Data201

Group Project Report

Take Five

Chathuranga Alwis | 81045233
Phua Sheng-He | 39267168
Jemin Lee | 75487642
Ziling Huang | 17053777
Jack Walsh | 31387258

Upon finishing all the work on the project, we were quite happy with how it turned out. The passion of the teammates and the effort in ensuring our work is very well documented and presented was exceptional. Although we were a bit slow in terms of getting work done on time, everyone equally contributed into the project and gave their fullest support. Furthermore, we would like to take this opportunity to thank the amazing lecturers, TA's and fellow colleagues as this wouldn't be possible without their support.

Table of Contents

1. Data Sources Used

➢ **Crash_Analysis_System__CAS__Data.csv**

Retrieved from :
https://data.govt.nz/

➢ **TrafficService.csv**

Retrieved from :
https://data.govt.nz/

➢ **RoadCondition.xlsx**

Retrieved from :
https://www.nzta.govt.nz/planning-and-investment/learning-and-resources/transport-ata/data-and-tools/

2. Why we chose these datasets

- Crash_Analysis_System__CAS__Data was picked as the first dataset as it has columns in various categories, from road names to crash severity to even light conditions. Although data analysis is not the focus of the project, the versatility of the dataset would open up to more logical relations to other types of datasets. Of which group members would be able to relate easier as compared to a relation with no logical reasoning behind it.
  On the other hand, this dataset was rather clean, meaning not much data cleaning was needed for it.

- TrafficService dataset was selected as it had some columns with inconsistent data, which meant data cleaning had to be done for it. It also could serve as a logical relation towards the original dataset, Crash_Analysis_System__CAS__Data, which will be explained in this report below.

- RoadCondition was used after considering the feedback received from the TAs during labs. This file contains information about the Road Network for the last 10 years with 3 different indexes.

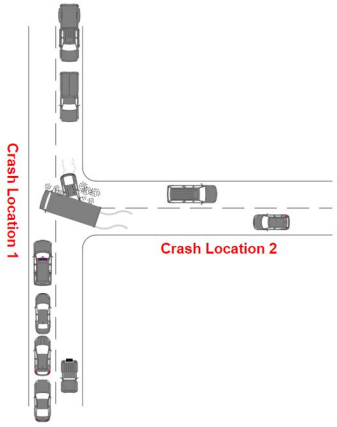


Figure 1 : How CrashLocation1 & CrashLocation2 is stored in CAS Dataset

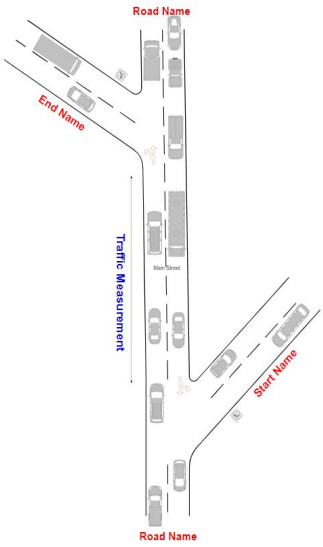


Figure 2 : How RoadName, StartName & EndName is stored in the TrafficService Dataset

3. Our Goal

★ Determine the number of Crashes for each Road Segment recorded in the Traffic Dataset.

★ Determine most crash dense regions and graph the crashes over time for specifically problematic roads

4. Difficulties

★ Gathering the ideas/datasets for the group project proved to be the hardest part of the assignment as it took 2-3 weeks, despite starting early, to get started on the project. The datasets were either too clean, or too specialised in each field of which restricted the flow of ideas from the group.

★ Planning the assignment through online communications also proved to be a challenge as different group members had different ideas at the beginning. So additional time was used to brief one another on the progress.

★ Using the Github for our code was challenging for all of us because we needed to work on the same file, but programming for different features. Branching was new to all of us, it was challenging to merge and resolve conflicts. However, we did a lot of google searches, and learnt more about version control.

Link to our Git Repo :
https://eng-git.canterbury.ac.nz/jle147/data201-group-project.git

5. A Brief Explanation of the Process and the Techniques we used

For this project we made use of the functionalities the following libraries offer.

```
library(tidyverse)
library(lubridate)
library(glue)
library(skimr)
library(visdat)
library(ggplot2)
library(stringr)
library(compare)
library(purrr)
library(splitstackshape)
library(knitr)
library(rvest)
library(RSelenium)
library(googleway)
library(readxl)
library(writexl)
```

Section 5.1 : General Cleaning

After importing the data from the files Crash_Analysis_System__CAS__Data.csv and TrafficService.csv we decided to reduce the datasets into smaller data frames before we clean them, using select() function:

```
#Extracting the needed columns from the Crash Dataset
sub_cas_df <- cas_df %>% select(Crash_Location1=crashLocation1,
                                Crash_Location2=crashLocation2,
                                Crash_Year=crashYear,
                                Region=region,
                                OBJECTID,
                                Speed_Limit = speedLimit,
                                Road_Surface = roadSurface,
                                Road_Lane = roadLane,
                                Flat_Or_Hill = flatHill)
```

For cleaning and formatting, we needed to change all abbreviated words to full words, remove all NA values, and unwanted parts of the variable strings.

