Communicating Economics:

Data Visualisation in R

Zaeen de Souza
Azim Premji University
3/05/21

Overview

Today

Designing a plot

Things to keep in mind while making a plot

Scatterplots

Understanding how ggplot works using a simple scatterplot

Themes

An example of a histogram, using a new theme

Transitions & flows:

An example of transition matrices using an alluvial plot

Gaps & Differences:

An example of a dumbbell plot!

What psychology has to say

There are two really nice animations on how the brain percieves shapes, colours, lines and so on. You can view them here and here.

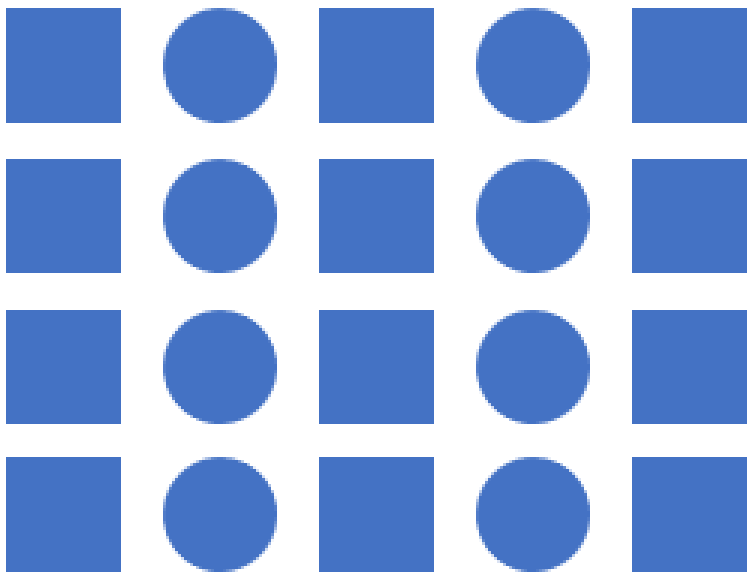
Here are 4 ways in which we organise/percieve information[1]

1. Similarity
2. Proximity
3. Continuity
4. Enclosure

[1] Taken from here

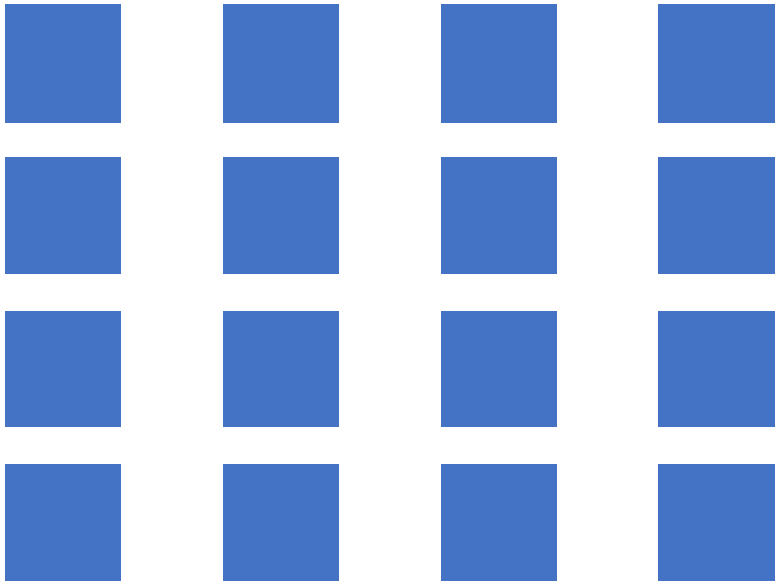
Similarity

Items that are similar to each other tend to be seen as a group, rather than single objects. The squares and circles in the graphic below are not seen as single objects but are rather perceived as grouped columns.



Proximity

Objects that are closer together are seen as a group and not individually. The squares below are perceived as 4 columns.

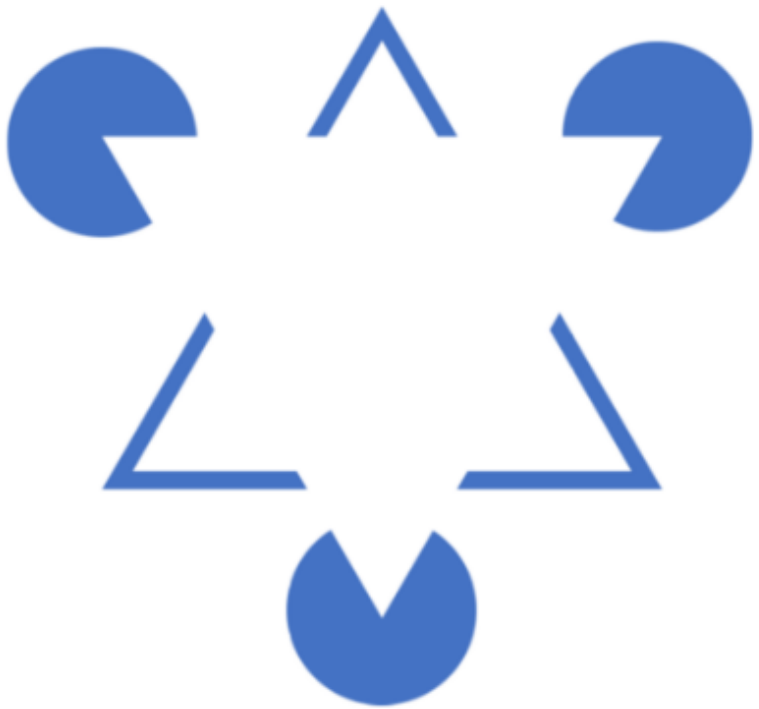


Continuity

The eye follows the smoothest path it can find. Did you notice how the eye follows the lighter points better?

Closure

Objects grouped together are seen as a whole. We ignore lines and fill in missing parts. Do you see the white triangle?



