Visualising Multi-dimensional Data

tutorial-sroles1 created by GitHub Classroom

View on GitHub

Visualising Multi-dimensional Data

High dimentional data refers to data frames which contain many variables and or levels. They are difficult to plot as only so much infomation can exist on a single figure whilst maintaining clarity. This tutorial is an introduction to several possible approaches to displaying multiple varibles whilst keeping plots clear and manageable. We will consider the use of interactive plots using the 'plotly' package to suppliment our figures throughout. The tutorial is broken down into the following sections:

Learning Objectives:

- To be able to name some of the dificulties of plotting multi-dimentional data
- To be able to plot diferent types of multi-dimentional data using the approaches shown including 'workarounds' to problems associated with diferent plot types
- To understand the uses, benefits and limitations of using interactive graphs
- To practice data wrangling and exploration techniques useful when handling multi-dimentional data

First, download the Github repository locally so that you can access the data sets and starter script required for this tutorial, through this link:https://github.com/EdDataScienceEES/tutorial-sroles1.git. Click on Clone/Download Zip, download the files and unzip them. The script for this tutorial is called X and the data frames 2020_ap_exit_polls_combined_2.xlsx, us_agri.csv and 2015_State_Top10Report_wTotalThefts.xlsx are in the data folder.

Mosaic plots: Multiple catagorical variables

Mosaic plots are derived from barcharts and spineplots and are used for displaying multiple catagorical variables. They are formed of a series of tiles whose sizes are proportional to the number of observations in their particular intersection. Mosaic plots lend themselves best to survey data. Survey data can become highly dimentional due with many catagorical columns with many repeating rows (count data) which will influence the size of each tile.

We will use data from the 2016 US election exit poll to explore and learn how to use mosaic plots.

```
# clear environment
rm(list=ls())

# libraries
library(tidyverse)
library(readxl)

# load exit poll data
exit_poll <- read_excel("data/2020_ap_exit_polls_combined_1.xlsx")

# see all the columns and their unique levels
```

```
ulist <- lapply(exit_poll, unique)
ulist
```

This data set contains 28 demographics and 50 states Mosaic plots work well for displaying multiple catagorical variables but can’t handle this many levels and remain clear. In this example we will choose our demogrpahic to focus on to be wage-brackets and will choose the four states with the most electoral power.

```
# Filter data set for only wage-bracket demographics
filt <- exit_poll %>% filter(Demographic == "Under $25,000"  |
                          Demographic == "$25,000 - $49,999" |
                          Demographic == "$50,000 - $74,999" |
                          Demographic == "$100,000+")

# Find out which states have most electoral power
state_value <- unique(exit_poll[c( "Electoral_Votes_Available",
"State_Abbr")])
state_value <- state_value[order(state_value$Electoral_Votes_Available,
decreasing = TRUE), ]

# Filter data set for top four states
filt <- filt %>% filter(State_Abbr == "CA" |
                             State_Abbr == "TX" |
                            State_Abbr == "FL" |
                           State_Abbr == "NY")
```

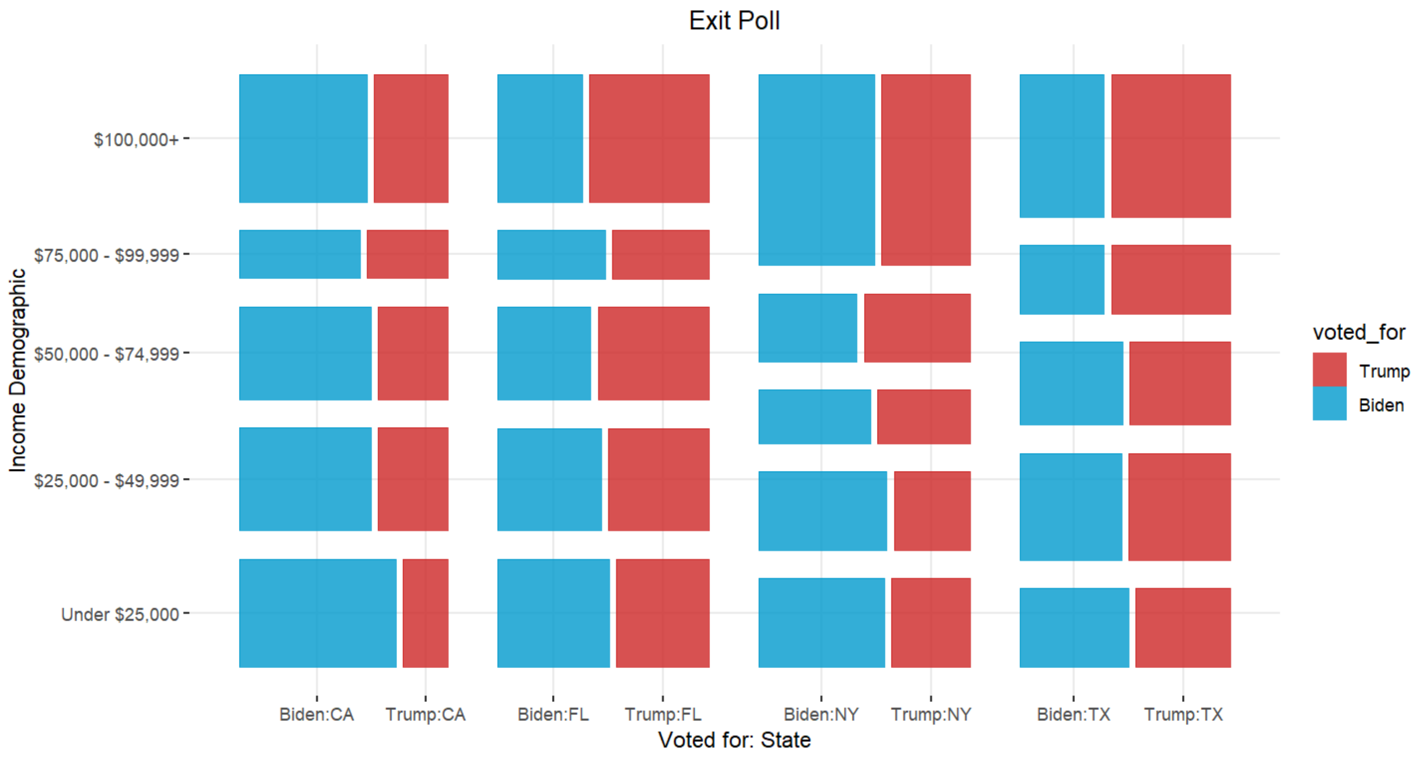
Our data frame originates from an exit poll survey taken nationwide. However, some ‘wrangling’ has already occured and what we are presented with is a summary data frame. Mosaic plots are best for displaying the raw data. If we had collected with data ourselves, using a mosaic plot would summarise our data for us without the need for the extra wrangling which has already occured. We will take a backwards step to return the data to a form representative of its raw form to allow the mosaic plot to deal with the count data it is best with.

```
# library
library(splitstackshape)

# Convert rows to count data for the percentage who voted for biden and
# the percentage who voted for trump.

# Create two new data frames in which to expand the rows and delete
# column where data was extracted from.
expanded_biden <- expandRows(filt, "Biden_%")
expanded_trump <- expandRows(filt, "Trump_%")

# Expand rows to show the proportion of the total population each
# demographic makes up.
```

```
expanded_biden <- expandRows(expanded_biden, "proportion")
expanded_trump <- expandRows(expanded_trump, "proportion")

# Add new column name to distinguish Trump voters from Biden voters
expanded2$voted_for <- "Trump"
expanded$voted_for <- "Biden"

# Remove names of column which has been deleted from the other data set
# in order to make them match, ready to combine
expanded_biden <- expanded_biden %>% select(-"Trump_%")
expanded_trump <- expanded_trump %>% select(-"Biden_%")

# Combine the two data sets
combined <- rbind(expanded_biden,expanded_trump)
```