Complete.cases() function was used in a defined function to remove all NA values for the selected columns as shown below.

```
clear_rows_NA <- function(df, desiredCols) {
  cleared_df <- complete.cases(df[, desiredCols])
  return(df[cleared_df, ])
}
```

Using regex and for loop, we formatted variable strings, removing unwanted parts, or changing abbreviations to full words:

```
sub_ts_df$Road_Name <- str_replace(sub_ts_df$Road_Name, " \\(.*\\)", "")
sub_ts_df$Start_Name <- str_replace(sub_ts_df$Start_Name, " \\(.*\\)", "")
sub_ts_df$End_Name <- str_replace(sub_ts_df$End_Name, " \\(.*\\)", "")


dict <- list(ST = "STREET", RD = "ROAD", LD = "LANE", PL = "PLACE", DR = "DRIVE", CRES = "CRESCENT", AVE = "AVENUE")

for(i in 1:length(dict)){
  sub_ts_df$Road_Name <- gsub(paste0("\\b", names(dict)[i], "\\b"), dict[[i]], sub_ts_df$Road_Name)
}

for(i in 1:length(dict)){
  sub_ts_df$Start_Name <- gsub(paste0("\\b", names(dict)[i], "\\b"), dict[[i]], sub_ts_df$Start_Name)
}

for(i in 1:length(dict)){
  sub_ts_df$End_Name <- gsub(paste0("\\b", names(dict)[i], "\\b"), dict[[i]], sub_ts_df$End_Name)
}

for(i in 1:length(dict)){
  sub_cas_df$Crash_Location1 <- gsub(paste0("\\b", names(dict)[i], "\\b"), dict[[i]], sub_cas_df$Crash_Location1)
}

for(i in 1:length(dict)){
  sub_cas_df$Crash_Location2 <- gsub(paste0("\\b", names(dict)[i], "\\b"), dict[[i]], sub_cas_df$Crash_Location2)
}
```

Section 5.2 : Formatting the Time Column

The Traffic Service dataset did have some columns which was not in a format we desired for our end product. One of these columns was the peak_hour column. Generally, it had 4 types of time formats in that column. They are as such:

- 700, which means 7am
- 7:00, which means 7am
- 0700, which means 7am
- 07:00 which means 7am, and has 5 characters

As seen above, these inconsistencies are not favourable for wrangling, so we decided to make all of the times consistent with type 4. The order of these if checks are important as it will slowly build the time format from any of the above types, to our desired type 4.

For example, the first ‘if’ check will check if the time format has 5 characters and a colon in the string. Doing so will allow for more efficiency so that times which are in the correct format will not need to follow through the remaining checks. glue() was mainly used in this function because most of the formats were lacking characters, of which we had to add onto it. substr() was also used because we had to split the string in half to insert the “:” in the middle.

```r
convert_proper_time_format <- function(time) {
    #get rid of all whitespaces in each time field
    time <- gsub(" ", "", time, fixed = TRUE)
    if ((nchar(time) == 5) && (str_detect(time, ":") == TRUE)) {
        return (time)
    }
    #if length of time field is 3, means its a time in the format like 800 or 500 which
    #represents 8am and 5am respectively we want things to be consistent, so we add a 0
    #in front of each of these types of times
    if (nchar(time) == 3) {
        time <- glue("0{time}")
    }

    #if ":" exists in the time field and the time field is of length 4, means the string
    #does not have the "0" in front of single digit times. thus, same solution as above,
    #add a "0" to the time
    if ((str_detect(time, ":") == TRUE) && (nchar(time) == 4)) {
        time <- glue("0{time}")
    }

    #if ":" does not exist, just add it in the middle of the string as by now, the string
    # would have at least 4 characters in the correct format
    if (str_detect(time, ":") == FALSE) {
        time <- glue("{substr(time, 1, 2)}:{substr(time, 3, 4)}")
    }

    #special cases whereby time values are "0:", so we just take it that these times are
    #00:00, instead of getting rid of them
    if (nchar(time) != 5) {
        time <- glue("0{time}00")
    }
    return (time)
}
```

Section 5.3 : Merging our Datasets

After the initial data cleaning and formatting, the team had two different ideas. Constructing a function to match the locations, or Using data joining functions. So we set out to implement both approaches and compare.

