

BIG DATA ANALYSIS IN PUBLIC TRANSPORTATION DATA

Submitted by:

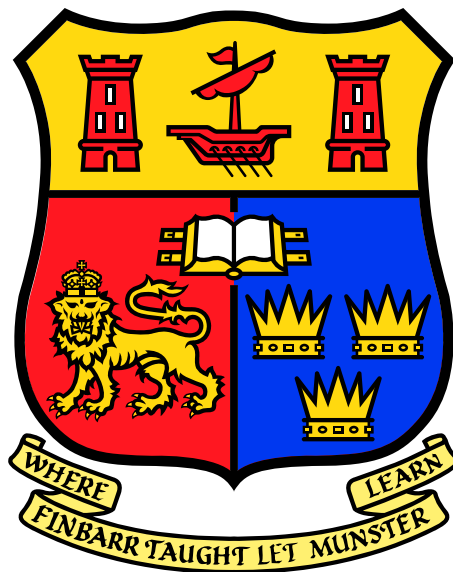
AAUSUMAN DEEP

Supervisor:

DR. ALEJANDRO ARBELAEZ

Second Reader:

DR. LAURA CLIMENT



MSc Data Science and Analytics

School of Computer Science & Information Technology
University College, Cork

September 4, 2020

Abstract

The data generated in the public transportation sector has an enormous potential to assist us in making key decisions to improve those services over which a vast majority of population depends for their travels and daily commutes. This can be achieved by properly cleaning, treating and processing the raw available data and then extract relevant information out of it through various analytical techniques.

In the following project, a month of public bus network data of Dublin Ireland has been taken as our subject and a big data analysis has been performed over it to harness some important details that can help the concerned department in better understanding the service that they work with, how it is performing and whether there is any room for improvement. Since the data generated at each instant in this network is quite detailed, the level of data analysis that can be performed in this regard is also quite comprehensive. Therefore, usage of a proper big data technology was on the cards and this is why Apache Spark on Hadoop has been used as the main technological tool in this research project.

The primary concerns related to public bus transportation networks is undoubtedly the delays with which they run on their respective lines throughout the day and the major factors which directly affect these delays, like congestions and busyness of those lines measured by the number of stops made by it. An algorithm has been initially produced to identify the most affected lines by these delays, with stepwise delve into deeper and more detailed delay analysis. A few prediction models including a time series analysis have also been performed over the average delays with which the buses run from the data that we have as our principal repository.

All the results obtained from these analysis have then been compiled in an attempt to give out a detailed explanation and present the key insights in a constructive manner.

Keywords - (1) Public Transportation Data, (2) Apache Spark, (3) Hadoop, (4) PySpark, (5) Big Data Analysis

Declaration

I confirm that, except where indicated through the proper use of citations and references, this is my original work and that I have not submitted it for any other course or degree.

Signed: _____

Aausuman Deep
September 4, 2020

Acknowledgements

I would like to show the greatest appreciation to my supervisor Dr. Alejandro Arbelaez for guiding me through this project and for assisting me in this journey, considering we were miles apart in the current pandemic situation of COVID-19. His continued direction in progressing through this thesis, with the efficient and effective scrutiny over my work with constructive feedback on a weekly basis over Microsoft Teams really helped me in completing this dissertation. This fulfilment won't have been possible without him and for this I am deeply grateful.

I would also like to take a moment to thank Dr. Eric Wolsztynski and Dr. Micheal Cronin for aligning me with the industry level machine learning algorithms and data analysis techniques which form the backbone of this analysis project. I would also like to thank Dr. Gregory Provan who introduced me to Deep Learning mechanisms and how we can work with them by utilizing online computing platforms like Google Collaboration environment. This knowledge had assisted me in understanding the distributive nature of the technology I have used in this thesis. Finally, I would also like to mention Dr. Kieran Hurley, whose collective classes on Python and its various data analysis specific capabilities laid the groundwork for me to choose PySpark as the technology in this thesis.

All the information that I gained and garnered here at UCC has cumulatively assisted me in performing the tasks I set out to do in the course of this dissertation and I appreciate everyone's efforts for working through this venture with me.

Dedication

'Without effort, there are no rewards. And to make effort, we need someone who believes in us.'

I would like to dedicate this research thesis to my parents, without whose support and countless sacrifices, I would not have been able to stand where I currently am in life.

Contents

Contents	vi
List of Tables	viii
List of Figures	ix
1 Introduction	1
1.1 What is Big Data?	2
1.2 Big Data Technologies	3
1.2.1 Data Storage	4
1.2.2 Data Mining	4
1.2.3 Data Analytics	4
1.2.4 Data Visualization	4
1.3 Tools and Technologies being used in this project	4
1.4 Familiarisation with our data set	5
2 Literature Review	9
2.1 Introduction to problem statement	9
2.2 Description of the Publications	12
2.2.1 An Architecture for Big Data Processing on Intelligent Transportation Systems	12
2.2.2 Effective Bus Arrival Time Prediction based on Spark Streaming platform	14
2.2.3 Short Term Bus Passenger Demand Prediction	16
2.3 Summary of Main Points	17
2.4 Contributions of these publications to Big Data in Public Transportation sector	18
3 Methodology	20
4 Detailed Process	23
4.1 Environmental Setup	23
4.2 Preliminary Analysis	25
4.3 Stop Locations	28
4.4 Busy Lines and Congestion	30

<i>Contents</i>	vii
4.5 Operator Influence and Latency	37
4.6 Average Delays	42
4.7 Average Reaching Delay at Stops	44
4.8 Time Series Analysis	47
4.9 Multiple Linear Regression	54
4.10 Random Forest Regression	57
4.11 GitHub Code Repository	59
5 Conclusion and Future Work	60
Bibliography	62

List of Tables

1.1	Data set structure	6
2.1	Highest number of Vehicles registered	10
4.1	Stops on Line 747	29
4.2	Busy Lines Result	32
4.3	Congestion Result	35
4.4	Number of vehicles per Operator	39
4.5	Average Delay/Operator on Jan 1	41
4.6	Average Delay of Line 747	45
4.7	Average Reaching Delay on stops for 747 line	47

List of Figures

1.1	Data and Knowledge	2
1.2	The 4Vs of Big Data	3
3.1	Methodology	21
4.1	Installation Verification	24
4.2	Number of records per day in January 2013	26
4.3	Number of stops per line on January 1	33
4.4	Congestion timestamp counts/line on January 1	36
4.5	Number of Lines facing congestion of specific timestamps on Jan 1	36
4.6	Number of vehicles per Operator	39
4.7	Average Delay/Operator on Jan 1	41
4.8	Average Delay of Line 747	44
4.9	Average Reaching Delay on stops of Line 747	48
4.10	Checking stationarity	49
4.11	Rolling mean standard deviation	50
4.12	Auto-correlation graph for inferring value of 'p'	50
4.13	Partial Auto-correlation graph for inferring value of 'q'	51

Chapter 1

Introduction

What 'buzzwords' have you heard pertaining to the information technology industry over the last few years? Every day when we open up our workstations and our phones and arrive at the feed where we obtain our daily dosage of information before getting ready for work, we come across a few repeated words which claim to be the next big thing in the technological industry. Artificial Intelligence, Internet of Things, Block chain and Quantum computing are to name a few. Every couple of years one of these words creates a hype and seeds itself into the minds of each and every professional trying to make in the industry with an idea of their own. It won't exactly be correct to say that they are over-hyped, but we have to admit, people have been working at them in one or the other way for years already and it seems just a fancy way of reigniting curiosity in us.

But, sometimes one of them comes along and sweeps us off our feet. 'Big Data' was already huge by the time I associated myself with it, but to be honest it was worth the wait. I thought it was again one of those hyped up frameworks which will die down to normality eventually after people get bored off it, but it did not and that's where its sheer potential was first revealed. If a technology, after flowing through each technical valley in the world, still managed to maintain its reputation amongst those same people which breezed through each new tech and discarded them, it was a real candidate. And am I glad that I got into it.

The usage of predictive analysis dates back to the 1940s when governments began using early computers to carry them out. So, it's easy to say that this is also a very fundamental area of technology, but the big data introduction in recent years has unlocked a whole new level of information processing that can be performed by us.

The need for processing the vast amounts of data being generated every minute in every industry is ever increasing. It is quite important to derive methods to reduce the production of redundant data in this data pool. Technologies are there at our disposal, but how we effectively use them is what makes an analytical algorithm into a successful analytical funnel. This extracted knowledge can then be used to perform further tasks

for predictions and decision making.

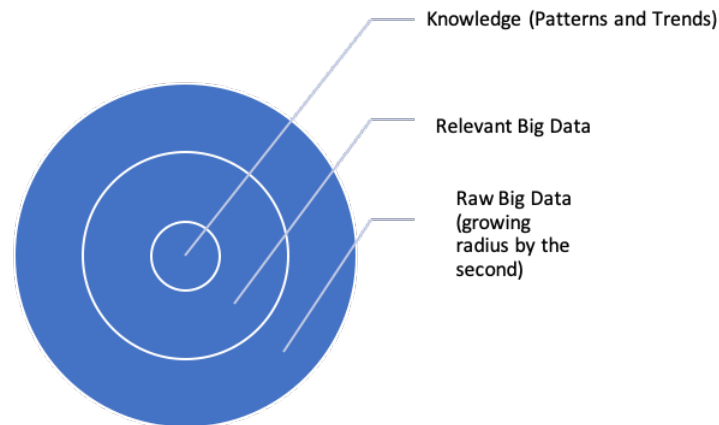


Figure 1.1: Data and Knowledge

As seen in the illustration above, the amount of big data generated increases every second, and we need to model out ways to identify relevant data out of it (this is called data cleaning) and after this subsequently we need to extract knowledge from that relevant data.

1.1 What is Big Data?

The first question that we need to address is what exactly does Big Data mean. By its very definition, it comprises of a set of extremely large data sets that may be analyzed computationally to reveal patterns, trends and associations. It is formed by a combination of structured, semi structured or unstructured data.

When talking about Big Data, its 3Vs take the forefront in explaining its significance.

1. Volume - The huge volume of data in various environments.
2. Variety - The varied nature of the data types collected.
3. Velocity - The speed at which the data is generated, collected and processed.

These characteristics of Big data were introduced by Doug Laney on 6 February 2001, who at that time was an analyst at META Group. He highlighted a few facts in his article. [Laney 2001]

1. Within a year, organizations would increase their usage of a centralized data warehouse to align internal and external practices for a more standard driven approach.

2. In a couple of years, data quality and integration problems will be eradicated by data profiling technologies (for generating metadata).
3. In a span of 4-5 years, data, document and knowledge management will amalgamate through a single schema indexed strategy.

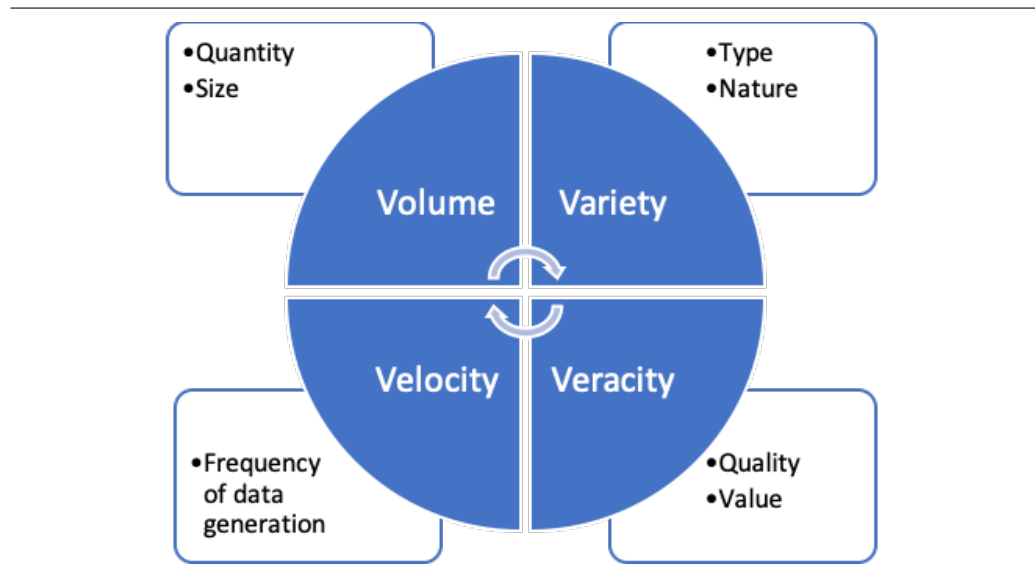


Figure 1.2: The 4Vs of Big Data

The above illustration clearly defines the major associations with Big data and what they signify. Considering all these practices have been brought to realism over the defined timelines and not only grown but exponentially evolved over the past couple decades, it is fair to say that this marked the birth of big data in a formal manner. More recently, 'Veracity' has been added as the fourth V in this structure which signifies quality and value of data at hand.

1.2 Big Data Technologies

There are various technologies associated with big data. They are various software utilities that are capable of analyzing, processing and extracting information from an extremely large complex dataset. From the initial step of collection, across multiple stages of cleaning, transformation and processing to the end of decision making through predictions, the tasks needed to be done are immensely complicated.

Top big data technologies are divided into 4 major categories -

1. Data Storage

2. Data Mining
3. Data Analytics
4. Data Visualization

1.2.1 Data Storage

Hadoop - A framework developed by Apache Software Foundation to store and process datasets on a distributed environment i.e. a collection of systems breaking up chunks of tasks for faster processing.

MongoDB - In comparison to basic rigid schemas of a relational database system like SQL, MongoDB offers a larger flexibility making it capable of handling large datasets of varied datatypes.

1.2.2 Data Mining

Elasticsearch - A search engine based on Lucene library.

Presto - It is an open source distributed SQL query engine capable of running interactive analytic queries on data sources.

1.2.3 Data Analytics

Kafka - It is a distributed streaming platform with 3 major capabilities - publisher, subscriber and consumer.

Spark - Another hugely popular tech with in memory computing capabilities to deliver high speed and a generalised execution model to support various types of applications.

1.2.4 Data Visualization

Tableau - A hugely popular data visualization tool used in business intelligence industry.

Plotly - Another tool used for creating graphs in an efficient manner.