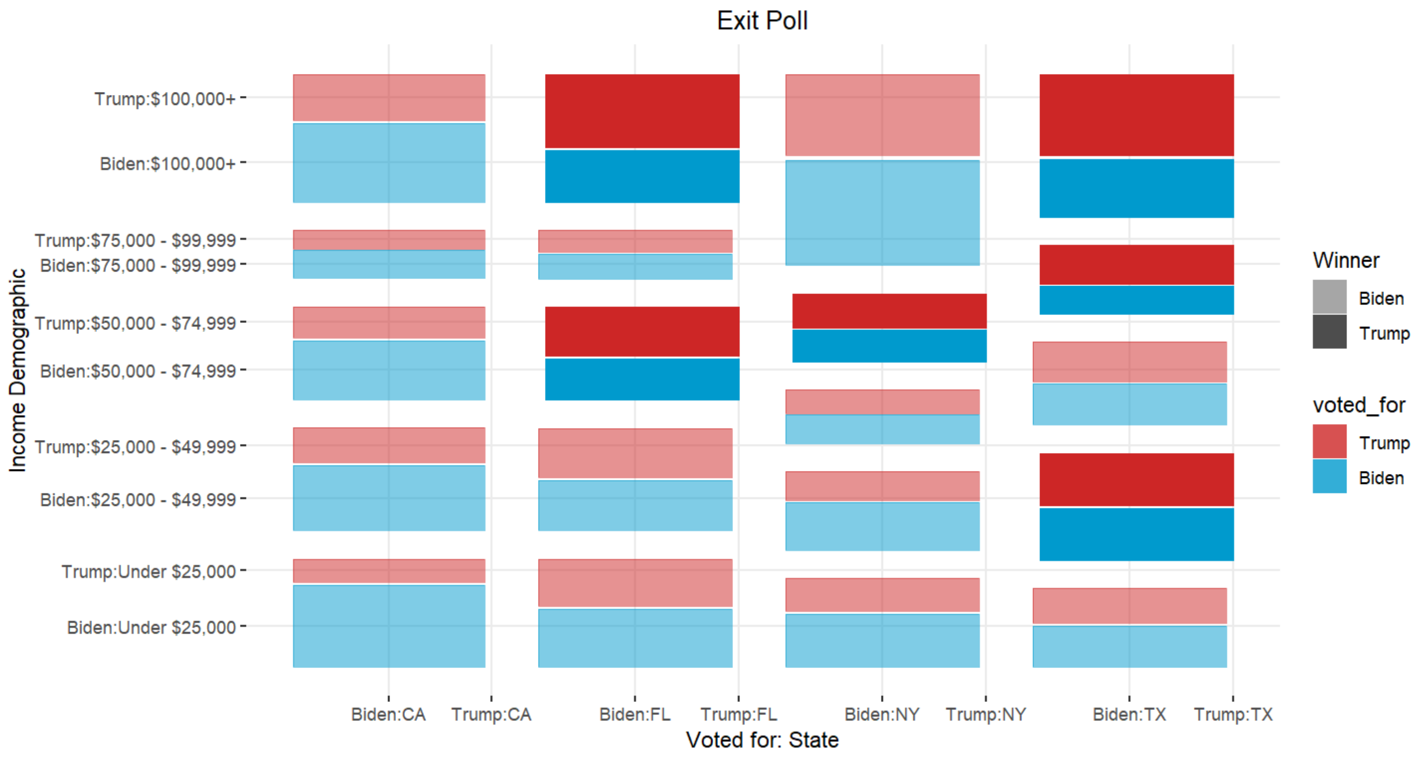
Now we are going to plot our data. This plot will show us what proportion of each demographic voted for Trump and Biden, the proportion each wage demographic makes up of the whole population, and how these varies by state.

```
# Library
library(ggmosaic)

# Reorder demographics into ascending order
combined$Demographic <- factor(combined$Demographic,
levels=c("Under $25,000", "$25,000 - $49,999", "$50,000 - $74,999",
"$75,000 - $99,999", "$100,000+"))

# Plot mosaic figure
(mosaic_plot <- ggplot(data = combined) +
    geom_mosaic(aes(x=product( Demographic, State_Abbr ),
                    fill = voted_for, colour = Demographic),
                    offset = 0.05) +
       theme(axis.text.x = element_text(angle = 90,
       hjust = 1, vjust = .5)) +
    labs(y="Income Demographic", x="Voted for: State",
    title = "Exit Poll") +
    scale_fill_manual(values = c("Trump" = "firebrick3",
    "Biden" = "deepskyblue3")) +
    theme_bw() + theme(panel.border = element_blank(),
                       panel.grid.minor = element_blank(),
                       plot.title = element_text(hjust = 0.5),
                       axis.line = element_blank())
  )
```



This plot is clear, informative and well labled. However, there is limitations to gg_mosaic. Mosaic plots idealy are able to show more catagorical variables than is displayed here. For example, by the use of transparency to distinguish between another variable. However when using this in gg_mosaic it causes complications. Try showing who was the overall winner in each group using the alpha argument in the geom_mosaic aesthetic.

```
# Plot mosaic figure
(mosaic_plot_alpha <- ggplot(data = combined) +
    geom_mosaic(aes(x=product( Demographic, State_Abbr ),
                    fill = voted_for, colour = Demographic, alpha = Winner,),
                    offset = 0.05) + # Alpha argument included
    scale_alpha_manual(values =c(.5,1)) +
    theme(axis.text.x = element_text(angle = 90, hjust = 1, vjust = .5)) +
    labs(y="Income Demographic", x="Voted for: State",
    title = "Exit Poll") +
    scale_fill_manual(values = c("Trump" = "firebrick3",
    "Biden" = "deepskyblue3"))+
    theme_bw() + theme(panel.border = element_blank(),
                       panel.grid.minor = element_blank(),
                       plot.title = element_text(hjust = 0.5),
                       axis.line = element_blank())
)
```

