script for extracting the junction links from the master text file to be generic and reusable, as new SCOOT links are added to it on a semi-regular basis and are even sometimes removed. The client then wanted a script that could match each link-to-link connection to its corresponding coordinates so that the traffic flow between them could be displayed on a map. Finally, a script to update the traffic flow at each junction every 15 minutes, that would also pull in the historical flow data was desired as this would allow for a more informative system than the one currently in place. At present each junction is just a single point on a static map of Glasgow with a simple colour coded number to denote the flow volume of traffic, which makes it difficult to update the map and to judge, at a glance, the direction in which traffic is moving.

In terms of the overall data flow, the client wants the data from the master text file and APIs to be extracted and transformed into a useable state, before being passed onto a Global Information System (GIS) mapping software- preferably one which the council has already licensed, such as ArcGIS, or an opensource software.

The structure of the dissertation is outlined below. It will begin with an exploration of the literature surrounding big data in local government and traffic management, as well as some literature behind the main problems to be solved in the project. This went on to inform the direction of the methodology used in constructing this project, which will consist of the data extraction, prototyping the processing and transformation of the data, followed by an exploration of the methods used to display the data in a useful form and finishing with a conclusion explaining the results of the project and recommendations for the client to carry out in the future.

# 4.0 Literature Review

The literature review is split into a number of sections: First a brief overview of the usage of data in Local Government, specifically for use in Traffic Management, followed by some more pertinent research into best practice methods for working with text, spreadsheets and other data formats found in the project as well as creating maintainable data pipelines. Finally, some research into the different ways geospatial data can be stored, and more practically relevant to the project, how this data can then be best visualised for the Council’s specific use case.

## 4.1 Data in Traffic Management

Using data for the purpose of Traffic Management is a practice used across the globe with many governments having invested in Intelligent Transport Systems. While initially a rather vague term it was properly defined in the 2010/40/EU directive of the European Unions as:

‘…systems in which information and communication technologies are applied in the field of road transport, including infrastructure, vehicles and users, and in traffic management and mobility management, as well as for interfaces with other modes of transport.’

*(Official Journal of the European Union, 2010)*

There are several systems in use across the UK and Europe to manage traffic, with the UK systems united under the broader umbrella of ITS (UK) *(ITS UK, 2017)*. The current system in use by Glasgow City Council is the Split Cycle Offset Optimisation Technique, or SCOOT system, a real time adaptive traffic control system for controlling and coordinating traffic signal across an urban road network *(Department for Transport, 1995)*. SCOOT has been shown to consistently outperform other methods of traffic control, with 23% and 30% delay reductions in Worcester and Southampton respectively when compared to other systems, and in Glasgow specifically it has reduced vehicle delays by as much as 12% *(Robertson, D.I. and Hunt, P.B., 1983)*. A SCOOT network is divided into various regions, each of which contains a number of nodes (traffic signal junctions and pedestrian crossings). The SCOOT network keeps a track of the current red and green light timings and optimises these timings by adjusting them in small increments in order to improve the overall performance of each region’s traffic signal network. By a combination of these relatively small changes to traffic signal timings the SCOOT system can adapt and respond to short term local traffic peaks, while also maintaining a capability to follow long term traffic trends (*Department for Transport, 1995*).

Diagram

Description automatically generated

Data Flow in a SCOOT system

Much of the research and development on SCOOT has been carried out by the European Union, and the data from the European wide SCOOT network is stored in a central database, ASTRID (*Hounsell, N. and McLeod, F., 2005*). Data for this database is stored across Europe in a unified format; the DATEX II standard. Created by the European Union in collaboration with its various member states, it sets forth a specification for how traffic data should be stored, allowing for data to be easily shared across European countries to enable better management of the European Road Network (*Easyway, 2012*). It contains many key features including the ability to show the level of service of a road network, travel times and various incidents such as road accidents, road works and other obstructions. The existence of such a data standard is a key lynchpin in European ITS’s and is the primary method of storing traffic data throughout Scotland (*Transport Scotland, 2015*).

Currently the SCOOT system in use at Glasgow contains data on various nodes situated around Glasgow as well as what other nodes they connect to. Separate to this is a traffic flow dataset which contains the traffic flowing through a particular node at 15-minute intervals dating back to May 2018. However as these are two distinct datasets it is impossible with the current incarnation of the SCOOT network to track the flow of traffic from node to node, and therefore the key aim of this project is to combine the unstructured node data used in the SCOOT system with the structured big dataset containing the flow data for each node, to allow a visualisation of directional traffic flow to be created.

## 4.2 Working with Unstructured Data

Oftentimes when working with real world data it tends not to be in a neat and tidy format as typically seen in classroom projects (*Hernández et al., 1998*). As a result of this it is often necessary to extract data from files that can be difficult for a computer to parse, be it from a raw text file written for explicitly human use, a spreadsheet with missing data or other more exotic formats.

### 4.2.1 Unstructured Data and it’s Issues

In a recent survey it was found that data scientists estimate that up to 80% of a project is spent on acquiring and transforming the data for its final use (*Press, 2016*). This is a significant percentage and suggests that the majority of this project will be spent preparing the data for visualisation rather than adjusting the visualisation itself. In order to be able to use multiple different datasets it is necessary for them to be ‘transcodeable’, that is they must have enough attributes in common and a similar enough structure that they can be matched up closely enough to be used together. This requirement can be further complicated by the presence of missing data, as blank fields can lead to myriad errors when joining datasets (*Van Beek, 2018*).

In the case of the text file used in this project, the data does contain a roughly consistent structure, with an almost tabular layout and with each junction containing mostly the same information. However, each junction has a variable record length, with several containing blank values, and so basic methods such as simply extracting ten lines for every fifty won’t work. As a result, a more intelligent method of data extraction is required.

### 4.2.2 Regex

The Regex Quick Syntax Reference describes a regular expression as ‘a finite character sequence defining a search pattern’ (*Nagy, 2018*). These patterns are most often used to test whether a string matches a certain user-defined search expression and are particularly useful for extracting information from text and configuration files, such as the one used by Traffcom.

Regex is however not useful in and of itself and requires a text streaming package of some description to allow it to be used for a practical purpose. Blum and Bresnahan (2015) more pragmatically define a regular expression as ‘a pattern template you define that a Linux utility uses to filter text.’ Regex originally originated within the Unix shell (*Aho, 1992*), the precursor to Linux and so it’s most powerful functionality can be best used via the Linux shell, or on a Windows system via the Bash shell (a Windows compatible version of the Linux shell) (*Loewen, 2021*). Linux contains a number of utilities capable of parsing and filtering text data, including the sed editor, the gawk program and the grep utility.

While on the surface these programs seem like different flavours text editing packages, they actually have a number of key distinctions that help to narrow down the options when deciding which one to use for creating the shell script.

The Linux Command Line and Shell Scripting Bible (2015) defines sed as ‘a stream editor … which edits a stream of data based on a set of rules you supply ahead of time before the editor processes the data.’ It is best used for text files with a semi-regular structure in which minimal processing is required beyond extracting the matching pattern. Gawk on the contrary provides ‘a programming language instead of just editor commands.’ It allows the user to define variables to store data and to use more advanced programming concepts such as if-then statements and loops to add logic to the data processing.

Ultimately due to the relative simplicity of the text file used for this project, it was decided to proceed using sed, as this would require less processing power and would be able to run quicker than Gawk. If in the future the text file becomes significantly larger, then adjusting the script to gawk is a relatively simple process due to the interchangeability of the Linux commands.

## 4.3 Creating Generic and Reusable Data Pipelines

When creating a reusable data pipeline, particularly one that involves unstructured data, it is important to ensure that each stage of the pipeline is generic enough to always produce results of a similar quality, while also specific enough to ensure those results don’t change.

Van Beek (2018) sets forth a five-step process for preparing unstructured data as follows: 1. Ingesting the data – i.e., collecting all the raw data relevant for the project. 2. Convert – ensure that all the data is in a format that can be read in some way by a computer. 3. Classify – Assign each section of the data its own distinct category, for instance in the Traffcom file there are parent nodes, upstream nodes and links and downstream nodes and links. 4. Identify – this involves using selecting the specific data needed, in our case using regex. Once this is completed the dataset can be searched through by the computer. 5. Extract – Now that the pipeline for preparing the data is complete, the specific subsection needed can be extracted using a set of rules and saved into its own more usable format.

A screenshot of a computer

Description automatically generated with medium confidence

Van Beek’s Unstructured Data Preparation Process

Of course, creating a data pipeline is next to irrelevant if we don’t know what we want its output to be. ‘The questions are the important thing, and the tools are the means to reach the answer to these questions’ *(Marazzi, 2016*). When creating the pipeline, it is essential that we know ahead of time what data we need from it, and in what structure it will be stored. Once this is decided and the prerequisite data sources are collected, only then can the pipeline be built to process the data step by step, and if correctly designed this process should be fully automated, assuming of course the data sources are always in the same format. If not and the input changes in an unexpected way this can cause the entire pipeline to collapse. (*Monroe, 2018*)

The most common type of data pipeline used is an ETL Pipeline (Extract, Transform, Load). While not well suited for working with the unstructured data, once that has been converted into a better structured form an ETL pipeline is best suited for carrying out the rest of the data preparation process for the visualisation. As the Council are not reliant on using real time data, the ability of ETL to run multiple batches at set time periods is a massive advantage over a real time pipeline in terms of computer resource requirements (*Tobin, 2020*).