1.3 Tools and Technologies being used in this project

In this project, considering the data set that we had at our hands, which we will elaborate upon in the next section, it had been decided to utilise Spark for our big data analysis procedure. The sheer amount of data available had already made sure that a proper and efficient big data tool needs to be relied upon, after that it was just the point

of identifying the right one. Apache Spark has already been a proven technology for handling big data sets with ease and its distributed nature would have undoubtedly made the tasks at hand a lot easier. [*Spark Overview* n.d.]

The public availability of the data set, and our usage of just a month's data has prevented us from using a database system like MongoDB.

For the choice of programming language, Python and its integration in the Spark environment as 'PySpark', made going along with it the easiest decision.

1.4 Familiarisation with our data set

The data set that we are working on in this thesis is the government provided data of bus network of the public transportation sector of Dublin, the daily commute lifeline of the multi-million people residing in the capital city of Republic of Ireland. [*Dublin Bus GPS sample data from Dublin City Council (Insight Project)* n.d.]

It has been published by the Dublin City Council onto the data.gov.ie website, which is the centralised government repository for all generally available data for usage by subsequent parties in Ireland and abroad. The license with which this data is available is 'Creative Commons Attribution 4.0 International' which allows the users of these data sets to -

1. Freely share - copy and redistribute the material in any medium or format.
2. Freely adapt - remix, transform, and build upon the material for any purpose, even commercially.

We are under one important term by using this data. Attribution - We must give appropriate credit, provide a link to the license, and indicate if changes were made. We may do so in any reasonable manner, but not in any way that suggests the licensor endorses us or our use. And hereby we have done exactly that by providing an insight as to where we exactly obtained this dataset and have clearly defined usage requirements.

Next, moving on to the data structure. The dataset have been provided to us in csv format, and each datapoint (row in the CSV file) has the following entries -

1. Timestamp
2. Line ID
3. Direction
4. Journey Pattern ID
5. Time Frame

6. Vehicle Journey ID
7. Operator
8. Congestion
9. Lon
10. Lat
11. Delay
12. Block ID
13. Vehicle ID
14. Stop ID
15. At Stop

Following is a table showing a sample of this dataset.

Table 1.1: Data set structure

Timestamp	LineID	Direction	JourneyPatternID	TimeFrame	VehicleJourneyID	Operator	Congestion	Lon	Lat	Delay	BlockID	VehicleID	StopID	AtStop
1.357E+15	747	0	7470001	31/12/12	3493	SL	0	-6.236852	53.425327	-709	747006	40040	7411	0
1.357E+15	27	0	null	31/12/12	3883	RD	0	-6.233417	53.342232	0	27017	33521	395	0
1.357E+15	40	0	null	31/12/12	2226	HN	0	-6.27825	53.416683	0	40206	33142	6071	0
1.357E+15	7	0	71003	31/12/12	6106	D1	0	-6.231633	53.317768	0	7019	43004	3222	1
1.357E+15	747	0	7471001	31/12/12	3531	SL	0	-6.254617	53.355484	-454	747007	40039	1445	0
1.357E+15	56	0	056A1001	31/12/12	1830	RD	0	-6.233183	53.342201	0	56001	33488	2379	0
1.357E+15	25	0	025A0001	31/12/12	2866	CD	0	-6.296867	53.3475	0	25007	33604	4604	0
1.357E+15	747	0	7470001	31/12/12	3493	SL	0	-6.238668	53.425789	-687	747006	40040	7411	0
1.357E+15	27	0	null	31/12/12	3883	RD	0	-6.2334	53.342232	0	27017	33521	395	0
1.357E+15	4	0	null	31/12/12	4243	HN	0	-6.279	53.416683	0	4001	43043	7226	0

As can be seen in each of these records, all the columns denote some significance to the dataset at hand. Delay column will be one of the most important variable of interest to us because it is the one which will be a key decision maker, due to its relation to the actual performance of a bus network, once the appropriate model has been trained.

For the grouping scenario for division of dataset, we will be starting at the Line ID, because logically it will be the one which ideally separates the performance of the buses within their own defined patterns.

- Timestamp - It denotes the exact timestamp at which that particular row was collected.
- LineID - It denotes the numerical ID of the Line to which that particular row belongs to.
- Direction - It informs us of the direction that particular bus is running in. It is denoted by a '0' if the bus is going from the source station towards the destination station or '1' if the bus is going in the opposite direction towards the source.

- JourneyPatternID - It is a division within the LineID. Various kinds of patterns are predefined within a LineID and JourneyPatternID is there to present that exact information.
- Timeframe - Denotes the timeframe within which that particular row of data falls into. (The start date of the production timetable - in Dublin the production timetable starts at 6am and ends at 3am).
- VehicleJourneyID - It denotes a given run within a journey pattern.
- Operator - This denotes the Bus Operator initials (not to be confused with the driver)
- Congestion - This denotes that whether at that exact timestamp, the bus is experiencing any congestion on the route it is currently running on. It is 0 if no, otherwise 1 if yes.
- Lon - Longitude value in WGS84 format. WGS stands for World Geodetic System and it is a standard used in cartography and satellite navigation including GPS. WGS84 is an earth centered, earth-fixed terrestrial reference system falling under WGS.
- Lat - Latitude value in WGS84 format
- Delay - This signifies the delay with which the bus is currently running. It is negative if the bus is running ahead of schedule.
- BlockID - This denotes a section ID of the journey pattern.
- VehicleID - As the name suggests, it denotes the ID of the vehicle within this particular row of data.
- StopID - As the name suggests, it denotes the ID of the stop that the bus is closest to (usually the next stop, or the current stop it is standing on), with respect to the current row of data.
- AtStop - It denotes the status value whether a bus is at the stop it is showing in the previous column i.e. StopID, or not. It is 0 if no, otherwise 1 if yes.

The dataset available to us, is present in this similarly structured set of 31 csv files, all containing data of each individual day in the month of January 2013. Due to the large volume of data combined in these files, this dataset at our hand can be defined as big data. Approximately 44 million records are present in these files combined.

We are going to apply a few big data analysis algorithms, which have been discussed further in this thesis document. However, it is always a beneficial method to delve deep into previous works that have been done in the same field to get an idea on how the industry demands information pertaining to such problems.

A few such research papers have been referenced, which are all related to analysis of data in the public transportation sector. The information presented in them ranges from a simple analytical project to some deep layered architectural approaches. A detailed review of those works is in the next chapter.

Chapter 2

Literature Review

When we talk about the public transportation sector, the data generation points are numerous and they provide us with amazing opportunities of capturing some really interesting sets of data. If we think about this topic from a real world perspective, the rising population has led the people seeking some mode of transportation to move about. Granted, in current times of COVID-19, this is something we really can't stress upon, but when life does go back to normality, this will always be a pressing matter. In this technological age, we will undoubtedly set up some sort of a pipeline where data is collected from these modes of transportation for various reasons, be it for analytical purposes or just surveillance. Everything from the person using the transportation to the functioning attributes of it will be extracted. [Elgendy, and Elragal 2014]

2.1 Introduction to problem statement

Every individual in this world can't afford a vehicle of their own. A majority of inhabitants of varying demographics depend on the public transportation that have been provided by their city councils and facilitated to them. These modes range from land, aquatic and airborne and some time or the other we do need them.

Also, along with the affordability being one factor, the rising congestion problems due to increasing vehicle numbers on the roads is another such problem. Managing this issue has a very successful tried and tested technique of focusing on proper development of public transportation which have the capacity of carrying more people with the similar on-street footprint. By ensuring a good operating frequency and high efficiency in such a way that it meets the demand of the urban population around them, we can somewhat curb this problem.

And not to mention, the rising problem of global warming and climate change is directly affected by the emissions made by the vehicles. The higher the number, the bigger the problem gets. Although, there have been numerous modifications to existing trans-

portation technologies to reduce these emissions, a decrease in vehicle number will be of much greater help. That's where public transportation can also chime in by carrying more amount of people with minimum regulated vehicles.

In the following data, we focus on the number of registered vehicles per 1000 people residing in different parts of the world. This data will be indicative for us to map out the rising number of vehicles in a country and how public transportation can be useful in reducing these numbers.

Table 2.1: Highest number of Vehicles registered

Country	Number of vehicle per 1000 population
San Marino	1263
Monaco	899
New Zealand	860
United States	838
Iceland	824
Liechtenstein	773
Malta	766
Finland	752
Australia	730
Brunei	721
Switzerland	716
Canada	685
Guam	677
Luxembourg	670
Italy	655

These above 15 nations rank the highest number of registered vehicles with respect to their total populations. These high numbers can definitely be reduced by introduction of higher focus on the public transportation.

As cited by researchers, in recent years there has been an immense advancement in the field on intelligent transportation systems. Due to this, the public transportation authorities across the world have developed efficient data acquisition techniques and workflows. The sweeping in of the big data era has not only helped this situation but introduced the analyst teams with amazing opportunities to mitigate forthcoming problems even before their arrival.

The availability of multiple sourced data in different forms, with cell phone data, GPS data, surveillance data, Wi-Fi data to name a few, these acquisition techniques had to be refined in an iterative manner to become efficient over time. However, the collection

of these data sets was not an easy task, because traditionally, surveys has always been the go to methods for data collection at a large scale. The operation and planning of the public transportation systems depends a whole lot on these factors of demand and supply, and surveys were not able to effectively work in collection of the detailed spatial and temporal data we need.

That's why the usage of smart transportation systems has had such an impact.

- The Geo-positioning data monitored via our consent through our phones and satellites
- Urban transportation smart cards for riders
- Occasional surveys
- Road sensors
- Radio frequency identification readers
- Social media feeds
- Cameras
- Microphones

These methods produce massive amounts of data without hindering the convenience of the system users. They agree to it once, and then the data is automatically collected according to the needs of the analytical companies who work continuously to acquire the relevant data that can assist them in their protocols. This collection of data has a standard set of characteristics which are essential for big data processing. These include

- Continuous nature instead of discrete
- Wide coverage rather than a clustered set of survey data
- Comprehensive information
- Updates in a dynamic nature

Since it was obvious to the world that prioritizing public transport was necessary, the reliability of them was the next concern. Resource allocation, network planning and frequency setting are parameters that substantially depend on the short term passenger demand. But why stop there? As we already have intelligent transportation systems in place which pour in streams of information by the second, dynamic decision making was possible for forecasting delays and subsequent scheduling of buses and tweaking there frequencies. Accuracy in these predictions will not only reduce the operation cost but also increase the service quality as a whole.

2.2 Description of the Publications

Following are a few related publications to the topic of Big Data processing and its impact on the public transportation sector.

2.2.1 An Architecture for Big Data Processing on Intelligent Transportation Systems

In this publication at the Universidade Nova de Lisboa, Portugal, the researchers have aimed at proposing an ETL architecture for intelligent transportation systems for addressing an application scenario of dynamic toll charging on the highways. [Guerreiro et al. 2016]

They have utilised big data technologies to store and process large datasets from various sources provided by different highway operators. According to this research, the bigger challenge after collection of data is the processing of huge volumes of unstructured data for subsequent analytics. The data available in such cases is also full of inconsistencies due to missing data and unaligned nature. The traditional approaches towards data analysis do not work efficiently on them, and this is the reason why they have utilized an ETL approach. The proposed architecture has three major capabilities -

1. Account for the available data quality.
2. Ability to mould itself according to already existing data standards under the Intelligent Transport System domain.
3. Furnish a robust and scalable storage system.

Apache Spark lies at the soul of this architecture. SparkSQL works simultaneously along with it for the processing needs and MongoDB has been used for data storage requirements. These technologies allow the parallel in-memory processing of data.

To successfully implement the structure defined by them for their workflows, CRISP-DM methodology has been followed by them in the project. It stands for Cross Industry Standard Process for Data Mining and is a highly regarded framework in data analysis. It consists of six basic steps -

1. Business understanding - Success criteria, objectives. From this perspective, the data was generated by interviews with several highway operators.
2. Data Understanding - Data collection, quality check and developing introductory insight into data.
3. Data Preparation - Cleaning and transformation of data to more closely suit the needs of the project.

4. Modelling - Application of appropriate modelling technique after comparative selection.
5. Evaluation - Evaluation of obtained model in the previous step and decision making on how to interpret the results for collective benefit.
6. Deployment - Charting out use cases of the obtained knowledge and results.

Moving on to the actual model developed by the team, the primary objective was to encourage drivers to use national highways by dynamically affecting toll prices of national roads during peak hours. This was done to restore a balance on the streets and manage financial ratios in infrastructure management in the long run, simultaneously increasing the quality of life for all inhabitants by preventing them to project themselves through the same roads causing congestion and pollution.

There were several factors in play while developing this toll pricing model.

- Real time conditions of road networks
- Quality of service
- Road safety
- Environmental data
- Cost maintenance
- Toll revenues
- Congestion
- Traffic events
- Weather conditions

All these data values were collected and subjected through the Spark engine. MongoDB was used for storing and managing the data. The direct integration of this NoSQL database system with Spark and its scalability made it the obvious choice. After the pre-processing step of data cleaning, classification into two categories was done - tolling data and vehicle counting data, with both different in the essential meaning they supplied to the model.

Tolling data described the actual number of vehicles that were tolled within a 5-minute interval at a specific area of the highway in question. Vehicle Counting data described the total number of vehicles counted within a 5 minute interval at a specific area of the highway in question.

Since the data had already been subjected to transformation prior to being processed

through the model, the trained model obtained was appropriate for future DATEX-II transformed data as well (DATEX-II was the data transformation standard used while data cleaning step).

The results obtained using the classic approaches took approximately double the time than that it took by using the Spark approach. The conclusion was driven that usage of big data technologies had improved efficiency and computational timelines. Since the data used was of historical nature, plans were laid out to use it in the future on much larger data sets.

2.2.2 Effective Bus Arrival Time Prediction based on Spark Streaming platform

This research touched upon the discussion we had before in our problem statement of the reliability of public transportation systems. In order for the operations to be smooth a lot of analysis has to be invested into perfecting it.