Datasets for today

Load all the datasets from the google drive folder.

```
ex_1 <- read.csv("ex_1.csv")
ex_2 <- read.csv("ex_2.csv")
ex_3 <- read.csv("ex_3.csv")
map_data <- read.csv("map_data.csv")
```

Packages

```
library(pacman)
p_load(ggplot2, dplyr,
       ggforce, ggthemes,
       ggalluvial, ggtext)
```

GGplot Review

All the data visualisation that we will do here uses either the standard `ggplot2` package, or its extensions, such as `ggmaps`, `ggalluvial`, and so on. You can read more about the ggplot2 package here.

`ggplot` relies on three core arguments:

1. `data`
   This is where you numbers (variables) come from.
2. `aes()`
   This maps variables to different aesthetics—colours, shapes etc.
3. `geom_type()`
   This takes the other arguments and builds up your graphic using geometries

Note 1: `geom_type()` is a placeholder; `type` will be replaced by the geometry you choose to use; for e.g: `geom_point()` .

Note 2: `aes()` can also contain functions. For instance:

```
aes(x = age_months/12, y = log(wages))
```

ggplot2: Building Blocks

We need to map the following:

`data` → `aes()` → `geom_type()`

New layers/blocks are added ontop of each other, using the `+` sign

Visualisation 1

Load data

Let's use the dataset called `ex_1`, and use mapping the aes() using `x=`, `y=`, to three variables from our dataset.

1. `incgr` : Change in Income (%)
2. `mobility` : Change in Mobility (%)
3. `state_month` : State ID + Month

Scatterplots Step 1: Making the Canvas

```
plot_1 <- ggplot()
plot_1
```


Figure 1: An Empty Canvas

Scatterplots Step 2: Mapping Variables

```
plot_1 <- ggplot(ex_1,
# we mapping the x, and y to two variables from the data
       aes(x = mobility,
           y = incgr))
plot_1
```

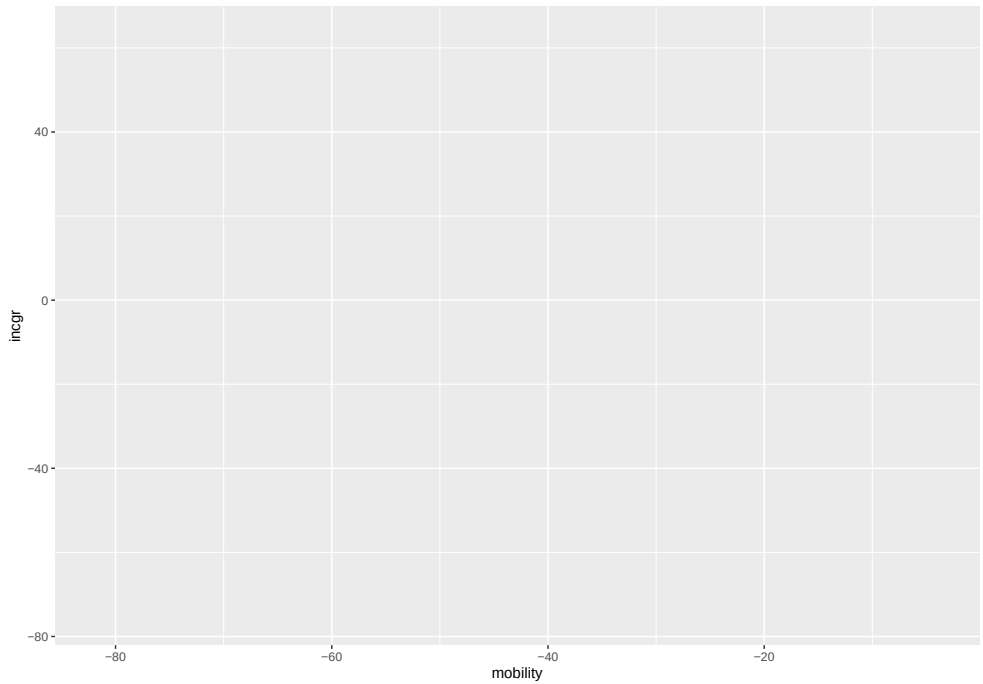


Figure 2: Canvas + Mapped Variables

Scatterplots Step 3: Adding A Geom

```
plot_1 <- plot_1 + geom_point()
plot_1
```

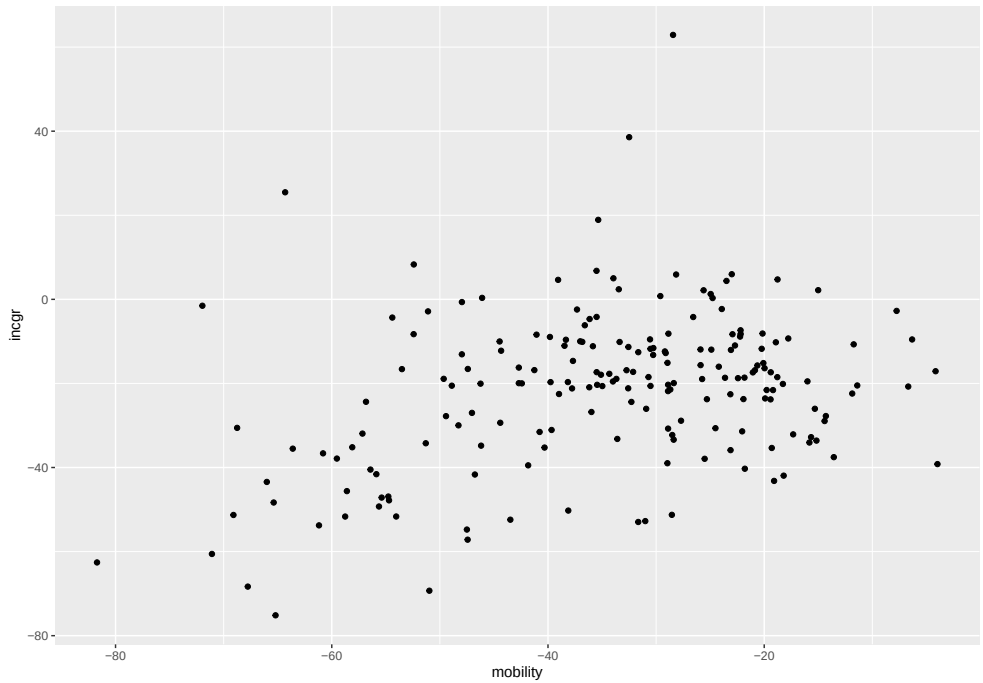


Figure 3: A Scatter Plot

Scatterplots Step 4: Editing the aes()

```
plot_1 <- ggplot(ex_1,
          aes(x = mobility,
              y = incgr,
              colour = state)) +
  geom_point()
plot_1
```