There are a number of software options that can be used for creating data pipelines. A wide variety of free and open-source software are available, including the Python standard library, pandas- a more programmatic version of Microsoft Excel, Apache Airflow which allows for pipelines to be scheduled and monitored and also Postgres, a version of SQL that supports additional unstructured data types such as JSON (*Keboola, 2020*). There are also several paid software options available, with some of the most popular including FME (the Feature Manipulation Engine), which is designed to allow less technical users to create their own data pipelines, Stitch which allows for ETL pipelines to be designed with less overhead than traditional methods and Microsoft SQL Server, which allows developers to create pipelines using their own preferred language and environment (*G2, 2021* and *Safe Software, 2021*).

Both main options have their advantages and disadvantages. Open-source software is of course available for free and can be fully customised to suit the specific needs of the user. It also has a wide array of community support and resources available, all without needing to be locked into a vendor contract. However, this customisation is also its main drawback, as it can be difficult to recreate the custom solution when changing or updating the system. They can often be difficult to scale and require a certain amount of expertise to be adjusted which can lead to quickly mounting costs (*Keboola, 2020*).

Proprietary software options have their own advantages, chief among them being the option of professional technical support available direct from the vendor and a minimal number of bugs. Some options, in particular FME, can allow for non-technical users to design pipelines for their own specific use cases while also allowing more proficient users to build more complex solutions. Of course, the biggest downside of paid solutions is the cost, with software subscriptions creating an ongoing cost that needs to be paid. There is also often a need to pay for higher tiers to gain access to required features or for additional licences to allow the software to be run on more than a single system. (BBC, 2021)

Ultimately following on from the research carried out it was decided to use a multiple software solution for constructing the pipeline. Python was to be used for the bulk of the processing, as it is an extremely powerful yet also comprehensible language and can interface with most systems the council uses including Pandas, various manners of APIs and NoSQL. FME was then to be used for more programmatically difficult sections of the pipeline, as the Council have already licenced the software and currently have a large amount of technical knowhow on its usage. FME is also ideally suited for working with geospatial data, which will be discussed in further detail in the next section (*GISGeography, 2021*).

## 4.4 Visualising Geospatial Data

When it comes to visualising geospatial data, the best option is to use what is known as a GIS or Geographic Information System (*esri, 2021*). As defined by National Geographic, a GIS is a computer system for capturing, storing, checking, and displaying data related to positions on Earth’s surface. This allows people to see, analyse and understand patterns present in the geospatial data more easily (*National Geographic, 2021*).

The Ordnance Survey (2021) recognises and recommends two main GIS packages, Esri ArcGIS and Quantum GIS (QGIS), as well as several other smaller scale packages including Snowflake GO Loader, Cadcorp and Leaflet. While these all have their own strengths and weaknesses, the leading proprietary GIS software is ArcGIS (*Technavio, 2021*) while QGIS is also widely used and is the most predominant open-source GIS software (*GISGeography, 2021*).

ArcGIS is a mature software and has been used for mapping spatial data for a long time. This longevity contributes greatly to how well known the package is, with it being relatively easy to source and hire ArcGIS experts for projects. Esri also provide excellent support for ArcGIS users with a wide variety of training available, which is essential giving the complexity of the software package. However, this complexity is what allows ArcGIS to be able to accomplish exactly what its users need it to, even with the steep learning curve necessary (*GrindGIS, 2019*).

QGIS on the other hand is a much simpler, open-source GIS system that is better suited for small projects with smaller amounts of data. It is especially suited for smaller organisations, due both to its streamlined interface and the fact that it is available completely free. QGIS also makes use of generic file formats, allowing it to interface with ArcGIS specific files, and this coupled with its plethora of plugins and extensions allow it to accomplish as many things as ArcGIS can (*Capterra, 2021*).

Both services come with their own unique limitations which are ultimately the deciding factor when it comes to choosing which software to use. Due to its age ArcGIS can be poorly optimised for certain systems with a large amount of computer processing power needed to use it to its fullest extent, with long load times and save times for larger files. ArcGIS is a rather expensive software package too, with advanced analytical options costing extra money still. QGIS on the other hand, suffers from limitations common to open-source software, notably that while plenty plugins exist to expand QGIS’s functionality, these are not located in a central repository and so can be difficult to track down. QGIS may be easier than ArcGIS, but it is still a complex software to learn, and unlike its rival, QGIS lacks the training resources and technical support of ArcGIS. A potential dealbreaker is the lack of guarantee for any particular QGIS build’s stability and its likelihood of crashing, which is not ideal for a system designed for traffic management (*TrustRadius, 2020*).

In the end, ArcGIS was decided upon for a number of reasons. As the Council have already licenced ArcGIS for their systems this eliminates the expense as a disadvantage while also ensuring there are several trained ArcGIS users already present in the Council. While not mentioned earlier, the latest version of ArcGIS also boasts the ability to host visualisations online, which is incredibly useful for the Data Team’s end goal of releasing a version of the visualisation for public use. It also mitigates the processing power and long save times of the Desktop version by offloading this to an online server. A final deciding factor is the interoperability of ArcGIS with FME, which makes preparing geospatial data significantly easier and allows for any pre-existing FME and ArcGIS solutions already in use by the Council to be integrated into the final visualisation.

# Methodology

## 5.1 Introduction

Initially the data used in this project came from two major sources:

1. The Master text file used by the Traffcom that contains the network data for the entire GCC SCOOT network.
2. The Glasgow Open Data Traffic Application Programming Interface (API) – this is an API that allows a user to query the central GCC database and retrieve information on each SCOOT link’s geospatial position and the traffic flowing through it at 15-minute intervals.

As the project progressed several other data sources were also included:

1. An internally maintained Traffcom databases with the coordinates of each SCOOT node in Northings and Eastings.
2. A further series of Traffcom databases including corrections to location data present in both the API and previous Traffcom dataset.
3. An ArcGIS shapefile containing point data with the geographic location of each SCOOT detector.
4. The raw Cosmos DB behind the aforementioned API which could be directly queried using SQL to retrieve the last set time period of traffic flow data.

## 5.2 Methodology

### 5.2.1 Overview of the Master Text File

To begin with the initial goal was to find a way to convert the raw text file from a document laid out like this:

Text

Description automatically generated with medium confidence

Fig 1. Initial Structure of Network Data

Into a more structured dataset of road connections more like this:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| From\_Node | To\_Node | To\_Link | From | To | hasEntryLink |
| DD1271 | DD1251 | C | DD1271 | DD1251\_C | FALSE |
| DD1351 | DD1251 | D | DD1351 | DD1251\_D | FALSE |
|  | DD1251 | E |  | DD1251\_E | TRUE |
| GA0151 | GA0201 | W | GA0151 | GA0201\_W | FALSE |
| GA0251 | GA0201 | T | GA0251 | GA0201\_T | FALSE |
| GA0451 | GA0201 | S | GA0451 | GA0201\_S | FALSE |
| GA5201 | GA5151 | T | GA5201 | GA5151\_T | FALSE |
| GA5551 | GA5151 | R | GA5551 | GA5151\_R | FALSE |
| GA510A | GA5151 | V | GA510A | GA5151\_V | FALSE |
| GA5171 | GA5151 | P | GA5171 | GA5151\_P | FALSE |

Fig 2. Proposed Final Layout of Network Data

While the initial data structure is laid out in an intuitive manner for a human to read and understand, it is not well structured for a machine to interface with. It also contains a lot of extraneous information that is not needed for this particular project, including many lines of extra descriptive data as well as varying amounts of whitespace between each section of data.

### 5.2.2 Bash Script with Regex

To extract only the relevant data, a script needed to be created that would be able to detect the general pattern of the relevant SCOOT information (i.e., the name of the SCOOT node as well as the upstream and downstream nodes and links it is connected to) without picking up any extraneous data. After carrying out some research into the various methods used to extract text data from files (particularly section 4.2.2), it became clear that a program based on using regular expressions, or regex, would be the way forward.

By making use of the Regex Quick Syntax Reference *(Zsolt Nagy, 2018)* the below regex pattern was constructed to match the necessary data:

'/SCN/,/^\s\*$/p'

The pattern works by first finding the letters ‘SCN’ which precede the junction’s name at the start of each section of data and then it captures all lines after the SCN until it reaches a line that is entirely whitespace, as this denotes the break between the junction connection section of the text file and further data.

The regex pattern by itself is only capable of matching the first occurrence of the pattern it represents and so to capture the data for each individual junction a more powerful text editing software would be needed. As I had some prior experience in working with the UNIX command line, a software program that contains several text-streaming and string manipulation programs, I decided to proceed using one of the pre-existing features. This also had the additional benefit of allowing a script that would require several lines of code in other programming languages to be condensed into a single line that could be run directly from the Windows Command Prompt, as well as massively decreasing the runtime of the script due to UNIX interacting directly with the C programming language *(Loewen, 2021)*.

Ultimately the package chosen was sed (short for stream editor) as it includes an option to capture all patterns that match within a text file and to continue matching until no further patterns can be found. The final shell script, ‘ext-jun-links.sh’ contains the following code:

sed -n -e '/SCN/,/^\s\*$/p' $@ > Junction-Links.txt

The script executes like so:

1. sed calls the sed function from the UNIX shell
2. -n tells the sed program to continue looping until a certain criterion is satisfied
3. -e tells the program to match as many regex patterns in the text as possible
4. $@ allows the user to pass the name of the file to extract junction data from. In the case of this project this would be substituted for NETL1405.txt
5. > Junction-Links.txt – this passes the results of the previous script into a new text file which contains only the relevant data.

A picture containing text, screenshot, receipt

Description automatically generated

Fig 3. Text file after executing Bash Script (Junctions-Links.txt)

### 5.2.3 Cleaning and Splitting Text File with Python

As can be seen above, the bash script is able to remove the majority of the unnecessary data from the text file. However, a number of other steps had yet to be carried out before it could be directly fed into a program to convert it into a table like the one seen in figure 2.