The first idea was to combine CrashLocation1, CrashLocation2 and make a column containing a list [CrashLocation1, CrashLocation2] and for each row of cas_df. Similarly, we created a column by combining RoadName, StartName, and EndName for each row in ts_df. This was by using unite(), and concat.split() functions.

```r
sub_ts_road_name_combined_df <- sub_ts_df %>%
                                unite(location, sep=", ", road_name:end_name, remove=FALSE, na.rm = TRUE)

sub_ts_road_name_combined_df <- concat.split(sub_ts_road_name_combined_df, 2, sep = ",", structure = "list")

sub_cas_locations_combined_df <- sub_cas_df %>%
                                 unite(location, sep=", ", crashLocation1:crashLocation2, remove=FALSE, na.rm = TRUE)

sub_cas_locations_combined_df <- concat.split(sub_cas_locations_combined_df, 1, sep = ",", structure = "list")
```

Then, check if the list [CrashLocation1, CrashLocation2] is a subset of the list [RoadName, StartName, EndName] for each row. Therefore, we constructed a function that would take two lists as parameters and use a for-loop to return a logical result if the crash location matches the traffic location, using the vector() function.

The code block below is the function constructed:

```r
location_match <- function(crash_location, traffic_location) {

    crash_location_len = length(crash_location)
    match_vector = vector(mode = "logical", length = crash_location_len)
    print(crash_location_len)
    for (i in 1:crash_location_len) {
        match_vector[i] = crash_location[i] %in% traffic_location
        if(length(which(crash_location[i] %in% traffic_location)) != 0){
            traffic_location = traffic_location[-match(crash_location[i], traffic_location)]
        }
    }
    result = all(match_vector)

    return(match_vector)
}
```

However, indexing these list columns in the functions we made was quite challenging. Further, as the CAS dataset contained approx. 700,000 rows, we realized that using nested for-loops would be very inefficient.

As a result, our second idea turned out to be a more efficient way. instead of using for-loops we figured that we could get the same desired output using the R function INNER-JOIN().

When using INNER-JOIN() to join the datasets, we used CrashLocation1 and CrashLocation2 as keys. Also, because we had to do it for every possibility, we used UNION() to combine multiple merged datasets together. However, as some of the merged datasets did not contain certain columns, in order to make it UNION compatible we had to remove some columns when using UNION.

The above merged dataset was mainly to retrieve information such as traffic peak hour and percentages of different types of vehicles using that given road where the crash has occurred. However, this was not our final target.

Thus, in order to achieve our final goal of studying and analyzing road segments, we used the same method as above to combine the CAS dataset to the Road Traffic dataset, using Road_Name, Start_Name and End_Name as keys.

Below code snippet illustrates how we combined the CAS dataset to the Road Traffic dataset.

```
# ============================================================================================================================

x<-inner_join(sub_ts_df, sub_cas_df, by =
           c("Road_Name" = "Crash_Location1", "End_Name" = "Crash_Location2"))

y<-inner_join(sub_ts_df, sub_cas_df, by =
           c("Road_Name" = "Crash_Location1", "Start_Name" = "Crash_Location2"))

#Combining the above two tibbles but making them union compatible
combined <- union(x, y)
combined <- combined %>% distinct(OBJECTID.x, .keep_all = TRUE)

#============================================================================================================================

x<-inner_join(sub_ts_df, sub_cas_df, by =
           c("Road_Name" = "Crash_Location2", "End_Name" = "Crash_Location1"))

y<-inner_join(sub_ts_df, sub_cas_df, by =
           c("Road_Name" = "Crash_Location2", "Start_Name" = "Crash_Location1"))

#Combining the above two tibbles but making them union compatible
combined <- union(x, y)
Merged_to_Traffic <- combined

#============================================================================================================================
Merged_to_Traffic %>% sample_n(10)
```

Section 5.4 : A little bit of Cleaning on the Merged Dataset.

Throughout the file, functions like skim(), summary() or str() were used to assess the types of our columns, to know what we were dealing with. This method allows us to check for inconsistencies and NA values which are undesirable for wrangling. Of which, it allowed us to find one particular NA value which could have been missed if not for the skim() function. The row had “Region” as NA as seen below.

	skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
*	<chr>	<int>	<dbl>	<int>	<int>	<int>	<int>	<int>
1	Crash_Location1	0	1	5	35	0	3771	0
2	Crash_Location2	0	1	5	29	0	4777	0
3	Region	1	1.00	12	24	0	12	0
4	Road_Surface	0	1	4	11	0	4	0
5	Road_Lane	0	1	4	8	0	4	0
6	Flat_Or_Hill	0	1	4	9	0	3	0
7	Peak_Hour	0	1	5	5	0	66	0

A tibble: 1 × 16

Crash_Location1	Crash_Location2	Crash_Year	Region	OBJECTID.x	Speed_Limit	Road_Surface	Road_Lane	Flat_Or_Hill	OBJECTID.y	ADT	Perc_Hea
<chr>	<chr>	<dbl>	<chr>	<dbl>	<dbl>	<chr>	<chr>	<chr>	<dbl>	<int>	
BLOCKHOUSE BAY ROAD	NEW NORTH ROAD	2019	NA	598309	50	Sealed	2-way	Null	3423	23694	