This publication at Inner Mongolia University proposed a particle filtering algorithm to formulate a bus arrival time prediction model. Going a further step in achieving higher optimization in the incepted model, the prediction error of using particle filter was reduced by accounting in latest bus speeds for collaborative data analysis. Apache Spark Streaming technique was used in the development phase, because achieving real time analysis was only possible by setting up a real time data stream and processing it on the go. [Liu, and Xiao 2019]

It was identified that the simultaneous processing of both real time and historical data was necessary in this project. Also, each 15 second ejected record was not required to be put through computation, and small batch processing of the real time stream was considered as a better approach.

The successful execution of particle filter algorithm is dependent on acquiring relatively recent data continuously and then correcting and updating the prediction results defined in the previous iteration with the real time computation on the recently acquired data set. Since there is a pre-existing issue with particle filter algorithms on weight degradation in the iterative process, in order to avoid this problem of particle becoming less effective in representing the probability distribution of variable over time, they used a re-sampling method to curb it. Sequential Importance Sampling was selected for this step.

The two main problems in particle filter algorithms usage for a bus time prediction model, which are discussed in short in previous paragraphs, are -

1. Prediction error

2. Acquisition of records

The first problem had been mitigated by introducing average speed of the latest previous bus in the same road segment, and the second by construction of optimized observations.

The core algorithm adopted in the model was as follows -

1. Initialise the basic parameters
2. Categorizing all pre-processed GPS records
3. Gathering all bus data which are not at terminals, i.e. in transit
4. Initialising the particle swarm and taking the sequence number of key road segments as 0 to obtain the data of latest two buses.
5. Computing number of key road sections where that particular bus trip is located at that particular timestamp.
6. Classifying the data for each bus trip according to the number of key road segments.
7. Computing the historical average speed of each bus trip on that key road segment.
8. Calculating the particle group expectation
9. Calculating the weight of each particle based on the weight calculation formula and putting it through a normalisation funnel.
10. Calculate degree of degradation and deciding whether to resample or not
11. Performing the resampling if necessary
12. Showing the final prediction result

After the formulation of the final algorithm, the prediction model was designed. Since streaming in the form of batch processing had already been decided upon before, the streaming window was only set for particular amount of time. HDFS was used because of the efficiency and reliability of distributed architecture in such a case.

It was identified that as the number of particles go up, the distribution of the characterisation gets better. After setting the particle number to 2000, the prediction simulation was ran. This prediction process actually predicted the travel time in the next key road segment according to the driving condition of the previous key segment of the bus, on the constraint that the driving speed of the bus in these two sequence segments remains unchanged.

The results of this experiment were categorised for each bus into two periods - peak

hours and non-peak hours. During the non-peak hours, the maximum absolute error was 78.16 seconds. During the peak hours the maximum absolute error was 270 seconds.

In conclusion, the utilization of Spark platform, facilitated these researchers to effectively create a fruitful algorithm for arrival time prediction and played a key role in reducing the computational complexity considerably.

2.2.3 Short Term Bus Passenger Demand Prediction

This research article is based on Time Series Model and Interactive Multiple Model Approach and was published at Shanghai Jiao Tong University. It focuses on short term passenger demand forecasting through the help of time series analysis. This will in turn assist in improving the dynamic bus scheduling techniques and their management. Also, accurately predicting the demand will also help in increasing operation efficiency and overall reliability amongst the passengers. [Xue, Sun, and Chen 2015]

The objectives of this research were as tabulated below

1. Analyse characteristics of historical data with prime focus on stationarity, periodicity and volatility on differing time scales. After this, formulation of three time scales has to be done, one in a week, one in a day and last in 15 minutes which were constructed based on different characteristics.
2. Design separate prediction models which illustrate the characteristics of data to effectively predict weekly, daily and 15 minutes time series.
3. Dynamically amalgamate the prediction models estimations for further usage of outputting the hybrid predictions by applying IMM algorithm (Interactive Multiple Model) and evaluate its performance.

Time Series modelling on the data available to predict the bus passenger demand has been done in a hybrid manner. Both the data available in historical repositories and the real time extracted streams have been utilised in this modelling framework.

A four step process was followed in development of these time series

1. Based on the characteristics of historical data, three time series were made
2. After correlation analysis, each time series was further skimmed down to each having a separate weekly, daily and 15 min interval time series.
3. Adjust the time dependent transition probability matrix based on performance values of models based on historical data.
4. Combine the models' predictions using IMM algorithm.

The data collected for a couple of months comprised primarily of the passenger boarding data. Since this was acquired through the smart card usage of the passengers, Intelligent Transport System standard apply to this research.

This historical data was then subjected through various analytical techniques to map out a set of descriptive statistics. To further analyse this historical data's characteristics, three time series were constructed for the weekly, daily and 15 minutes intervals to illustrate upon the stationarity, periodicity and volatility of real time passenger demand.

After these time series were developed, next came the implementation of the IMM algorithm and its validation. Its basic idea was to match the varying snapshots of data and with different models and zero in on usage of a specific model with minimal error.

The process followed while implementing this is as mentioned below.

1. Calculate the mixed state and covariance at time 't' based on transition probability matrix.
2. Update the estimations for each model using Kalman filter algorithm and calculate the prediction residual and covariance with the input of real time passenger demand.
3. Update the probability for each model based on likelihood function of each model using prediction residual and covariance before.
4. Calculate the final estimation at time 't' by combining the updated estimations at time 't'.

To conclude, the real time prediction modelling facilitated by the usage of Time Series has been of prime importance in this research and has helped in increasing the operational efficiency and reliability of the bus network. This performance evaluation has optimised real time decision making capabilities of the network administrators and furnished them a way to achieve high functioning dynamic scheduling for better usage throughout the city.

2.3 Summary of Main Points

From the above literature review, we can set aside a few points that might be considered as the foundation steps in moving forward with deeper dives in works in this field. The related work mentioned above, in addition to minor associative articles read online, has been the driving force to move ahead in this dissertation.

The main points to focus upon from this literature review are as mentioned below

1. Data availability and variety of sources where we will accumulate it from is the first thing we need to concern ourselves with when working with a big data project. Sometimes the sheer numbers of sources make the process harder to keep a uniform nature in the incoming data and we need to formulate a plan on how to act in such a scenario.
2. Pre-processing techniques that are present in the industry have a set of standards that need to be followed in order to achieve proper data cleaning and transformations.
3. Sometimes the same source of data, may provide an irregular or alienated structure of incoming records and we need to be prepared for such a scenario with dynamic updates to our pre-processing techniques.
4. Public transportation sector is a gold mine when it comes to available streams of real time data. In order to prevent an anticlimactic situation at the point of maximum output, we need to select an appropriate model for our analysis algorithm development.
5. Big Data techniques need to be subjected upon harmonized data to the platform we have decided to use and have a hybrid setting in their mind for maximum information gathering. A stream of real time data going through Kafka engine will readily be sometimes of reduced usage without the Spark engine working on historical data first.
6. When big data is in the question, the approach of using a traditional relational database may need to be rethought. A lot of works in this field have employed the capabilities of NoSQL database systems like MongoDB or PostgreSQL for their data storage needs and have proven track record of efficient results.
7. Mining for data has various industry standard methodologies which are quite popular amongst miners in the data analytics industry. CRISP-DM is an example of this.

These few keynotes from the years of related works by noted scholars and professionals in this field has led us to inherit techniques which are proven to work in our cases and we move ahead with a strong foundation. Identifying out problem statement at the very beginning was essential and then walking through these publications has provided us with essential knowledge that will go a long distance in helping us achieve our bench-marked sub objectives.

2.4 Contributions of these publications to Big Data in Public Transportation sector

The publications brushed through above, along with the other sources cited by them, have done a whole lot of groundwork and made significant contributions to the Big Data

domain as a whole. In the recent years, since the Big Data terminology has compounded within the Information Technology sector, the research works have helped pave the paths of aspirants like us.

Considering the Public Transportation sector specifically, the contributions have been marvellous. The advancement in methods to acquire data and moving those techniques from traditional survey based sheets of paper handed out to the travellers, which more often than not, they threw away because they simply didn't have the time, to the automated collection of data using smart card in this era of Intelligent Transportation Systems is a boon.

The implementation of these big data techniques onto the vast transport datasets, and their appropriate usage in the publications above, is an example of how the public transportation sector is moving in the right direction, and with more relevant researches on the way, painstakingly trying to make all these tasks even simpler, we can only imagine the wonderful future ahead in this domain.

Chapter 3

Methodology

Considering the standard processes followed in the industry to conduct successful data analysis projects like CRISP-DM (Cross Industry Standard Process of Data Mining), [Crisp DM methodology 2020] and TDSP (Team Data Science Project), the lack of manpower behind this project drilling down to just an individual has led me to adopt a simpler workflow to meet the needs of the project.

A phased out road-map of steps with set objectives is the simplest approach for any project, not just in the data analysis field. The methodology followed for this project strongly resembles the 11 independent benchmarks that are set out in any data analysis project. Even though this is a big data scenario, the key established routes for progressing in the analysis remain same to their core.

1. Collaborate your needs - Under the supervision of Dr Alejandro Arbelaez we initially discussed the need to do this project.
 - How analysing the public bus data for Dublin could be of assistance to the authorities?
 - Why big data was going to play a part in this?
 - How Apache Spark would serve our purpose?
2. Establish Questions - The foundation of any project are the questions we are pursuing to answer through the work put into it. The primary ones set out by me were to
 - Work with the Delay data and congestion data and analyse them and see was there an effect of one over the other.
 - Also, which bus lines are struggling the most i.e. which have the maximum delay over time.
3. Harvest Data - Harvesting or identifying the main source of data was not an issue in this project, because we had a clear cut idea from the start that importing the

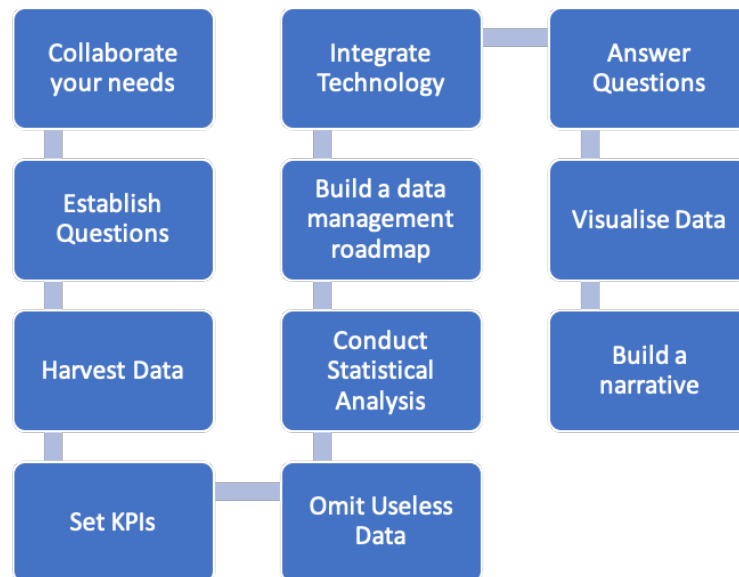


Figure 3.1: Methodology

January month's data of 2013 from the government website of Irish datasets was the primary source.

4. Set KPIs - Next we drew a clearer picture of the Key Performance Indicators of the project. The raw data collected was definitely of use, but we still have to filter out which features are going to be of more importance than compared to the others. For instance, in this dataset the KPIs are -
 - Delay
 - Congestion
 - AtStop

These three columns clearly mark the situation that particular bus was in at that timestamp.

5. Omit Useless Data - Data cleaning is an essential part of a data analysis life cycle. There are records which have incomplete information or mismanaged values and they need to be omitted to have a clearer picture of the analysis being performed by us. In this data set, there are a lot of records which have one or the other feature with an empty value. Instead of imputing those values, we decided to remove those records altogether. This was because imputing them becomes tricky, as the features missing like Timestamp would be useless when imputed on the basis of any aggregate measure.

6. Conduct statistical Analysis - We conducted some preliminary statistical analysis to identify the amount of data we were going to process and some key aspects of the various features in the data. This sole part of the process was performed using R language, as we performed this on just the first day of data i.e. January 1st, 2013 to get an idea of the data set in general.
7. Build a data management roadmap - Since the data in consideration is of the static form, its upload to a cloud repository is the easiest way to manage it. DataBricks was the main platform of usage in this project for processing of big data. It is an Azure service provided by Microsoft Corporation.
8. Integrate technology - DataBricks has native support for PySpark which is a subsidiary tool of Apache Spark Technology. It has been utilised for the entirety of the project, with minor levels of the algorithms replicated onto the local machine first, which is running a Spark engine with just the local system cores as the essence of its distributed architecture.
9. Answer Questions - After performing all the analysis by using the constructive algorithm techniques, we were able to answer the questions that we had initially constructed.
10. Visualise Data - The result obtained after any analysis project are effectively projected onto the viewers by the help of graphs and charts. This is why the visualisation techniques used are of prime importance.
11. Create a narrative - In the end of the process life-cycle, I am writing this dissertation document to summarise and accumulate all the findings and results obtained from the methods and analysis done throughout. This document is an attempt to completely project all my research and analysis and provide that to the reader in such a way that they can replicate all this from scratch.

Chapter 4

Detailed Process

4.1 Environmental Setup

The first order of business attended to in the environment setup phase was to download the dataset from the website. It comprised of 31 separate csv files, one for each day of January 2013. The file size for this entire download was a zipped 1.01 GB and expanded to over 4.3 GB once unpacked.

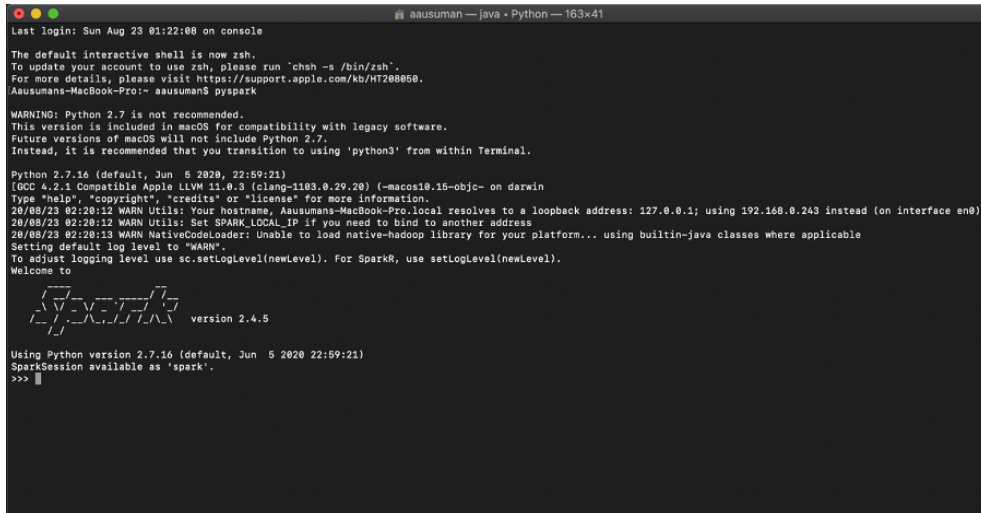
The next step was to setup a local installation of PySpark, because we wanted to make sure to run the jobs with a minor set of data before moving onto the cloud services and run it over the entire dataset. This was to be certain of the algorithm that we were going to scale up and attempt on the data.