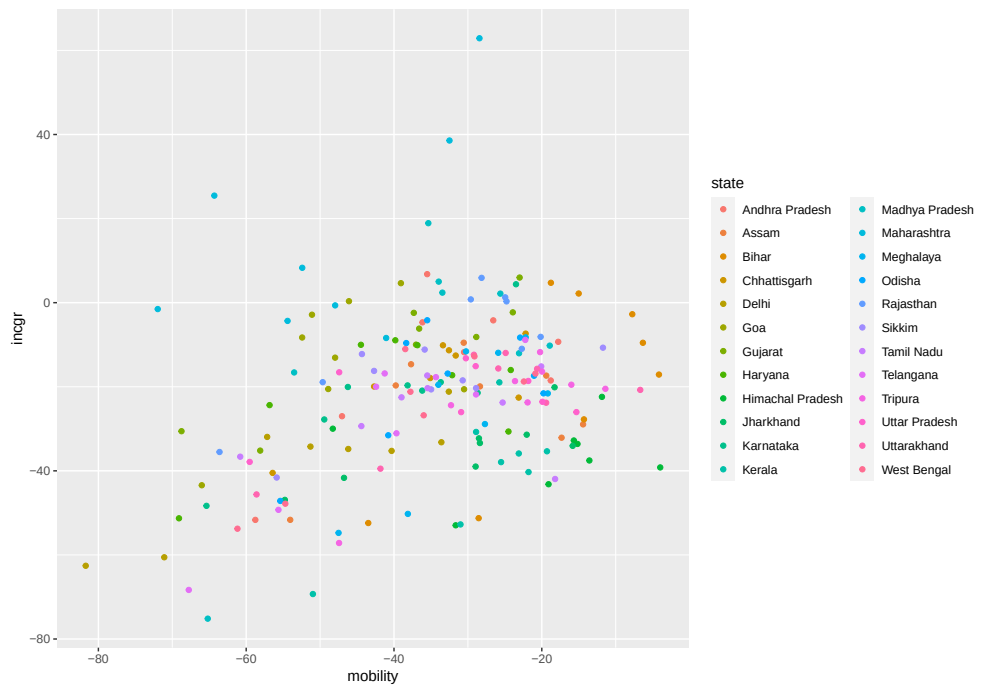


Figure 4: A Scatter Plot with Colours

Scatterplots Step 5: Regression Line + Labels

```
plot_1 <- ggplot(ex_1,
                 aes(x = mobility,
                     y = incgr)) +
  geom_text(aes(label = state_month)) +
  geom_smooth(method = "lm")
# try without method="lm", i.e geom_smooth()
plot_1
```

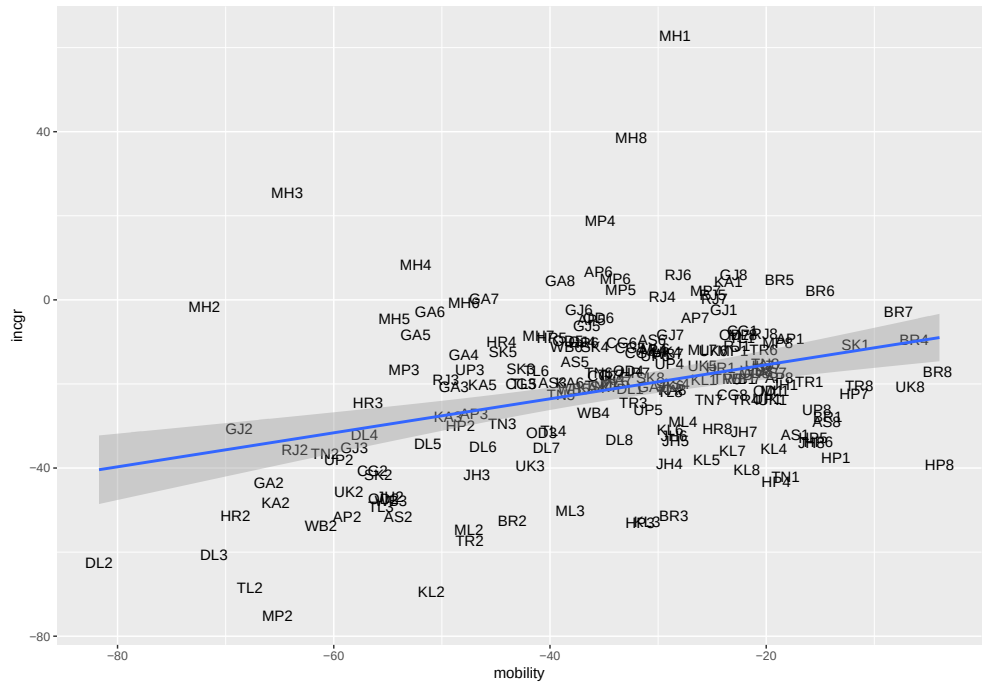


Figure 5: A Scatter Plot with Labels

In short

What we did

1. Add data
2. Mapp variables
3. Customise the aesthetics
4. Add geoms (‘geom_point‘, ‘geom_text‘, ‘geom_smooth‘)

What you need to do:

1. Add axis labels
2. Add a title, subtitle, caption
3. Add (Make!) a theme
4. Add annotation to the plot—notes, reference lines etc

Hint: Use `annotate()` , `geom_hline()` , `geom_vline()`

Geoms: Some Common Geoms

1. `geom_histogram()` histogram
2. `geom_bar()` bar plot
3. `geom_col()` bar plot 2
4. `geom_density()` Kernal density plot
5. `geom_point()` Scatterplot
6. `geom_smooth(method=)` regression lines
    - `geom_smooth()`
    - `geom_smooth(method="lm")`

There is a massive list of geoms and as well, ggplot options here. Make sure to check this out.