The initial idea was to scrape from google search results or google maps by searching Crash_Location1 and Crash_Location2. But to do that, we need to wait for the page to load for at least 1 second before we can proceed to scrape from it. As such, RSelenium was used for its web scraping capabilities, in particular, the setImplicitWaitTimeout function, so that we can wait for the page to load and then extract the information.

```
# binman::rm_platform("phantomjs")
# wdman::selenium(retcommand = TRUE)
# driver <- RSelenium::rsDriver(browser = "chrome",
#                               port = 4837L,
#                               chromever =
#                                 system2(command = "wmic",
#                                         args = 'datafile where name="C:\\\\Program Files (x86)\\\\Google\\\\Chrome\\\\
#                                         stdout = TRUE,
#                                         stderr = TRUE) %>%
#                                 stringr::str_extract(pattern = "(?<=Version=)\\d+\\.\\d+\\.\\d+\\.") %>%
#                                 magrittr::extract(!is.na(.)) %>%
#                                 stringr::str_replace_all(pattern = "\\.",
#                                                          replacement = "\\\\.") %>%
#                                 paste0("^",  .) %>%
#                                 stringr::str_subset(string =
#                                                       binman::list_versions(appname = "chromedriver") %>%
#                                                       dplyr::last()) %>%
#                                 as.numeric_version() %>%
#                                 max() %>%
#                                 as.character())
# remote_driver <- driver[["client"]]

#this is the function that would use the R selenium capabilities to extract the region
# get_specific_region <- function(road_name, remote_driver) {
#     url_road_location <- (glue("https://www.google.com/search?q={road_name}"))
#     remote_driver$navigate(url_road_location)
#     remote_driver$setImplicitWaitTimeout(milliseconds =3000)
#     address_element <- remote_driver$findElement(using = 'class', value = 'desktop-title-subcontent')
#     region <- address_element %>%
#                 html_text()
#     return (address_element)
# }
# get_specific_region("BLOCKHOUSE+BAY+ROAD", remote_driver)
```

Unfortunately, we were unable to get RSelenium to work for the google scrape, as it was too complicated. Above is our attempt at using RSelenium to create a driver to access google search for us. After some research, we found out that Google has their own API which would allow us to get the results easily without using RSelenium. In the Google library, there was a function called google_places() which we could get the region based on a road name.

```
#attempt to use the google api to get results but they required an API key
# google_places(search_string = "BLOCKHOUSE BAY ROAD")
```

The next issue came whereby they required a Google API key, which we were unable to get, and thus, we decided to abandon the idea.

The most straightforward idea was to get the region directly from our original datasets, because for each road, there was more than one row related to it and chances are, those rows would have a non-NA Region in it. We used functions like filter() to get the rows with that road name, and select() to focus only on the region column, groupby() and summarise() to get a count of the number of regions that came up. Because there were multiple possibilities with crashLocation1, crashLocation2, we had to consider all of them. We then combined these possibilities by binding the rows together and summing them up with summarise_all(). The end result will show the different types of regions that appeared for that road and their count. The region with the highest count would then be selected to replace the NA value.

```r
# The simple way to solve the NA column is to use the unaltered CAS dataset and sift through all crash location
# which has the same name and get the region.
# The description is as follows:
# We know for sure each road would be generalised to one region from the dataset.
# However, a road name can appear in other regions too.
# As such, we will get the counts of a region appearing in the filtered dataset by the target road_names,
 #and get the most commonly appearing region and set the NA value region as that.

#returns a string region from two input road names and the given dataset
get_specific_region <- function(road_name1, road_name2, cas_df) {
    #since each road name can appear in crashLocation1 and crashLocation2, and vice versa, we have to get all four
    #ombination tallies to get the most commonly appeared region for that road
    unique_regions_count1 <- cas_df %>% filter(crashLocation1 == road_name1) %>%
                                       select(region) %>% group_by(region) %>% summarize(count=n())

    unique_regions_count2 <- cas_df %>% filter(crashLocation2 == road_name2) %>%
                                       select(region) %>% group_by(region) %>% summarize(count=n())

    unique_regions_count3 <- cas_df %>% filter(crashLocation1 == road_name2) %>%
                                       select(region) %>% group_by(region) %>% summarize(count=n())

    unique_regions_count4 <- cas_df %>% filter(crashLocation2 == road_name1) %>%
                                       select(region) %>% group_by(region) %>% summarize(count=n())

    #combine all the results, and get the summed result of all the possible combinations.
    merged_result <- bind_rows(unique_regions_count1,
                               unique_regions_count2,
                               unique_regions_count3,
                               unique_regions_count4) %>%

                               group_by(region) %>%
                               summarise_all(funs(sum(., na.rm = TRUE)))

    #the highest appearing row is the region that will replace the NA value
    region_row <- merged_result[which.max(merged_result$count), ]
    return (region_row$region)
}
#replace the NA value with the majority region function above
Merged_to_Crash$Region[is.na(Merged_to_Crash$Region)] <- get_specific_region(Merged_to_Crash$Crash_Location1,
                                                                            Merged_to_Crash$Crash_Location2,
                                                                            cas_df)
```