Text

Description automatically generated

Fig 4. RemoveLines function python code

Firstly, a function was defined to remove any unneeded lines from the file by removing lines that contained specific strings. In this case the lines containing the strings Modified, Type and Region contained no other data needed to construct the table of node connections and so they were eliminated by running the function above.

As only the first part of the line identifying the junction was needed, the rest of the line that explained where the junction was located to the human reader could be discarded. A function was written to split the line on a specific delimiter, in this case ‘At’, and then keep the part of the string before that delimiter. If carried out on the first line present in figure 3. the function would return only the ‘SCN DD1251’ part of the string and drop the rest.

Text

Description automatically generated

Fig 5. TrimLine function python code

A final function was then written to finish cleaning the text file, which split each individual junction’s dataset into its own separate file to allow for a more generic program to extract the main junction title from the top of each file and then match it to the other junctions in the file.

Text

Description automatically generated

Fig 6. Split\_junctions function

An overall function to pre-process the text file was created which packages these 4 subroutines into one function was created to allow for the text file processing to be more quickly run whenever the junction file is updated.

Text

Description automatically generated with low confidence

Fig 7. Example of Individual Junction file

### 5.2.4 Extracting Location Data from Traffic API

In comparison to the code needed to extract the necessary data from the text file, the code needed to pull in the link, location and flow data from the API was a much simpler affair. As the GCC had already set up an API to interact with the central traffic database it was simply a matter of sending the correct request link to pull in the needed data.

Text

Description automatically generated

Fig 8. API data pull function

The function in figure 8 works by requesting the data be sent in a csv format and then iterating through the predefined number of pages (currently the dataset contains 20 pages so by setting the page\_num variable to 30 it allows for a substantial amount of location data to be added before the program needs updating.) It retrieves the link name- a node followed by the relevant link suffix- as well as the geographical position of each link in WGS84 Latitude and Longitude.

Text

Description automatically generated

Fig 9. API flow table

### 5.2.5 Converting the text file into a csv table

While the text files have been pared down to only the data needed to create a table of link connections it is still not quite ready to be read directly by a computer. One last cleaner function is used to remove the leading SCN from the junction title and then strip out any extra whitespace to ensure all the strings extracted from the table contain only characters.

Text

Description automatically generated

Fig 10. Final Cleaning function

Now that each of the text files have been completely cleaned, the junction and link information can be extracted using two different regex patterns. These match the layout of both normal junctions (such as GG4812\_R) and links without a preceding junction (such as simply T). After matching one of the two patterns, it is then appended to either a list of incoming connections or outgoing connections. After this is carried out for each junction text file, the resulting list is converted into a Pandas series. This is the initial step needed to convert a list into a more useful data frame and allows for any brackets or other extra characters to be more easily removed.

Text

Description automatically generated

Fig 11. Part 1 of function to extract junction information into Pandas series

Text

Description automatically generated

Fig 12. Part 2 of function to convert Pandas series into csv file

Once in a Pandas series the datapoints are split into two columns, one containing the upstream junction and the other containing the downstream junction to which it connects. These are then each split into two again, with a column denoting the Node and a column denoting the Link for each junction. Finally, to complete the table an additional column is added to show whether a given junction has an entry link or not. Any row in the table that doesn’t contain an originating ‘From’ junction is marked as an entry link while all normal rows with a From and To junction are marked as not being entry links. The resulting table is then output into a csv file laid out like the figure below:

Table

Description automatically generated

Fig 13. csv output from text file extraction

### 5.2.6 Combining Connection and Location Datasets

While initially the plan was to join the resulting junction link table with the API flow table, this would only match the junctions in the ‘To’ column. Because of this another source of data was needed to provide location data for the ‘From’ junctions. Ultimately a table of Traffic signal coordinates was provided by Traffcom, which contained a list of every junction in Glasgow as well as it’s corresponding Eastings and Northings. Since the positional data from the API table was in Latitude and Longitude and the positions for the ‘To’ junctions were also available in Eastings and Northings, it was decided that it would make more sense to proceed using only the location data in the Traffcom dataset. This eliminates the need for coordinate conversion as this is a complex mathematical procedure that can introduce significant errors into the geospatial data.

Before the traffic data could be used however it needed to be prepared. As there was lots of data in addition to the site and coordinates these were dropped from the dataset and additional links from Dunbartonshire Council were added to enable the council boundary junctions to be included in the data. A brief analysis of the coordinates included in the dataset revealed that several of them were missing (had their eastings and northings set as 0,0) or had slightly off values, in some cases having the eastings and northings switched around and in other cases having an incorrect number of decimal places which causes the points to correspond to locations in Wales rather than Glasgow.

Table

Description automatically generated with medium confidence

Fig 14. Subset of Traffcom Location dataset after preparation

After simplifying the dataset and acquiring a table of corrected location data from Traffcom, the corrected data was substituted into the complete dataset in place of the original incorrect data, as seen in Appendix 7.1.

The final stage of the Python pipeline was to merge the location data with the table of link connections. This was achieved by first joining the position data on the From column, to attach the coordinates of the ‘From’ node, and then carrying out a second join on the ‘To’ column, creating a final dataset with each node followed by its corresponding location.

Table

Description automatically generated

Fig 15. Subset of completed location table

After creating the completed location dataset, the bearing nodes were separated into a separate table to allow for their angles to be calculated at a later stage, and the table of complete node-node pairs was exported as a csv for use in the next stage.

Diagram

Description automatically generated

Fig 16. Process Flow diagram of Python Pipeline

### 5.2.7 Generating and Collecting Flow Dataset from the Traffic API

#### 5.2.7.1 Complete Flow Data Pipeline

Now that the location data had been collected and collated, the final step before the data could be properly visualised was to collect the flow data. There were two main methods considered for how to pull in the flow data from the GCC’s servers, both with their own benefits and drawbacks. The first method was to create an FME pipeline that would directly query the Cosmos DB database that contained the flow data and ask it to send all flow data collected between the current datetime and the past 7 days. This was achieved with the SQL query below:



Fig 17. SQL query to pull in all API data from the past 7 days

While this query was able to pull in the complete data from the last 7 days, the sheer volume of data being pulled in caused the query to take an extremely long time to execute. When interfacing directly with the Cosmos DB server on the web, the query would take approximately 2 minutes to pull in the complete dataset, well within the 15-minute range between fresh readings being added to the database. However, when used in FME, due to the query having to be first sent through the Azure Rest API, this caused the fetch procedure to take closer to an hour, which is much too slow to allow the traffic operators to notice any changes in traffic in anywhere close to real time.

Diagram

Description automatically generated

Fig 18. FME SQL Pipeline to pull in complete data from past week

#### 5.2.7.2 Partial Flow Data Pipeline

The second method used was much more lightweight. It worked by querying the GCC’s Open Data Traffic APIs rather than the Cosmos DB directly, pulling in all the real time flow data for the current datetime, and then pulling in the corresponding data from the historical flow API exactly a week before the current datetime. This method was able to pull in the data far faster than the first method, taking only a few seconds to pull in the requested data. However, this came at the cost of less data being available to the user at any one time as only two time periods would be available at any one time, and these would be constantly changing every 15 minutes. This would drastically limit the usefulness of any analysis that could be carried out by Traffcom engineers and also removes the ability for different time periods to be selected, one of the key features requested in the original problem definition.

Diagram

Description automatically generated

Fig 19. FME SQL Pipeline to pull in current data and data from past week

#### 5.2.7.3 Ideal Flow Data Pipeline

While there wasn’t enough time to implement it during the 12-week project timetable, a solution which would combine aspects of both approaches was proposed. Initially the first pipeline would be run in order to pull in the complete set of traffic data to date, or to whatever timespan as Traffcom currently needs. After pulling in the full dataset, a modified version of the second pipeline would then be scheduled to run at 15-minute intervals, which would append the latest datapoints to the overall table. This eliminates both the drawback from the long runtime of the first pipeline, as it only needs now to be run a single time, as well as the lack of data being pulled in by the second pipeline as it is instead merely updating the master dataset with fresh data every 15 minutes.

However, for the purposes of the current project the second pipeline has been implemented to provide a proof of concept for how the final visualisation could work. It will allow for the majority of the desired features to be demonstrated, particularly the direction of traffic flow and a comparison of traffic flows over different time periods (albeit only two).

## 5.2.8 Preparing Data for Visualisation

The method to be used for visualising the final set of data was settled on at an early stage of the project, ArcGIS, in particular ArcMap. This was decided upon for several key reasons, the most important being that it is already currently used within the Council for a number of other similar projects.

As a result of this there are already several people within the council with in-depth knowledge of the entire ArcGIS software suite, which will allow for the final visualisation to be maintained well into the future. It will even allow for the visualisation to be improved as it will be interconnectable with other ArcGIS packages the Council is already making use of, such as their cycling map and bus and taxi lanes map and will also allow for any additional maps the Council creates in the future to be added to the visualisation. This will make the visualisation useful to various decision makers within the Glasgow City Council, in addition to just Traffcom.

To prepare the data already collected and processed for visualisation, a final pipeline was created. Initially both the current flow data as well as the historical flow data from a week prior are loaded into FME. The two tables are both then joined on their site columns, allowing for the current and historical data for each site to be contained in a single table. Once the tables are joined a new column, dist\_flow, is added to the table which takes the difference between the current and historical data flow, allowing for a comparison to be made once the data is loaded into the ArcMap.

