Self Guided Study

Zahlen Zbinden

2024-03-18

Summary

- Skills Gained: geom_segment, geom_hline, geom_density_ridges, geom_sankey, make_long, fct_reorder, fct_relevel, facet_wrap, melt, ggplot, theme, element_text, element_blank, element_rect, element_line, element_point, element_path, element_polygon, element_ribbon, element_tile, element_rect

Reflection

My planned learning goals were met and exceeded, as I was not only able to find out about new plot and geometry styles, but I was able to discover new function and enhanced data pipeline techniques to improve and continue my learning.

Future Goals

Continue to learn new and interesting plotting techniques in ggplot.

During this exploration I will be looking at ggplot2, and some of the geoms, and interesting combinations of geoms that we didn't cover in class.

Common themes.

It is possible to build a common theme for your ggplot object so that it may be applied to all plots in the paper, such that they have a common look and feel, and are not just the default, this will also stop the user from writing out many of the common theme elements everytime to get that distinct look and feel that they are going for. It also doesn't stop the writer from adding new theme elements to the plot.

Lollipop

The first plot we will be covering is the lollipop plot. Which is very useful in viewing the differences between many groups. Typically you would want something that is categorical, and perhaps has many categories along the X-axis, and then a value to map to the Y-axis. This type of plot could be viewed as a replacement of the bar plot. In my opinion it is more visually appealing, and therefor can be more effective in communication.

GeomRidges

The second plot types that we will be discovering are those of the gg_ridges package. A fantastic way to view distributions of variables overlaid on one another to really get an idea of what the main differences are between the groups, in a common figure.

Sankey

The third and final figure that we will be looking into is the sankey plot. This is a fantastic way to visualize the flow of data, it can be used to visualize things such as segmentation and grouping, were we can see how one group flows into other groups, and ends up at the final target group.

Common Theme

This theme will set the size of the axis titles (both x and y), the size of the plot tite, and adjust the x, y axis title to be more visually appealing and in a slightly better location than the default. It also gets rid of the minor grid lines by making use of the element_blank() function, and sets a visually appealing background color so that we are away from just the standard white of GGPLOT. The ... in the function call allows for more variables to be passed, which will allow more theme elements to be added to the common theme function as if it were just the normal theme function.

```r
com_theme <- function(...) {
  theme(
    plot.title=element_text(size=12, face="bold"),
    axis.title=element_text(size=8, face="bold"),
    axis.title.x=element_text(vjust=-1),
    axis.title.y=element_text(vjust=2),
    panel.grid.minor=element_blank(),
    panel.background=element_rect(fill="#FFFFED"),
    plot.background=element_rect(fill="#FFFFED")
  ) +
  theme(...)
}
```

Lollipop Plot

We will look at the cars data, it contains the speed that a car was traveling and how long it took it to stop, measured in ft. We can see that there are 19 different unique values of speed. We will group by the speed variable, and aggregate the mean of the distance variable, and use this data for our plot.

```
data(cars)
```

```
length(unique(cars$speed))
```