Themes

Themes

`ggplot` comes with a range of in-built themes that are ready to be used. These can be accessed using `library(ggthemes)` and then adding a theme object to your ggplot.

Themes: Classic

```
plot_1 +
  theme_classic()
```

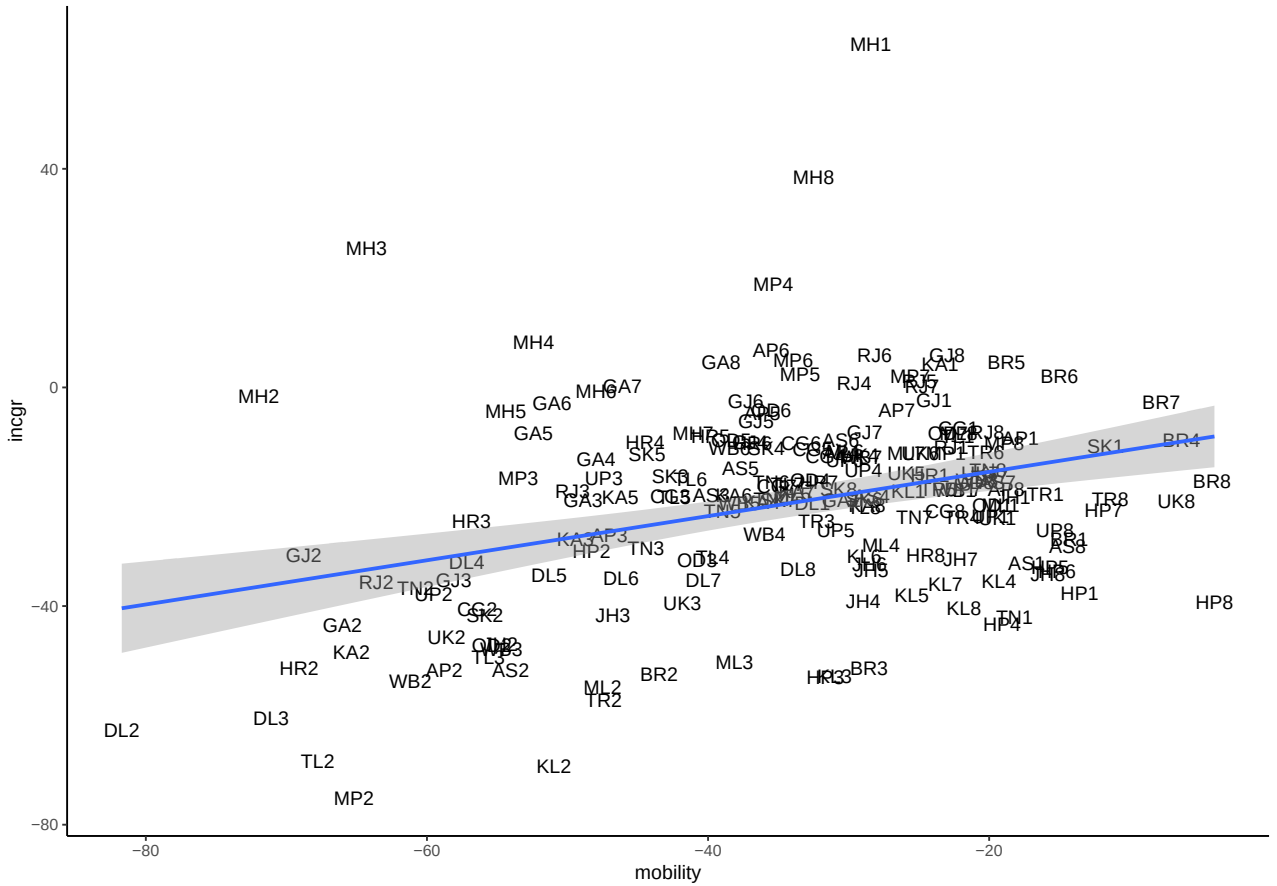


Figure 6: Classic

Themes: fivethirtyeight

```
plot_1 +
  theme_fivethirtyeight()
```

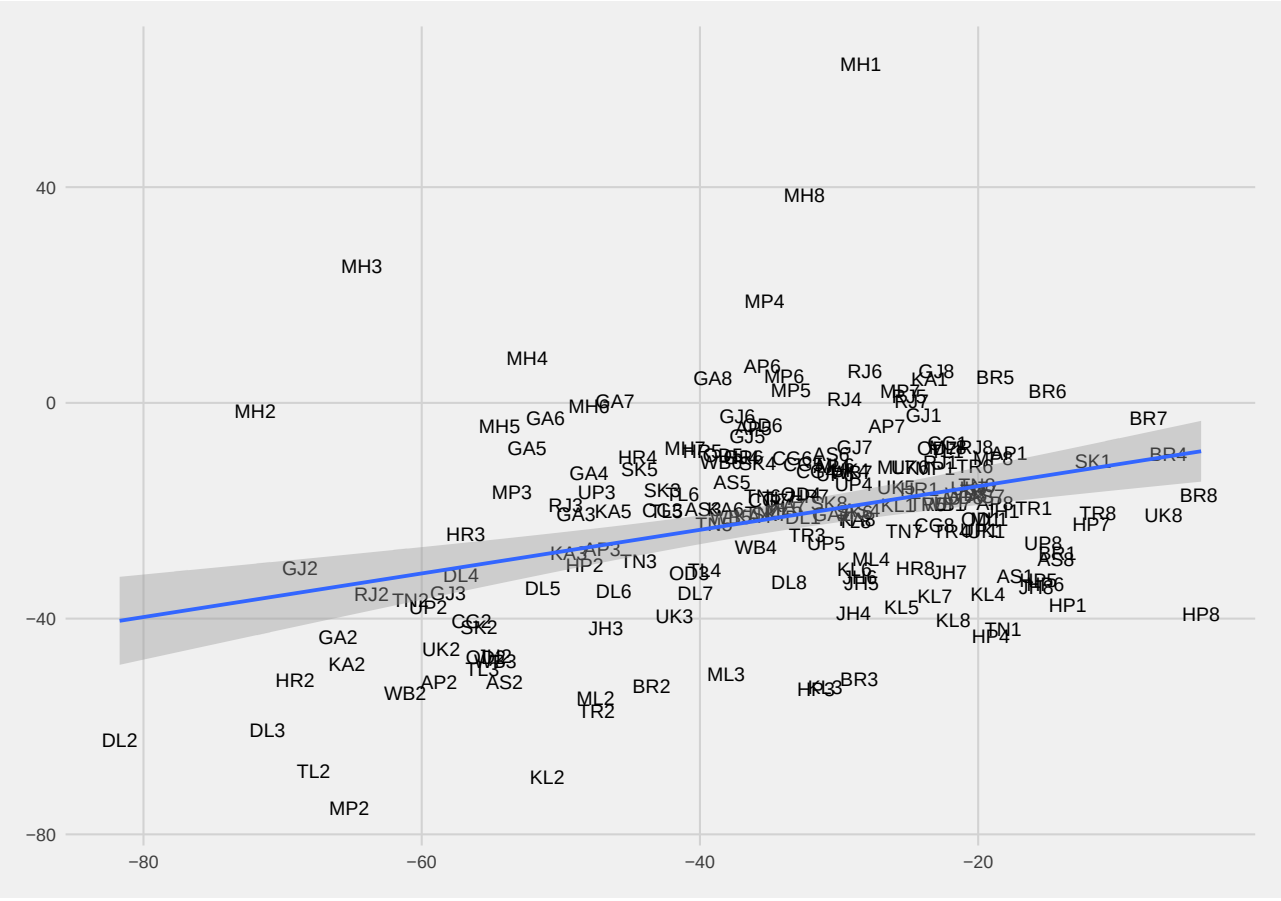


Figure 7: fivethirtyeight

Themes: The Economist

```
plot_1 +
  theme_economist()
```

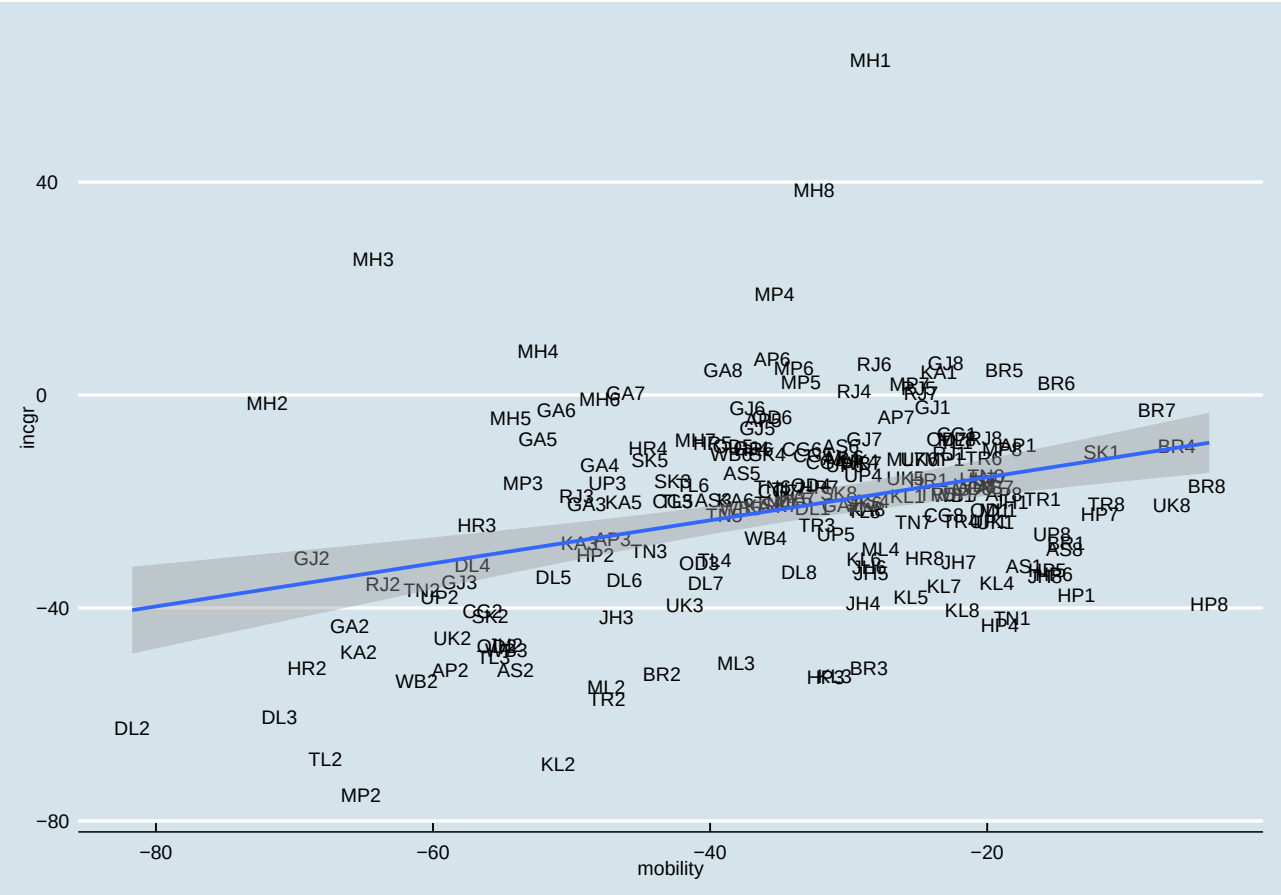


Figure 8: The Economist

Themes: A Real Classic

```
plot_1 +
  theme_stata()
```

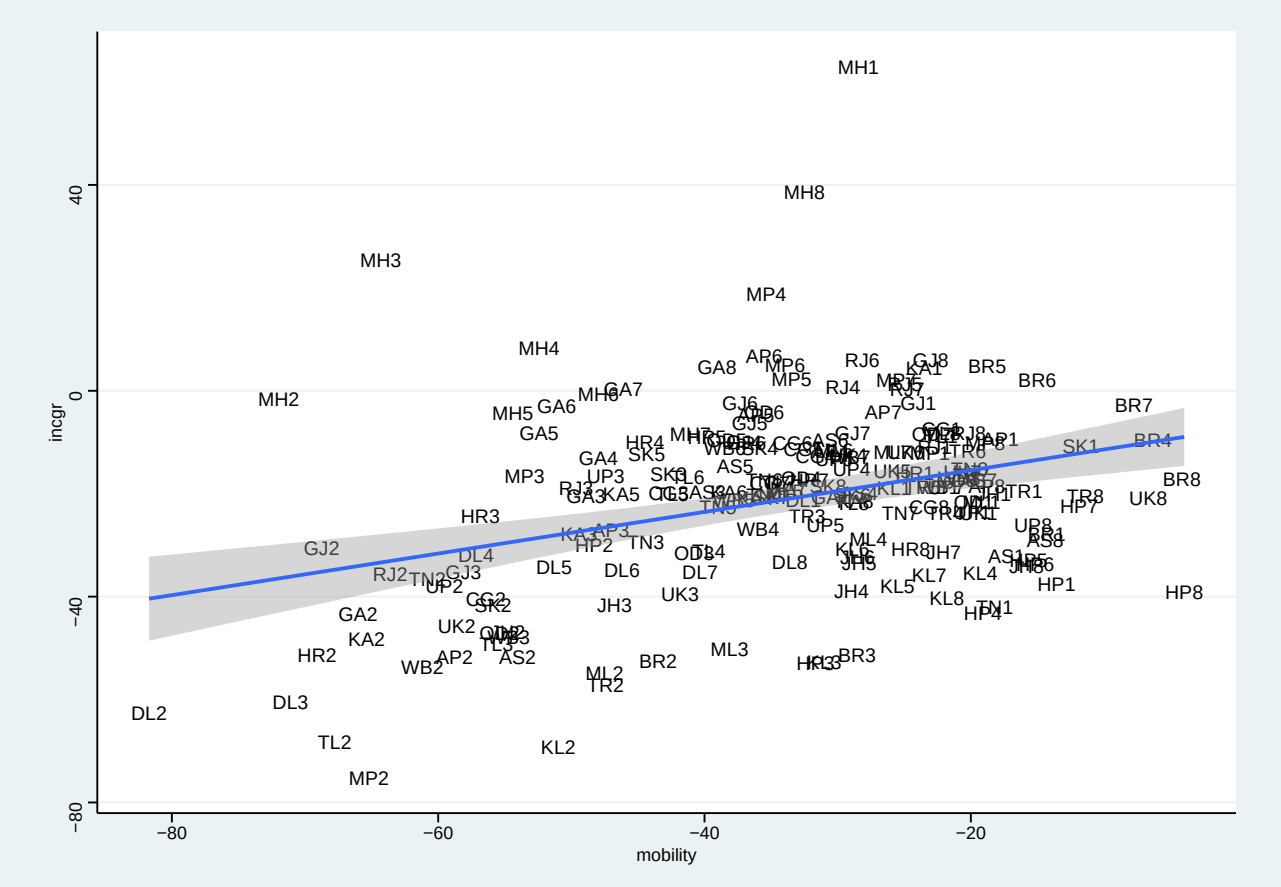


Figure 9: Stata

Modifying a theme

`ggplot` is very flexible. You can customize every single part of a graph. This is done using the `theme()` function, and selecting different options.

Each plot, consists of a range of different parts—axis lines, gridlines, labels, backgrounds, titles, shapes and so on.

Modifying a theme: Options

Type `?theme` to see the full list of options. You should see something like this in your `help` tab.

```
theme(
line,
rect,
text,
title,
aspect.ratio,