Concurrently, the location dataset is loaded into FME, and the eastings and northings for both the from and to sites are converted into vertices so they can be loaded into ArcMap directly as connected lines. Following this, the now converted line data is joined with the combined flow table on the site column, allowing the current, historical and difference in flow to be seen for each junction pair. Finally, the data is converted into an ArcGIS shapefile containing both the location and flow data which is output in an easily portable format that can be used by any ArcGIS package.A picture containing text, sky, map

Description automatically generated

Fig 20. FME Pipeline to join flow and location data into an ArcGIS shapefile

## 5.2.9 Visualisation and Final Amendments

#### 5.2.9.1 Acquiring an Appropriate Base Map

When creating the final ArcMap visualisation, a simple, but essential first step needed to be carried out- selecting a base map. As the data used for the visualisation makes use of the British National Grid coordinate system, the default base maps included in the ArcGIS suite could not be used as they require data to use the WGS84 coordinate system. Due to this, a base map supporting the BNG system was downloaded from the official ArcGIS map depository and loaded into ArcGIS.

Diagram, engineering drawing

Description automatically generated

Fig 21. BNG Base map for Traffic Flow

#### 5.2.9.2 Adding in Raw SCOOT Link and Junction Shapefile

After the base map was loaded in, the point data for the SCOOT sensors placed around Glasgow was loaded in to allow the directional data to be sense checked. This SCOOT location data had already been made public by members of the GCC Data Team and was available for download directly using ArcGIS’ online repository of data sources. While the junctions and links don’t match up directly with the SCOOT sensor positions, and some junctions contain multiple sensors, by comparing the placement of the sensors and the final junctions it allows us to see if the junctions have been placed correctly.

Diagram

Description automatically generated

Fig 22. Map with SCOOT sensor point data added

The initial output from the visualisation pipeline can be seen in the figure below. A series of connected points representing the incoming and outcoming node for each junction can be seen, with each junction joining on to the next junction, giving the user a network map of the connections between each junction. However, by default there is no way to tell the direction of the traffic flow, as the node connections are simply displayed as lines with no arrows or other indicators to show the origin and terminal nodes. In order to combat this lack of directional information, further processing was required within the ArcGIS program itself.

Chart

Description automatically generated with medium confidence

Fig 23. Traffic Flow Data when initially loaded into ArcGIS

#### 5.2.9.3 Displaying the Directional Data

To display the direction of traffic flow, the default shapefile shape (in this case a straight line) for the traffic flow dataset should be selected by double clicking on it. Then the ‘Arrow at End’ symbol should be selected from near the bottom of the symbol menu, and for ease of clarity the width of the symbol was increased from 0.40 up to 2.00. This resulted in an ArcGIS map as seen in figure 25 below.

Graphical user interface, application

Description automatically generated

Fig 24. Adding arrows to Traffic Flow

Graphical user interface, chart

Description automatically generated

Fig 25. Traffic Flow with direction added

#### 5.2.9.4 Adding Colour Scales, Labels and Other Relevant Visual Information

While being able to see the direction of traffic flow is useful in and of itself, it doesn’t allow for any deeper level of analysis to be carried out. ArcGIS does offer additional functionality to allow for more data relating to both the junctions and their traffic flow to be displayed up front without requiring the user to delve deep into menus once they have been set up. By right clicking on the traffic flow data layer and selecting layer properties it brings the user to a menu similar to the one seen below in figure 26. Selecting Symbology followed by selecting quantities in the Show column allows for the user to adjust the colours, value ranges and the labels of the symbols used in the layer according to a user’s self-defined criteria. In the case of the traffic flow data, these criteria are based on the flow difference quantity which is a calculation of the difference in traffic flow across two different time periods.

For Traffcom, comparing traffic flow as a universal value across multiple junctions doesn’t make much sense as a 5-lane junction can support far more cars driving on it than a 1 lane junction yet if both had a raw traffic flow value of say 30 cars, they would show as suffering from the same amounts of congestion. Due to this it makes far more sense to compare the same junction to itself over different time periods, and as a result of this the flow difference is only calculated for a specific junction at different time periods. The resulting colour scheme for this comparison would work as follows:

* If traffic at the present time is more than 20% worse (i.e., the value of present traffic flow is 20% greater than the past traffic flow) than traffic at the past time to which it is being compared, then the arrow should be coloured red.
* If traffic at the present time and traffic at the past time are within +/- 20% of one another then the arrow should be coloured orange to show that the traffic is relatively similar.
* If traffic at the present time is more than 20% better (i.e., the value of present traffic flow is 20% less than the past traffic flow) than traffic at the past time to which it is being compared, then arrow should be coloured green.

.Graphical user interface, text, application

Description automatically generated

Fig 26. Adding arrows to Traffic Flow

After selecting these ranges in the Symbology tab, each arrow is then changed to match the colours specified above. Labels are also added to provide a brief explanation of what each colour means to any unaware users. Finally, by navigating to the Labels tab in the figure above, the site field is selected as a label to allow each junction to be quickly identified at a glance, with the deeper information still remaining just a single right click away.

# 6.0 Conclusion and Future Work

## 6.1 Results to Date

Map

Description automatically generated

Fig 27. Final Traffic Flow

The results of the project have so far been very promising, with nearly all the original requirements desired by the Council being met and several additional features being added. The map currently displays over 1000 individual link connections, overlaid on to the streets of Glasgow. The map view is capable of being moved around the city to the user’s area of interest and can be zoomed in and out to the desired extent needed for the user’s query.

Map

Description automatically generated

Fig 28. View of All Links Displayed on Map

Each link’s traffic flow direction can be ascertained from the directional arrow present on each link, allowing for the user to easily follow traffic flow from junction to junction across the desired subsection of the network needed. The junctions have also been colour coded to reflect the flow differences that were discussed in section 5.2.9.4, allowing the user to compare the relative difference in traffic flow across the two time periods at a glance, with toggleable labels being added for each of the junctions to allow the user to tell what the name of a given junction is without having to access the menu, whilst simultaneously allowing them to turn the label off to prevent certain areas of the map from becoming too cluttered.

Of course, the most useful functionality added is the ability for the user to select a junction and then right clicking on it and selecting identify to view further information on the junction’s properties. The user is able to see the IDs of the two nodes that make up the junction and their spatial coordinates, as well as being able to see the junction’s current flow value, the equivalent flow value from a past time period and ultimately the flow difference between the two.

Map

Description automatically generated

Fig 29. Sample of Viewable Properties for each Junction in Visualisation

A final useful feature is the ability to manually search for a desired junction, even if the user is unsure of its position on the map. This can be achieved by navigating to the Selection tab in the ArcGIS taskbar and then using the Select by Attributes option. This brings up a popup box that allows the user to query the flow dataset in a variety of ways, such as selecting a list of unique junctions in the dataset, searching for a specific junction by its name or highlighting all junctions that have flow values or positional coordinates within a certain range to limit the user’s analysis to a specific area or subset of the dataset.

Graphical user interface, text, application, chat or text message

Description automatically generated

Graphical user interface, map

Description automatically generated  
Fig 30. Using the Selection Search to Locate a Junction

## 6.2 Application of Results

Now that a flow visualisation including both the value of traffic flow and the corresponding direction, as well as the underlying data containing all the information needed to create it, has been realised it allows for various different types of users to benefit from its use. These can be broadly split into Governmental Bodies who directly gather and manage the data, private entities who can make use of the data for their own needs but who don’t have direct access to the traffic control system, and Glasgow City residents who will be able to use both the data and visualisation for their own personal use. In the next few sections, some more in-depth use cases for different end users will be explored.

### 6.2.1 Traffic Engineers

Traffic Engineers will be able to use the ArcGIS visualisation to extend the functionality of the pre-existing traffic manager in a number of ways. While the traffic flow at each individual junction can be seen in the original SCOOT Urban Traffic Control (UTC) system, there is no further detail available for the junctions and it is currently impossible to follow the flow of traffic across more than a single junction without significant trial and error. With the new system, not only can the engineers easily follow the traffic flowing from junction to junction across the entire city network, they’ll also be able to quickly search for any specific junctions they need using the inbuilt search function- a function which is not present in any form in the original system.

More in-depth analysis will be able to be carried out by the traffic engineers in the actual visualisation, without needing to go back to the raw dataset. Traffic flow across different time periods can be easily compared for each junction, which will allow the engineers to discover trends across junctions. This could be noting that a particular junction tends to get busy at the same time on a Tuesday each week, to more overarching trends such as being able to track the increase in traffic flow in certain areas of the city depending on the time of the year, such as city centre streets drastically increasing in traffic flow around the Christmas period. It could also help the traffic engineers to make better decisions by using the already existing data. A prime example would be deciding the best time to carry out roadworks or other services on a given junction. By monitoring the traffic flow through the junction at various times throughout the week, the engineers would be able to find the period with the lightest traffic flow and timetable the road closures necessary for work to be carried out around this, reducing the impact closing the road will have on the surrounding junctions’ traffic flow. Further expansion of this capability could ultimately lead to a model for predicting future traffic conditions based on past data.

A final key feature that doesn’t directly affect the traffic engineer’s day-to-day responsibilities but is no less useful is the ability to quickly add new junctions to the visualisation. With the current UTC system, for a new junction to be added all the data containing its connections, location, and other related features needs to be manually entered both into the master text file, and into the actual system itself. This is a classic case of data redundancy as it requires the same data to be entered several times and can lead to all sorts of issues down the line if the two sets of data don’t exactly match. Fortunately, with the new system, the data for a new junction just simply needs to be added to the master text file once, and it can then be passed forwards into the visualisation automatically, simply by rerunning the Python/FME pipeline to process the new data. This dramatically cuts down the time needed to add a new junction and removes the risk of any data inconsistencies, allowing the traffic engineers more time to focus on their actual duties.