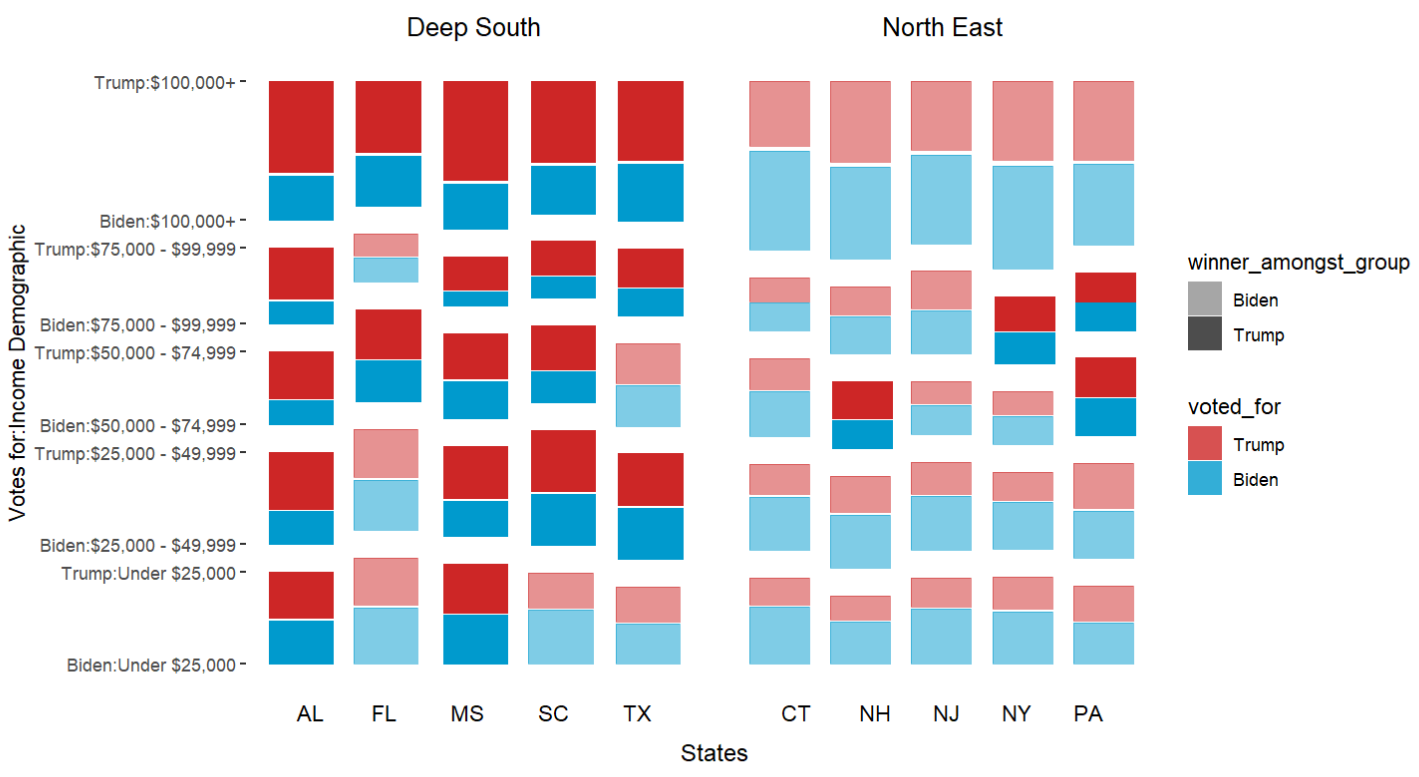


We now have a plot which displays four catagorical variables. However, a complication with gg_mosaic is that it is very hard to alter axis labels. This is due to the fact that although we know that variables are essentially discrete, in gg_mosaic they are continuous as this is how the proportionate sizes of boxes based on the count data are generated. This means it is not possible to customise axis, in an intuitive way. This has made our axis a little complex and comprimised the clarity of out plot, as is seen on the x-axis where there are redundant labels. In the next plot we will show one approach to tackling this problem.

If we wanted we could take this plot a step further and display a fifth categorical variable- region. We want to use a faceted approach but cannot use facet_grid() or facet_wrap() because in this case we want our x-axis to be diferent according to which states are found in a region. If we had equivilent data for the 2016 exit poll we could use facet_wrap() by year as the same x axis would be required. We will instead use a ‘work-around’ which creates two seperate plots based on region and combines them using grid.arrange(). We will repeat the previous code and add in another varible (region) to display. You will have to remove several aspects of the graphs in order to combine them cleanly. We have also improved the clarity of the x-axis labels, see if you can spot the way we got around the dificulites with changing axis labels in gg_mosaic.

```
# Repetition of the previous section but to allow for facetting
# according to region.


# Filter data set for only wage-bracket demographics
region_data <- exit_poll %>% filter(Demographic == "Under $25,000"  |
                               Demographic == "$25,000 - $49,999" |
                               Demographic == "$50,000 - $74,999" |
                               Demographic == "$75,000 - $99,999" |
                                     Demographic == "$100,000+")


# Filter deep south states
deep_south <- region_data %>% filter(State == "south-carolina" |
                                     State == "mississippi" |
                                     State == "florida" |
                                     State == "texas" |
                                     State == "alabama" )

# Filter north east states
north_east <- region_data %>% filter(State == "connecticut" |
                                     State == "new-hampshire" |
                                     State == "new-jersey" |
                                     State == "new-york" |
                                     State == "pennsylvania")


# Add region column
north_east$region <- "North East"
deep_south$region <- "Deep South"

# Combine the two data sets
region_data <- rbind(north_east, deep_south)


# Convert rows to count data for the percentage who voted for biden and
# the percentage who voted for trump.

# Create two new data frames in which to expand the rows, also deletes
# column where data was extracted from.
expanded_biden_region<- expandRows(region_data, "Biden_%")
expanded_trump_region <- expandRows(region_data, "Trump_%")

# Expand rows to show the proportion of the total population each
# demographic makes up
expanded_biden_region <- expandRows(expanded_biden_region, "proportion")
expanded_trump_region <- expandRows(expanded_trump_region, "proportion")

# Add new column name to distinguish Trump voters from Biden voters
expanded_trump_region$voted_for <- "Trump"
expanded_biden_region$voted_for <- "Biden"

# Remove names of column which has been deleted from the other data set
# in order to make them match, ready to combine
expanded_biden_region <- expanded_biden_region %>% select(-"Trump_%")
expanded_trump_region <- expanded_trump_region %>% select(-"Biden_%")

# Combine the two data sets
combined_region <- rbind(expanded_biden_region,expanded_trump_region)
```

```
deep_south_data <- subset(combined_region, region == "Deep South")
north_east_data <- subset(combined_region, region == "North East")

# North East plot
(mosaic_plot_ne <-north_east_data %>%
    arrange(Demographic) %>%
    mutate(Demographic=factor(Demographic, levels=c("Under $25,000",
    "$25,000 - $49,999", "$50,000 - $74,999", "$75,000 - $99,999",
    "$100,000+"))) %>%
    ggplot() +
    geom_mosaic(aes(x=product( Demographic, State_Abbr ),
                    fill = voted_for, colour = Demographic,
                    alpha = winner_amongst_group,), offset = 0.05) +
    scale_alpha_manual(values =c(.5,1)) +
    theme(axis.text.x = element_text(angle = 90, hjust = 1,
    vjust = .5)) +
    labs(x="CT        NH       NJ       NY       PA"      ,
    title = "North East") +
    scale_fill_manual(values = c("Trump" = "firebrick3",
    "Biden" = "deepskyblue3")) +
    theme_bw() + theme(panel.border = element_blank(),
                       panel.grid.minor = element_blank(),
                       panel.grid.major = element_blank(),
                       plot.title = element_text(hjust = 0.5),
                       axis.line = element_blank(),
                       axis.title.y=element_blank(),
                       axis.text.y=element_blank(),
                       axis.ticks.y=element_blank(),
                       axis.text.x=element_blank(),
                       axis.ticks.x=element_blank())
)

# Deep South plot
(mosaic_plot_ds <- deep_south_data %>%
    arrange(Demographic) %>%
    mutate(Demographic=factor(Demographic, levels=c("Under $25,000",
    "$25,000 - $49,999", "$50,000 - $74,999", "$75,000 - $99,999",
    "$100,000+"))) %>%
    ggplot() +
    geom_mosaic(aes(x=product( Demographic, State_Abbr ),
                    fill = voted_for, colour = Demographic,
                    alpha = winner_amongst_group,), offset = 0.05) +
    scale_alpha_manual(values =c(.5,1)) +
    theme(axis.text.x = element_text(angle = 90, hjust = 1,
    vjust = .5)) +
    labs(y="Votes for:Income Demographic",
    x = "AL        FL          MS          SC          TX",
    title = "Deep South") +
    scale_fill_manual(values = c("Trump" = "firebrick3",
    "Biden" = "deepskyblue3")) +
    theme_bw() + theme(panel.border = element_blank(),
                       panel.grid.minor = element_blank(),
                       panel.grid.major = element_blank(),
                       plot.title = element_text(hjust = 0.5),
                       axis.line = element_blank(),
                       #   axis.text.x = element_text(angle = 90,
                       vjust = 0.5, hjust=1),
                       axis.text.x=element_blank(),
                       axis.ticks.x=element_blank(),
                       legend.position = "none")
)

# Arrange both plots together to give the appearance of a facet
grid.arrange(mosaic_plot_ds, mosaic_plot_ne, ncol=2, bottom = "States")
```