The installation of PySpark on the local MacBook was easily done by following the steps mentioned in a specific article webpage and an acknowledgment is necessary to the author of this page for this derived assistance received from it. [Vecmanis 2019]

After following the process in this webpage, we test whether we have been successful in doing that or not. To test this we open a terminal window, enter 'PySpark' command and press the 'return' button (these will be replaced by a command prompt window and 'Enter' button on a Windows machine). The screenshot below confirms that Spark is up and running on the machine.

Please note that in the figure 4.1 it says that 'Python 2.7' is not recommended to perform any tasks as it has been depreciated, and we should have an explicit installation of Python3 on the machine for the fulfilment of this project's tasks.

The next step was to procure a proper cloud environment to process the big data that we had at our hands. There were multiple options like GCP's (Google Cloud Platform) service 'Dataproc', AWS's (Amazon Web Services) 'Elastic MapReduce' also known as



```

Last login: Sun Aug 23 01:22:08 on console

The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208856.
Aausumans-MacBook-Pro:~ aausuman$ pyspark

WARNING: Python 2.7 is not recommended.
This version is included in macOS for compatibility with legacy software.
Future versions of macOS will not include Python 2.7.
Instead, it is recommended that you transition to using 'python3' from within Terminal.

Python 2.7.16 (default, Jun 5 2020, 22:59:21)
[GCC 4.2.1 Compatible Apple LLVM 11.0.3 (clang-1103.0.29.20)] (-macos10.15-objc- on darwin
Type "help", "copyright", "credits" or "license()" for more information.
20/08/23 02:20:12 WARN Utils: Your hostname, Aausumans-MacBook-Pro.local resolves to a loopback address: 127.0.0.1; using 192.168.0.243 instead (on interface en0)
20/08/23 02:20:12 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Welcome to

      ____      __
     /  _/_____/  /
    _/  /_  _/  _/
   /___/  /___/  /
  version 2.4.5

Using Python version 2.7.16 (default, Jun 5 2020 22:59:21)
SparkSession available as 'spark'.
>>>

```

Figure 4.1: Installation Verification

EMR. But in the end I, along with the guidance of my supervisor, decided to move ahead with a similar offering of the Microsoft's cloud platform called 'Databricks'. Databricks is an industry leading data engineering tool that is used for the processing and transformations of massive quantities of data and their exploration through various machine learning models. It has been recently added to the Azure environment post-acquirement of it by Microsoft.

Since this is a 'pay as you go' service, utilization of the Community edition of it is recommended at academic levels to avoid paying hefty fees for its features. This only provided access to a reduced performing distributed architecture of Hadoop and limited storage, but considering the tasks at hand it served all the purposes.

The main architecture provided by a Hadoop framework is a cluster of machines over which we run our Spark jobs. One machine is called the driver node and the rest are called the worker nodes. The driver node, as the name suggests, is responsible for initiating the job and carrying out the distribution of the data into chunks and providing them to various worker nodes to effectively and efficiently process them. Higher the distribution in the architecture, better is the efficiency in data processing. The community edition account of Databricks only provides access to a network of one driver node and no other worker nodes i.e. that driver node itself is responsible for the processing of the data. Naturally it is not a powerful architecture as compared to one with multiple worker nodes, but considering the static nature and size of the data at hand, it worked well enough in this project.

The cluster created was having the following configuration:

- 1 driver node, 0 worker nodes
- 15.3 GB of memory
- Spark 3.0.0 version
- Support for Python3
- Availability zone of US West

We uploaded all the data files to the repository on the Databricks called the 'FileStore' within the Databricks File System (DBFS). It is a straightforward repository, however there are a few quirks associated with it.

1. To remove a folder or file from FileStore, use the following command in Databricks notebook. `dbutils.fs.rm("/FileStore/<>", True)`
2. To download a file from Databricks FileStore open the following URL in browser. <https://community.cloud.databricks.com/files/<folder>/<file>?o=8736540974339418>

The long number in the URL above is the unique token associated with this project's account. It will be different for each account and will have to be pulled out separately by each user.

There is native support for Python notebooks in the web UI of Databricks, and has been utilised for the project.

The entire code for this project is available in the form of a Git repository at the following url. <https://github.com/Aausuman/Thesis>

4.2 Preliminary Analysis

An initial exploration of the data was done. Since the total amount of records for the entire month of dataset are in excess of 44 million, a preliminary analysis needs to be done to get an estimated idea of the data we are dealing with.

The records collected are collected from all bus operators at regular intervals and the number of records are different on different days of the week.

As seen by the illustration in figure 4.2, the number of records by the authorities sees a fall in the weekend i.e. Saturday and Sunday, which suggests a decline in service during these days. The chart starts from a Tuesday because 1st January 2013 was a Tuesday.

This aggregation of data was performed by using the RDD datatype which is a native to Spark framework. RDD stands for a Resilient Distributed Data set and is at the core of the distributed architecture of Apache Spark, with the key concept being division

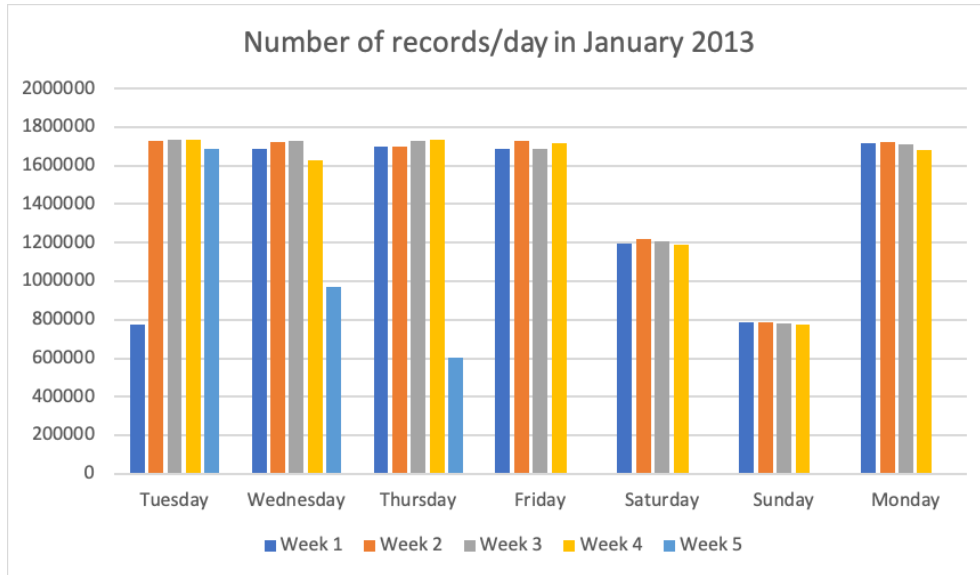


Figure 4.2: Number of records per day in January 2013

of data set onto different nodes and then simultaneous execution of those subsets as part of one job. [*RDD Programming Guide* n.d.]

The next step was to identify the number of records which consisted of missing data. This is an essential part of any data analysis project as cleaning of the data becomes necessary in preserving the accuracy of any model we train onto the training data set.

Out of the 44 million, around 5 million records were identified to be consisting of some missing data in one or more features. Since imputation was not useful considering the nature of the features, it was decided to remove those records altogether from the data set to get a clearer picture through analysis.

Certain more metrics were calculated while doing preliminary analysis. They are as listed follows -

1. 8 bus operators were present in this Dublin bus network during that time period.
2. 59 bus lines were present in this network.

The pseudo code for this above algorithm is as follows -

```
Importing SparkConf, SparkContext and SQLContext from pyspark library
Importing countDistinct from pyspark.sql library
```

```
function preprocessing(record){
```

```
        Separate each record string using ',' operator and return as  
        separate fields  
    end  
}
```

Initializing an empty rdd variable

```
For file in file_list:  
    Read each file as an rdd as a variable  
    Union this variable to the empty rdd variable
```

Remap the total records rdd through the preprocessing function

Converting rdd to a dataframe with predefined schema

Using countDistinct method to get different levels of Operators and LineIDs from this dataframe

The data cleaning function talked about previously was also utilised in the same code snippet. The pseudo code explanation for the same is as below

```
Function cleaning(dataframe){  
    Extract a list of columns in the dataframe  
  
    Create a filter expression to check for null values  
  
    Run the dataframe through this filter criteria  
  
    Drop duplicates from the filtered dataframe  
  
    Return the cleaned dataframe  
end  
}
```

This concluded the preliminary analysis performed upon the dataset. I also explored the dataset to ensure that the understanding which was mentioned on the website related to the description of the data, checked out by confirming those points on the 1st day of data in RStudio, such as no wrong values were present in any of the columns other than the ones which were supposed to be present in them.

Next, we start working with the more detailed and complex algorithms for the analysis of the data.

4.3 Stop Locations

This algorithm consisted of mapping out the location of each StopID and mapping that onto an actual map of Dublin. This would verify that the data was accurate and the subsequent analysis that we perform on the dataset would be completely true with respect to the features furnished by it.

A couple of assumptions were made before constructing this method. Number one, only the first day data would be required for this analytical algorithm, because all the stops would be covered by one or the other bus in a single day within the entire city. And second, since the latitude and longitude values given in records are at a very granular level, the mapping at the end of it may be deviating a little bit because of inaccuracies in procuring a map of the exact same scale.

The algorithm was divided into two sections, firstly to gather the approximately accurate values of the latitude and longitude of each stop of each line and then secondly plotting those coordinates. After setting up the three Spark global level variables - a Spark Configuration variable, a Spark Context variable and a SQL Context variable, the actual processing of data begins.

The steps followed in the algorithm are as follows -

1. Importing the day 1 csv in the form of an rdd.
2. Pre-processing the rdd with the 'cleaning' function mentioned in the previous section of preliminary analysis. This will omit the incomplete records in the dataset.
3. Converting the rdd into a data frame with a defined schema. The conversion of rdd into a data frame is done using the 'toDF' function over the rdd variable. This accepts one parameter of schema which is essentially a list of all the column names in the dataset.
4. Creating an empty data frame with a set schema of the relevant data we will be extracting out of this algorithm. We create a schema using the StructType and StructField methods imported from the pyspark.sql.types library.
5. Next, converting the data frame just created into a key-value pair rdd, with LineID feature as the key.
6. Use groupByKey() transformation function to group the rdd by lineIDs and use collect() method to store the results as a list.
7. Iterate over that list and walk through the records line by line.
8. For each line, convert that subset of data into another key value pair rdd with StopID as keys.

9. Further use `groupByKey()` transformation to group that rdd by StopID and use `collect()` method as before to store the result as a list.
10. For each stop, convert that subset of data into an rdd. Subsequently convert that rdd into a data frame with a schema.
11. Register that data frame as temporary table by using the function `dataframe.registerTempTable('nameforthetable')`.
12. Utilise Spark SQL and select the LineID, StopID, Latitude and Longitude values from that temporary table where the AtStop value is 1. Limiting the result to one to avoid repeated records of same Latitude and Longitude values.
13. Since a Spark SQL query returns a data frame type record set, unionising the result to the empty data frame created at the beginning.
14. Exporting that final data frame as a csv file for better and universal readability.

This entire process gives us quite an accurate picture of the coordinates of the various stops present around the bus network in Dublin city.

The resultant csv looks a bit as follows.

Table 4.1: Stops on Line 747

LineID	StopID	Latitude	Longitude
747	7456	-6.260201	53.344318
747	3665	-6.243983	53.428101
747	7401	-6.240617	53.426517
747	3669	-6.235045	53.428631
747	7399	-6.22705	53.348381
747	2002	-6.269407	53.343739

This table above is a collection of a few stops present in the LineID 747. The third and fourth column are Longitude and Latitude respectively, with the second column being the StopID. This result type similarly details out the values for the remaining lines as well.

The next order of business was to plot these coordinates onto an actual map of Dublin to check for accuracy of results. By downloading an open text map from a website, we move ahead with the process for this, and the steps followed are as listed below.

1. Importing the csv file previously exported by the SQLContext's read.csv command as a data frame.
2. Typecasting the Latitude and longitude values in this data frame from string type to double type.
3. Extracting min and max values for both latitude and longitude to establish a boundary box on the map.
4. Using matplotlib library to create a plot of the coordinates on the map.

After export the plot and comparing it to real world data, the data generation authorities can be certain that the data is verified.

This procedure, although not exactly an analysis algorithm and more of an explanatory method, gave a small insight into the data and how it can be utilised to replicate the real life scenario. The credibility of the data-set has also been established and we move on to the next analytical algorithms.

4.4 Busy Lines and Congestion

This analytical algorithm in our project life-cycle gives us an idea of what lines present in the bus network experience the most number of stops to make in a day and which of these buses face congestion on the roads and at what times. This bit of information will be useful in identifying which lines require constant monitoring and maintenance by the authorities so as to make the experience for the passengers a smoother one. Also having an idea of a generalised pattern of congestion faced, be it because of rush hours or other reasons, the vehicle can be prepared for alternative routes during these times.

The Busy lines identification algorithm is depicted by the following steps -

1. Importing and cleaning our data set using the previously discussed pre-processing and cleaning methods.
2. Creating an empty data frame with a user defined schema. The schema has four columns - 'LineID' of integer type, 'Number of Times at Stops' of integer type, 'Date' of string type, 'Day' of string type.
3. Since we are iterating through day wise data in this scenario, push each set of records through a function which extracts the day and date on which those set of records were generated.