### 6.2.2 City Stakeholders and Decision Makers

There are multiple city stakeholders and decision makers who would both benefit from and potentially contribute to the traffic flow visualisation. These include but are not limited to:

1. Hauling Companies – Drivers and Fleet managers can better plan their routes using the traffic flow data to avoid high traffic areas. This helps them to optimise their routes, saving both time and fuel, whilst helping to reduce the emissions produced by the vehicles. This is of particular importance to Glasgow as they are hosting the upcoming COP26 climate change conference, and this will help them to meet their targets in time.
2. Vehicle for hire companies – Companies that carry people from one place to another, such as Taxi firms or Uber, will benefit from free and enhanced access to traffic flow data. This will allow them to better optimise the algorithms they use for route planning, which can enable them to transport higher volumes of passengers and lower costs. This would likely incentivise more members of the public to use these companies instead of their own vehicles, helping to reduce the number of vehicles on the road and minimise congestion.
3. Retail companies, along with bars and restaurants – Companies like these which rely on a steady flow of people passing through an area to do business, would be able to make use of the traffic flow data to locate areas of high and low traffic. As a result of this, decision makers within the companies would be able to better choose locations for their stores to maximise the number of people passing through that could potentially shop there, ultimately leading to an increase in business. At a higher level still, it would help both public and private decision makers to identify potential areas for retail parks and industrial estates.

### 6.2.3 Glasgow City Residents

The residents of Glasgow themselves will also be able to benefit both directly and indirectly from the release of the visualisation and it’s supporting data. Less tech literate citizens will now be able to view the traffic around the city without needing the coding abilities necessary to work with the APIs currently released by the Council. Citizens will also be able to keep themselves informed of traffic policy to a much higher degree due to the release of this data, which will further allow them to form their own opinions on traffic policy and other sectors of the government. It will also enable more tech literate citizens to access the current existing system and augment it in innovative ways, which could lead to greater collaboration between the public and private sectors and lead to an explosion in new products and ideas.

As well as being directly able to make use of the visualisation and data, citizens will also benefit from this in other indirect ways. For instance, due to traffic around the city being easier to manage for traffic engineers, this will hopefully lead to a knock-on effect of reduced congestion, which will greatly improve the quality of citizens’ lives due to having to spend less time in traffic and more time on personal or professional growth. Furthermore, as traffic engineers will be better able to predict where and when high congestion will occur, additional public transport can be brought in to help offset this increase. If more citizens make use of public transport, particularly trains and the subway, although buses will still help to a lesser degree, this will help to reduce the congestion at these peak times even more, which can then allow Glasgow Council to better enact its future plans to increase pedestrianisation and green areas around Glasgow.

## 6.3 Limitations

While the project was successful in allowing the direction and intensity of traffic flow to be accurately displayed on a map of Glasgow, it still suffers from a number of limitations. Currently the system only contains data from two particular time periods, which have to be hardcoded into the FME pipeline for fetching flow data. This unfortunately limits the current usefulness of the visualisation as comparing different time periods requires both a tech savvy user and a not insignificant amount of coding. This needs to be expanded upon to include a wider time period of flow data to compare, without slowing down the speed of importing data to a greater than 15-minute wait (as this is the amount of time between live data updates).

The visualisation itself suffers from a few imperfections, mainly that not all the links perfectly match their corresponding road. While the majority of the junctions match up cleanly, more complicated road systems or even just lanes that have a large degree of curvature don’t match up. Additionally, in denser areas such as the city centre that have multiple junctions on the one road, the junctions can end up being overlaid on top of one another, making it difficult to ascertain which junction is flowing in what direction. The only fix for this requires a user to manually adjust the road links to make them better match their roads or to slightly offset one another to improve clarity.

There are also some deficiencies in the data behind the visualisation. As some of the junctions are missing either positional or flow data, this means that when a join is carried out during the visualisation pipeline, any junctions without all the necessary data don’t get matched up and are therefore not displayed on the visualisation. The pipeline itself is also not completely future-proofed as while it currently has a hard coded data pull limit of roughly twice the size of the current Cosmos DB dataset, this will eventually require someone with sufficient understanding of Python to manually increase it once the size of the dataset has surpassed this threshold. Further to this, the save locations for the intermediate files used when processing the text and location data are manually specified and will need to be adjusted when the project is handed over to the Council, either to allow each individual user to use the necessary files, or more likely to allow it to run on the Council’s server set up.

A final potential limitation involves the coordinate system that the visualisation is built upon- the Ordnance Survey National Grid reference system. This is a geographic grid referencing system that measures coordinates from an origin point off the southwest coast of Great Britain, in eastings and northings, both in metres. While this allows for a more accurate ability to locate places in Britain, it is being increasingly phased out by the majority of geographic systems, in particular the EU and Google, and nowadays the main system in use is the Universal Transverse Mercator coordinate system, which makes use of latitude and longitude measured from the equator in metres, rather than from an arbitrary point near the British coast. As a result of this, and the fact that both systems use slightly different projections of the globe, it is difficult to use the two coordinate systems interchangeably, due to complex mathematical transformations necessary for conversions, and this severely limits what datasets could be added to the visualisation (Ordnance Survey 2020). Both solutions are costly and time consuming, either converting all the Council’s datasets to use latitude and longitude or converting all other future datasets that may be used into eastings and northings.

## 6.4 Suggested Further Work

### 6.4.1 Data Gaps and Data Augmentation

At the time of writing, there are several missing data points for both the position and the traffic flow for many of the SCOOT links. The position for the missing data points could be found and added to the master database to increase the number of links the ArcGIS visualisation is able to display, while further investigation could be taken into the reason for missing periods of flow data in the dataset.

The project currently also excludes a number of more difficult to model links, including entry, exit and filter links. While it is theoretically possible to calculate the positioning of entry links by taking a bearing halfway between the two links it is connected to, this is programmatically difficult and so has not been included in the current incarnation of the project. Even more difficult still are exit and filter links, as there is no way to easily calculate their positioning from the data contained in the master text file alone, and so additional data would be required to supplement the original data and allow for their inclusion.

The utility of the visualisation could also be improved by adding additional data or transforming the currently used data. Glasgow City Council already collects a wide variety of geospatial data from across the city including pedestrian activity, bike hire and cycle traffic as well as train and subway usage. These could be layered on top of the existing map to provide further insight into transport patterns across Glasgow. A heatmap or some sort of visual symbology to differentiate the volume of traffic flow across the city could be used to highlight areas of key interest- these could be averaged out over a variety of time periods to allow traffic engineers and urban planners to see trends more clearly than from the live map alone. Time periods of particular interest as suggested by the Council include:

1. Weekdays AM peak (from 7 to 10 am)
2. Interpeak (Between 10am and 3pm)
3. Weekdays PM peak (from 3 to 7pm)
4. The same time periods as above but for the weekend

Breaking the data into these subdivisions would allow users to better compare time periods to note any striking differences between them. For instance, they could compare an AM peak weekday from 2020 and compare it to an AM peak weekday from 2019 to ascertain what effect covid may have had on traffic flow around Glasgow. Similarly, neutral time periods from months with fewer public/school holidays or reasons for non-standard traffic patterns (e.g., a time period from November) could be used as a benchmark to compare the rest of the data against. Aggregating the data like this could also help to improve the speed bottleneck when loading in the traffic flow data, as if the data is pre-aggregated it requires less processing power, and therefore time to load into the visualisation.

### 6.4.2 App Improvements and Data Sharing

As of the completion of the project, the product is currently a collection of pipelines, datasets and a final visualisation in ArcGIS which currently need to be stored on the same computer system to function. A way to significantly improve the usefulness of the software would be for Glasgow City Council to host the completed application on their internal server, to allow users from across the council to access the app and its underlying data without needing to acquire all the relevant files and adjust the file paths. This has already been done for several other Council visualisations, including visualisations of parking permit areas and bus lanes across the city. As well as this, the aggregations mentioned in 6.4.1 could also be hosted online to allow the data pipeline supplying the visualisation to pull in larger volumes of data in less time, thanks to the increased processing power offered by being hosted server side.

As the Council is eager to promote public data access, an alternate version of the ArcGIS map with all the confidential and personal information removed from it, could be hosted on the Council’s web portal to allow members of the public to access the traffic flow visualisation. The pipeline to produce it and the raw data behind the visualisation could also be added as an API to the Council’s traffic API system. This would allow people from outside of the council to analyse the data for their own needs and to produce their own visualisations, whilst also allowing entrepreneurs and private entities such as traffic data companies access to good quality data to produce new apps of their own. This ties neatly into the Council’s Digital Glasgow Strategy and will particularly help to boost the digital economy following the pandemic and to lead to a more sustainable and low carbon city.

The ability of the app to inform traffic management decisions could also be improved by including data from the council areas surrounding Glasgow City Council. As it stands there is a limited amount of flow and positional data available for links near the border of the Council’s, with the only data currently available supplied by the neighbouring East Dunbartonshire Council. As a result of this it is currently impossible to track the volume of traffic entering and existing the Glasgow City Council’s boundaries, leading to a deficiency in the ability to reliably monitor and predict traffic in the Greater Glasgow Metropolitan Area. There are currently 7 councils surrounding Glasgow city proper, these are:

1. East Dunbartonshire
2. West Dunbartonshire
3. North Lanarkshire
4. South Lanarkshire
5. East Renfrewshire
6. Renfrewshire
7. Inverclyde

By entering into a data sharing agreement with these councils, Glasgow City Council will be able to better manage the city and meet its proposed future goals, not just for traffic management, but in their overall Digital Glasgow Strategy much more effectively.