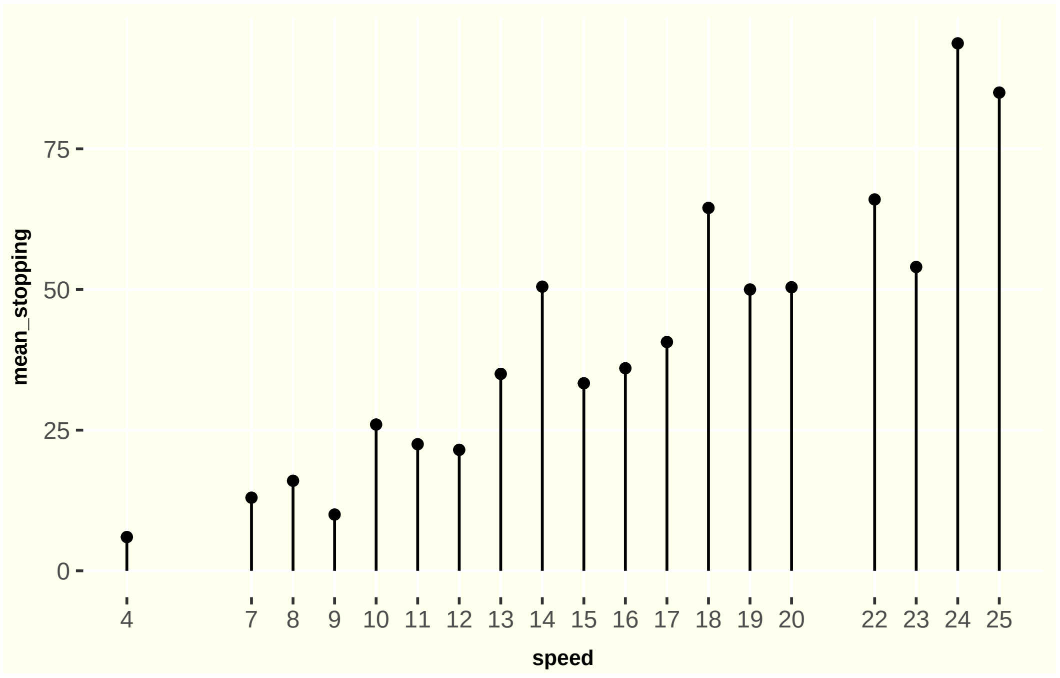
```
[1] 19
```

Storing the unique values of the car speeds will allow us to use them to set breaks on the plot.

```
car_speeds <- unique(cars$speed)
```

To create the lollipop we use the geom_segment, with each segment having a start and end on the x axis based of the categorical variable, and the y axis start at 0 and end at the value of that categorical variable. This will create all the vertical lines, then all we need to add is the scatter plot of the points, which will be done with geom_point().

```
cars |>
  group_by(speed) |>
  summarize(mean_stopping=mean(dist)) |>
  ggplot(aes(x=speed, y=mean_stopping)) +
    geom_segment(
      aes(
        x=speed,
        xend=speed,
        y=0,
        yend=mean_stopping
      )
    ) +
    scale_x_continuous(breaks=car_speeds) +
    geom_point() +
    com_theme()
```



It is also easier to see the relationship between the variables if you sort the column based off of the values so that the unique speed with the highes value for mean_stopping distance appears first, and then the descending order is used. This really highlights which of the speeds has the highest to lowest mean stopping distance.

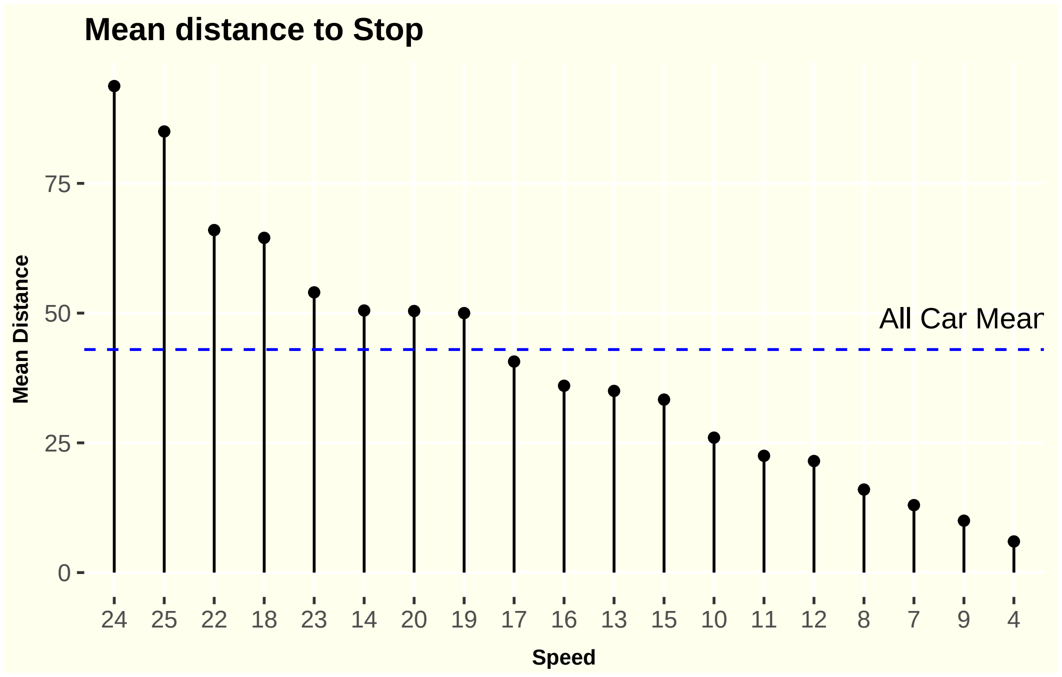
```
# first group the data, and then set the speed as a factor
cars_grouped <- cars |>
                  group_by(speed) |>
                  summarize(mean_stopping=mean(dist)) |>
                  mutate(speed=factor(speed, levels=car_speeds))
```

```
mean_stop_allcars <- mean(cars$dist)
```

```
cuttoff <- data.frame(yintercept=mean_stop_allcars, Lines="Avg Stopping Distance")
```