axis.title,
axis.title.x,
....
)
```

Modifying a theme: Graph Components

You could choose to edit any of these—or something else. There are 3000+ permutations that you could experiment with!

1. `axis.line =`
2. `axis.ticks =`
3. `axis.ticks.length =`
4. `legend.position =`
5. `panel.background =`
6. `panel.grid.major =`
7. `panel.grid.minor =`

There is a complete, exhaustive list here.

Modifying a theme: Graph Components & Elements

Each graph component, maps to a particular `element type`. The 4 main types are:

1. `element_blank(arguments)`
2. `element_line(arguments)`
3. `element_text(arguments)`
4. `element_rect(arguments)`

Modifying a theme: Graph Components & Elements

1. `theme(axis.title = )` → `element_blank()` →remove axis titles
2. `theme(axis.line = )` → `element_line(colour="black")` →axis line colour
3. `theme(axis.title = )` → `element_text(colour="red")` →axis title colour
4. `theme(plot.background = )` → `element_rect(fill="green")` →background colour

Modifying a theme: Setup

```
my_theme <- function(base_size = 12,
                     base_family = "") {
# we will use theme_minimal as a base
theme_minimal(base_size =
                base_size,
              base_family =
# this tells R that we are replacing some elements in theme_minimal
                base_family) %+replace%
```

Modifying a theme: Gridlines

```
# starting the modification using theme(.....)
    theme(
      panel.grid.minor = element_blank(),
      panel.grid.major.x = element_blank(),
      panel.grid.major.y  =  element_line(
# you can add any common colour here - "red", "green" or a hexcode
        colour = "#c9c0b7",
# dotted, dashed, dotdash, longdash, solid, blank
        linetype = "dotted",
# controls line width 0 -> inf
        size = .5),
```

Modifying a theme: Axis Lines & Ticks

```
axis.ticks = element_line(
  size = .5,
  colour = "#c9c0b7"),
axis.ticks.x = element_blank(),
axis.line = element_line(
  size = .5,
  colour = "#c9c0b7",
  linetype = "solid"),
axis.line.y = element_blank(),
```

Modifying a theme: Axis Text

```
axis.text.y = element_text(
  colour = "black",
# margin(r,l,t,b)
  margin = margin(r = 2),
  hjust = 1),
axis.text.x = element_text(
  colour = "black"),
# we are done!
complete = TRUE)
# remember to close the curly bracket which we opened!!
}
```