# 7.0 Appendix

def Prepare\_Positional\_Data():

    # Removes all columns bar Site ID and the Eastings and Northings

    Positions = pd.read\_csv("Traffic Signals Co-ordinates 2021.csv",float\_precision='round\_trip')

    Positions = Positions[["FEID", "EASTINGS", "NORTHINGS"]]

    # Loads in Dunbartonshire links

    DD\_Positions = pd.read\_csv("Replacement for Missing Data/East Dunbartonshire (DD) Links.csv",float\_precision='round\_trip')

    # Adds DD Link values

    Positions = pd.concat([Positions, DD\_Positions])

    # Loads in incorrect or missing links not in Dunbartonshire

    Position\_replacement = pd.read\_csv("Replacement for Missing Data/missing data eastings and northings corrected bd v1.csv")

    # Replaces incorrect values with corrected values

    Positions.set\_index("FEID",inplace=True)

    Position\_replacement.set\_index("FEID",inplace=True)

    Positions.loc[Position\_replacement.index, "EASTINGS"] = Position\_replacement["Corrected Eastings"]

    Positions.loc[Position\_replacement.index, "NORTHINGS"] = Position\_replacement["Corrected Northings"]

    Positions.reset\_index(inplace=True)

    Positions.to\_csv("Traffic Signals Co-ordinates 2021.csv", index=False)

Prepare\_Positional\_Data()

From\_Positions = pd.read\_csv("Traffic Signals Co-ordinates 2021.csv",float\_precision='round\_trip')

From\_Positions.rename(columns={'FEID': 'From'}, inplace=True)

To\_Positions = pd.read\_csv("Traffic Signals Co-ordinates 2021.csv",float\_precision='round\_trip')

To\_Positions.rename(columns={'FEID': 'To\_Node'}, inplace=True)

# Merges Junctions with Positions on From

Junctions = Junctions.merge(From\_Positions, left\_index=False, right\_index=False, how='left', on=['From'])

Junctions.rename(columns={'EASTINGS': 'FROM\_EASTINGS', 'NORTHINGS': 'FROM\_NORTHINGS'}, inplace=True)

# Merges Junctions with Positions on To\_Node

Junctions = Junctions.merge(To\_Positions, left\_index=False, right\_index=False, how='left', on=['To\_Node'])

Junctions.rename(columns={'EASTINGS': 'TO\_EASTINGS', 'NORTHINGS': 'TO\_NORTHINGS'}, inplace=True)

# Drops unneeded index column

Junctions.drop(['Unnamed: 0'], axis=1, inplace=True)

Junctions.drop\_duplicates(keep=False, inplace=True)

# Flags all links that are bearings and removes to separate table

bearings = Junctions.loc[Junctions['hasEntryLink'] == True]

Junctions.drop(Junctions.loc[Junctions['hasEntryLink']==True].index, inplace=True)

bearings.to\_csv("Bearing\_Links.csv")

# Saves complete table to csv file

Junctions.to\_csv("BNG Links without Bearings.csv", index=False)

7.1. Function to prepare positional dataset for joining followed by joining script

# Finds the savepath for each text file, removes the unnecessary preceding SCN from the junction name, then strips any excess whitespace

def read\_main\_junction(filename, savepath):

    completeName = os.path.join(savepath, filename)

    with open(completeName) as f:

        main\_junc = f.readline()

        main\_junc = main\_junc.replace('SCN','')

        main\_junc = main\_junc.strip()

        return(main\_junc)

# Creates lists of Junctions and corresponding links

def map\_junctions\_to\_series\_to\_dataframe(files\_in\_directory, upstream\_pattern1, upstream\_pattern2):

    # Assigns empty arrays of links and junctions

    links = []

    junctions = []

    # Iterates through each file in junction directory

    for i in files\_in\_directory:

        completeName = os.path.join(JunctionSavePath, i)

        with open(completeName) as f:

            for line in f:

                if re.search(upstream\_pattern1, line):

                    links.append(re.findall(upstream\_pattern1, line))

                    junctions.append(read\_main\_junction(i, JunctionSavePath))

                if re.search(upstream\_pattern2, line):

                    links.append(re.findall(upstream\_pattern2, line))

                    junctions.append(read\_main\_junction(i, JunctionSavePath))

                links = list(filter(None, links))

                links = list(map(str, links))

                JunctionSeries = pd.Series(links, dtype='str')

                JunctionSeries = JunctionSeries.str.strip("[()]")

    # Process series and converts into a table of Node to Node connections

    df = JunctionSeries.str.split(', ', expand=True) # Splits Series on ,

    # Drops columns with non relevant info

    cols = [2,3]

    df.drop(df.columns[cols],axis=1,inplace=True)

    # Removes quotes from strings

    for j, col in enumerate(df.columns):

        df.iloc[:, j] = df.iloc[:, j].str.replace("'", '')

    # Names Columns and adds To\_Node from junctions list to column

    df.columns = ['From\_Node', 'To\_Link']

    df.insert(1, 'To\_Node', junctions)

    df['From'] = df['From\_Node']

    df['To'] = df['To\_Node'] + "\_" + df["To\_Link"]

    # Strips all strings of whitespace

    df['From\_Node'] = df['From\_Node'].str.strip()

    df['To\_Link'] = df['To\_Link'].str.strip()

    df['From'] = df['From'].str.strip()

    df['To'] = df['To'].str.strip()

    df['hasEntryLink'] = np.where(df['From']== '', True, False) # Marks blank From columns as Entry links for future processing

    df.to\_csv('Junction-Links.csv') # Saves output to CSV

# Combines all previous steps including textfile processing up to creating the dataframe

def create\_junction\_dataframe():

    preprocess\_text\_file()

    # Using regex to extract upstream node pairs

    upstream\_pattern1 = re.compile(r"(\w{2}\d{3}\w\*)(?:\s{6})(\w)") # 1st group Node, 2nd group Link

    upstream\_pattern2= re.compile(r"(\s{20})(\w)") # 1st group space, 2nd group Link

    # gets list of files in savepath

    arr = os.listdir(JunctionSavePath)

    map\_junctions\_to\_series\_to\_dataframe(arr, upstream\_pattern1, upstream\_pattern2)

create\_junction\_dataframe()

# Loads in Junction Dataframe and Positonal Dataframe

Junctions = pd.read\_csv('Junction-Links.csv',float\_precision='round\_trip')

7.2: Code to Extract Junction connections from formatted text file and convert to csv

# Import Location data from API

import http.client, urllib.request, urllib.parse, urllib.error, base64, pandas as pd

from io import BytesIO

import time

import os, os.path, subprocess, re, numpy as np, math

def get\_API\_positions(page\_num):

    headers = {

    # Request headers

    'Ocp-Apim-Subscription-Key': '62f8d7952d654af6b59958605b79a001',

    }

    params = urllib.parse.urlencode({

    })

    for i in range(0,page\_num):

        try:

            conn = http.client.HTTPSConnection('gcc.azure-api.net')

            conn.request("GET", "/traffic/v1/movement/now?page={}&format=csv&%s".format(i) % params, "{body}", headers)

            response = conn.getresponse()

            data = response.read()

            df1 = pd.read\_csv(BytesIO(data))

            if i == 0:

                df = df1

            else:

                df = df.append(df1)

            conn.close()

        except Exception as e:

            print("[Errno {0}] {1}".format(e.errno, e.strerror))

    # Saves data collected to dataframe with the following columns

    df.columns = ['latitude', 'longitude', 'flow', 'concentration', 'site', 'lastupdateutc', 'lastupdate', 'timestamp']

    df.to\_csv('link\_positions.csv')

def load\_and\_correct\_junction\_positions():

    EN = pd.read\_csv("Traffic Signals Co-ordinates 2021.csv")

    EN\_corrections = pd.read\_csv("Replacement for Missing Data/missing data eastings and northings corrected bd v1.csv")

get\_API\_positions(30)

7.3: Code to fetch location data from API

# Sets a savepath to extract each junction's data too

JunctionSavePath = 'E:/elija/OneDrive/OneDrive - University of Strathclyde/Dissertation/1. Extract Junction and Link Data/Junction Text Files/Individual-Junctions/'

# Removes irrelevant lines from the Junction-Links file

def RemoveLines(input\_file, output\_file, strings= []):

    with open(str(input\_file) + ".txt", "r") as file\_input:

        with open(str(output\_file) + ".txt", "w") as output:

            for line in file\_input:

                if any (s in line.strip("\n") for s in strings) == False:

                    output.write(line)

# Cuts out unneeded section of line, by splitting it on a set delimiter

def TrimLine(input\_file, output\_file, delimiter, strings= []):

        with open(str(input\_file) + ".txt", "r") as file\_input:

            with open(str(output\_file) + ".txt", "w") as output:

                for line in file\_input:

                    if any (s in line.strip("\n") for s in strings) == True:

                        split\_line = line.split(str(delimiter),1)

                        output.write(split\_line[0] + "\n")

                    elif any (s in line.strip("\n") for s in strings) == False:

                        output.write(line)

# Separates each Junction and it's related information into it's own text file for easier computer and human manipulation

def split\_junctions(input\_file):

    save\_path = JunctionSavePath

    with open(input\_file, 'r') as file:

        i = 0

        output = None

        for line in file:

            if not line.strip():

                if output:

                    output.close()

                output = None

            else:

                if output is None:

                    i += 1

                    print(f'Creating file "{i}.txt"')

                    completeName = os.path.join(save\_path, str(i)+".txt")

                    output = open(completeName,'w')

                output.write(line)

        if output:

            output.close()

    print('-fini-')

# Combines all previous steps that involve preparing the textfile

def preprocess\_text\_file():

    # Runs shell script which strips out irrelevant data, to leave only lines between junction name and it's links

    subprocess.run(["bash", "./ext-jun-links.sh","NETL1405.TXT"], shell=True)

    # Removes lines with the strings Modified, Type and Regions

    strings = ("Modified", "Type", "Region")

    RemoveLines("Junction-Links","Removed-Junction-Links", strings)

    # Splits lines on At and keeps the first half with the parent junction name

    TrimLine("Removed-Junction-Links","Trimmed-Junction-Links", "At", strings=["At"])

