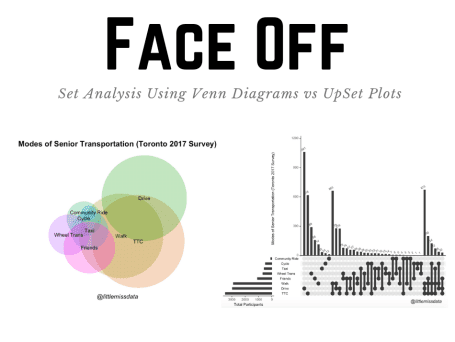
It’s time for me to come clean about something; I think Venn diagrams are fun! Yes that’s right, I like them. They’re pretty, they’re often funny, and they convey the straight forward overlap between one or two sets somewhat easily.I considered sharing with y’all how to create Venn diagrams in R. But I couldn’t do that in good conscious without showing an alternative for larger and more complex set analysis.

Folks, what you are about to witness is a set analysis face off! We will be pairing off Venn diagrams and UpSet plots in a variety of scenarios for a true battle royale. Winner takes all and is able to claim the prize of set analysis master.



**Working Environment**

For this tutorial, we are going to be using R as our programming language.

**Round 1: Tiny and Fun Set Intersections**

Kind folks, this is our warm up. In this round, we will be creating some fun and simple set intersections. Specifically, we will just be creating a very important graph which describes why I love Twitter.

To get started, we are going to install and load the packages required for this tutorial. If you do not have the packages already installed, please uncomment the install.packages() commands by removing the hashtag(#).

**Install and Load Packages**

# install.packages("rJava")

# install.packages("UpSetR")

# install.packages("tidyverse")

# install.packages("venneuler")

# install.packages("grid")

library(rJava)

library(UpSetR)

library(tidyverse)

library(venneuler)

library(grid)

**Format the data**

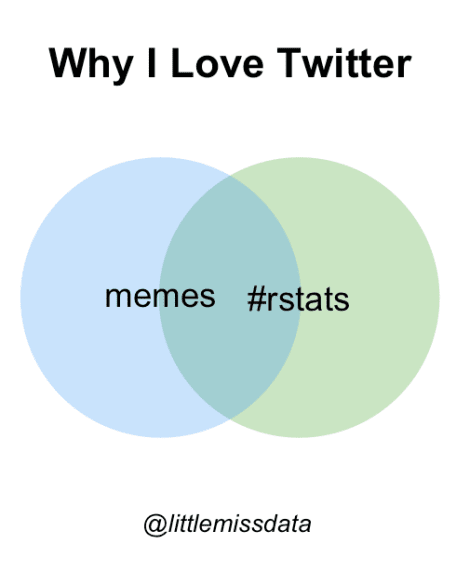
We will create a basic list which specifies the values of each of the circles and their overlap.

# Set the chart data

expressionInput <- c(`#rstats` = 5, memes = 5, `#rstats&memes` = 3)

**Create a Venn diagram**

To create a simple Venn diagram, you can just pass in the list with the specified set and overlap values into the venneuler() function. The remaining code is just formatting to set the font size, title and subtitle.



# Create the Venn diagram

# note on set up for java v11 jdk (v12 does not work with this)

myExpVenn <- venneuler(expressionInput)

par(cex=1.2)

plot(myExpVenn, main = "Why I Love Twitter")

grid.text(

"@littlemissdata",

x = 0.52,

y = 0.2,

gp = gpar(

fontsize = 10,

fontface = 3

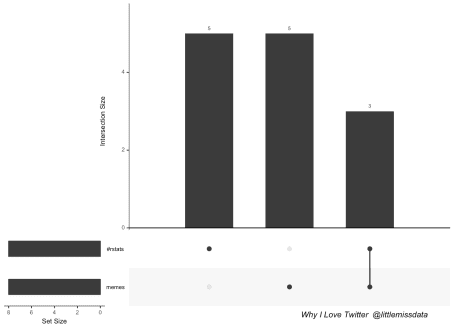
)

)

**Create an UpSet Plot**

The great thing is that we can also create an UpSet plot using the same basic expression list. You simply pass the fromExpression() function into the upset() function. The remaining code is to format the labels and font size.