4. Remapping the rdd into a key-value pair rdd with LineID as the key.
5. Grouping the dataset using the `reduceByKey()` function so that we have groups of data for different LineIDs.
6. Iterating through each LineID's data and doing the same grouping with StopID as key.
7. Iterating through each StopID's data and extracting records in which a bus stopped at any stop and in the next immediate record left from that stop.
8. Adding up the count of records for this scenario within each LineID
9. Converting that particular LineID, counts of stops, Day and Date into a data frame with the same schema as the empty data frame we created before.
10. Exporting that dataframe as a csv file for better readability and universal understanding.

The pseudo code for this algorithm is as depicted below.

```
Importing SparkConf, SparkContext and SQLContext from pyspark library
Importing types from pyspark.sql library
Importing time library
```

```
Initialising the Spark environment variables
```

```
Create pre-processing function as described before
```

```
Create cleaning function as described before
```

```
function date_and_day(data frame) {
extract first timestamp value from the data frame
```

```
convert that timestamp value into a readable one using
the time package
```

```
return day and date as separate values
}
```

```
Create an empty data frame with a explicit schema
```

```
Importing the data frame and pre-process it through the function created above
```

```
Putting this dataset through the cleaning function created above
```

Putting this dataset through the day and date function created above

Converting rdd into pair-wise rdd with LineID as key

Using reduceByKey to group data by LineID

Iterating through each LineID and then parallelizing this dataset into an rdd

Converting this rdd into a pair-wise rdd with StopID as key

Using reduceByKey to group data by StopID

Iterating through each StopID and then converting that dataset into a dataframe

Registering that dataframe as a temp table using registerTempTable command

Using SparkSQL command to extract our dedicated records.

```
with temp as (select row_number()over(order by Timestamp ASC)
as row, * from records)
```

```
select t2.Delay from temp t1 INNER JOIN temp t2
t1.row = t2.row+1 where t2.AtStop = 1 and t1.AtStop = 0"
```

Creating a dataframe row of our results for this LineID

Union to the empty data frame

Coalesce and write the resultant dataframe as a csv

The resultant csv has the following type of result.

Table 4.2: Busy Lines Result

1	660	Jan-01	Tue
4	540	Jan-01	Tue
7	1742	Jan-01	Tue
9	1553	Jan-01	Tue
11	826	Jan-01	Tue
13	1781	Jan-01	Tue
14	1270	Jan-01	Tue
15	3345	Jan-01	Tue
16	2341	Jan-01	Tue
17	122	Jan-01	Tue

The first column is the Line ID with the second column being the number of stops

made by buses running within that Line ID in a day. The date and day are the third and fourth columns respectively. This result is only the first 10 lines of Jan 1. Let's have a look at a graphical visualization of the busy lines on Jan 1.

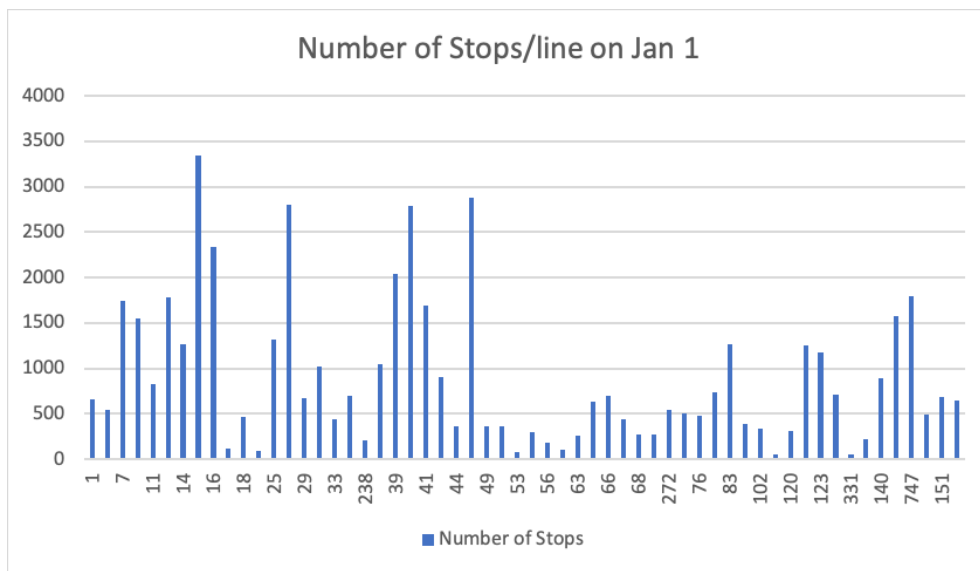


Figure 4.3: Number of stops per line on January 1

As seen above, some lines make more number of stops in a day as compared to the rest. This result can be utilised to analyse that which lines are having more load and are more susceptible to delays if some mitigation methods are not employed. These methods can be something as follows -

1. Increasing number of vehicles on the line.
2. Reducing the frequency of stops made by each vehicle at some stops where the passenger count is low.

This will go a long way in assisting management of the already loaded few lines in the city. Talking about a similar situation that is faced by these vehicles on the daily basis is the congestion factor.

The algorithm used to calculate the peak times of congestion occurring on the roads is as follows.

1. Importing and cleaning our data set using the previously discussed pre-processing and cleaning methods.
2. Creating an empty data frame with a user defined schema. The schema has four columns - 'LineID' of integer type, 'Congestion' of integer type, 'Date' of string type, 'Day' of string type.

3. Since we are iterating through day wise data in this scenario, push each set of records through a function which extracts the day and date on which those set of records were generated.
4. Remapping the rdd into a key-value pair rdd with LineID as the key.
5. Grouping the dataset using the reduceByKey() function so that we have groups of data for different LineIDs.
6. Iterating through each LineID's data and converting that dataset into a dataframe and using SparkSQL to get those timestamp in which the lineID faces congestion
7. Converting that result for the LineID into a data frame and unionising that into the initially created empty data frame. Exporting the result into a csv file.

The pseudo code for this above described algorithm will be as follows.

```
Importing SparkConf, SparkContext and SQLContext from pyspark library
Importing types from pyspark.sql library
Importing time library
Importing lit from pyspark.sql.functions library
```

```
Initialising the Spark environment variables
```

```
Create pre-processing function as described before
```

```
Create cleaning function as described before
```

```
Create date_and_day function as described before
```

```
Create an empty data frame with an explicit schema
```

```
Importing the data frame and pre-process it through the function created above
```

```
Putting this dataset through the cleaning function created above
```

```
Putting this dataset through the day and date function created above
```

```
Converting rdd into pair-wise rdd with LineID as key
```

```
Using reduceByKey to group data by LineID
```

```
Iterating through each LineID and then parallelizing this dataset into an rdd
```

```
Converting this rdd into a data frame and registering it as a temporary table
using registerTempTable function
```

Using SparkSQL command to extract our dedicated records.

```
select MIN(LineID) as LineID, Timestamp from records where Congestion = 1
and AtStop = 0 group by Timestamp order by Timestamp
```

Creating a dataframe row of our results for this LineID

Union to the empty data frame

Coalesce and write the resultant dataframe as a csv

The resultant csv will look something like the following.

Table 4.3: Congestion Result

1	1.357E+15	Jan-01	Tue
1	1.3571E+15	Jan-01	Tue
1	1.3571E+15	Jan-01	Tue
1	1.3571E+15	Jan-01	Tue
4	1.357E+15	Jan-01	Tue
4	1.357E+15	Jan-01	Tue
4	1.3571E+15	Jan-01	Tue
4	1.3571E+15	Jan-01	Tue
4	1.3571E+15	Jan-01	Tue
4	1.3571E+15	Jan-01	Tue
7	1.357E+15	Jan-01	Tue
7	1.357E+15	Jan-01	Tue

The first column is the Line ID and the second column is the set of timestamps where that LineID faced congestion. This result is just an initial part of the actual result. Following is a simple visualisation which tells us which LineID faces more number of congestion timestamps on Jan 1.

As evident from the figure 4.4, we clearly cannot extract any relevant information, as there are a number of line IDs facing congestion for same number of times in a day. It would be better if we can identify if there is a specific timestamp which occurs the most.

The figure 4.5 gives a clearer picture. It tells us the number of lineIDs which face congestion on the six timestamps through the day of Jan 1 on which congestion occurs somewhere in Dublin. Except for these 6 timestamps there is no congestion.

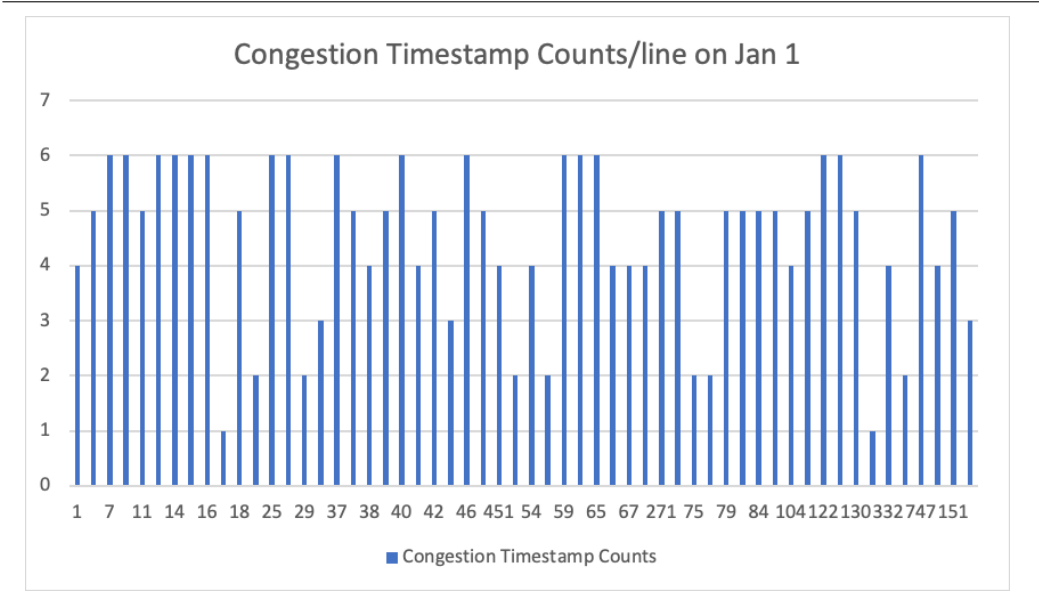


Figure 4.4: Congestion timestamp counts/line on January 1

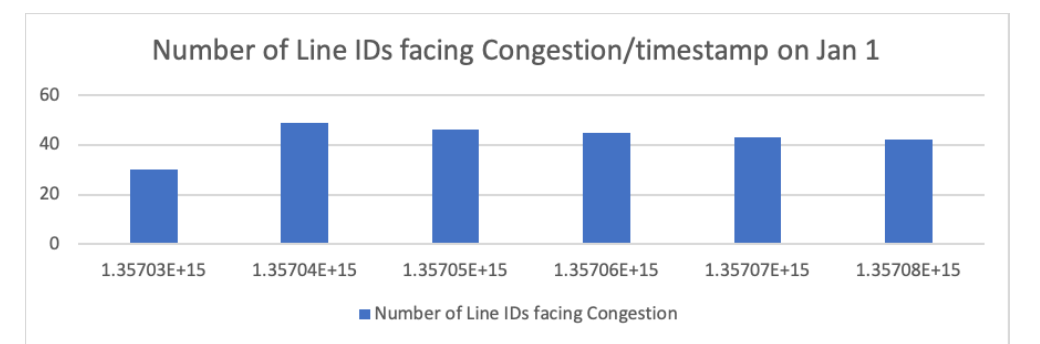


Figure 4.5: Number of Lines facing congestion of specific timestamps on Jan 1

These two above algorithms gave a clearer insight into how the authorities can take action from the logistics perspective. The bus operators that take part in this Dublin bus operations also play a vital role in shaping up the performance of this network in the city. In the next section we focus a bit on how the operators are an integral part in this network.

4.5 Operator Influence and Latency

The bus operators participating in the Dublin bus network have a vital role to play in the overall performance of the transportation in the city. 8 operators are a part of this network, as identified before in the preliminary analysis. These operators have abbreviations assigned to them in the data set and distinguished solely on this basis, with the next level division coming in the vehicle IDs assigned to the buses that are deployed by them individually on the streets.

The number of vehicles owned by an operator and taking rounds on a particular lineID can be a useful metric to extract if the amount of delay faced by an operator is projected side by side. This will lead to the operator taking crucial decisions in the number of vehicles it has in the network, and whether they need to modify that number a bit with respect to the current situations.

The two terms in the title of this section mean exactly the two things mentioned in the previous paragraph - influence signifies the number of vehicles of an operator in the network, while latency defines the average delay of an operator. Influence will be an important metric in defining the market share of the operator and how the Dublin bus authority will do a cost benefit analysis in managing their role in the city. Latency is what they will be measured against.

The algorithm for measuring influence is as follows.

1. Importing and cleaning our data set using the previously discussed pre-processing and cleaning methods.
2. Using groupBy operator on the total records data frame's 'Operator' column with an aggregate function of countDistinct on 'Vehicle IDs'
3. This will give us the number of distinct vehicle IDs within each operator
4. Exporting this result set into a csv file of better readability and universal understanding.

The pseudo code for this algorithm is as below.

Importing SparkConf, SparkContext and SQLContext from pyspark library

```
Importing types from pyspark.sql library
Importing countDistinct from pyspark.sql.functions library

Initialising the Spark environment variables

Create pre-processing function as described before

Create cleaning function as described before

Importing the data frame and pre-process it through
the function created above

Putting this dataset through the cleaning function
created above

Using groupBy operator on the obtained clean data frame
with respect to 'Operator' column. Applying an aggregate
method of 'countDistinct' with respect to 'VehicleID' column

operator_influence_df = records_df.groupBy("Operator").\
agg(countDistinct("VehicleID"))

Coalesce and write the resultant dataframe as a csv
```

The resultant csv will look something like the following.