6. What We Achieved

After we were done with the initial cleaning and all the steps mentioned above in section 5, we got a dataset containing the necessary data that's needed for our two goals as shown below.

A grouped_df: 4762 × 17

Region	Road_Name	Start_Name	End_Name	Number_of_Crashes	ADT	Peak_Hour	Perc_Cars	Perc_Light_CommVehicles	Perc_HeavyVehicles	Speed_L
<chr>	<chr>	<chr>	<chr>	<int>	<int>	<chr>	<int>	<int>	<int>	<i
Auckland Region	AARTS AVENUE	CRAMPTON PLACE	AWAKINO PLACE	1	955	08:00	99	0	1	
Auckland Region	ABBOTTS WAY	GRAND DRIVE	KENNETH SMALL PLACE	15	21569	11:30	96	0	4	
Auckland Region	ABBOTTS WAY	KENNETH SMALL PLACE	NGAHUE DRIVE	8	22298	16:30	94	0	6	
Auckland Region	ABBOTTS WAY	KORAHA STREET	GRAND DRIVE	19	23764	16:30	97	0	3	
Auckland Region	ABBOTTS WAY	LADIES MILE	KORAHA STREET	44	22681	16:30	96	0	4	
Auckland	ABERCROMBIE	COOK STREET	MOORE	10	932	17:00	98	1	1	

This is our Final Dataset containing all important information for each Road Segment.

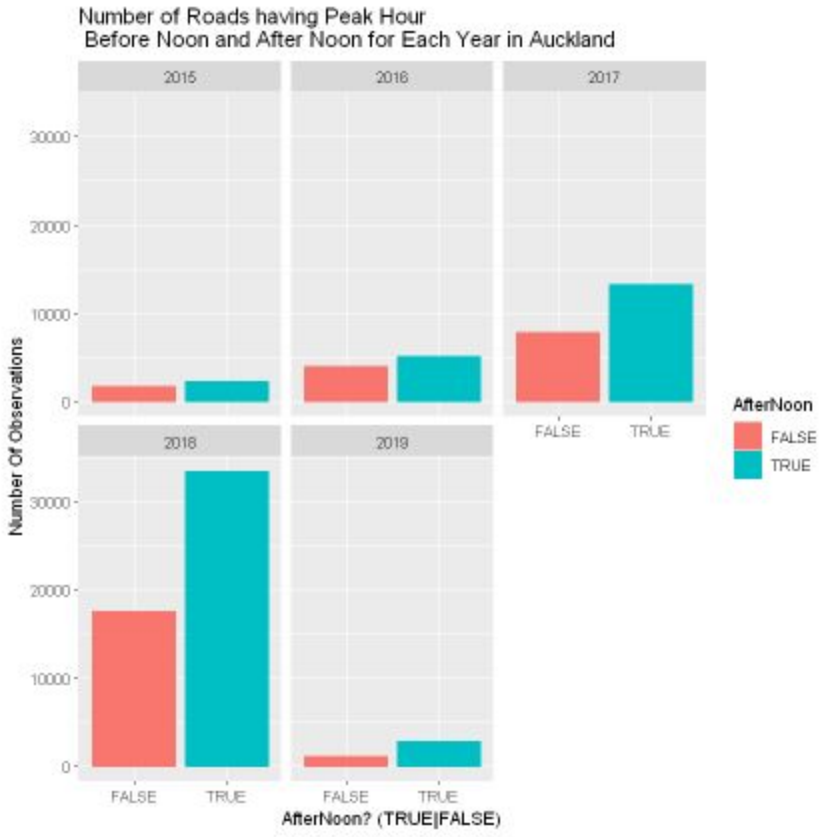
This dataset contains the three names of the roads that we are interested for each road segment, average daily traffic, peak hour, number of crashes in that road segment, average percentage of different types of vehicles, and so on,

Further, we combined the RoadCondition dataset into the above shown final dataset. Thus it now contains the pavement integrity index, road condition index and smooth travel exposure index for each region.

Therefore, our final dataset consists of an adequate amount of data needed for government officials to study a road segment, and use that in order to make improvements and for other decision making purposes.

