# SPRINT-III

| | |
|---|---|
| Date | 08 November 2022 |
| Team ID | PNT2022TMID12612 |
| Project Name | Project - Signs with smart connectivity for better road safety |

- Application Packages
- Main
- Slides
- Textcenter_analysis
- Task_Rmd

**Application Packages :**

```
package com.example.myhp.accidentprevention;
import android.app.Application;
import android.test.ApplicationTestCase;
/***
<ahref="http://d.android.com/tools/testing/testing_android.html">Testing
Fund>
*/
public class ApplicationTest extends ApplicationTestCase<Application>
{
public ApplicationTest()
{
super(Application.class);
}
}
```

**MAIN :**

\# Manipulating data {#data}
This section is an introduction to manipulating datasets using the `dplyr` package.
As outlined in the previous section, `dplyr` and `ggplot2` are part of the `tidyverse`, which aims
to provide a user-friendly framework for data science [@grolemund_data_2016].
Date
03 Nov 2022
Team ID
PNT2022TMID38493
Project Name
Project – Signs with Smart Connectivity for Better Road SafetyExperience of teaching R over the past few years suggests that many people find it easier to get

going with data driven research if they learn the 'tidy' workflow presented in this section.
However, if you do not like this style of R code or you are simply curious, we encourage you to
try alternative approaches for achieving the similar results using base R [@rcoreteam_language_2020]^[
Run the command `help.start()` to see a resources introducing base R, and [Chapter 6 on lists and
data frames](https://cran.r-project.org/doc/manuals/r-release/R-intro.html#Lists-and-data-frames)
in [An Introduction to R](https://cran.r-project.org/doc/manuals/r-release/R-intro.pdf) in
particular for an introduction to data manipulation with base R.
]
, the `data.table` R package [@R-data.table] or other languages such as
[Python](https://www.python.org/) or [Julia](https://julialang.org/).
If you just want to get going with processing data, the `tidyverse` is a solid and popular starting
point.
<!-- Todo: add new part here? -->
Before diving into the `tidyverse`, it is worth re-capping where we have got to so far as we have
covered a lot of ground.
Section \@ref(basics) introduced R's basic syntax; Section \@ref(rstudio) showed how to use the
Source Editor and other features of RStudio to support data science; and Section \@ref(pkgs)
introduced the concept and practicalities of R packages, with reference to `stats19`, `ggplot2` and
`dplyr`.
In this section, we will start with a blank slate.
In Section \@ref(basics) we learned that in R having a 'clear desk' means an *empty global
environment*.
This can be achieved by running the following command, which removes the `list()` of all
objects returned by the function `ls()`:
```{r}
rm(list = ls())
```
## tibbles
Although the data processing techniques in R are capable of handling large datasets, such as the
`crashes_2019` object that we created in the previous section, representing 100k+ casualties, it
makes sense to start small.
Let's start by re-creating the `crashes` dataset from Section \@ref(basics), but this time using the
`tidyverse` `tibble()` function. This is the `tidyverse` equivalent of base R's `data.frame`.

`tibble` objects can be created, after loading the `tidyverse`, as follows:
```{r, message=FALSE}
library(tidyverse)
crashes = tibble( casualty_type = c("pedestrian", "cyclist", "cat"),
casualty_age = seq(from = 20, to = 60, by = 20),
vehicle_type = c("car", "bus", "tank"),
dark = c(TRUE, FALSE, TRUE)
)
```
In the previous code chunk, we passed four vector objects as *named arguments* to the `tibble`
function, resulting in columns such as `casualty_type`.
A `tibble` is just a fancy way of representing `data.frame` objects, preferred by `tidyverse` users
and optimised for data science.
It has a few sensible defaults and advantages compared with the `data.frame`, one of which can
be seen by printing a `tibble`:
```{r}
class(crashes)
crashes
```
Note the `<chr>`, `<dbl>` or `<lgl>` text below each column, providing a quick indication of the
class of each variable - this is not provided when using `data.frame`.