We can also do things such as add in horizontal lines that will draw our readers attention to a certain point, for instance we can add in a horizontal line that indicates the mean stoppoing distance across all speeds, and then add in the annotation that calls out what this line is signifying. Notice that we can now easily identify the speed groups that fall above the mean stopping distance, and those that fall below it, as well as a very easy representation to decipher of which speed has the highest mean stopping distance, all the way down to the lowest.

```r
cars_grouped |>
  ggplot(aes(x=fct_reorder(speed, -mean_stopping), y=mean_stopping)) +
    geom_segment(
      aes(
        x=fct_reorder(speed, -mean_stopping),
        xend=fct_reorder(speed, -mean_stopping),
        y=0,
        yend=mean_stopping
      )
    ) +
    geom_point() +
    geom_hline(
      aes(
        yintercept=mean_stop_allcars,
        ),
      color="blue",
      linetype="dashed"
      ) +
    annotate("text", 18, 43, vjust=-1, label="All Car Mean") +
    scale_color_manual(values=c("blue"), labels=c("All Car Mean")) +
    labs(
      title="Mean distance to Stop",
      x="Speed",
      y="Mean Distance"
    ) +
    com_theme()
```



GeomRidges

We will simulate data from a chi-squared distribution, with degrees of freedom ranging from 1 to 5, and then plot these distributions with our geom_ridges package to get an idea of just how powerful the plot can be when comparing distributions across multiple groups/categories.

```
df_list <- c(1, 5, 10, 20)
df_sim <- data.frame(
  x=rchisq(n=1000, df=rep(df_list, each=250)),
  group=factor(
    rep(df_list, times=250),
    levels=df_list
  )
)
```

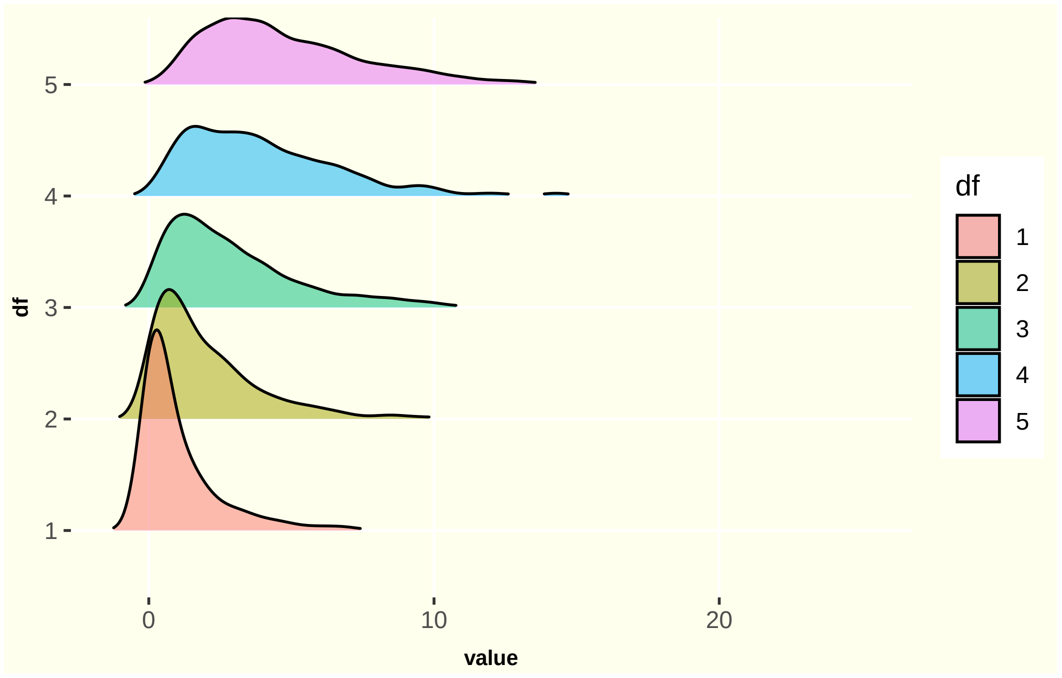
```
df_1 <- data.frame(value=rchisq(n=1000, df=1)) |>
          mutate(df="1")
df_2 <- data.frame(value=rchisq(n=1000, df=2)) |>
          mutate(df="2")
df_3 <- data.frame(value=rchisq(n=1000, df=3)) |>
```

```
          mutate(df="3")
df_4 <- data.frame(value=rchisq(n=1000, df=4)) |>
          mutate(df="4")
df_5 <- data.frame(value=rchisq(n=1000, df=5)) |>
          mutate(df="5")
```

```
plot_df = rbind(df_1, df_2, df_3, df_4, df_5)
```

```
plot_df |>
  ggplot(aes(x=value, y=df)) +
    geom_density_ridges(
      aes(fill=df),
      rel_min_height=0.01,
      alpha=0.5
    ) +
    com_theme()
```

```
Picking joint bandwidth of 0.464
```