Visualisation 2

Load data

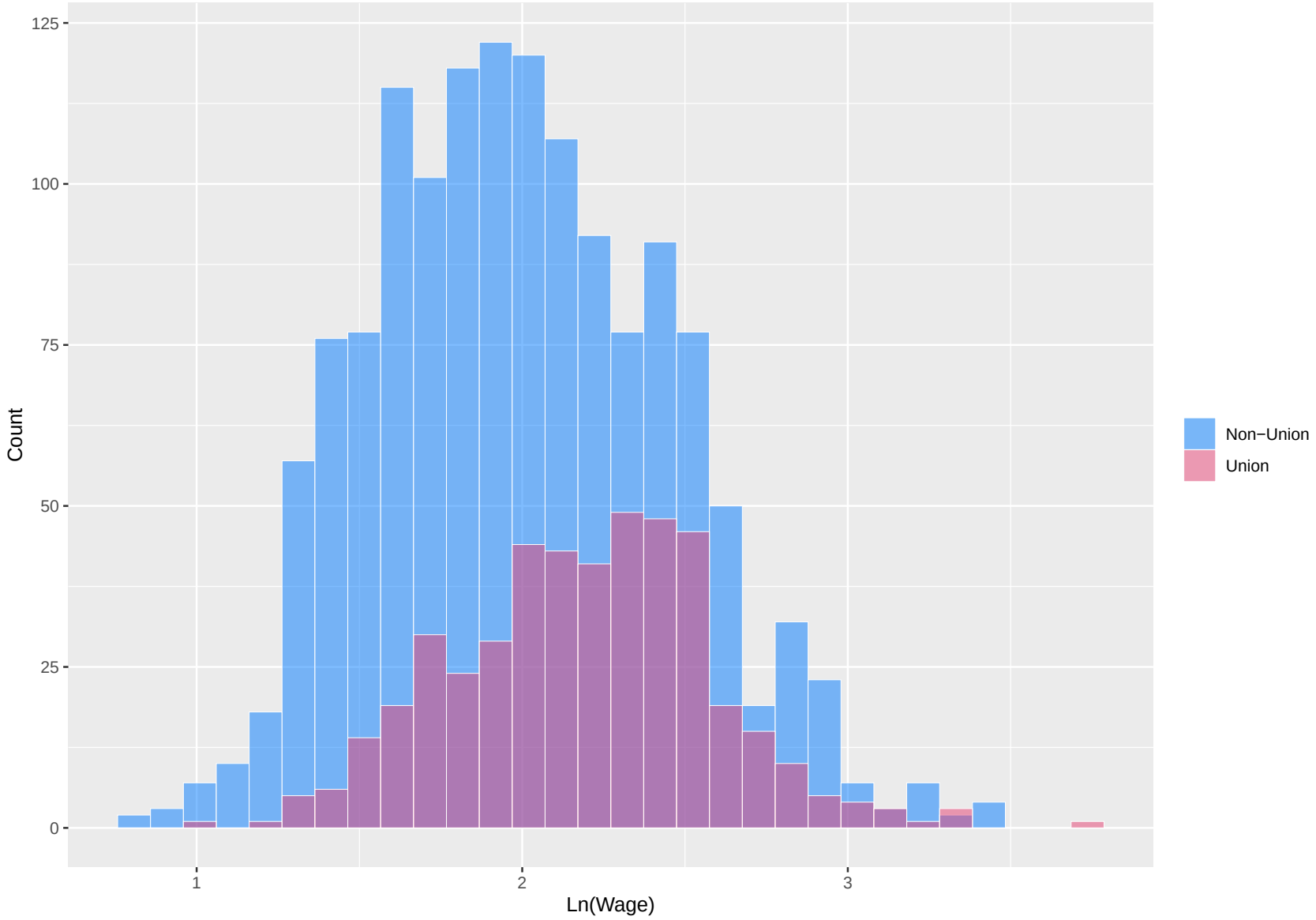
We will use the dataset called `ex_2`, to test our customised theme

1. `ln_wage` : ln(hourly wage)
2. `union` : Union Status

Modifying a theme: ggplot default

```
plot_2 <- ggplot(ex_2,
       aes(x = ln_wage,
                 fill = union)) +
  geom_histogram(position = "identity",
                 bins = 30,
                 size = 0.05,
                 alpha = .5,
                 colour = "white") +
  scale_fill_manual(values = c("#007bff", "#E64173"),
                    labels = c("Non-Union", "Union")) +
  ylab("Count") +
  xlab("Ln(Wage)") +
  labs(fill = "")
plot_2
```