## filter() and select() rows and columns

In the previous section, we briefly introduced the package `dplyr`, which provides an alternative
to base R for manipulating objects. `dplyr` provides different, and some would argue simpler,
approaches for subsetting rows and columns than base R.
`dplyr` operations for subsetting rows (with the function `filter()`) and columns (with the
function `select()`) are demonstrated below. Here we can also see the use of the pipe operator
`%>%` to take the dataset and apply the function to that dataset.
```{r}
crashes %>% filter(casualty_age > 50) # filters rows
crashes %>% select(casualty_type) # select just one column
```
It should be clear what happened: `filter()` returns only rows that match the criteria in the
function call, only observations with a `casualty_age` greater than 50 in this case.
Likewise, `select()` returns data objects that include only columns named inside the function call,
`casualty_type` in this case.
To gain a greater understanding of the functions, type and run the following commands, which
also illustrate how the `%>%` can be used more than once to manipulate data (more on this

soon):
```{r}crashes_darkness = crashes %>% filter(dark)
crashes_a = crashes %>% select(contains("a"))
crashes_darkness_a = crashes %>%
filter(dark) %>%
select(contains("a"))
```
Can you guess what the dimensions of the resulting objects will be?
Write down your guesses for the number of rows and number of columns that the new objects,
`crashes_darkness` to `crashes_darkness_a`, have before running the following commands to
find out. This also demonstrates the handy function `dim()`, short for dimension (results not
shown):^[
Note that the number of rows is reported before the number of columns.
This is a feature of R: rows are also specificiefied first when subsetting using the square brackets
in commands such as `crashes[1, 2:3]`.
]
```{r, eval=FALSE}
dim(crashes)
dim(crashes_darkness)
?contains # get help on contains() to help guess the output of the next line
dim(crashes_a)
dim(crashes_darkness_a)
```
Look at the help pages associated with `filter()`, `select()` and the related function `slice()` as
follows and try running the examples that you will find at the bottom of the help pages for each
to gain a greater understanding (note you can use the `package::function` notation to get help on
functions also):
```{r, eval=FALSE}
?dplyr::filter
?dplyr::select
?dplyr::slice
```
## Ordering and selecting the 'top n'
Other useful pipe-friendly functions are `arrange()` and `top_n()`. `arrange()` can be used to sort
data. Within. the `arrage()` function, optional arguments can be used to define the order in which
it is sorted. `top_n()`simply selects the top 'n' number of rows in your data frame.
We can use these functions to arrange datasets and take the top most ‘n’ values, as follows:
```{r}
crashes %>% arrange(vehicle_type)
crashes %>%
```

top_n(n = 1, wt = casualty_age)
```
&lt;!-- ## Long and wide data --&gt;
## Summarise
A powerful two-function combination is `group_by()` and `summarise()`.
Used together, they can provide *grouped summaries* of datasets.
In the example below, we find the mean age of casualties in dark and light conditions.
```{r}
crashes %>%
group_by(dark) %>%
summarise(mean_age = mean(casualty_age))
```
The example above shows a powerful feature of these pipelines. Many operations can be
'chained' together, whilst keeping readability with subsequent commands stacked below earlier
operations. The combination of `group_by()` and `summarise()` can be very useful in preparing
data for visualisation with a `ggplot2` function.
Another useful feature of the `tidyverse` from a user perspective is the autocompletion of column
names mid pipe.
If you have not noticed this already, you can test it by typing the following, putting your cursor
just before the `)` and pressing `Tab`:
```{r, eval=FALSE}
crashes %>% select(ca) # press Tab when your cursor is just after the a
```
You should see `casualty_age` and `casualty_type` pop up as options that can be selected by
pressing `Up` and `Down`.
This may not seem like much, but when analysing large datasets with dozens of variables, it can
be a godsend.
Rather than providing a comprehensive introduction to the `tidyverse` suite of packages, this
section should have offered enough to get started with using it for road safety data analysis.
For further information, check out up-to-date online courses from respected organisations like
[Data Carpentry](https://datacarpentry.org/R-ecology-lesson/index.html) and the free online
[books](https://bookdown.org/) such as [R for Data Science]((https://r4ds.had.co.nz/))
[@grolemund_data_2016].
## Tidyverse exercises1. Use `dplyr` to filter rows in which `casualty_age` is less than 18, and then 28.
2. Use the `arrange` function to sort the `crashes` object in descending order of age (**Hint:**
see the `?arrange` help page).
3. Read the help page of `dplyr::mutate()`. What does the function do?
```

4. Use the mutate function to create a new variable, `birth_year`, in the `crashes` data.frame
which is defined as the current year minus their age.
5. **Bonus:** Use the ` %>% ` operator to filter the output from the previous exercise so that
only observations with `birth_year` after 1969 are returned.
```{r dplyr, eval=FALSE, echo=FALSE}
# answers
crashes %>%
arrange(desc(casualty_age))
crashes %>% filter(casualty_age > 21)
crashes %>%
mutate(birth_year = 2019 - casualty_age) %>%
filter(birth_year > 1969)
```