We now have a plot which clearly displays 5 catagorical variables. Ideally this would contain a figure caption, perhaps giving the full names of each state instead of just the abbreviations. We could do this when knitting into a PDF for example using fig.cap = “”.

Plotly: An introduction to interactive visualisation

Plotly is a package which acts as an interface to the plotly javascript graphing library. It wraps javascript for multiple coding languages including R, Python and Matlab. It produces interactive and animated graphics of browser/ html based charts and visualisations with little code.

Although plotly is relitively simple to use, it is made even simplier by being able to improve existing ggplot code using ggplotly().

Interactive visualisations are beocming increasingly popular especially within popular web based news media. Interactive visuals have a particular use in displaying high density or multi-dimentional data and allowing users to gain a better insight into the data and hone-in on specific data points using hover info or focusing on subsets of data by selecting or deselecting groups or a they are curious about.

However, we must remember that interactivity alone does not make a good graphic. We must always refer to the best practices of data visualisation and design principles and know that static plots are better suited to certain scenarios. For example static plots are best for reports which emphasise what you the creator has highlighted.

Parallel coordinate plots: Multiple continuous variable

Parallel coordinate plots can display a large number of continuous variables along the x axis, each with its own vertical axis that has its own scale. Data is then plotted as series of lines which connect across each axis. Groups can be distinguished using colour and then distinct relationships can be seen by patterns in their response to variables.

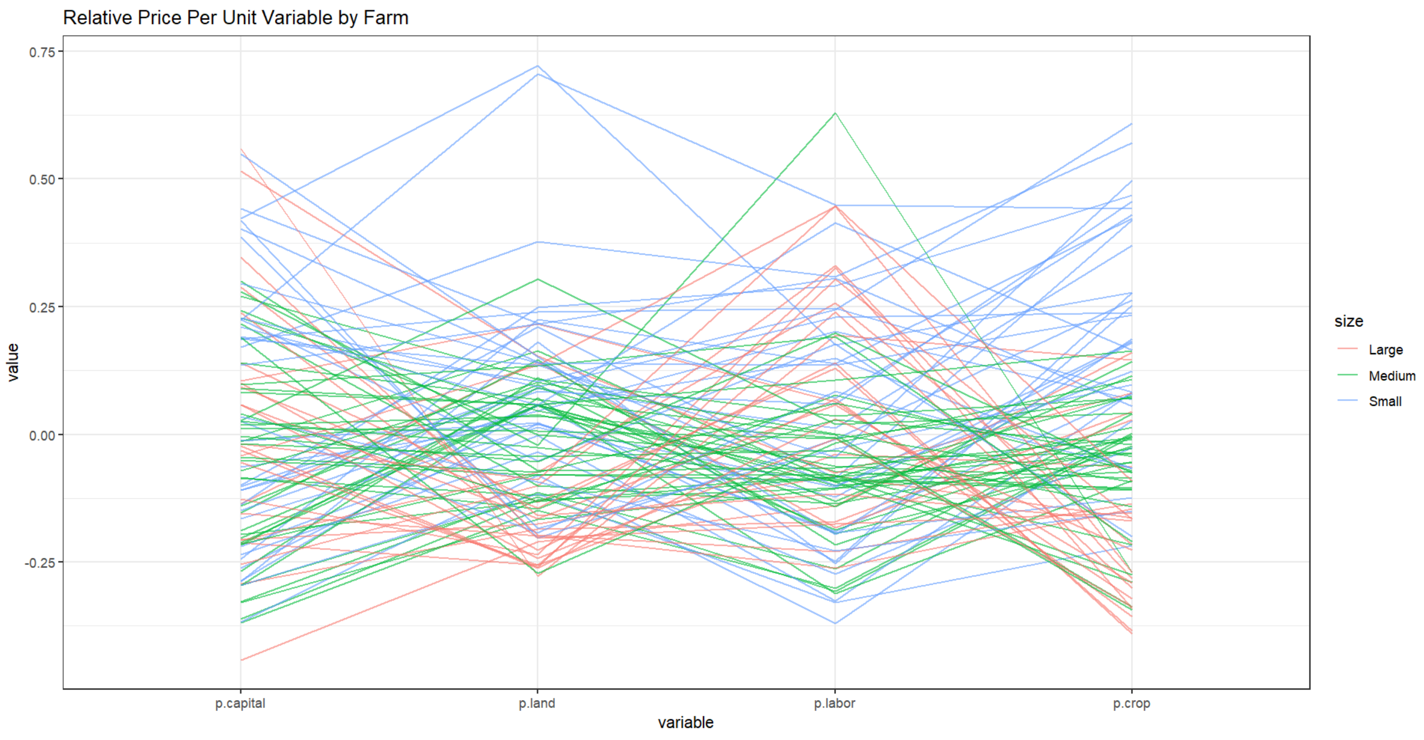
We will use agricultural data related to the realtive prices of variables such as labour and land, to explore how parallel coordinate plots can be used to display multiple continuous variables in the same plot.

```
# Clear environment
rm(list=ls())
```

```
# Load libraries
library(GGally)

# Read in agricultural data
agri_data<- read.csv("data/us_agri.csv")

# Investigate data frame structure and class of each variable
str(agri_data)
```

We can see that we have 4 continuous variables and a catagorical variable which groups the rows into size of farm. Now we are going to plot parallel coordinates plot to display all of the varibles.

```
# Plot parallel coordinates plot
(parcoord_plot <- ggparcoord(agri_data,
           columns = 1:4, groupColumn = 5, # Variable columns on x axis
           # and which to group lines by
           scale = "center", # Standardise and center variables
           title = "Relative Price Per Unit Variable by Farm", # Title
           alphaLines = 0.6)) + # Opacity of lines
           theme_bw()) # Opacity of lines
```



This plot is not useful for extracting specific values but is effective for viewing overall trends in the data. A question which may be posed by looking at this graph is why do small farms generally own land which is valued higher than larger farms? We used the colours red, green and blue to seperate each catagory, an improvement to this graph would be to consider a colour blind friendly colour palet. This style of plot can become confusing and unclear when more rows of data are added.