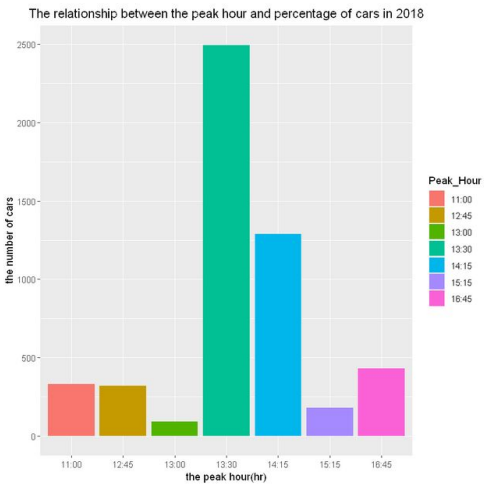
In order to get a clear visualization of some of the features of the dataset, we did some plots as shown below.

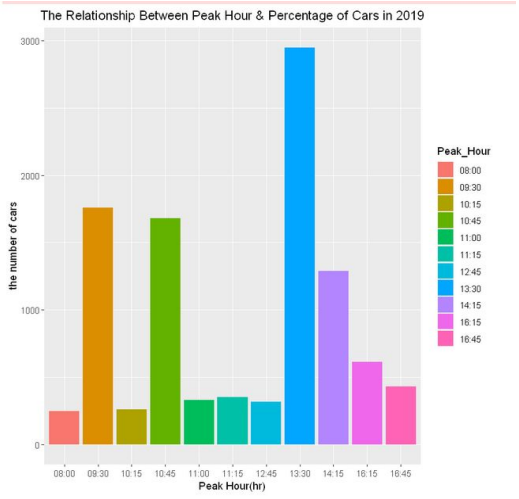
However, as the course doesn't focus much on the analysis part of the data, we did not spend too much time on the plots. Also, due to reasons such as the number of observations not being consistent for each road segment for each year, some of the plots might reflect some extreme outliers.



The plot above shows the number of roads having peak hour before-noon and afternoon for each year in Auckland. If we ignore the extreme case of 2018, it is quite clear from the above graphs that most roads have a peak hour in the afternoon. This might be caused by school traffic, and after work traffic.

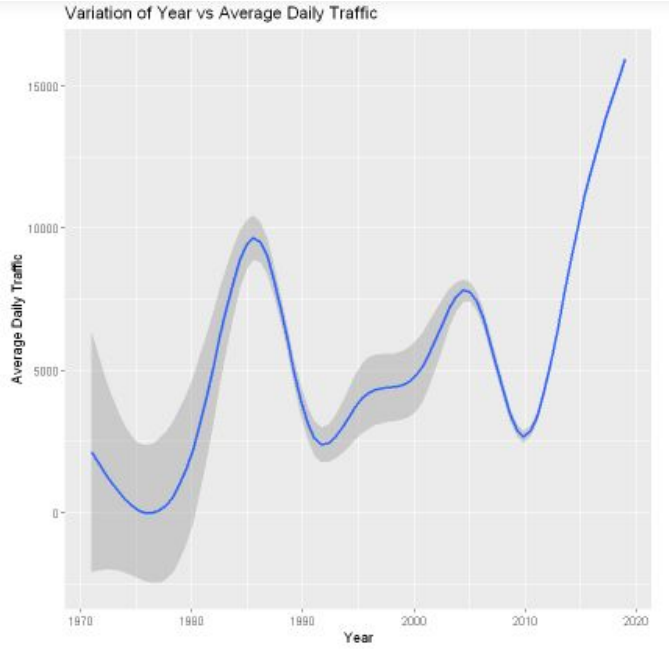
As we were quite curious with why that was the case, and to see if that's common with all regions, we plot the following two graphs for 2018 and 2019.





The above graphs shows the relationship between peak hour and the percentage of cars in 2018 and 2019. Through the above graphs it was clear that it was the case for all regions that the majority of the roads had a peak hour in the afternoon.

In order to get a broader idea of the dataset, we went on further and did some more plots as shown below.