## Slides :

```
---
title: " Road Safety (and transport) Research with R"
# subtitle: '`r emojifont::emoji("bike")`<br/>For England and Wales'
subtitle: '`r emojifont::emoji("rocket")`<br/>RAC Foundation, Data Driven'
author: "Robin Lovelace"
date: '2020'
output:
xaringan::moon_reader:
# css: ["default", "its.css"]
# chakra: libs/remark-latest.min.js
lib_dir: libs
nature:
highlightStyle: github
highlightLines: true
# bibliography:
# - ../vignettes/ref.bib
# - ../vignettes/ref_training.bib
---
```
```{r setup, include=FALSE, eval=FALSE}
# get citations
refs = RefManageR::ReadZotero(group = "418217", .params = list(collection =
"JFR868KJ",
limit = 100))
refs_df = as.data.frame(refs)
# View(refs_df)# citr::insert_citation(bib_file = "vignettes/refs_training.bib")
RefManageR::WriteBib(refs, "refs.bib")
# citr::tidy_bib_file(rmd_file = "vignettes/pct_training.Rmd", messy_bibliography =
"vignettes/refs_training.bib")
options(htmltools.dir.version = FALSE)
knitr::opts_chunk$set(message = FALSE)
library(RefManageR)
BibOptions(check.entries = FALSE,
bib.style = "authoryear",
```

cite.style = 'alphabetic',
style = "markdown",
first.inits = FALSE,
hyperlink = FALSE,
dashed = FALSE)
my_bib = refs
```
```{r, eval=FALSE, echo=FALSE, engine='bash'}
# publish results online
cp -Rv code/rrsrr-slides* ~/saferactive/site/static/slides/
cp -Rv code/libs ~/saferactive/site/static/slides/
cd ~/saferactive/site
git add -A
git status
git commit -am 'Update slides'
git push
cd -
```
# Slide/links
https://itsleeds.github.io/rrsrr/
https://bookdown.org/
https://www.pct.bike/
---
background-image: url(https://media.giphy.com/media/YlQQYUIEAZ76o/giphy.gif)
# Coding
Ideal:```{r, eval=FALSE}
od_test$perc_cycle = round(od_test$bicycle / od_test$all) * 100
l = od_to_sf(od_test, od_data_centroids)
r = stplanr::route(l = l, route_fun = journey)
rnet = overline(r, "bicycle")
```
--
![](https://media.giphy.com/media/3oKIPnAiaMCws8nOsE/giphy.gif)
Reality
---
## Transport software - which do you use?
```{r, echo=FALSE, message=FALSE, warning=FALSE}
u = "https://github.com/ITSLeeds/TDS/raw/master/transport-software.csv"
tms = readr::read_csv(u)[1:5]
tms = dplyr::arrange(tms, dplyr::desc(Citations))
knitr::kable(tms, booktabs = TRUE, caption = "Sample of transport modelling software in use by
practitioners. Note: citation counts based on searches for company/developer name, the product
name and 'transport'. Data source: Google Scholar searches, October 2018.", format = "html")
```
---
## Data science and the tidyverse

- Inspired by Introduction to data science with R (available free
[online](https://r4ds.had.co.nz/))
```{r tds-cover, echo=FALSE, out.width="30%"}
knitr::include_graphics("https://d33wubrfki0l68.cloudfront.net/b88ef926a004b0fce72
b2526b0b5
c4413666a4cb/24a30/cover.png")
```
---
## A geographic perspective
- See https://github.com/ITSLeeds/TDS/blob/master/catalogue.md
- Paper on the **stplanr** paper for transport planning (available
[online](https://cran.r-project.org/web/packages/stplanr/vignettes/stplanr-paper.html))
- Introductory and advanced content on geographic data in R, especially the [transport
chapter](https://geocompr.robinlovelace.net/transport.html) (available free
[online](https://geocompr.robinlovelace.net/))
- Paper on analysing OSM data in Python with OSMnx (available
[online](https://arxiv.org/pdf/1611.01890))
---
# Getting support
--
With open source software, the world is your support network!
--
- Recent example: https://stackoverflow.com/questions/57235601/
--
- [gis.stackexchange.com](https://gis.stackexchange.com/questions) has 21,314
questions
- [r-sig-geo](https://r-sig-geo.2731867.n2.nabble.com/) has 1000s of posts
- RStudio's Discourse community has 65,000+ posts already!
--
- No transport equivalent (e.g. earthscience.stackexchange.com is in beta)
- Potential for a Discourse forum or similar: transport is not (just) GIS
---