Plotly: Multiple continuous variables

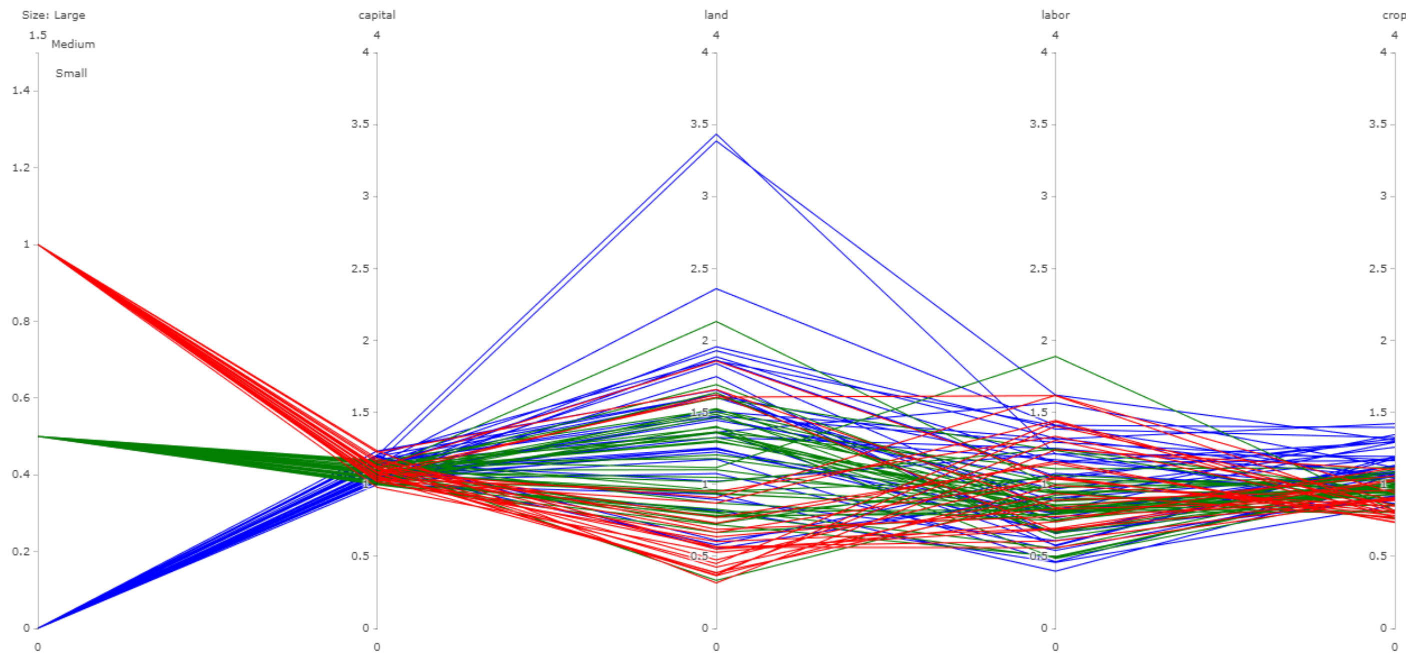
With plotly we can plot the same style of graph but allow for viewer interaction which can help to make trends clear by isolating parameters or regions of interest. However, Plotly parellel coordinate plots are not perfect. They struggle with customisation of labels, legends and catagorical data. We have implemented two ‘work-arounds’ in this code to overcome some of these problems. The first requires us to assign each level of the ‘size’ catagory a numerical value.

```
# Load libraries
library(plotly)
library(tidyverse)

# Add column giving 'size' numerical values for parcoords plot
agri_data$size_plotly <- ifelse(agri_data$size == "Large", 1,
                             ifelse(agri_data$size == "Small", 0, 0.5))
```

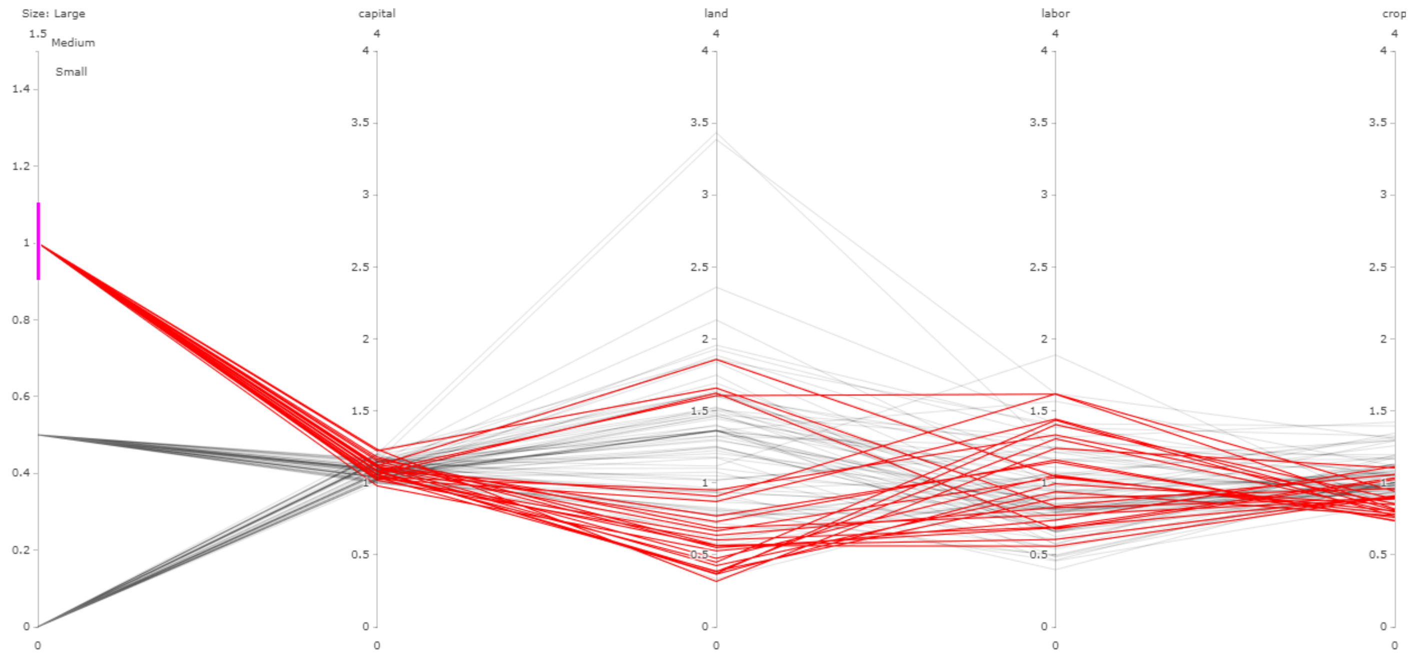
We will now plot the parallel coordinate graph. The numerical size column is a work-around to assign each level a colour and allows us to add a make-shift legend to seperate the levels. To do this we add size as an extra variable and use our assinged numerical values to distinguish each level. We will use the label of this column as a make-shift legend.

```
# Plot  a plotly parallel coordinate plot
plotly_parcoord <- agri_data %>%
  plot_ly(type = 'parcoords', line = list(color = ~size_plotly,
  # Distinguish sizes into three colours
                                          colorscale = list(c(0,'blue'),
                                          c(0.5,'green'),c(1,'red'))),
                                dimensions = list( # Define the scale
                                # ranges of each variable
                        list(range = c(0,1.5),
                             label = '         Size: Large \r\n
                    Medium \r\n
                 Small', values = ~size_plotly), # A make-shift legend
                        list(range = c(0,4),
                             label = 'capital', values = ~p.capital),
                        list(range = c(0,4),
                             constraintrange = c(0,4),
                             label = 'land', values = ~p.land),
                        list(range = c(0,4),
                             label = 'labor', values = ~p.labor),
                        list(range = c(0,4),
                             label = 'crop', values = ~p.crop))
          )

plotly_parcoord # Call plot
```