**How to read an UpSet plot:** UpSet plots offer a straight forward way for us to view set data by frequency. On the bottom left hand side horizontal bar chart, we show the entire size of each set. In this case, each set is of size 8. The vertical bar chart on the upper right hand side shows the sizes of isolated set participation. In the example, 5 values only belong to the #rstats set or only belong to the memes set. 3 values belong to both sets.



# Create an UpsetR Plot

upset(fromExpression(expressionInput), order.by = "freq")

grid.text(

"Why I Love Twitter @littlemissdata",

x = 0.80,

y = 0.05,

gp = gpar(

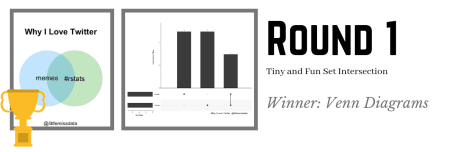
fontsize = 10,

fontface = 3

)

)

While the UpSet graph is an exciting new addition to our set analysis, I’m going to have to give this round to Venn diagrams. When trying to represent simple and easy to understand information, Venn diagrams are more visually appealing.



**Round 2: Complicated Sets**

Coming off of the round 1 win, Venn diagram may be feeling quite confident. However, the stakes are getting higher and we need to expect more of our visualizations in this round. We have more sets and interactions to visualize and more data to work with.

**Data Introduction**

The data is created using the [2017 Toronto Senior Survey](https://www.toronto.ca/city-government/data-research-maps/open-data/open-data-catalogue/community-services/#9ece3c85-08c9-097d-f4c8-bb7374fea6c1) from the [Toronto Open Data Catalogue](https://www.toronto.ca/city-government/data-research-maps/open-data/open-data-catalogue/). I feel strongly that data should be available to the people that pay for it.

This data set shows the output of a [2017 senior citizen survey](https://www.toronto.ca/city-government/data-research-maps/open-data/open-data-catalogue/community-services/#9ece3c85-08c9-097d-f4c8-bb7374fea6c1) to identify various needs of Toronto’s seniors’ population, in order to better inform decision making. To make our data processing easier, I have stripped down the columns that we will use and have performed a little pre-formatting. Please see below for a data dictionary and outline of what was changed.

| **Column** | **Source Column** |
| --- | --- |
| ID | Not previously included. This is a new unique key column. |
| physicalActivity | Survey Question: “1. In the past 3 months, how often did you participate in physical activities like walking?” |
| physicalActivityPerMonth | Survey Question: “1. In the past 3 months, how often did you participate in physical activities like walking?”. This has been transformed into numerical format. |
| volunteerParticipation | Survey Question: “5. During the past 3 months, how often did you participate in volunteer or charity work?” |
| volunteerPerMonth | Survey Question: “5. During the past 3 months, how often did you participate in volunteer or charity work?”. This has been transformed into numerical format. |
| difficultFinancial | Survey Question: “9. In the last year, have you had difficulty paying your rent, mortgage, Hydro bill, or other housing costs? For example, have you had to go without groceries to pay for rent or other monthly housing expenses?” |
| supportSystem | Survey Question: “13. Do you have people in your life who you can call on for help if you need it?” |
| postalCode | “Survey Question: 14. What are the first three characters of your postal code?” |
| employmentStatus | Survey Question: “15. What is your current employment status?” |
| sex | Survey Question: “16. What is your sex/gender?” |
| primaryLanguage | Survey Question: “18. In what language(s) would you feel most comfortable to receive services?” (first option listed) |
| ageRange | Survey Question: “19. Which age category do you belong to?” |
| ttcTransportation | Survey Question: “6. To get around Toronto, what modes of transportation do you use frequently? [TTC (bus, subway, or streetcar)]” |
| walkTransportation | Survey Question: “6. To get around Toronto, what modes of transportation do you use frequently? [Walk]” |
| driveTransportation | Survey Question: “6. To get around Toronto, what modes of transportation do you use frequently? [Drive]” |
| cycleTransportation | Survey Question: “6. To get around Toronto, what modes of transportation do you use frequently? [Cycle]” |
| taxiTransportation | Survey Question: ” 6. To get around Toronto, what modes of transportation do you use frequently? [Taxi or Uber]” |
| communityRideTransportation | Survey Question: “6. To get around Toronto, what modes of transportation do you use frequently? [Community Transportation Program, for example Toronto Ride or iRIDE]” |
| wheelTransTransportation | Survey Question: “6. To get around Toronto, what modes of transportation do you use frequently? [Wheel-Trans]” |
| friendsTransportation | Survey Question: “6. To get around Toronto, what modes of transportation do you use frequently? [Rides from family, friends or neighbours]” |
| ageRange | Survey Question: “19. Which age category do you belong to?”. |
| minAgeRange | Survey Question: “19. Which age category do you belong to?”. This has been converted to numerical format, taking the lowest age as the value. |