## Textcenter_analysis:

---
pagetitle: "Analysis based on test centres"
author: "Amol Nanaware"
date: "06/12/2019"
always_allow_html: true
output:
html_document: default
---```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```
```{r, warning=FALSE, echo=FALSE, include=FALSE}
packages <- c("plotly", "tidyverse")
newPackages <- packages[!(packages %in% installed.packages()[,"Package"])]
if(length(newPackages)) install.packages(newPackages)
library(tidyverse)
library(plotly)
```
```

````
```{r,echo=FALSE}
load("passfail.RData")
passfail <- passfail %>%
mutate(totalFails = Fail1 + ifelse(is.na(Fail2), 0, Fail2), Totalpass = Pass1 +
ifelse(is.na(Pass2),
0, Pass2))
```
```{r,echo=FALSE}
passfailGroup <- summarise(group_by(passfail, Centre), Pass1 = sum(Pass1), Fail1 =
sum(Fail1), Total1 = sum(Total1), Pass2 = sum(Pass2, na.rm = T), Fail2 = sum(Fail2,
na.rm =
T), Total2 = sum(Total2, na.rm = T), Totalpass = sum(Totalpass), totalFails =
sum(totalFails))
passfailGroup <- mutate(passfailGroup, Pass1prop = Pass1/Total1, Pass2prop =
Pass2/Total2,
totalPassProp = (Totalpass / (Total1 + Total2)), totalFailsProp = (totalFails / (Total1 +
Total2)))
```
```{r,echo=FALSE}
passfailGroup$totalPassProp = round((passfailGroup$totalPassProp * 100), digits = 2)
passfailGroup$totalFailsProp = round((passfailGroup$totalFailsProp * 100), digits =
2)
passFailGroup1 <- passfailGroup[c(1, 8)]
passFailGroup1$Test <- "Pass"
names(passFailGroup1) <- c("Centre", "Count", "Test")
passFailGroup2 <- passfailGroup[c(1, 9)]
passFailGroup2$Test <- "Fail"
names(passFailGroup2) <- c("Centre", "Count", "Test")
passFailcount <- rbind(passFailGroup1, passFailGroup2)
```### Analysis based on test centres
<br/>In this section we will analyse data from 2013 till 2018 about each test centre.
As shown in
the <a href = "https://github.com/NanawareAmol/R-project_Road
safety/blob/master/Result/loc_spread_across_ireland.JPG">map</a>, the test centres
are spread
across the Ireland and the number of centres is more in highly populated areas such as
dublin,
cork etc.
The bar chart shows the total number of tests that each centre performed and the total
pass and
fail counts as well as percentages. So, based on the test counts, the top 3 test centre
are,
*Fonthill(770685)*, *Deansgrade(767484)*, and *Northpoint 2(729661)*. The botton
3 centres
which performed less tests are, *Donegal Town(16315)*, *Cahirciveen(28806)* and
*Clifden(38683)*.
```{r,echo=FALSE, fig.width=9,fig.height=4}
p <- plot_ly(passfailGroup, x = ~passfailGroup$Centre, y = ~passfailGroup$Totalpass,
type =
'bar', name = 'Pass', text = paste("Total tests = ",
````
````