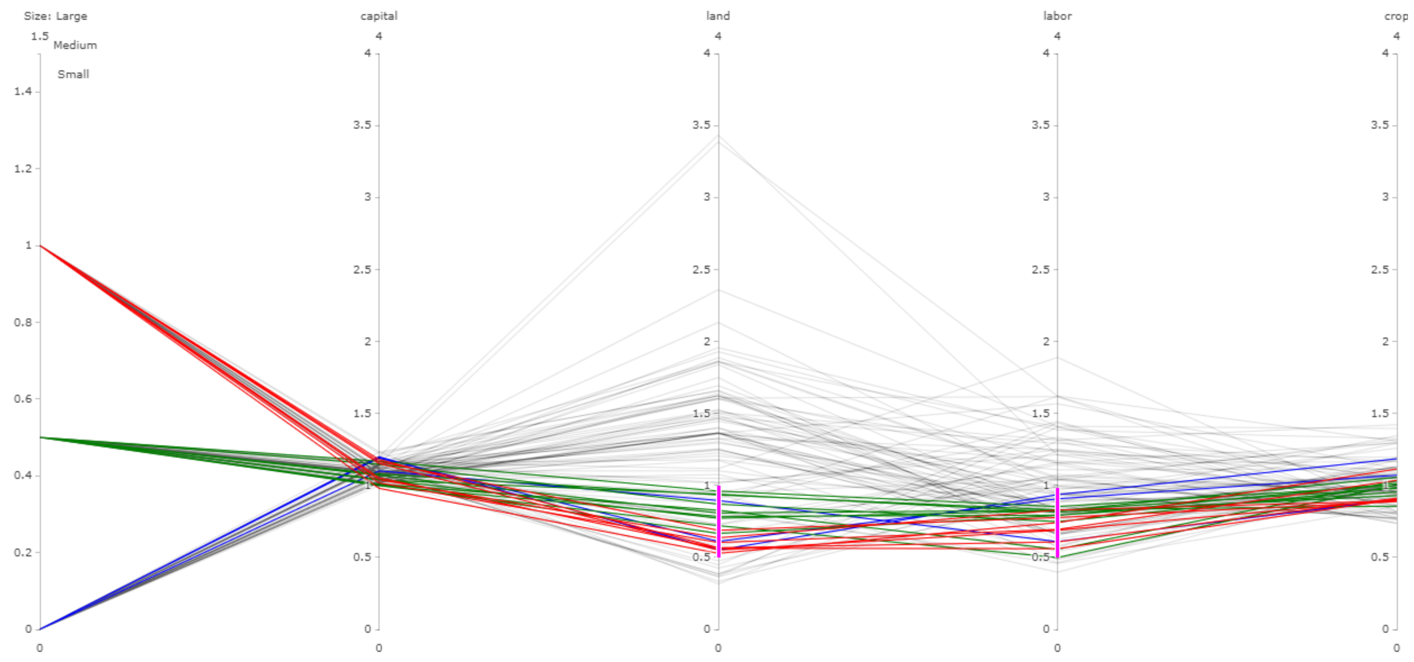
Output:



Example of how each level can be filtered:



Example of how mutiple conditions can be set to isolate data:



Faceted plots: A Mixture of catagorical and continuous

Facet plots are figures made up of mutiple subpplots based on a catagorical variable. This creates a panel-like plot with each subplot displaying the same variables but with diferent groupings. Facet plots are also useful for dealing with catagorical variables with a large number of levels. This is done by grouping these levels and facetting by these groups.

We will use 2015 US car theft data to explore how facet plots can be used to display a mixture of continuous and catagorical variables in the same plot.

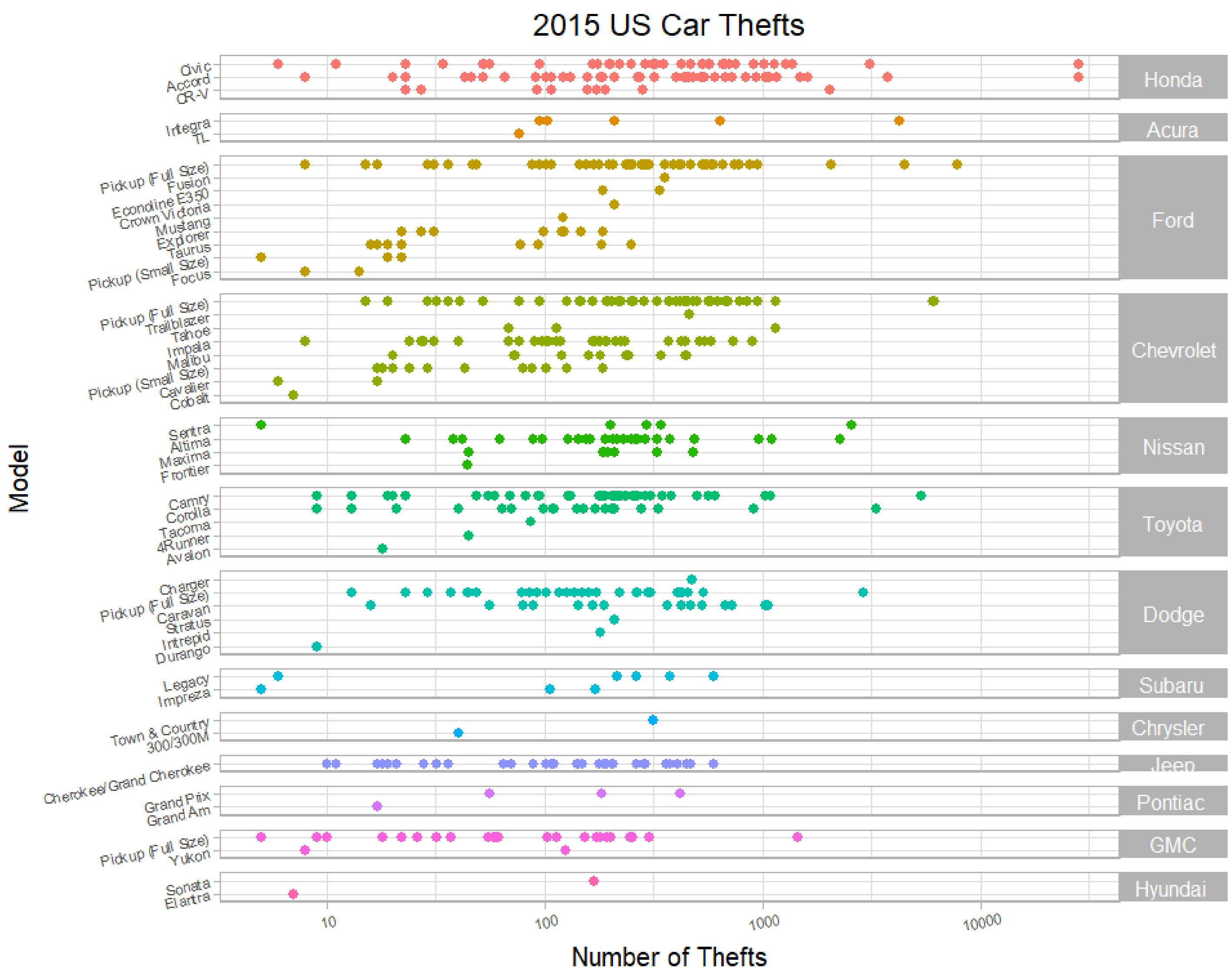
```
# clear environment
rm(list=ls())

# libraries
library(readxl)
library(tidyverse)
library(reshape2)

# read in car thefts data
theft_data <- read_excel("data/2015_State_Top10Report_wTotalThefts.xlsx")

# view unique values in each column
ulst <- lapply(theft_data, unique)
ulst
```