**Bring in the Data**

We will start by bringing in the data, replacing the NA’s and renaming the columns for easier display.

rawSets <- read.csv(

file = "https://raw.githubusercontent.com/lgellis/MiscTutorial/master/sets/seniorTransportation.csv",

header = TRUE, sep = ",", stringsAsFactors = FALSE

)

# Replace the NA's

rawSets[is.na(rawSets)] <- 0

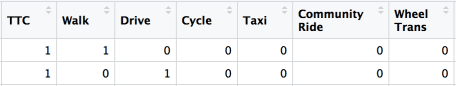
# Rename the columns for easier display

sets <- rawSets %>%

rename(TTC = ttcTransportation, Walk = walkTransportation, Drive = driveTransportation, Cycle = cycleTransportation, Taxi = taxiTransportation, `Community Ride` = communityRideTransportation, `Wheel Trans` = wheelTransTransportation, Friends = friendsTransportation)

dim(sets)

head(sets)



The data comes with the sets in the form of a binary matrix.

**Create a Venn Diagram**

Now it’s time to create our Venn diagram. The data is currently in the form of a binary matrix, but to pass it into the venneuler() function, we need to get it into a list of set, ID pairs.

# Prep the data for a Venn diagram

vennSets <- sets %>%

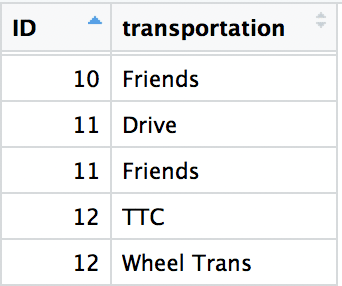
gather(transportation, binary,6:13) %>% # take all binary mappings and convert to be a the set indicator

filter(binary == 1) %>% # only include set matches

select(ID, transportation) %>% # only include ID and set category

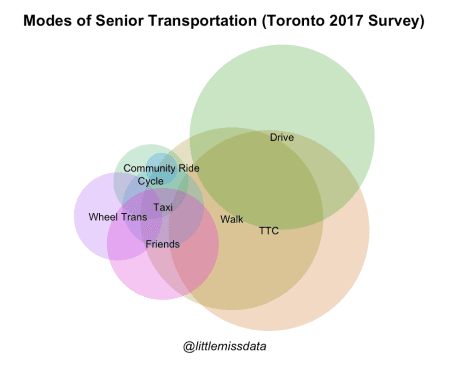
mutate(transportation = factor(transportation)) # set the transportation column as a factor

dim(vennSets)



The data has been transformed to have one set column and one ID column. An ID can be repeated for every set it belongs to.

Create the Venn diagram by passing the data frame into the venneuler() function. The rest of the code is for labelling and formatting.



v <- venneuler(data.frame(vennSets))

#Note that if you need to move around the labels so that they are not overlapping, you can use the new line breaks like the example below.