````
(passfailGroup$Totalpass+passfailGroup$totalFails), "<br>Passed =",
passfailGroup$totalPassProp,"%", "<br>Failed =", passfailGroup$totalFailsProp,"%"),
opacity =
0.5, marker = list(color = '#3AC3E3', line = list(color = '#0D6EB0', width = 1))) %>%
add_trace(y = ~passfailGroup$totalFails, name = 'Fails', opacity = 0.5,
marker = list(color = '#0E84FF', line = list(color = '#0D6EB0', width = 1))) %>%
layout(yaxis = list(title = 'Count'), xaxis = list(title = 'Test Centres'), barmode = 'stack')
p
```
#### <b>Total test passed for each test centre</b>
The following scatter plot show the total test pass count for each test centre from the
year 2013
till year 2018. The questions that can be answered by this graph are, <br/>
1. which are the top 3 and last 3 centres based on total pass count?<br/>
    <b>(Deansgrade, Northpoint 2, Fonthill and
Cahirciveen, Clifden, derrybeg resp.)</b><br/>
2. Which year has the highest and lowest total pass count?<br/>
    <b>2015 and 2014 respectively</b><br/>
But, in this graph we are not considering the total tests performed by the test centres
which
shows the actual performance of the tests. For this we will plot another graph.
<br/><br/>
```{r,echo=FALSE, fig.width=9,fig.height=4}
#scatter plot for centre total pass per year
ggplot(data = passfail, aes(x = fct_reorder(Centre, -Totalpass), y = Totalpass, color =
Year, size
= Totalpass)) + geom_point(alpha = 0.5) +
theme(axis.text.x = element_text(size=9, angle=-90, hjust = 0, vjust = 0.5),
axis.ticks.x =
element_blank(), panel.background = element_rect(fill = "white", colour =
"lightblue"),
panel.grid.minor = element_line(size = 0.5, linetype = 'solid', colour = "lightblue")) +
labs(x = "Test Centres", y = "Totol pass count")
```#### <b>Test performance for each test centre</b>
The graph gives the overall idea of the test performance based on pass rate and the
year.
As per the graph we can say that for year 2013, 2015, 2016, 2017 and 2018, the pass
rate is
higher that 55%. And the highest and lowest performance found in Kilkenny and
Monaghan test
centres respectively.<br/><br/>
```{r,echo=FALSE, fig.width=9,fig.height=4}
passfail$totPassPercentage <- round((passfail$Totalpass / (passfail$Totalpass +
passfail$totalFails)) * 100, digits = 2)
passfail$totFailPercentage <- round((passfail$totalFails / (passfail$Totalpass +
passfail$totalFails)) * 100, digits = 2)
#scatter plot for centre pass percetage per year
ggplot(data = passfail, aes(x = fct_reorder(Centre, -totPassPercentage), y =
totPassPercentage,
color = Year, size = totPassPercentage)) + geom_point(alpha = 0.5) +
````

~~~~
theme(axis.text.x = element_text(size=9, angle=-90, hjust = 0, vjust = 0.5),
axis.ticks.x =
element_blank(), panel.background = element_rect(fill = "white", colour =
"lightblue"),
panel.grid.minor = element_line(size = 0.5, linetype = 'solid', colour = "lightblue")) +
labs(x = "Test Centres", y = "Total Pass %")#title = "Test centre pass% per year",
```
#### <b>Total pass count limits per year</b>
The box plot shows the total pass count against each year. With this we can fetch the
details on
maximum and minimum pass counts per year, the meadian pass count and the
oustanding pass
count values which are shown as outliers (points) per year with the test centre
name.<br>
```{r,echo=FALSE, fig.width=9,fig.height=4}
p <- plot_ly(passfail, x = passfail$Year, y = passfail$Totalpass, color =
~passfail$Year, type =
"box", text = paste("Centre = ", passfail$Centre)) %>%
layout(title = "Yearly performance", yaxis = list(title = 'Total Pass Count'), xaxis =
list(title =
'Year'))
p
```
~~~~

## Task_Rmd :

~~~~
---
title: " Project – Signs with Smart Connectivity for Better Road Safety "
output: html_document
---
```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
#ggparcoord
#geom_polygon => packcircles
```## Libraries
```{r cars}
suppressMessages(library(readxl))
suppressMessages( library(dplyr))
suppressMessages( library(tidyr))
suppressMessages( library(ggplot2))
suppressMessages( library(MASS))
suppressMessages( library(GGally))
suppressMessages( library(ggExtra))
suppressMessages( library(plotly))
suppressMessages( library(packcircles))
```
## Preparing The Data For Analysis
```{r}
df <- read_excel("mmAll.xlsx")
#d13 <- read_excel("m_m2013.xlsx")
#d14 <- read_excel("m_m2014.xlsx")
~~~~