This tells us the number of vehicles operating for each operator company. The following chart will give us a better idea on this data. As evident from the pie chart in figure 4.6, more than half the vehicles operating in the Dublin Bus network belong to PO, HN and D1 operator companies. Since the influence they have on the transportation market in Dublin is more as compared to others, the actions performed by these operators to keep their operations running smooth will be essential towards maintaining a certain standard in the market.

Next we look at the average latency these operators have in their operations in the network. The algorithm for that is as follows.

1. Importing and cleaning our data set using the previously discussed pre-processing and cleaning methods.
2. Typecasting Delay column from a string data type to double data type
3. Using groupBy operator on the total records data frame's 'Operator' column with an average function on 'Delay' column.

Table 4.4: Number of vehicles per Operator

SL	49
D1	83
HN	71
PO	63
RD	48
D2	1
CF	39
CD	53

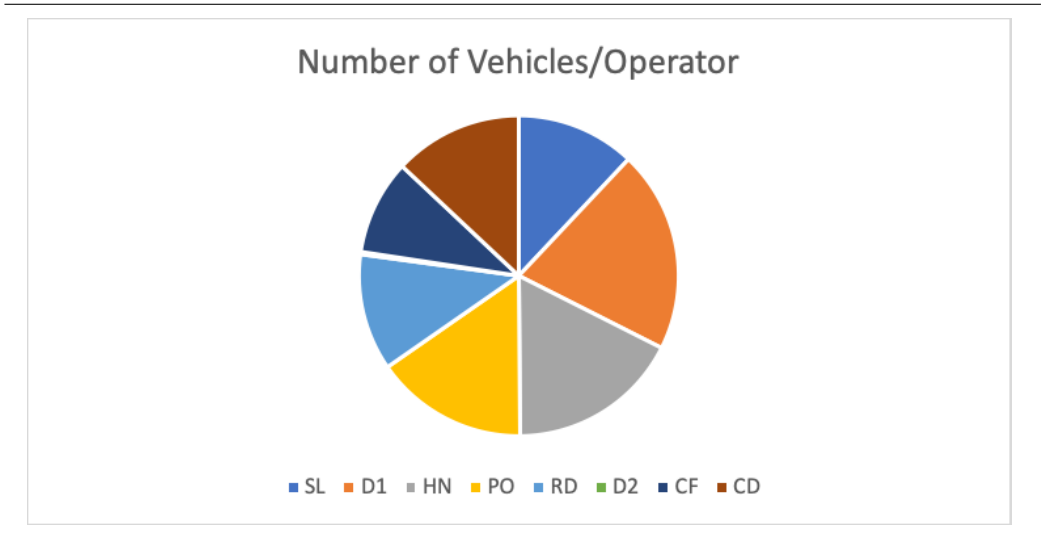


Figure 4.6: Number of vehicles per Operator

4. This will give us the average delay value within each operator.
5. Exporting this result set into a csv file of better readability and universal understanding.

The pseudo code for this algorithm is as follows.

```
Importing SparkConf, SparkContext and SQLContext from pyspark library
Importing types from pyspark.sql library
Importing DoubleType from pyspark.sql.types library
```

Initialising the Spark environment variables

Create pre-processing function as described before

Create cleaning function as described before

Importing the data frame and pre-process it through the function created above

Putting this dataset through the cleaning function created above

Typecasting the Delay column into a Double data type using
the cast(DoubleType()) method

Using groupBy operator on the obtained clean data frame
with respect to 'Operator' column. Applying an average
method onto the 'Delay' column

```
operator_latency_df = records_df.groupBy("Operator").avg("Delay")
```

Coalesce and write the resultant dataframe as a csv

The resultant csv file is displayed in table 4.5.

The Delay value with a negative sign denotes that bus generally run ahead of schedule. This is a good sign in the transportation network. Let's take a visual look at this result.

This result is just for January 1. We will get a more comprehensive insight into the data if we look at more number of days.

This analysis has given us yet another perspective of a descriptive analytics applied onto the dataset at hand. The detailed information being obtained can be utilised in numerous ways to enhance the services of the network.

Table 4.5: Average Delay/Operator on Jan 1

SL	-162.31024
D1	-183.91302
HN	-248.7742
PO	-195.63757
RD	-208.06848
D2	-72.098837
CF	-159.40135
CD	-141.41706

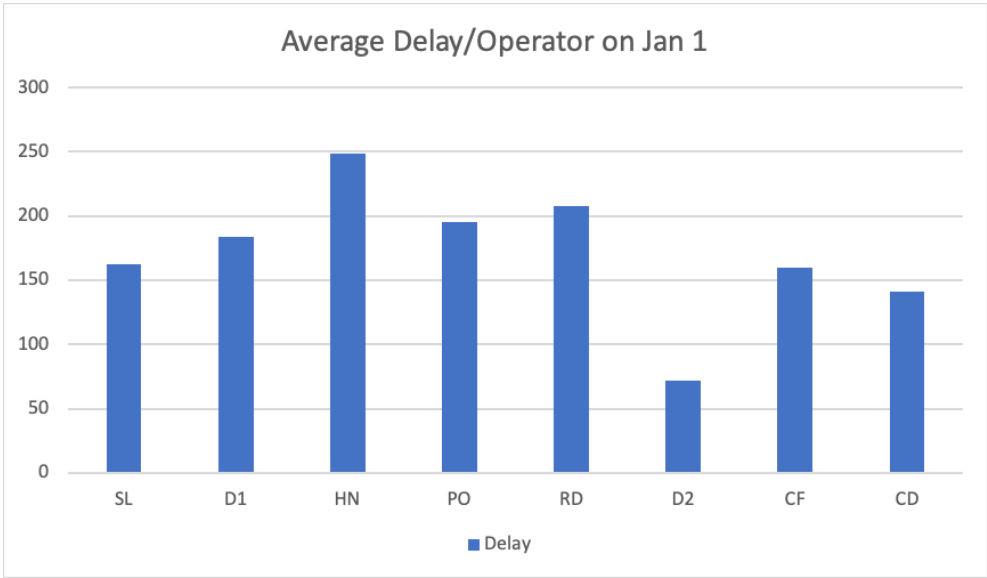


Figure 4.7: Average Delay/Operator on Jan 1

Next we look at the analysis of the delay column in a bit more detail because it is the most essential feature when we talk about the performance of a transportation medium.

4.6 Average Delays

The average delays experienced by separate line IDs is a direct metric to analyse their performance. For each record spitted out by the data collection system, there is a delay value attached to it, which signifies the delay with which that current bus is running. A negative value means that the bus is running ahead of schedule and vice versa for a positive value.

The steps for this algorithm are as follows.

1. Importing and cleaning our data set using the previously discussed pre-processing and cleaning methods.
2. Creating an empty data frame with a defined schema. The schema consists of four columns - 'LineID' of Integer type, 'Average Delay' of Integer type, 'Date' of String type, 'Day' of String type.
3. Since we are iterating through day wise data in this scenario, push each set of records through a function which extracts the day and date on which those set of records were generated.
4. Remapping the rdd into a key-value pair rdd with LineID as the key.
5. Grouping the dataset using the reduceByKey() function so that we have groups of data for different LineIDs.
6. Iterating through each LineID's data and converting each data set into a data frame.
7. Extracting average of Delay column through a user defined function.
8. Unionising that record result to the empty data frame created before.

The pseudo code for this algorithm is as follows

```
Importing SparkConf, SparkContext and SQLContext from pyspark library
Importing types from pyspark.sql library
Importing DoubleType from pyspark.sql.types library
```

```
Initialising the Spark environment variables
```

```
Create pre-processing function as described before
```

Create cleaning function as described before

```
Function average_of_column(data frame, column)
{
  Get total of delay column for the input data frame

  Get number of records count

  Calculate average by dividing total by records count

  Return average value
}
```

Create an empty data frame with an explicit schema

Importing the data frame for 7 days and pre-process it through the function created above

Putting this dataset through the cleaning function created above

Putting this dataset through the day and date function created above

Converting rdd into pair-wise rdd with LineID as key through the map function

Using reduceByKey to group data by LineID

Iterating through each LineID and then converting that data set into a data frame using the toDF function

Putting that data frame through the average_of_column function created above, with 'Delay' as the column argument

Union to the empty data frame

Coalesce and write result to a csv

The resultant csv looks something as illustrated in table 4.6. To get a better understanding of the result data we just look at one particular LineID, for instance '747' and its

graphical plot below.

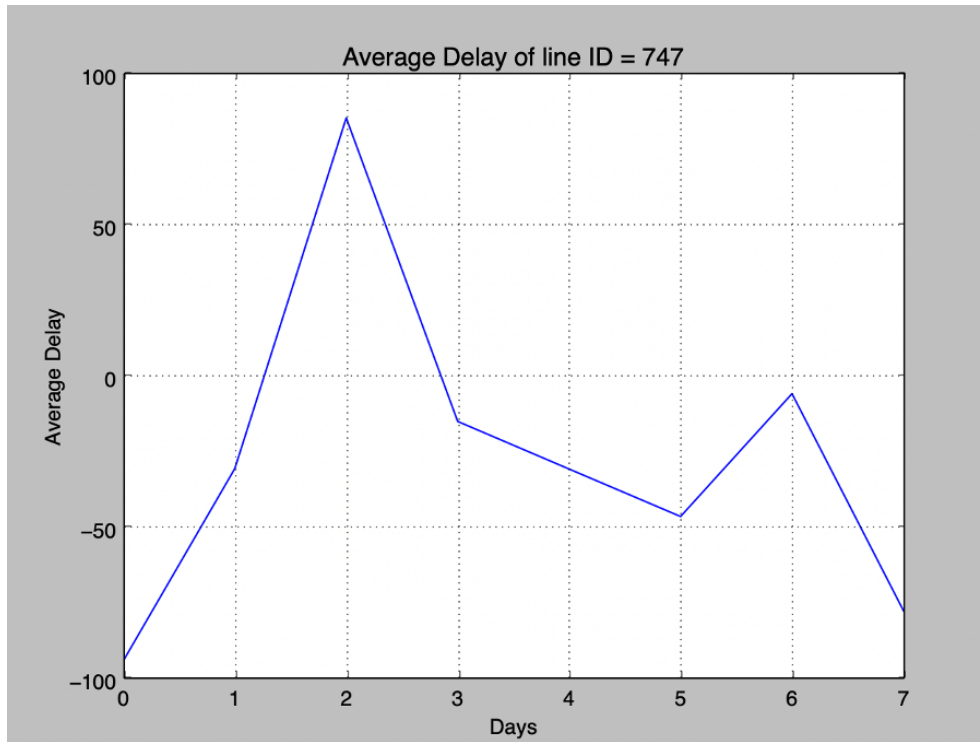


Figure 4.8: Average Delay of Line 747

The graph in figure 4.8 clearly depicts that there is an increased delay in third data point, which is a Thursday. For obvious space constraints, we could not include this detail for all other lineID. This illustration can be very beneficial in shuffling around the available resources in the form of number of vehicle that are dedicated to a particular line.

However, this data is a bit towards the generalised outlook. Getting an idea of the average delay with respect to the stops lying on the lines would furnish an even better insight into the data.

4.7 Average Reaching Delay at Stops

If we talk about the average delay with which buses reach a particular stop, that metric will be very important from the passenger's perspective. This will enable proper decision making for the people to reach the bus stops and also be a prediction feature against the real time bus arrival timings on a tracking platform.

Table 4.6: Average Delay of Line 747

747	-93.953652	Jan-01	Tue
747	-30.419853	Jan-02	Wed
747	85.5374886	Jan-03	Thu
747	-14.815726	Jan-04	Fri
747	-30.571386	Jan-05	Sat
747	-46.267576	Jan-06	Sun
747	-5.6498662	Jan-07	Mon
747	-77.403945	Jan-08	Tue

The algorithm for obtaining these values can be summarised by the steps below.

1. Importing and cleaning our data set using the previously discussed pre-processing and cleaning methods.
2. Creating an empty data frame with a user defined schema. The schema has four columns â 'LineID' of integer type, 'StopID' of integer type, 'Average Reaching Delay' of Integer type, 'Date' of string type, 'Day' of string type.
3. Since we are iterating through day wise data in this scenario, push each set of records through a function which extracts the day and date on which those set of records were generated.
4. Remapping the rdd into a key-value pair rdd with LineID as the key.
5. Grouping the dataset using the reduceByKey() function so that we have groups of data for different LineIDs.
6. Iterating through each LineID's data and doing the same grouping with StopID as key.
7. Iterating through each StopID's data and extracting those records where the AtStop value changes from 0 to 1 in two consecutive records with the help of following SparkSQL query.
8. Putting this obtained filtered data frame into the averageofcolumn function discussed in the previous algorithm and getting the average reaching delay value.
9. Converting this result row into a dataframe.
10. Unionising it to the empty data frame created above.
11. Coalesce and export this result set into a csv file.

The pseudo code for the above described algorithm will be as follows.

Importing SparkConf, SparkContext and SQLContext from pyspark library
Importing types from pyspark.sql library
Importing time library
Importing funtions from pyspark.sql library

Initialising the Spark environment variables

Create pre-processing function as described before

Create cleaning function as described before

Create date_and_day function as described before

Create average_of_column function as described before

Create an empty data frame with an explicit schema

Importing the dataset and putting it through the pre-processing function created above

Putting this dataset into the cleaning function created above

Putting this dataset through the day and date function created above

Converting rdd into pair-wise rdd with LineID as key

Using reduceByKey to group data by LineID

Iterating through each LineID and then parallelizing this dataset into an rdd

Converting this rdd into a pair-wise rdd with StopID as key

Using reduceByKey to group data by StopID

Iterating through each StopID and then converting that dataset into a dataframe

Registering that dataframe as a temp table using registerTempTable command

Using SparkSQL command to extract our dedicated records.


```

with temp as
(
select row_number()over(order by Timestamp ASC) as row, from records
)
select t2.LineID, t2.StopID, t2.Timestamp, t2.AtStop, t2.Delay
from temp t1
INNER JOIN temp t2
ON t1.row = t2.row+1
where t2.AtStop = 1 and t1.AtStop = 0

```

Putting that data frame through the `average_of_column` function created above, with `Delay` as the column argument

Union to the empty data frame

Coalesce and write result to a csv

The resultant csv will look something as mentioned in table 4.7

Table 4.7: Average Reaching Delay on stops for 747 line

747	7456	-105	Jan-01	Tue
747	620	-65	Jan-01	Tue
747	7400	36	Jan-01	Tue
747	7468	-55	Jan-01	Tue
747	7404	-182	Jan-01	Tue

The result is just for LineID 747 on Jan 1. If we analyse this data and look at the highest average reaching delay value then it can be deduced that there is an investigation opportunity near those stops which are causing this increased delay.