    # Splits each junction into it's own text file for future processing

    split\_junctions('Trimmed-Junction-Links.txt')

7.4: Code to Pre-process Text File

# Import Location data from API

import http.client, urllib.request, urllib.parse, urllib.error, base64, pandas as pd

from io import BytesIO

import time

import os, os.path, subprocess, re, numpy as np, math

def get\_API\_positions(page\_num):

    headers = {

    # Request headers

    'Ocp-Apim-Subscription-Key': '62f8d7952d654af6b59958605b79a001',

    }

    params = urllib.parse.urlencode({

    })

    for i in range(0,page\_num):

        try:

            conn = http.client.HTTPSConnection('gcc.azure-api.net')

            conn.request("GET", "/traffic/v1/movement/now?page={}&format=csv&%s".format(i) % params, "{body}", headers)

            response = conn.getresponse()

            data = response.read()

            df1 = pd.read\_csv(BytesIO(data))

            if i == 0:

                df = df1

            else:

                df = df.append(df1)

            conn.close()

        except Exception as e:

            print("[Errno {0}] {1}".format(e.errno, e.strerror))

    # Saves data collected to dataframe with the following columns

    df.columns = ['latitude', 'longitude', 'flow', 'concentration', 'site', 'lastupdateutc', 'lastupdate', 'timestamp']

    df.to\_csv('link\_positions.csv')

def load\_and\_correct\_junction\_positions():

    EN = pd.read\_csv("Traffic Signals Co-ordinates 2021.csv")

    EN\_corrections = pd.read\_csv("Replacement for Missing Data/missing data eastings and northings corrected bd v1.csv")

get\_API\_positions(30)

# Sets a savepath to extract each junction's data too

JunctionSavePath = 'E:/elija/OneDrive/OneDrive - University of Strathclyde/Dissertation/1. Extract Junction and Link Data/Junction Text Files/Individual-Junctions/'

# Removes irrelevant lines from the Junction-Links file

def RemoveLines(input\_file, output\_file, strings= []):

    with open(str(input\_file) + ".txt", "r") as file\_input:

        with open(str(output\_file) + ".txt", "w") as output:

            for line in file\_input:

                if any (s in line.strip("\n") for s in strings) == False:

                    output.write(line)

# Cuts out unneeded section of line, by splitting it on a set delimiter

def TrimLine(input\_file, output\_file, delimiter, strings= []):

        with open(str(input\_file) + ".txt", "r") as file\_input:

            with open(str(output\_file) + ".txt", "w") as output:

                for line in file\_input:

                    if any (s in line.strip("\n") for s in strings) == True:

                        split\_line = line.split(str(delimiter),1)

                        output.write(split\_line[0] + "\n")

                    elif any (s in line.strip("\n") for s in strings) == False:

                        output.write(line)

# Separates each Junction and it's related information into it's own text file for easier computer and human manipulation

def split\_junctions(input\_file):

    save\_path = JunctionSavePath

    with open(input\_file, 'r') as file:

        i = 0

        output = None

        for line in file:

            if not line.strip():

                if output:

                    output.close()

                output = None

            else:

                if output is None:

                    i += 1

                    print(f'Creating file "{i}.txt"')

                    completeName = os.path.join(save\_path, str(i)+".txt")

                    output = open(completeName,'w')

                output.write(line)

        if output:

            output.close()

    print('-fini-')

# Combines all previous steps that involve preparing the textfile

def preprocess\_text\_file():

    # Runs shell script which strips out irrelevant data, to leave only lines between junction name and it's links

    subprocess.run(["bash", "./ext-jun-links.sh","NETL1405.TXT"], shell=True)

    # Removes lines with the strings Modified, Type and Regions

    strings = ("Modified", "Type", "Region")

    RemoveLines("Junction-Links","Removed-Junction-Links", strings)

    # Splits lines on At and keeps the first half with the parent junction name

    TrimLine("Removed-Junction-Links","Trimmed-Junction-Links", "At", strings=["At"])

    # Splits each junction into it's own text file for future processing

    split\_junctions('Trimmed-Junction-Links.txt')

# Finds the savepath for each text file, removes the unnecessary preceding SCN from the junction name, then strips any excess whitespace

def read\_main\_junction(filename, savepath):

    completeName = os.path.join(savepath, filename)

    with open(completeName) as f:

        main\_junc = f.readline()

        main\_junc = main\_junc.replace('SCN','')

        main\_junc = main\_junc.strip()

        return(main\_junc)

# Creates lists of Junctions and corresponding links

def map\_junctions\_to\_series\_to\_dataframe(files\_in\_directory, upstream\_pattern1, upstream\_pattern2):

    # Assigns empty arrays of links and junctions

    links = []

    junctions = []

    # Itereates through each file in junction directory

    for i in files\_in\_directory:

        completeName = os.path.join(JunctionSavePath, i)

        with open(completeName) as f:

            for line in f:

                if re.search(upstream\_pattern1, line):

                    links.append(re.findall(upstream\_pattern1, line))

                    junctions.append(read\_main\_junction(i, JunctionSavePath))

                if re.search(upstream\_pattern2, line):

                    links.append(re.findall(upstream\_pattern2, line))

                    junctions.append(read\_main\_junction(i, JunctionSavePath))

                links = list(filter(None, links))

                links = list(map(str, links))

                JunctionSeries = pd.Series(links, dtype='str')

                JunctionSeries = JunctionSeries.str.strip("[()]")

    # Process series and converts into a table of Node to Node connections

    df = JunctionSeries.str.split(', ', expand=True) # Splits Series on ,

    # Drops columns with non relevant info

    cols = [2,3]

    df.drop(df.columns[cols],axis=1,inplace=True)

    # Removes quotes from strings

    for j, col in enumerate(df.columns):

        df.iloc[:, j] = df.iloc[:, j].str.replace("'", '')

    # Names Columns and adds To\_Node from junctions list to column

    df.columns = ['From\_Node', 'To\_Link']

    df.insert(1, 'To\_Node', junctions)

    df['From'] = df['From\_Node']

    df['To'] = df['To\_Node'] + "\_" + df["To\_Link"]

    # Strips all strings of whitespace

    df['From\_Node'] = df['From\_Node'].str.strip()

    df['To\_Link'] = df['To\_Link'].str.strip()

    df['From'] = df['From'].str.strip()

    df['To'] = df['To'].str.strip()

    df['hasEntryLink'] = np.where(df['From']== '', True, False) # Marks blank From columns as Entry links for future processing

    df.to\_csv('Junction-Links.csv') # Saves output to CSV

# Combines all previous steps including textfile processing up to creating the dataframe

def create\_junction\_dataframe():

    preprocess\_text\_file()

    # Using regex to extract upstream node pairs

    upstream\_pattern1 = re.compile(r"(\w{2}\d{3}\w\*)(?:\s{6})(\w)") # 1st group Node, 2nd group Link

    upstream\_pattern2= re.compile(r"(\s{20})(\w)") # 1st group space, 2nd group Link

    # gets list of files in savepath

    arr = os.listdir(JunctionSavePath)

    map\_junctions\_to\_series\_to\_dataframe(arr, upstream\_pattern1, upstream\_pattern2)

create\_junction\_dataframe()

# Loads in Junction Dataframe and Positonal Dataframe

Junctions = pd.read\_csv('Junction-Links.csv',float\_precision='round\_trip')

def Prepare\_Positional\_Data():

    # Removes all columns bar Site ID and the Eastings and Northings

    Positions = pd.read\_csv("Traffic Signals Co-ordinates 2021.csv",float\_precision='round\_trip')

    Positions = Positions[["FEID", "EASTINGS", "NORTHINGS"]]

    # Loads in Dunbartonshire links

    DD\_Positions = pd.read\_csv("Replacement for Missing Data/East Dunbartonshire (DD) Links.csv",float\_precision='round\_trip')

    # Adds DD Link values

    Positions = pd.concat([Positions, DD\_Positions])

    # Loads in incorrect or missing links not in Dunbartonshire

    Position\_replacement = pd.read\_csv("Replacement for Missing Data/missing data eastings and northings corrected bd v1.csv")

    # Replaces incorrect values with corrected values

    Positions.set\_index("FEID",inplace=True)

    Position\_replacement.set\_index("FEID",inplace=True)

    Positions.loc[Position\_replacement.index, "EASTINGS"] = Position\_replacement["Corrected Eastings"]

    Positions.loc[Position\_replacement.index, "NORTHINGS"] = Position\_replacement["Corrected Northings"]

    Positions.reset\_index(inplace=True)

    Positions.to\_csv("Traffic Signals Co-ordinates 2021.csv", index=False)

Prepare\_Positional\_Data()

From\_Positions = pd.read\_csv("Traffic Signals Co-ordinates 2021.csv",float\_precision='round\_trip')

From\_Positions.rename(columns={'FEID': 'From'}, inplace=True)

To\_Positions = pd.read\_csv("Traffic Signals Co-ordinates 2021.csv",float\_precision='round\_trip')

To\_Positions.rename(columns={'FEID': 'To\_Node'}, inplace=True)

# Merges Junctions with Positions on From

Junctions = Junctions.merge(From\_Positions, left\_index=False, right\_index=False, how='left', on=['From'])

Junctions.rename(columns={'EASTINGS': 'FROM\_EASTINGS', 'NORTHINGS': 'FROM\_NORTHINGS'}, inplace=True)

# Merges Junctions with Positions on To\_Node

Junctions = Junctions.merge(To\_Positions, left\_index=False, right\_index=False, how='left', on=['To\_Node'])

Junctions.rename(columns={'EASTINGS': 'TO\_EASTINGS', 'NORTHINGS': 'TO\_NORTHINGS'}, inplace=True)

# Drops unneeded index column

Junctions.drop(['Unnamed: 0'], axis=1, inplace=True)