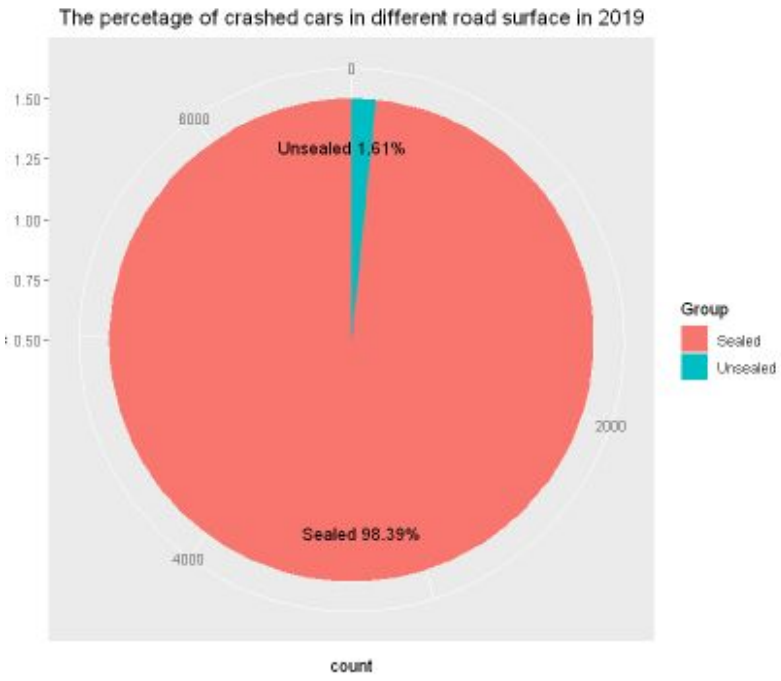


This graph depicts the variation of year vs average daily traffic.

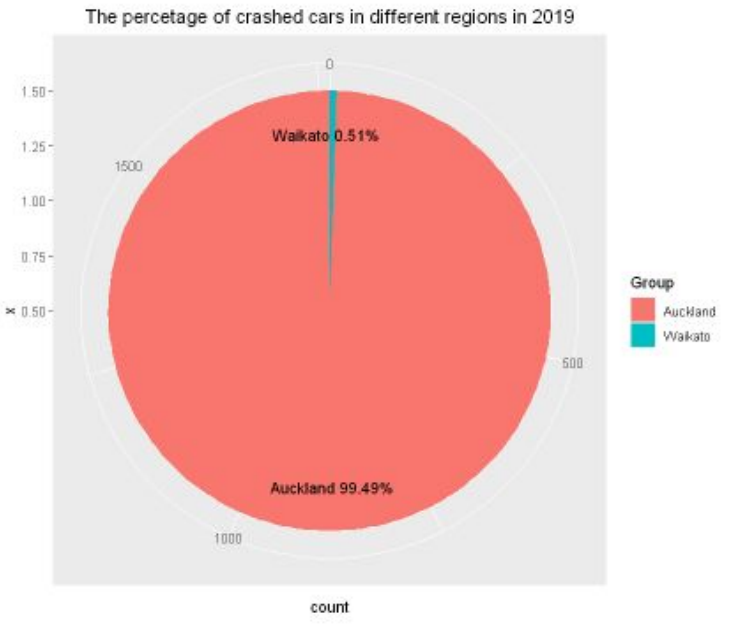
As it is seen on the graph, there's a minor upward trend. Which means that the average daily traffic increases as time passes. Further, after 2010 it has increased drastically.

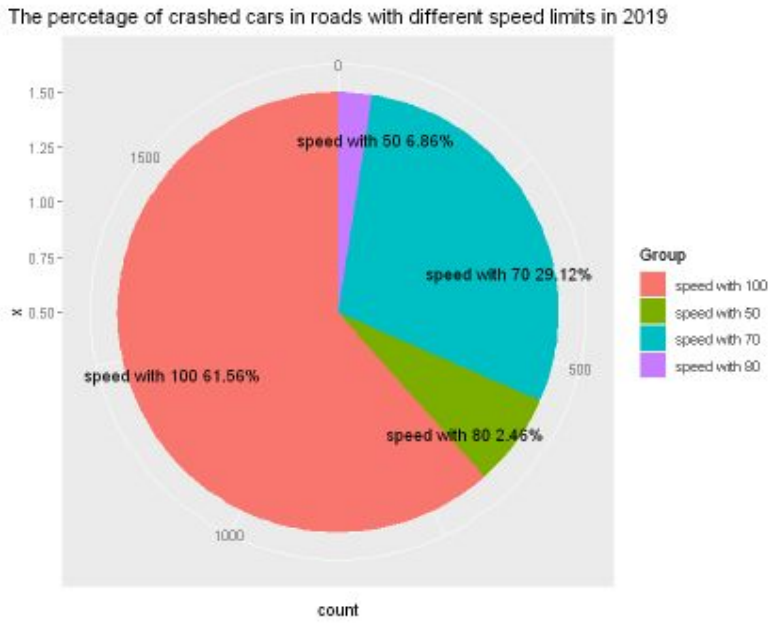
The pie chart to the right shows the ratio of crashes that happened on sealed roads in comparison to that aren't sealed.