We can also pair this plot with a dataset like Iris, where we have the three categorical variables that we would like to consider distribution across 4 or more different numerical variables. This plot highlights the distribution of each of the numerical variables, grouped by the plant species, using the facet_wrap() function to create a grid of plots segregated by the variable column that we create when we melt the dataframe into long format.

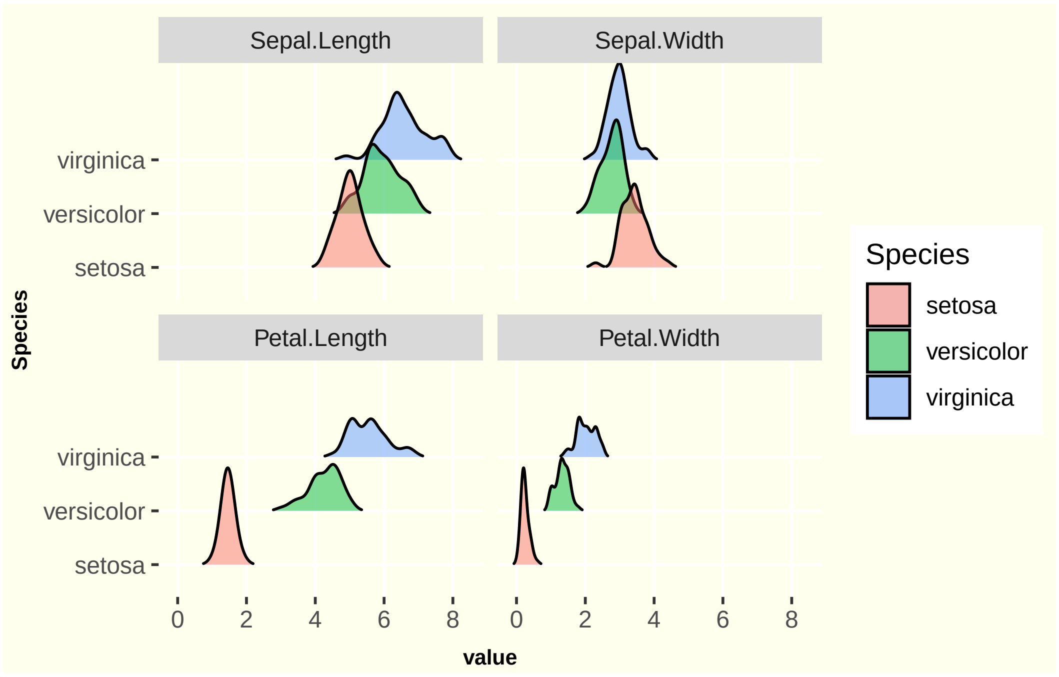
```
iris |>
  melt(
    id.vars="Species"
  ) |>
  ggplot(aes(x=value, y=Species)) +
    geom_density_ridges(
      aes(fill=Species),
      rel_min_height=0.01,
      alpha=0.5
    ) +
    facet_wrap(~variable) +
    com_theme()
```

```
Picking joint bandwidth of 0.181

Picking joint bandwidth of 0.13

Picking joint bandwidth of 0.155

Picking joint bandwidth of 0.075
```