#v$labels <- c("TTC", "Walk", "Drive", "Cycle\n\n\n", "\nTaxi", "Community Ride", "Wheel Trans", "Friends")

par(cex = 0.7)

plot(v, main = "Modes of Senior Transportation (Toronto 2017 Survey)", cex.main = 1.5)

grid.text(

"@littlemissdata",

x = 0.52,

y = 0.15,

gp = gpar(

fontsize = 10,

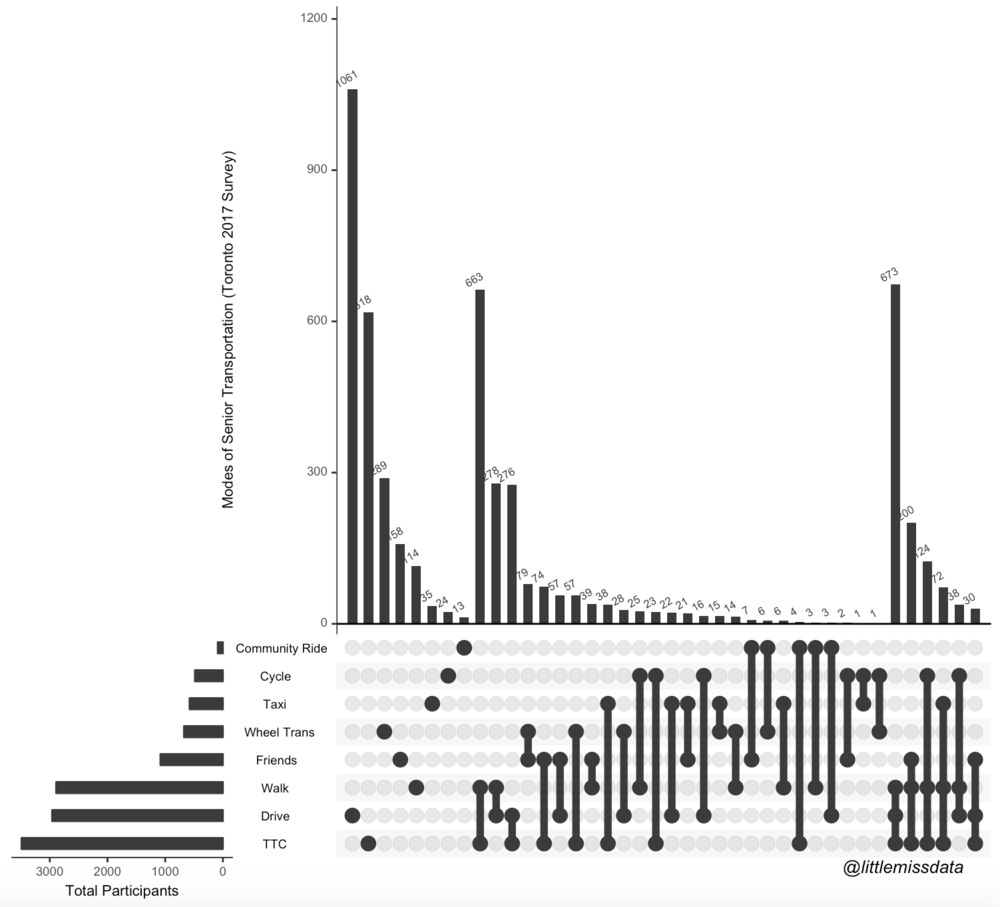
fontface = 3

)

)

**Create an UpSet Plot**

Create an UpSet plot by passing the original binary matrix into the upset() function. Other than the upset() function, the rest of the code is for labels and formatting.



upset(sets,

nsets = 10, number.angles = 30, point.size = 3.5, line.size = 2,

mainbar.y.label = "Modes of Senior Transportation (Toronto 2017 Survey)", sets.x.label = "Total Participants"

)

grid.text(

"@littlemissdata",

x = 0.90,

y = 0.05,

gp = gpar(

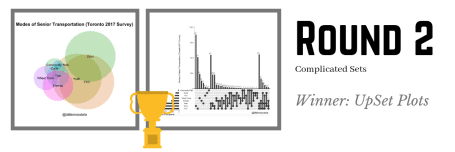
fontsize = 10,

fontface = 3

)

)

Unfortunately, I think when the stakes got higher, Venn diagrams just could not keep up. While I think the Venn diagram is quite pretty, I really can’t make much sense out of it. The clarity provided by the UpSet plot can’t be matched. Round 2 goes to UpSet plots!



**Round 3: Explore In Context Set Information**

We are all tied up as we enter round 3, and it’s time to raise the stakes. In this round, we want to explore information about other variables in the data set within the context of the sets.