Junctions.drop\_duplicates(keep=False, inplace=True)

# Flags all links that are bearings and removes to separate table

bearings = Junctions.loc[Junctions['hasEntryLink'] == True]

Junctions.drop(Junctions.loc[Junctions['hasEntryLink']==True].index, inplace=True)

bearings.to\_csv("Bearing\_Links.csv")

# Saves complete table to csv file

Junctions.to\_csv("BNG Links without Bearings.csv", index=False)

7.5: Complete Python Pipeline

Shape

Description automatically generated with medium confidence

7.6 Diagram of Regex Process



7.7: Missing Data from API

Table

Description automatically generated

7.8: Missing Data from Traffcom Dataset

<https://github.com/Wizzzzzzard/Glasgow-City-Council-Project>

7.8: GitHub of Complete Project

# 8.0 References

* Aho, Alfred V.; Ullman, Jeffrey D. (1992) ' Chapter 10. Patterns, Automata, and Regular Expressions', in (ed.) *Foundations of Computer Science.*Stanford: W. H. Freeman., pp. 529-590.
* Barceló, J., Kuwahara, M. and Miska, M. (2009) 'Traffic Data Collection and Its Standardization', in Barceló, J. and Kuwahara, M. (ed.) *Traffic Data Collection and its Standardization.*New York: Springer, pp. 1-10.
* BBC (2021) *Open source and proprietary software,*Available at: *https://www.bbc.co.uk/bitesize/guides/zhx26yc/revision/11* (Accessed: May 2021).
* Bresnahan, C. and Blum, R. (2015) 'Chapter 19 Introducing sed and gawk', in Bresnahan, C. and Blum, R. (ed.) *Linux Command Line and Shell Scripting Bible.*USA: Wiley, pp. 505-533.
* Bresnahan, C. and Blum, R. (2015) 'Chapter 20 Regular Expressions', in Bresnahan, C. and Blum, R. (ed.) *Linux Command Line and Shell Scripting Bible.*USA: Wiley, pp. 534-560.
* Capterra (2021) *QGIS Reviews,*Available at: *https://www.capterra.com/p/185949/QGIS/reviews/* (Accessed: June 2021).
* Carbonnell, J. (2019) *SMART-CITY: Stakeholders roles and needs,*Available at: *https://juliencarbonnell.medium.com/smart-city-stakeholders-roles-and-needs-8e3679764d2a* (Accessed: July 2021).
* Ciravegna, F. (2001) *Challenges in Information Extraction from Text for Knowledge Management*, Sheffield: University of Sheffield.
* Department for Transport (1995) *The "SCOOT" Urban Traffic Control System*, London: Department for Transport.
* Director of Governance and Solicitor to the Council (March 2018) *CORPORATE ASSET MANAGEMENT PLAN - UPDATE*, Glasgow: Glasgow City Council.
* EasyWay (2012) *DATEX II – The standard for ITS on European Roads*, Europe: European Commission.
* ESRI (2018) *Create lines from distance and bearing,*Available at: *https://solutions.arcgis.com/defense/help/distance-direction/arcmap/create-lines-from-distance-and-bearing.htm* (Accessed: July 2021).
* ESRI (2021) *Visualization and Exploration: Make sense of your data,*Available at: *https://www.esri.com/en-us/arcgis/products/spatial-analytics-data-science/capabilities/visualization-exploration* (Accessed: June 2021).
* ESRI Technical Support (2018) *How To: Create connection lines between two points,*Available at: *https://support.esri.com/en/technical-article/000012092* (Accessed: July 2021).
* G2 (2021) *FME Alternatives & Competitors,*Available at: *https://www.g2.com/products/fme/competitors/alternatives* (Accessed: May 2021).
* GISGeography (2021) *30 Best GIS Software Applications [Rankings],*Available at: *https://gisgeography.com/best-gis-software/* (Accessed: June 2021).
* GISGeography (2021) *FME Software – Feature Manipulation Engine (Review),*Available at: *https://gisgeography.com/fme-software/* (Accessed: May 2021).
* Glasgow City Council (2018) *Digital Glasgow*, Glasgow: Scottish Government.
* GrindGIS (2019) *Pros and Cons of ArcGIS,*Available at: *https://grindgis.com/software/pros-and-cons-of-arcgis* (Accessed: June 2021).
* Hernández, M.A., et al (1998) 'Real-world Data is Dirty: Data Cleansing and The Merge/Purge Problem', *Data Mining and Knowledge Discovery*, 2, pp. 9-37.
* Hounsell, N. and McLeod, F. (2005) *ASTRID: Automatic SCOOT Traffic Information Database*, UK: TRL Contractor Report 235..
* Hu, Y. (2018) 'Geo-text data and data-driven geospatial semantics', *Geography Compass,*12(11)
* ITS (2017) *INTELLIGENT TRANSPORT SYSTEMS (ITS) AND THEIR BENEFITS*, ITS: May 2021.
* Keboola (2020) *The 7 best data pipeline tools of 2021,*Available at: *https://www.keboola.com/blog/the-7-best-data-pipeline-tools-of-2020* (Accessed: May 2021).
* Leleux, C. et Webster, W. (2018) 'Delivering Smart Governance in a Future City: The Case of Glasgow', *Media and Communication,*6(4), pp. 163-174.
* Liu, Z. and Wu, K. (2021) 'Chapter 2 Urban Traffic Monitoring from Mobility Data', in Liu, Z. and Wu, K. (ed.) *Mobility Data-Driven Urban Traffic Monitoring.*Singapore: Springer, pp. 11-16.
* Loewen, C. et al (2021) *Windows Subsystem for Linux Installation Guide for Windows 10,*Available at: *https://docs.microsoft.com/en-us/windows/wsl/install-win10* (Accessed: May 2021).
* Marazzi, A. (2016) *Building a Data Pipeline from Scratch,*Available at: *https://medium.com/the-data-experience/building-a-data-pipeline-from-scratch-32b712cfb1db* (Accessed: May 2021).
* Microsoft (2020) *Windows Subsystem for Linux Documentation,*Available at: *https://docs.microsoft.com/en-us/windows/wsl/* (Accessed: May 2021).
* Monroe, R. (2018) *Data Pipeline,*Available at: *https://xkcd.com/2054/* (Accessed: May 2021).
* National Geographic (2021) *GIS (Geographic Information System),*Available at: *https://www.nationalgeographic.org/encyclopedia/geographic-information-system-gis/* (Accessed: June 2021).
* Official Journal of the European Union (2010) *DIRECTIVE 2010/40/EU OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 7 July 2010 on the framework for the deployment of Intelligent Transport Systems in the field of road transport and for interfaces with other modes of transport*, Europe: European Union.
* OpenTransportNet (2017) *Visualization of traffic volume changes during road works in the city of Plzen ,*Available at: *https://joinup.ec.europa.eu/release/visualization-traffic-volume-changes-during-road-works-city-plzen* (Accessed: July 2021).
* Ordnance Survey (2020) *A Guide to Coordinate Systems in Great Britain: An introduction to mapping coordinate systems and the use of GNSS datasets with Ordnance Survey mapping*, SOUTHAMPTON: Ordnance Survey.
* Ordnance Survey (2021) *Software for mapping,*Available at: *https://www.ordnancesurvey.co.uk/business-government/tools-support/gis/software* (Accessed: June 2021).
* Ordnance Survey (2021) *Using GIS software,*Available at: *https://www.ordnancesurvey.co.uk/business-government/tools-support/gis/using-gis* (Accessed: June 2021).
* Press, Gill (2016) *Cleaning Big Data: Most Time-Consuming, Least Enjoyable Data Science Task, Survey Says,*Available at: *https://www.forbes.com/sites/gilpress/2016/03/23/data-preparation-most-time-consuming-least-enjoyable-data-science-task-survey-says/?sh=799b09d46f63* (Accessed: May 2021).
* Robertson, D.I. and Hunt, P.B. (1983) *Estimating the benefits of signal co-ordination using TRANSYT or SCOOT optimisation.*, London: Proceedings 53rd ITE Conference.
* Safe Software (2021) *What is FME?,*Available at: *https://safe-software.gitbooks.io/introduction-to-fme-desktop/content/1.getting-started/1.02.what-is-fme.html* (Accessed: May 2021).
* Shi, J. et al (2019) 'Engaging Stakeholders in Urban Trac Restriction Policy Assessment Using System Dynamics: The Case Study of Xi’an City, China', *Sustainability,*11(3930)
* Technavio (2021) *Top 6 Vendors in the GIS Market from 2017 to 2021: Technavio,*Available at: *https://www.businesswire.com/news/home/20170110005543/en/Top-6-Vendors-in-the-GIS-Market-from-2017-to-2021-Technavio* (Accessed: June 2021).
* Thompson, Ken (1968) 'Programming Techniques: Regular expression search algorithm', *Communications of the ACM,*11(6), pp. 419-422.
* Tobin, D. (2020) *ETL Pipeline vs. Data Pipeline: What's the Difference?,*Available at: *https://www.xplenty.com/blog/etl-pipeline-vs-data-pipeline/* (Accessed: May 2021).
* Transport Scotland (2015) *TSIS DATEX II v2*, Scotland: European Commision.
* TrustRadius (2020) *ArcGIS vs QGIS,*Available at: *https://www.trustradius.com/compare-products/arcgis-vs-qgis* (Accessed: June 2021).
* UK Parliament Session 2010-12 (2010) *Effective road and traffic management*, London: Crown.
* Van Beek, Taylor (2018) *Data Extraction Techniques to Mitigate Unstructured Data Challenges,*Available at: *https://www.adlibsoftware.com/blog/2018/september/data-extraction-techniques-to-mitigate-unstructured-data-challenges.aspx* (Accessed: May 2021).