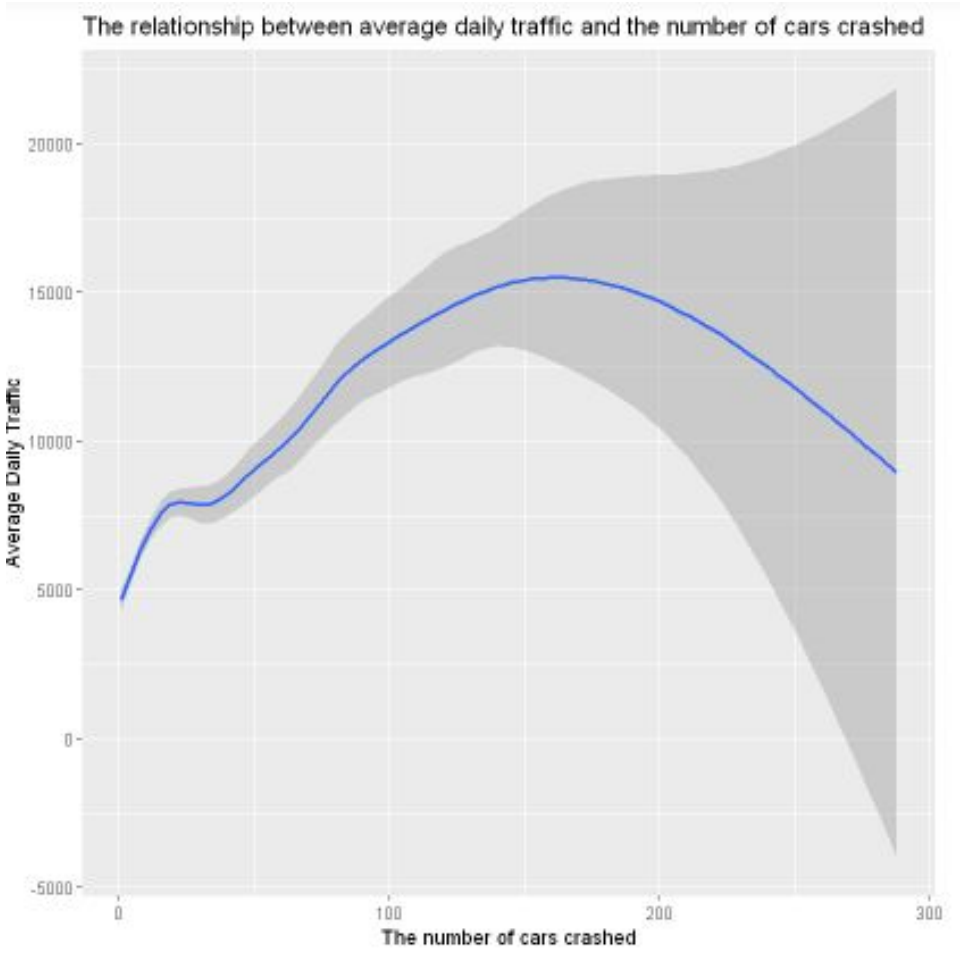
It is clear from the graph that 98.39% of the crashes occurred on sealed roads, and only 1.61% of the crashes occurred on roads that are not sealed. However, this might have been caused due to an inconsistent number of crashes being recorded.



Some more interesting plots.







7. What We Failed To Achieve

As we have shown above, we were able to successfully achieve our main goals for the project. However, it needs to be pointed out that the quality, accuracy, and complexity of the final dataset we produced was not perfect. Due to lack of time, lack of expertise and lack of resources we had to make some compromises on the final goal during the execution of our project.

One of the major ones were the assumptions we had to make. We assumed that the Crash Location 1 and Crash Location 2 was always a T junction as shown in Figure 1, and Road Name, Start Name and End Name was a road segment shown in Figure 2. This was due to the increasing complexity of the project otherwise.

Although at the start the of the project we did not think of it, interestingly at the presentation fellow colleagues raised a few questions such as,

- What if the crash occurred in a Roundabout, or an intersection involving 3 or more roads?
- How would you measure traffic in a Roundabout?
- What if the road names had some errors in them? Is ‘Width Change’ an actual road name? (This was raised by our lecturer Giulio Dalla Riva)

These problems haunted us all the way to this time we wrote this document. While we took every possible effort to try to resolve them, unfortunately it all failed. One of the methods we used to resolve these was also mentioned above in section 5.3 which was to use techniques such as RSelenium and API of Google. This failed due to numerous issues, one of the major ones being it was way too complicated and we did not have the required expertise.

We were not able to find a road named ‘Width Change’ in New Zealand on Google Maps, however as this was mentioned along with another road name on CAS dataset we assumed that this might be a term they use in the industry. Because otherwise it’s quite hard to believe that the name ‘Width Change’ was mistyped on both Datasets with the same adjoining road names.

Conclusion

To conclude, although a lot of time was lost brainstorming ideas, we managed to make the most of this data despite its many inconsistencies and errors. We managed to use most of the techniques learned throughout the semester and also learn some new ones trying to wrangle this data. It was definitely a good opportunity to not only apply what we learnt in class but also to work together as a team on a project. The technical challenges as well as the challenges we faced whilst working together as a team would indeed be beneficial for all of us in our future.