As seen in the figure 4.9, the stop 2002 experiences the maximum delay of nearly 75 milliseconds on an average. By utilising this same algorithm for more lines and more number of days, we can make an even more comprehensive analysis. This marks the end of descriptive analysis. Next we move towards the predictive module for this.

4.8 Time Series Analysis

The data at hand has been collected on the order of consecutive timestamps, with multiple features consisting of varying values. Since the repository is only of a month however, the prediction may not be that accurate. For a proper delay analysis and specially in

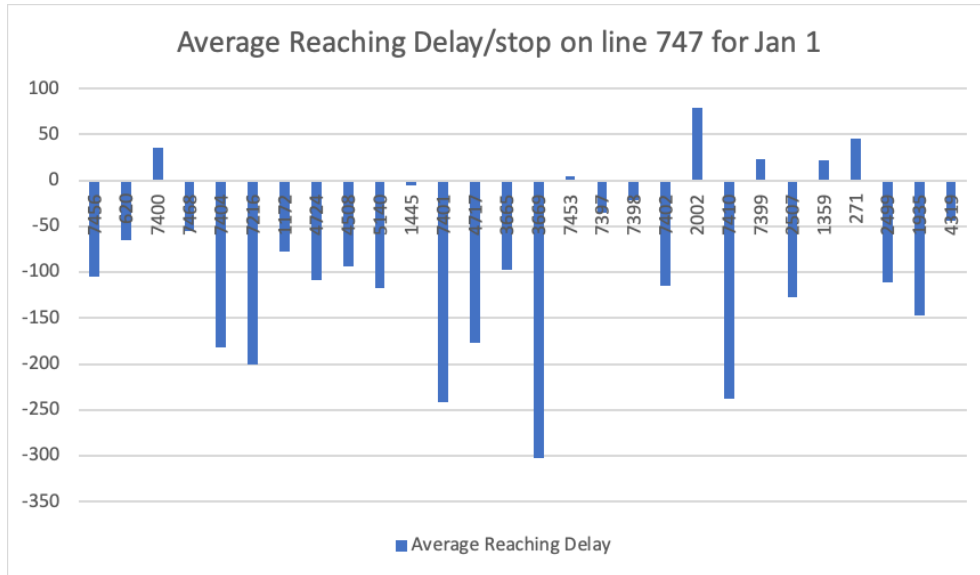


Figure 4.9: Average Reaching Delay on stops of Line 747

the field of public transportation, the seasonality plays an important role, because devoid of that only the pattern of lesser delays on the weekends and lowered frequency of buses will exist. But nevertheless, a time series analysis can be performed to check whether prediction using this technique can be fruitful with the data-set possessed by us.

The basic features of a time series analysis are the time ordered data and the feature we want to work with in that scenario. Through the Spark algorithms we have worked with in the previous pages, the accumulation of Average Delays for the various lines in the Dublin Bus network can be a prime variable of interest. Since working with all of them and depicting their results won't be a possibility in this report, the main focus will be on the bus line with ID 747.

The algorithm for the time series analysis is as follows -

1. Importing the csv file we created before which consists of our relevant data related to the line 747 and putting it through the pre-processing function as discussed before.
2. Converting the obtained spark data frame into a pandas data frame
3. Converting the timestamp column in it to a datetime type and only retaining the date part of it, and converting the average delay column from string type to numeric type for its compatibility with the functions.
4. Dropping LineID column as its value will only be 747 throughout and since it's a

known parameter, there is no relevance of it being in the data frame as a separate entity.

5. Setting an index on the Date column.
6. Plotting that indexed data set to analyse the stationarity in the dataset.

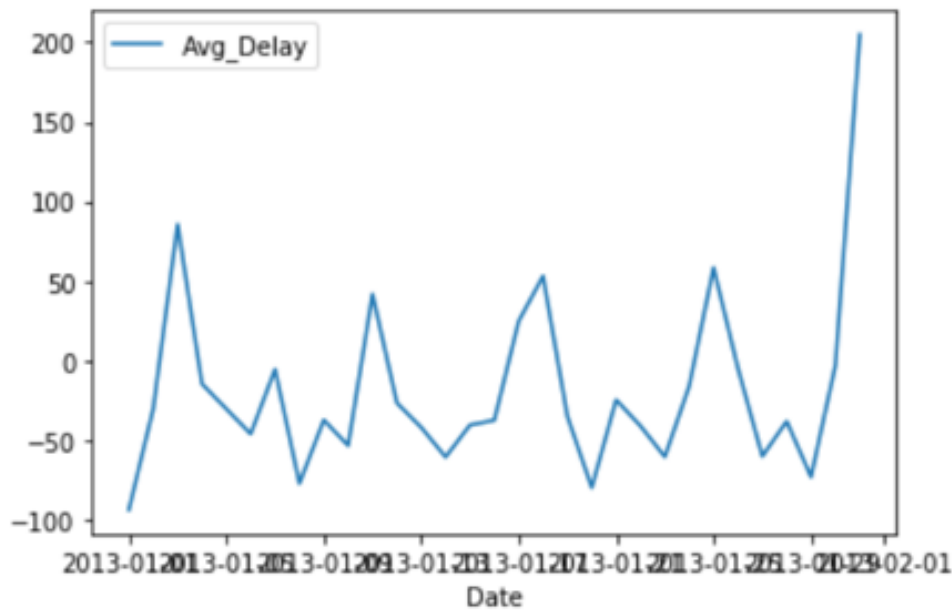


Figure 4.10: Checking stationarity

7. As evident from the graph above, there is no stationarity and also there seems to be an absence of a strong enough trend to perform similar analysis. We need to check the rolling statistics like mean and standard deviations for a better of the data.
8. These rolling statistic lines are much smoother than the original curve itself. We now move ahead with the series analysis and make a base model against which to compare our time series model of ARIMA.
9. Calculating the root mean squared error of the base model
10. Plotting the acf graph to get an estimate on the probable values of the $\hat{\rho}$ parameter in the ARIMA
11. As seen in figure 4.12, only 0th point is above the blue area which suggests that the value of p should be 0. Next we plot the pacf plot to similarly get an estimate of the ' q ' parameter in the ARIMA

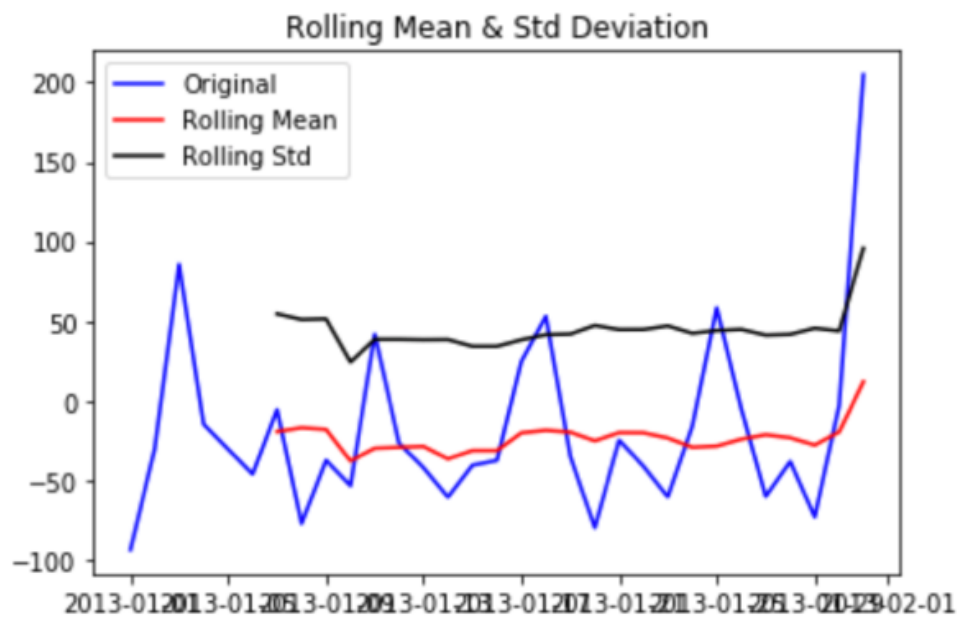


Figure 4.11: Rolling mean standard deviation

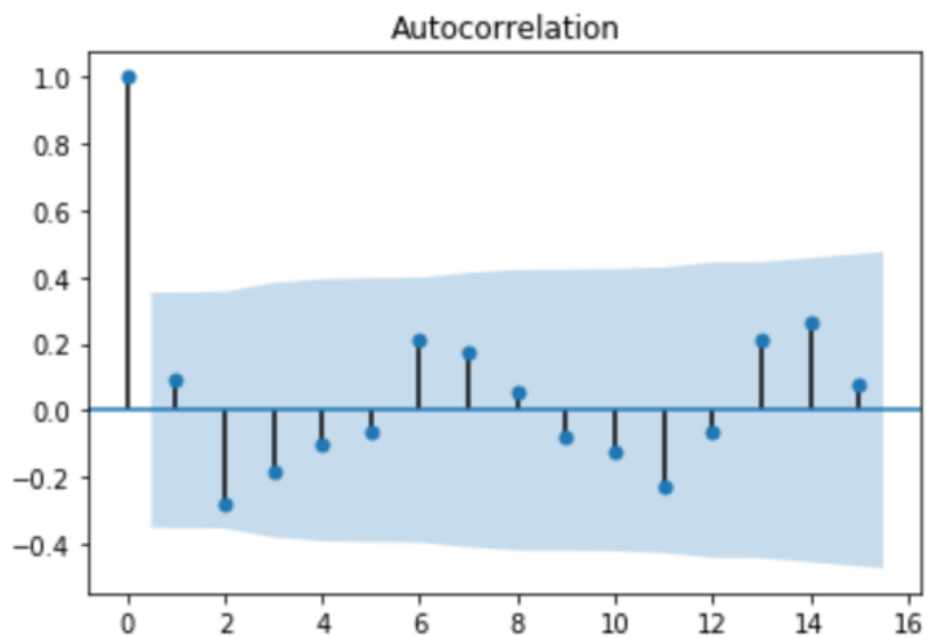


Figure 4.12: Auto-correlation graph for inferring value of 'p'

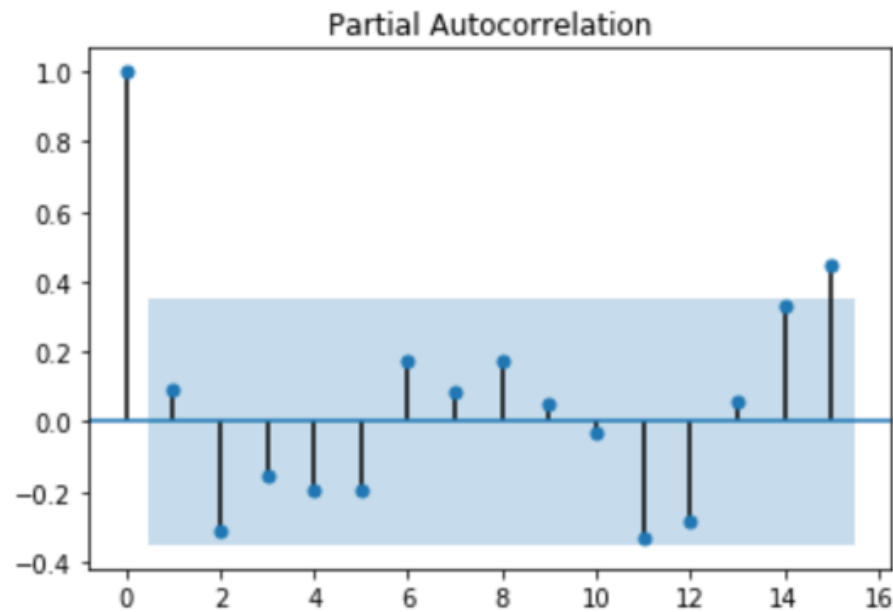


Figure 4.13: Partial Auto-correlation graph for inferring value of 'q'

12. As seen in the figure 4.13, similar to the acf graph before, here also only the 0th point is above the blue area which suggests that the value of 1 should be 0.
13. The d value of difference in ARIMA can vary and we would try out multiple values between 0 and 2 to find out that ideal parameter value.
14. Next we split the data into a training set and testing sets.
15. Then we move on to the application steps of ARIMA model and check the aic value and rmse value
16. Next we tune the p, q and d values by trying out various value sets for them in a specified range
17. Lastly, we apply multiple models and check which has the lowest rmse

The Pseudo code for this analysis is as follows.

```

Importing SparkConf, SparkContext and SQLContext from pyspark library
Importing types from pyspark.sql library
Importing pandas library
Importing numpy library
Importing matplotlib.pyplot library

```

```
Importing mean_squared_error from sklearn.metrics library
Importing plot_acf and plot_pacf from statsmodels.graphics.tsaplots library
Importing ARIMA from statsmodels.tsa.arima_model library
```

Initialising the Spark environment variables

Create pre-processing function as described before
Reading in the data and putting it through the pre-processing
function created above

Converting the spark data frame into a pandas one with
toPandas() function

Converting 'Date' column into datetime type with
to_datetime() function and retaining only date part
with .date function

Converting 'Avg_Delay' column into numeric type with
to_numeric() function

Setting index on 'Date' column using set_index() function

Plotting the indexed dataset using plot() method

Determining the rolling mean and standard deviation
using rolling(window=7) method. Window of 7 signifies
mean and standard deviation over a rolling set of 7 days.