We can see that the make_and_model column is has 48 levels and provides us with two bits of infomation which could be displayed clearer and lends itself to a grouping varible. So, we seperate the columns into make and model, make being the variable we will use to group.

```
# Create new data frame with make and model extracted and seperated
make_model_sep <- colsplit(theft_data$make_and_model," ",c("make",
"model"))
# Bind data frames back together
theft_data <- cbind(make_model_sep, theft_data)
# Remove original make_and_model column
theft_data <- select(theft_data, -make_and_model)
```

Now we are going to plot our data. Our plot will display infomation about the make, model and number of thefts.

```
# Plot car thefts facetted by make

# Order from highest thefts to lowest, between models and in overall
# facet plot
theft_data$model <- reorder(theft_data$model, theft_data$thefts)
theft_data$make <- reorder(theft_data$make, -theft_data$thefts)
# plot

(theft_facet <- ggplot(theft_data, aes(x = thefts, y = model,
colour = make )) +
    geom_point() +
    ggtitle("2015 US Car Thefts") +
    scale_x_continuous(trans = "log10", name = "Number of Thefts") +
    scale_y_discrete(name = "Model") +
    facet_grid(make ~ ., scales = "free", space = "free") +
    theme_light() + theme(text=element_text(size=7, angle=12),
                          strip.text.y = element_text(angle = 0,
                          size = 8),
                          legend.position="none",
                          title = element_text(angle = 0, size =10),
                          plot.title = element_text(hjust = 0.5))
)
```



Plotly: A Mixture of catagorical and continuous

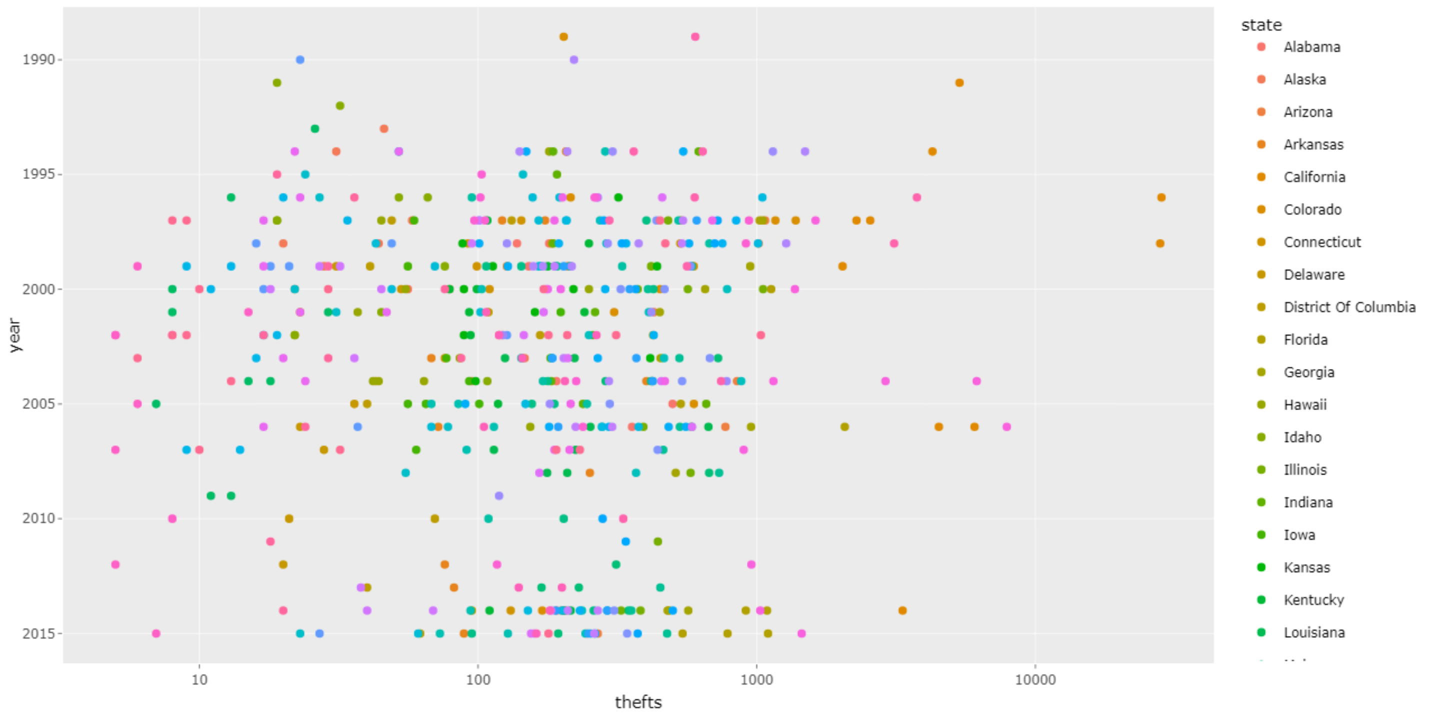
Now, using facet_grid() we have managed to display an extra variable. We used the grouping variable ‘make’ as to prevent there being too many levels to our catagories. This would have resulted in an extremely large and unmanagable plot. If we wanted to display the ‘state’ catagory, we could have created groupings by region with which to group our facets by.

Although this is an effective plot, our data frame still has three more variables we might be intersting in displaying. We could do this by presenting multiple figures but this might become confusing for our audience. We are going to use the package `plotly` to help us display extra variables and improve our ability to explore data within managable figures.