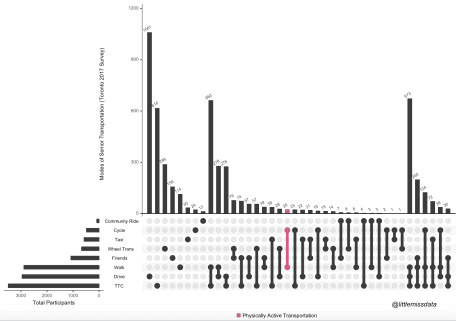
**Provide Context with Plot highlighting**

We will start by using colors to highlight specific areas of the graph that we care about.

**Highlight Seniors Who Both Walk and Cycle Using “Query=Intersects”**

UpSet plots have a very cool parameter called queries. Queries can be used to define a subset of the data that you would like to highlight in your graph. The queries property takes in a list of query lists which means that you can pass multiple queries into the same graph. Each query list allows you to set a number of properties about how the query should function.

In this example we are viewing the Cycle and Walk set intersection (query and params). We want the query to be highlighted in a nice pink (color). We want to display the query as a highlighted overlap (active) and we will give it a name that we add to the chart legend (query.name)



upset(sets,

query.legend = "bottom", nsets = 10, number.angles = 30, point.size = 3.5, line.size = 2,

mainbar.y.label = "Modes of Senior Transportation (Toronto 2017 Survey)", sets.x.label = "Total Participants",

queries = list(

list(

query = intersects,

params = list("Cycle", "Walk"),

color = "#Df5286",

active = T,

query.name = "Physically Active Transportation"

)

)

)

grid.text(

"@littlemissdata",

x = 0.90,

y = 0.05,

gp = gpar(

fontsize = 10,

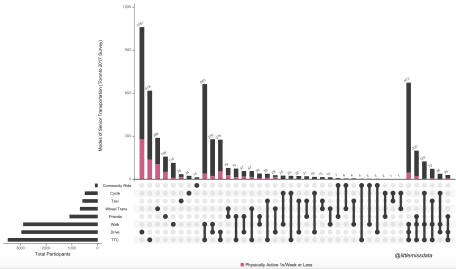
fontface = 3

)

)

**Highlight Seniors Who Exercise 1x/Week or Less Using “Query=Elements”**

In our next example, we are looking to highlight other data in the data frame within the context of the sets. In the normal UpSet graph, we want to highlight the rows identified as physically active less than 1x/week or less (queries, params) across all sets. We want the query to be highlighted in a nice pink (color). We want to display the query as a highlighted overlap (active) and we will give it a name that we add to the chart legend (query.name)



upset(sets,

query.legend = "bottom", nsets = 10, number.angles = 30, point.size = 3.5, line.size = 2,

mainbar.y.label = "Modes of Senior Transportation (Toronto 2017 Survey)", sets.x.label = "Total Participants",

queries = list(

list(

query = elements,

params = list("physicalActivityPerMonth", 0,4),

color = "#Df5286",

active = T,

query.name = "Physically Active 1x/Week or Less"

)

)

)

grid.text(

"@littlemissdata",

x = 0.90,

y = 0.05,

gp = gpar(

fontsize = 10,

fontface = 3

)

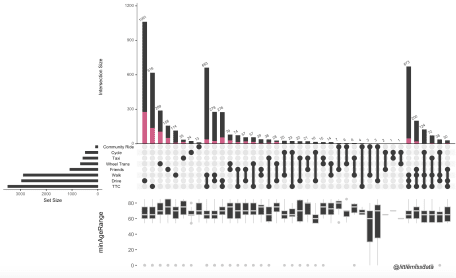
)

**Provide Context with Additional Graphs Called “Attribute Plots”**

Beyond highlighting within the UpSet main graph, we also have the option of bringing in additional plots which can display information about other variables within the context of the sets.