Sankey Plot

This code is mostly addapted from a TidyTuesday submission. I can no longer find the exact submission so I am not able to add credit to it here, but roughly 50% of my code comes from that submission. As the sankey plot is very hard to wrap your head around.

```
survivalists <- readr::read_csv('https://raw.githubusercontent.com/rfordatascience/tidytue
```

```
Rows: 94 Columns: 16
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (10): name, gender, city, state, country, reason_tapped_out, reason_cate...
dbl  (5): season, age, result, days_lasted, day_linked_up
lgl  (1): medically_evacuated

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
font_add_google("Caveat", "cave")
txt_font = "cave"
```

```
bg_col = "grey10"
txt_col = "white"
highlight_col = "grey40"
```

This sankey plot will follow contestens from different categories from an original mapping of agegroup, all the way through to what place the contestant ended up getting at the end of the alone season. It will show reasons for leaving, what the sex of the contestant was, which agegroup they came from, and of course the result.

First we select for the columns that we want. Then we notice that reason_category is blank/na if the contestant was a winner, as they didn't have a reason for leaving. Then we will cast the result as numeric so that it will be ordered correctly in our plot, and not as a string which would put 10th place in a funky spot in the ordering. We then set a case to group the ages of the contestants, as currently we have a very wide range, and many unique ages. Then we set the name for the columns.

Make long is the key to setting up the sankey plot. It takes our original dataframe and breaks it down into a much longer format. the new columns are going to be x, node, next_x, and next_node. If the original columns that we are passing to make_long, are AgeGroup, ReasonForLeaving, Gender, and Result, the make long function will take these categories and map them as such. The first row: x = AgeGroup, node = what group contestant belonged to(value from agegroup column), next_x = ReasonForLeaving, next_node = what group contestant belonged to(value from reasonforleaving column). second row, continues on with the previous rows final node, and then continues to the next node. This continues until each of the columns passed to make_long are used as the x and node columns in the new dataframe.

We then set the factor levels so that our plot will have things organized in a way that we can comprehend/like.