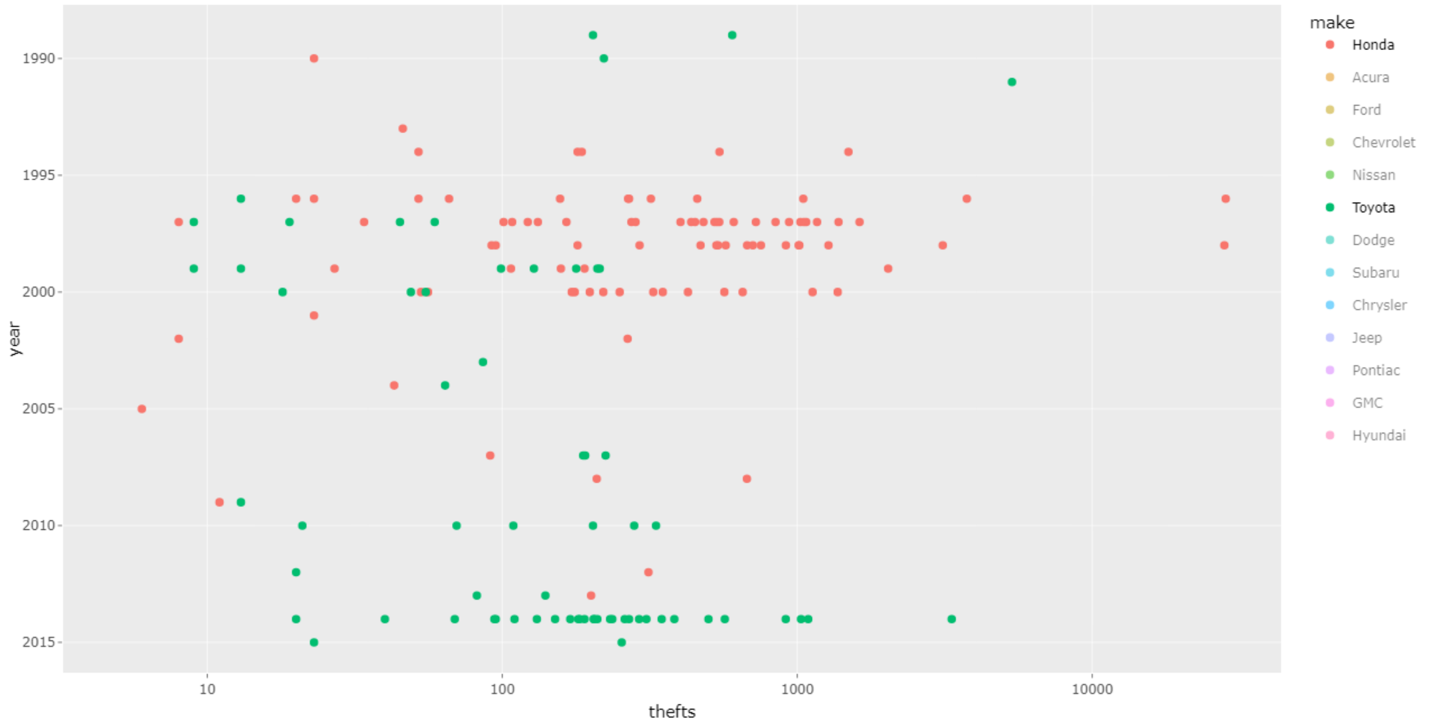
We will plot two seperate figures, one with a legned which allows filtering of states and the other for car make. Idealy we could incorporate these two legends into a single plot however, plotly does not support this.

```
# Load library
library(plotly)

# Plot thefts against year, coloured by make
(make_plot <- theft_data %>%
    ggplot(aes(label = model, label_2 = rank, label_3 = state)) +
    geom_point(aes(x = thefts, y = year, colour = make)) +
    scale_x_continuous(trans = "log10") +
    scale_y_reverse()
)
# Convert to ggplotly
ggplotly(make_plot)

# Plot thefts against year, coloured by state
(state_plot <- theft_data %>%
    ggplot(aes(label = model, label_2 = rank, label_3 = state)) +
    geom_point(aes(x = thefts, y = year, colour = state)) +
    scale_x_continuous(trans = "log10") +
    scale_y_reverse()
)
# Convert to ggplotly
ggplotly(state_plot)
```

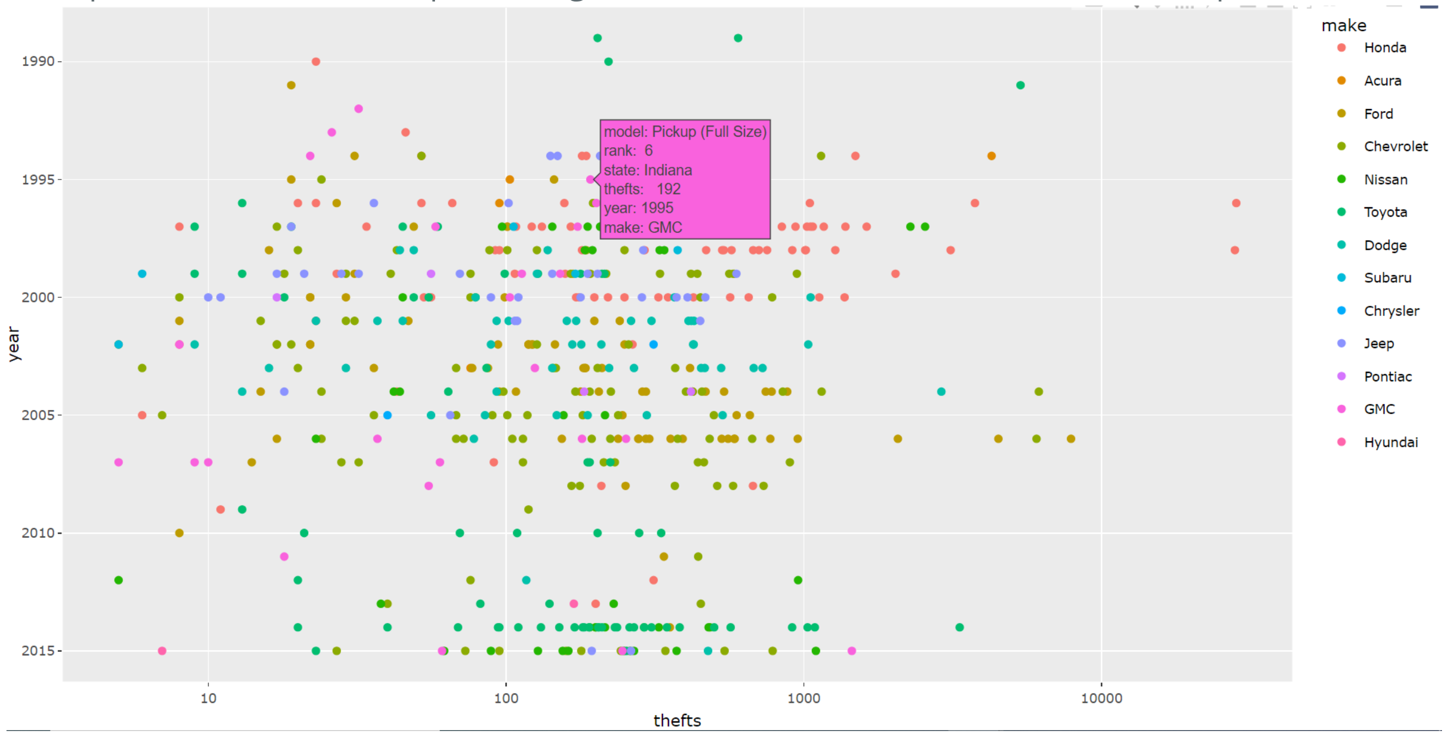
Output of the state filtered plot:



Output of the make filtered plot showing how groups can be isolated.



Example of hover-over tooltips which give additional infomation about each data point.



In these plots, all 6 of the variables are available to the viewer to find infomation about by hovering over data points. Data can be explored by selecting and deselecting data points and particualr regions can be zoomed in on. All items in the legend can be selected and deselected by double clickling.

Conclusion

Well done for getting through the tutorial! Here's a summary of what you have learned:

- Why multi-dimentional data is hard to plot

- How to approach the visualisation of multi-dimentional data through the conisderation of its class and stucture
- How and when to incorporate interactivity into graphs and the usefullness and limitations it has

Feedback on the tutorial is very welcome. If you have any comments or questions related to this tutorial feel free to contact me at s1869354@ed.ac.uk

Check out our Useful links page where you can find loads of guides and cheatsheets.

If you have any questions about completing this tutorial, please contact us on ourcodingclub@gmail.com

We would love to hear your feedback on the tutorial, whether you did it in the classroom or online!

- Follow our coding adventures on Twitter!

tutorial-sroles1 is maintained by EdDataScienceEES.
This page was generated by GitHub Pages.