**Display an in context box plot of age for each set using boxplot.summary() function**

In our next example, we are looking to display a boxplot of the minimumAgeRange for every single set. We can do this very easily by just passing in the boxplot.summary parameter with the variable that we would like to summarize.



upset(sets,

query.legend = "bottom", nsets = 10, number.angles = 30, point.size = 3.5, line.size = 2,

queries = list(

list(

query = elements,

params = list("physicalActivityPerMonth", 0,4),

color = "#Df5286",

active = T,

query.name = "Physically Active 1x/Week or Less"

)

),

boxplot.summary = c("minAgeRange")

)

grid.text(

"@littlemissdata",

x = 0.90,

y = 0.05,

gp = gpar(

fontsize = 10,

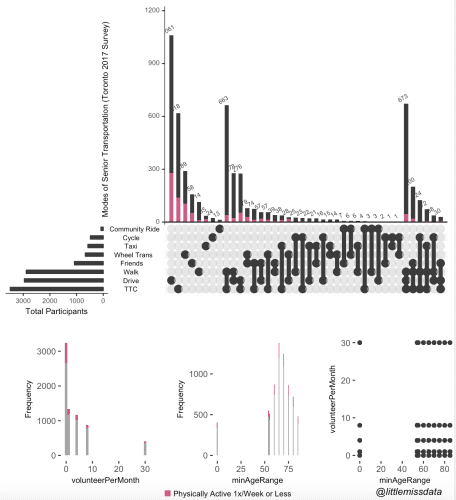
fontface = 3

)

)

**Using “Attribute Plots” Display In-Context Aggregate Statistics for Other Columns**

Like queries, UpSet plots also allow you to pass in a list of attribute.plots which can display additional graphs depicting the full data frame within the context of your sets. In the example below, we keep our “Physically Active 1x/Week or Less” query and add three attribute plots; 2 histograms and a scatterplot. All have been set to also carry the query highlighting throughout these new plots.



upset(sets,

query.legend = "bottom", nsets = 10, number.angles = 30, point.size = 3.5, line.size = 2,

mainbar.y.label = "Modes of Senior Transportation (Toronto 2017 Survey)", sets.x.label = "Total Participants",

queries = list(

list(

query = elements,

params = list("physicalActivityPerMonth", 0,4),

color = "#Df5286",

active = T,

query.name = "Physically Active 1x/Week or Less"

)

),

attribute.plots = list(gridrows = 50,

plots = list(list(plot = histogram, x = "volunteerPerMonth", queries = T),

list(plot = histogram, x = "minAgeRange", queries = T),

list(plot = scatter\_plot, x = "minAgeRange", y="volunteerPerMonth", queries = F)

),

ncols = 3

)

)

grid.text(

"@littlemissdata",

x = 0.9,

y = 0.02,

gp = gpar(

fontsize = 10,

fontface = 3

)

)

**Display Information About the Categories With “Set Metadata”**

Finally, we can use the set.metadata parameter to display aggregate statistics about the core sets. It is quite simple to implement. We start by creating a data frame with summarized set statistics. We need to convert from binary format to list format, and then we will aggregate and summarize the variable values grouping by the sets.

In this example we are going to display the average physical activity per month of each set.

aggregate <- sets %>%

gather(transportation, binary,6:13) %>%

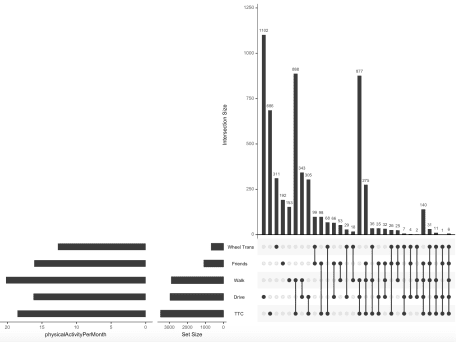
filter(binary == 1) %>% # only include set matches

group\_by(transportation) %>% #get summary stats per transportation category

summarize(physicalActivityPerMonth = mean(physicalActivityPerMonth))

aggregate

Now that the hard part is done, we simply specify the set.metadata parameter to have the aggregate data set and we are ready to get our set summary data on the bottom left hand plot.



upset(sets, set.metadata = list(data = aggregate, plots = list(

list(

type = "hist",

column = "physicalActivityPerMonth",

assign = 50

)

)))

By now may be wondering why we haven’t been talking about Venn diagrams in round 3. Simply put, they had to sit out of this round. While you could do some creative ideas to display context through color, it really isn’t on a comparable level to UpSet charts. As such, Venn diagrams are disqualified and I need to give this round to UpSet charts!