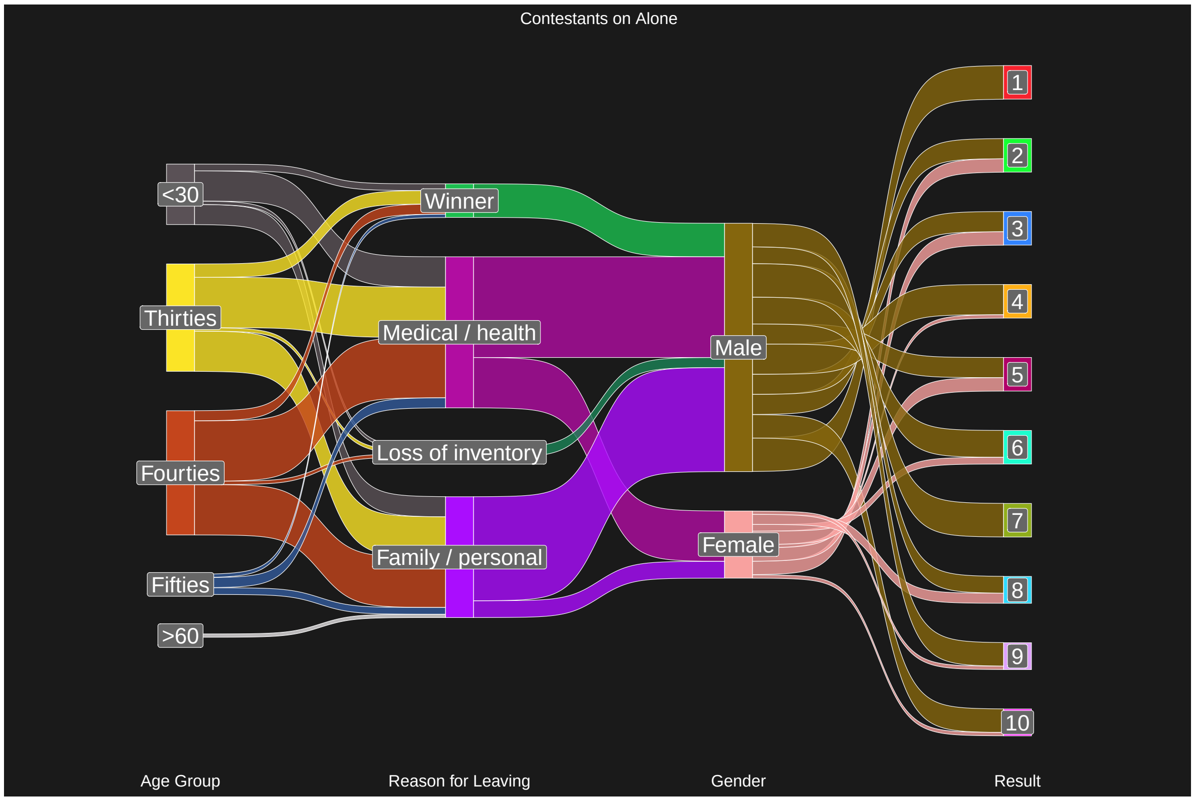
```
plt_data <- survivalists |>
              select("gender", "age", "reason_category", "result") |>
              mutate(reason_category = if_else(is.na(reason_category), "Winner", reason_ca
              mutate(result = as.numeric(result)) |>
              mutate(age_group = case_when(
                age <= 30 ~ "<30",
                age > 30 & age < 40 ~ "Thirties",
                age >= 40 & age < 50 ~ "Fourties",
                age >= 50 & age < 60 ~ "Fifties",
                age >= 60 ~ ">60"
              )) |>
              mutate(
                "Age Group" = age_group,
                "Reason for Leaving" = reason_category,
                "Gender" = gender,
                "Result" = result
              ) |>
              make_long("Age Group", "Reason for Leaving", "Gender", "Result") |>
              mutate(
                node = factor(node),
                node = fct_relevel(
                  node,
                  c("10", "9", "8", "7", "6", "5", "4", "3", "2", "1")
                ),
                node = fct_relevel(
                  node,
                  c(">60", "Fifties", "Fourties", "Thirties", "<30")
                )
              )
```

Here we can see the plot, this data takes the x and next_x and uses those are the first mapping, from an age group to a reason for leaving. Then it continues to the next row and uses the previous rows next_x as the new x, and maps to the new next_x node. This continues until there are no next_x, which happens after result is the x and node. This is the final node.

This graph represents each individual and there segmentation as they go trough each of the groups. We can see that contestants over 60 all left the show for family/personal reasons. We can see that all females that left the show did so for Medical/Health reasons or Family/personal reasons.

There are many other possible segmentations that we would be able to gleen insight with from this style of plot. I look forward to the chance to utilize it more in my future. It is a hard one to wrap your head around, and I would have been lost attempting to use it for my own purposes without the help of the tidytuesday submission that I used as a base/starting point for my code.

```
plt <- plt_data |>
          ggplot(
            aes(
              x = x,
              next_x = next_x,
              node = node,
              next_node = next_node,
              fill = factor(node),
              label = node
            )
          ) +
          geom_sankey(
            node.color = "white",
            flow.alpha = 0.8,
            linewidth = 0.25,
            color = "white"
          ) +
          geom_sankey_label(
            size = 7,
            color = "white",
            fill = highlight_col
          ) +
          labs(
            title = "Contestants on Alone"
          ) +
          scale_fill_manual(values = unname(polychrome())) +
          theme(
            plot.title = element_text(
              size = 15,
              hjust = 0.5,
              vjust = 0.5,
              color = txt_col
            ),
            plot.caption = element_text(color = "white", size = 10),
            plot.background = element_rect(fill = bg_col, color = bg_col),
            panel.background = element_rect(fill = bg_col, color = bg_col),
            panel.grid = element_blank(),
            axis.ticks = element_blank(),
            axis.title = element_blank(),
            axis.text.y = element_blank(),
            axis.text.x = element_text(
              size = 15,
              color = txt_col
            ),
            legend.position = "none"
          )
plt
```


Contestants on Alone
<30
Thirties
Fourties
Fifties
>60
Winner
Medical / health
Loss of inventory
Family / personal
Male
Female
1
2
3
4
5
6
7
8
9
10
Age Group
Reason for Leaving
Gender
Result