Plotting these statistics over the previous graph for comparison

Making the base model \hat{a} by a shift of 1 day to compare
the current day's delay value with the previous one
as its forecast

Check the mse using mean_squared_error() over the
two comparable columns of Actual delay and forecasted
delay we just made

Rooting the mse value to bring it to the scale of
actual delay values using numpy's sqrt function

Putting the indexed dataset through the plot_acf
and plot_pacf function for checking values of p

and q respectively

Apply the ARIMA model using ARIMA() function and order of (0,1,0) standing for (p,d,q)

Check the rmse of the fitted model

Parameter tuning by running the above two steps in a loop of modifying range of p, q and values

The rmse in the base model made initially in this algorithm was 66.83 approximately. The application of ARIMA model in this had a purpose to further reduce that rmse. However, upon obtaining the results, the rmse value only increased. The best performing ARIMA model was with the parameter (4,1,2) and that too had a rmse value of 81.51

```
ARIMA(1, 0, 2) RMSE = 84.86
ARIMA(1, 0, 3) RMSE = 91.71
ARIMA(1, 0, 4) RMSE = 89.03
ARIMA(1, 1, 0) RMSE = 88.85
ARIMA(1, 1, 1) RMSE = 86.48
ARIMA(1, 1, 2) RMSE = 87.20
ARIMA(1, 1, 3) RMSE = 84.95
ARIMA(1, 2, 0) RMSE = 84.75
ARIMA(1, 2, 1) RMSE = 95.53
ARIMA(1, 2, 4) RMSE = 108.09
ARIMA(2, 0, 0) RMSE = 86.15
ARIMA(2, 0, 1) RMSE = 85.12
ARIMA(2, 0, 2) RMSE = 81.52
ARIMA(2, 0, 3) RMSE = 81.75
ARIMA(2, 0, 4) RMSE = 82.01
ARIMA(2, 1, 0) RMSE = 97.19
ARIMA(2, 1, 1) RMSE = 87.19
ARIMA(2, 1, 2) RMSE = 86.77
ARIMA(2, 1, 3) RMSE = 89.41
ARIMA(2, 1, 4) RMSE = 98.05
ARIMA(2, 2, 0) RMSE = 88.93
ARIMA(2, 2, 1) RMSE = 102.36
ARIMA(2, 2, 4) RMSE = 91.13
ARIMA(3, 0, 0) RMSE = 81.52
ARIMA(3, 0, 1) RMSE = 86.16
ARIMA(3, 0, 2) RMSE = 85.55
ARIMA(3, 0, 3) RMSE = 87.22
ARIMA(3, 0, 4) RMSE = 90.87
```

ARIMA(3, 1, 0) RMSE = 89.36
ARIMA(3, 1, 1) RMSE = 81.54
ARIMA(3, 1, 2) RMSE = 87.32
ARIMA(3, 1, 4) RMSE = 85.28
ARIMA(3, 2, 0) RMSE = 140.51
ARIMA(3, 2, 1) RMSE = 148.10
ARIMA(3, 2, 2) RMSE = 103.41
ARIMA(3, 2, 3) RMSE = 113.92
ARIMA(4, 0, 0) RMSE = 81.53
ARIMA(4, 0, 1) RMSE = 84.22
ARIMA(4, 0, 2) RMSE = 85.87
ARIMA(4, 0, 3) RMSE = 86.67
ARIMA(4, 0, 4) RMSE = 87.13
ARIMA(4, 1, 0) RMSE = 92.95
ARIMA(4, 1, 1) RMSE = 81.53
ARIMA(4, 1, 2) RMSE = 81.51
ARIMA(4, 1, 4) RMSE = 93.60
ARIMA(4, 2, 0) RMSE = 144.49
ARIMA(4, 2, 1) RMSE = 107.02
ARIMA(4, 2, 2) RMSE = 93.72

This leads to a few conclusions -

1. The results obtained from above suggest that the naive base model was better than ARIMA.
2. Hence, autoregressive model might not be the way to go in this.
3. And transformation of data points into logarithmic form is also not the way to go because of the negative nature of the average delay values in certain cases.
4. There might not be that strong a trend to perform a time series analysis in this data.

Since time series analysis was not able to provide us with a proper defined method of performing predictions, we need to fall back on the classic regression techniques in a hope of finding better ways of doing predictive analysis in this dataset. The other question that starts to come to mind is that whether predictive training might be best practice in this limited data, the ideal way to go or not?

4.9 Multiple Linear Regression

By obtaining the numerous metric values from the previous big data analysis, we can cumulate that data in a structured manner and train a couple of models to do an analysis

on the delay value.

Since this is a discrete value scenario, we can create a regression model. Random forest regression might be the best model to train in this data, however comparing this to a multiple linear regression model would also be a good approach.

Cumulation and transformation of the data is done in the following manner.

1. Collection of features from the Busy lines algorithm - the number of stops made by a particular line in a day
2. Collection of features from the Congestion algorithm - the number of times the line faced a congestion on the streets.
3. Collection of Average Delay values over the entire month data per line.

The main line of interest in this algorithm is 747. The similar exploration in this analysis for other lines can be done by just changing the lineID in the algorithm to obtain values for the same.

The algorithm can be described as steps below -

1. Importing the csv file we created before which consists of our relevant data related to the line 747 and putting it through the pre-processing function as discussed before.
2. Converting the obtained spark data frame into a pandas data frame.
3. Converting the timestamp column in it to a datetime type with a new column name of 'Date' and only retaining the date part of it.
4. Dropping the previous Timestamp column
5. Remodelling the data to extract the x variable as an array of two features of 'numberofstops' and 'numberofcongestions' and y variable of Average Delay.
6. Splitting the dataset into training and testing set.
7. Applying the multiple regression model onto the training set.
8. Predicting through this model onto the x testing data.
9. Calculating the mean squared error and rmse of the prediction.

The pseudo code for this algorithm is as follows -

```
Importing SparkConf, SparkContext and SQLContext from pyspark library
Importing types from pyspark.sql library
Importing pandas library
```

```
Importing numpy library
Importing LinearRegression from sklearn.linear_model library
Importing train_test_split from sklearn.model_selection library
Importing mean_squared_error from sklearn.metrics library
```

Initialising the Spark environment variables

Create pre-processing function as described before

Importing and pre-processing our dataset through the function created above

Converting the spark data frame into a pandas one with toPandas() function

Converting 'Date' column into datetime type with to_datetime() function and retaining only date part with .date function

Remodelling the data to have 2nd and 3rd column of each row through iterrows() function and append that as a paired list to a separate list

Converting this into a x vector of numpy arrays of float64 type

Converting 'Avg_Delay' of data frame into y vector of numpy array of float64 type

Splitting into test and train sets using train_test_split of sklearn.model_selection library

Training the LinearRegression() on train data

Checking rmse through np.sqrt over mean_squared_error() on test and predicted y values

The rmse comes out to approximately 64 in this case, which is not considered good in any circumstance. The accuracy of less than 50% in the model is an outright indication that there is no linearity in the data. We may have to consider a different tree based regression algorithm to get better performance in predictions.

4.10 Random Forest Regression

As seen previously the performance of the Multiple Regression model was not up to the mark and the introduction of Random Forest Regression algorithm in the case is an attempt to further improve the accuracy in prediction. If the accuracy rises, we can consider that model to be a better indicator of the data to come and take that as the standard in usage as a prediction technique over this Dublin bus data.

Similar to before, this algorithm has also only been performed on to the line ID 747 to maintain consistency in this dissertation report. If the need arises to analyse any other line, the Id can be changed in the code and aligned results can be obtained.

The algorithm used can be broken down in the following steps -

1. Importing the csv file we created before which consists of our relevant data related to the line 747 and putting it through the pre-processing function as discussed before.
2. Converting the obtained spark data frame into a pandas data frame.
3. Converting the timestamp column in it to a datetime type with a new column name of 'Date' and only retaining the date part of it.
4. Dropping the previous Timestamp column
5. Remodelling the data to extract the x variable as an array of two features of 'numberofstops' and 'numberofcongestions' and y variable of Average Delay.
6. Splitting the data-set into training and testing set.
7. Applying the Random forest algorithm and fitting it onto the train data.
8. Predicting the y values using this model on the test set of x vector.
9. Checking mean squared error over the predicted values of y and the test set of y values.
10. Checking the rmse of that error by square rooting the mean squared value.

The pseudo code of this algorithm is as follows -

```
Importing SparkConf, SparkContext and SQLContext from pyspark library
Importing types from pyspark.sql library
Importing pandas library
Importing numpy library
Importing RandomForestRegressor from sklearn.ensemble library
Importing train_test_split from sklearn.model_selection library
Importing mean_squared_error from sklearn.metrics library
```

Initialising the Spark environment variables

Create pre-processing function as described before

Importing and pre-processing our dataset through the function created above

Converting the spark data frame into a pandas one with `toPandas()` function

Converting 'Date' column into datetime type with `to_datetime()` function and retaining only date part with `.date` function

Remodelling the data to have 2nd and 3rd column of each row through `iterrows()` function and append that as a paired list to a separate list

Converting this into a x vector of numpy arrays of float64 type

Converting 'Avg_Delay' of data frame into y vector of numpy array of float64 type

Splitting into test and train sets using `train_test_split` of `sklearn.model_selection` library

Training the `RandomForestRegressor()` on the train data

Predicting the y values through this model using `predict()`

Checking `mean_squared_error` between the predicted y values and test set of y values.

The rmse comes out to approximately 43 in this case. This is certainly an improvement over the previous multiple linear regression model with it rmse of around 64, but in no way is a standalone significant model to train over our existing data. There are certainly better ways of doing predictions over this data, but we may need to have a different insight into the data to perform this analysis to an optimal level.

4.11 GitHub Code Repository

The entire code constructed in relation to this project and explained in detail in the previous pages can be found at the following URL. I would humbly welcome any comments and additions to that repository with a pull request and association for a future work.

<https://github.com/Aausuman/Thesis>

Chapter 5

Conclusion and Future Work

The process followed in the previous pages has been a chartered plan of detailed analysis to identify relevant insights into the data available to us from the source. This can lead us to establish certain criteria points which can be helpful in filtering out the key aspects where focus can be laid down to improve the service as a whole and where we are obtaining the most relevant information.

The dataset in the raw form has been of multiple uses to us. The features available within have played a crucial role in performing multiple algorithms to get an insight into the network as a whole. The descriptive analysis along with the results furnished by it not only was a huge success, but also paved the way towards a future work wherein we can dig even deeper into this data and extract even more comprehensive knowledge.

One of the biggest advantages of having this dataset for our research was the real world application system ready to take influence from it for its benefit. The trends and pattern identified during these past months can be submitted as a formal result to the Dublin bus network authority and they can have an advantage of having this visually informative data to improve and work upon their service without any data preparation on their own end. From the analysis of the busy lines and congestions to the performance metrics of bus operators playing the role of actors within this network, the value of information curated here is substantial. The average reaching delay was yet another example of an excruciatingly detailed analysis at a granular level in the bus network.

However, the restricted nature of the dataset in terms of duration has definitely been a bit of challenge in creating a prediction plan for key features. If the data would have been of more number of months or even a year, the seasonality trend could have played a vital role in a time series analysis for the delay factor. Since a public transportation system heavily depends on the number of passenger it picks up within a tenure, the variability of the number on the basis of seasons is very crucial in this decision making, be it a manual process or by the help of an algorithm.

The variety of conclusions that can be drawn after the completion of this project are as listed below.

1. The dataset's Delay feature was the most important and influential in this research from a perspective of the subject's performance.
2. Congestion was a building block in the direct analysis of dependent nature of Delay.
3. Weekend's results of the delay analysis informed us that the reduction in service counts was directly proportional to fall in delay.
4. Prediction of the delay can't solely be done on this data-set to an optimal level. More data and more importantly, a feature rich data is required.

If we talk about what more features can be collected for an even detailed analysis of the data, a few methods come to mind.

1. How many passengers are on board at a given point in time on a bus.
2. The fares of the buses, there is a correlation with the price of a ticket and a person's willingness to board a cheaper more accessible bus rather than an expensive one.
3. How long a bus ride usually takes from one significant stop to another.

The facilitation of availability of these above listed features will be highly beneficial in a better analysis of the performance of Dublin bus network. Having the real time passenger count can easily tell us a busy line on a given day, and a bus duration time can also assist in identifying what people prefer and what they don't. These are a few ways wherein this project can be expanded upon in the future.

The learnings from this research venture have been amazing. The technologies that we got to experience and the professional guidance received during this project was nothing short of fulfilling. From the introduction to Apache Spark all the way up to the application of Random forest model onto the filtered data frame it was an amazing journey and we hope that the effectiveness and efficiency of the results was up to the mark and can be of assistance in other academic and professional works in the field.

Bibliography

- Crisp DM methodology* [June 2020]. URL: <https://www.sv-europe.com/crisp-dm-methodology/>.
- Dublin Bus GPS sample data from Dublin City Council (Insight Project)* [n.d.] URL: <https://data.gov.ie/dataset/dublin-bus-gps-sample-data-from-dublin-city-council-insight-project>.
- Elgendy, Nada, and Ahmed Elragal [Aug. 2014]. “Big Data Analytics: A Literature Review Paper”. In: vol. 8557, pp. 214–227. ISBN: 978-3-319-08975-1. DOI: 10.1007/978-3-319-08976-8_16.
- Guerreiro, G. et al. [2016]. “An architecture for big data processing on intelligent transportation systems. An application scenario on highway traffic flows”. In: *2016 IEEE 8th International Conference on Intelligent Systems (IS)*, pp. 65–72.
- Laney, Doug [2001]. “3D data management: Controlling data volume, velocity and variety”. In: *META group research note* 6.70, p. 1.
- Liu, J., and G. Xiao [2019]. “Efficient Bus Arrival Time Prediction Based on Spark Streaming Platform”. In: *2019 IEEE 23rd International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, pp. 416–421.
- RDD Programming Guide* [n.d.] URL: <https://spark.apache.org/docs/2.4.5/rdd-programming-guide.html#rdd-programming-guide>.
- Spark Overview* [n.d.] URL: <https://spark.apache.org/docs/2.4.5/index.html>.
- Vecmanis, Kevin [May 2019]. *Complete Guide to Installing PySpark on MacOS*. URL: <https://kevinvecmanis.io/python/pyspark/install/2019/05/31/Installing-Apache-Spark.html>.
- Xue, Rui, Daniel (Jian) Sun, and Shukai Chen [2015]. “Short-Term Bus Passenger Demand Prediction Based on Time Series Model and Interactive Multiple Model Approach”. In: *Discrete Dynamics in Nature and Society* 2015, 1â11. DOI: 10.1155/2015/682390.