~~~
#d15 <- read_excel("m_m2015.xlsx")
#d16 <- read.csv("m_m2016.csv", header = T)
#d17 <- read_excel("m_m2017.xlsx")
#d18 <- read_excel("m_m2018.xlsx")
names(df)[9] <- "Vehicle and Safety Equipment"
names(df)[10] <- "Vehicle and Safety Equipment %"
names(df)[22] <- "Chassis and Body %"
names(df)[26] <- "Suspension Test %"
names(df)[36] <- "Incomplete Tests %"
df$reportYear <- as.factor(df$reportYear)
```
## Which part failed the most per report year?
```{r, echo=FALSE}
######DATA#######
cols <- c("VehicleMake", "Vehicle and Safety Equipment", "Lighting and Electrical",
"Steering
and Suspension", "Braking Equipment", "Wheels and Tyres", "Engine, Noise and
Exhaust",
"Chassis and Body", "Side Slip Test", "Suspension Test", "Light test", "Brake Test",
"Emmissions", "OTHER")
m <- df %>% dplyr :: select(c("reportYear", cols)) %>% group_by(reportYear) %>%
summarise_if(is.numeric, mean, na.rm = TRUE)
m <- gather(m,-reportYear, key=Part, value= Failures)
``````{r}
######PLOT#######
#ggplot(m, aes(x=factor(reportYear), y=, colour=supp, group=supp)) + geom_line()
library(MASS)
library(GGally)
# Vector color
library(RColorBrewer)
palette <- brewer.pal(3, "Set1")
my_colors <- palette[as.numeric(m$reportYear)]
#names(x) <- c("2013","2014","2015","2016","2017","2018")
#p <- ggparcoord(m, columns=2:13, groupColumn =
"reportYear")+geom_line(size=0.3)+theme_minimal() + geom_point()+
# xlab("Car Part")+ylab("Average failure rate")
ggplotly(ggplot(data=m, mapping = aes(x = reportYear, y = Failures, colour = Part,
group=1))+
geom_point()+
geom_line()+xlab("Report Year")+ ylab("Average Number of Failures")
)
```
## Equipment Failures - Overall Statistics
```{r}
library(ggplot2)
cols <- c("Vehicle and Safety Equipment", "Lighting and Electrical", "Steering and
Suspension",
"Braking Equipment", "Wheels and Tyres", "Engine, Noise and Exhaust", "Chassis
and Body",
~~~

````
"Side Slip Test", "Suspension Test", "Light test", "Brake Test", "Emmissions",
"OTHER")
a <- df %>% dplyr::select(cols)
b<-colSums(a)
c <- data.frame(Part = names(b), Percent = unname(b)/sum(df$Total)*100)
ggplot(c)+
geom_col(mapping = aes(x = reorder(Part, -Percent), y = Percent, fill = Percent),
col="black")+
xlab("")+
ylab("Failure Percentage(%)") +
scale_fill_gradient(low = "orange", high = "tan")+
coord_flip()
```
### There is a bug in this code. Can anybody fix it?
```{r, eval=FALSE}
####The polygon graph representation of the above data####
l <- data.frame(Part = names(b), Total = unname(b))
packing <- circleProgressiveLayout(l$Total,sizetype='area')l$packing <- packing
l
dat.gg <- circleLayoutVertices(packing, npoints=50)
p <- ggplot() + geom_polygon(data = dat.gg, aes(x, y, group = id, fill=as.factor(id)),
colour =
"black", alpha = 0.6) + geom_text(data = l, aes(x, y, size = Total, label =
Part))+scale_size_continuous(range = c(1,4)) +theme_void()
+theme(legend.position="none") +
coord_equal()
ggplotly(p, tooltip = c("Total", "Part"))
```
```{r}
z <- df %>% group_by(VehicleMake) %>% summarise(tot=sum(Total),res =
sum(PASS)/sum(Total)) %>% arrange(desc(tot)) %>% print(Inf())
```
```

{r}
require(scales)
q <- z %>% arrange(desc(tot)) %>% slice(1:15)
ggplot(q)+
geom_col(mapping = aes(x = reorder(VehicleMake, -tot), y = tot, fill = "green"))+
xlab("Vehicle Make")+ylab("Number of Vehicles") + coord_flip()+
theme(legend.position =
"none")+
scale_y_continuous(labels = comma)
#ggMarginal(g, type = "histogram", fill="transparent")
```
## Pass Percentage versus Number of Vehicles for a given VehicleMake
```{r}
require(scales)
library(plotly)
p <- ggplot(q, aes(x = tot, y = res*100))+
geom_line(color = "red")+
````

```
geom_point(aes(text = VehicleMake))+xlab("Number of Vehicles") + ylab("Pass
Percentage
(%)") +
scale_x_continuous(labels = comma)
ggplotly(p, tooltip = "text")
```