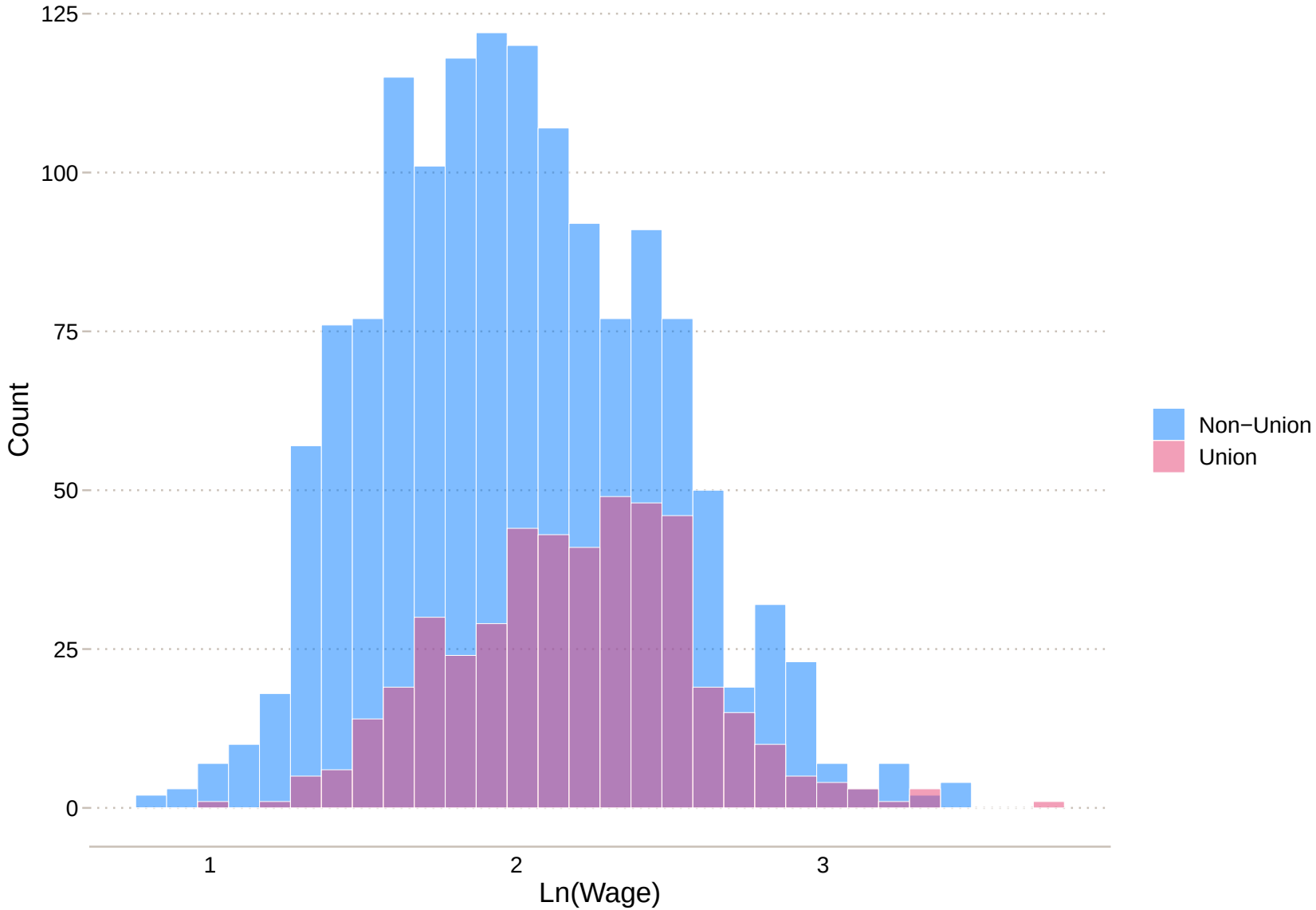
Modifying a theme: ggplot default



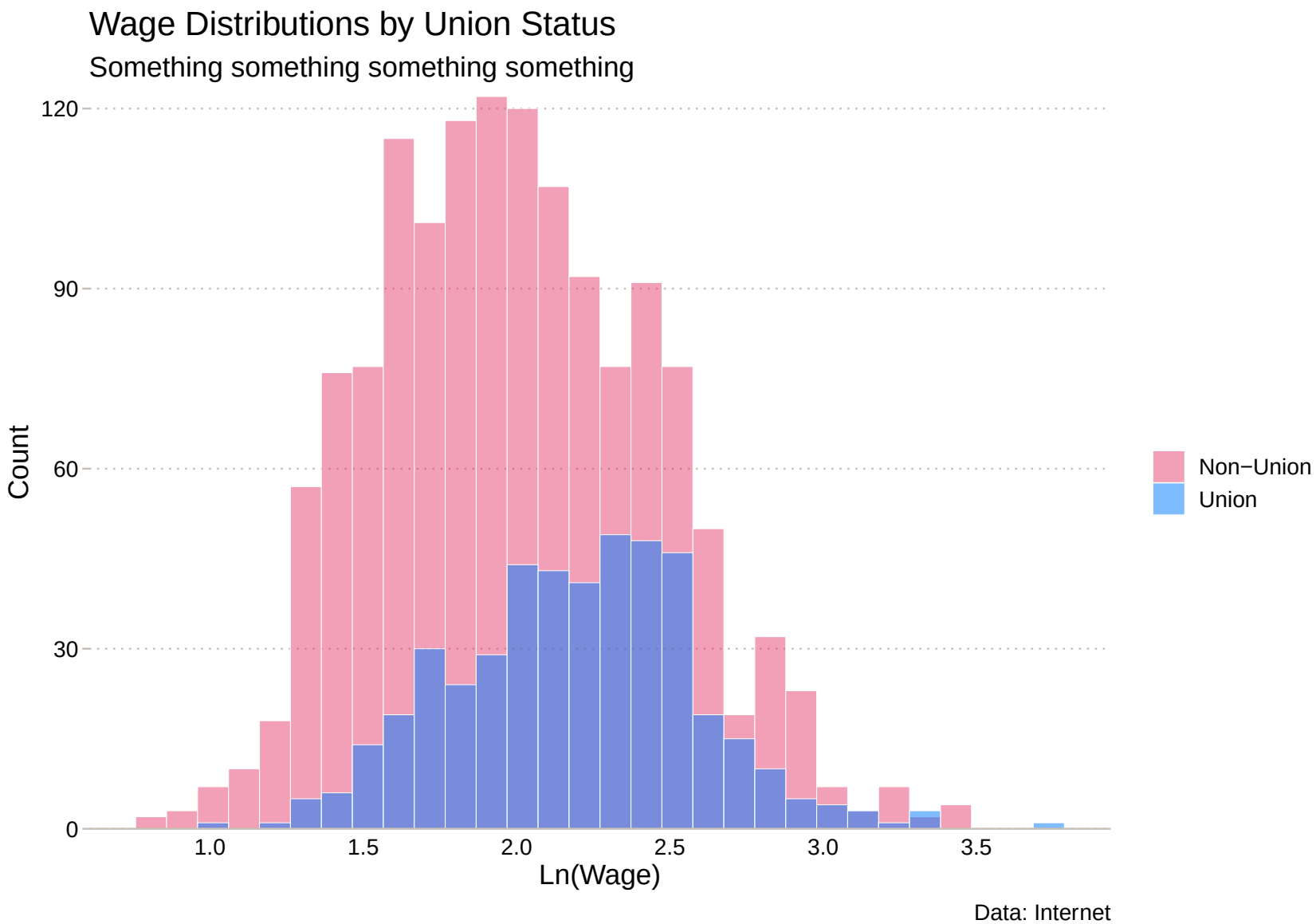
Modifying a theme: Using out theme instead

```
plot_2 <- plot_2 + my_theme(base_size = 15)
plot_2
```

Modifying a theme: Our theme!



Modifying out theme a bit more—try to do this yourself!



Visualisation 3

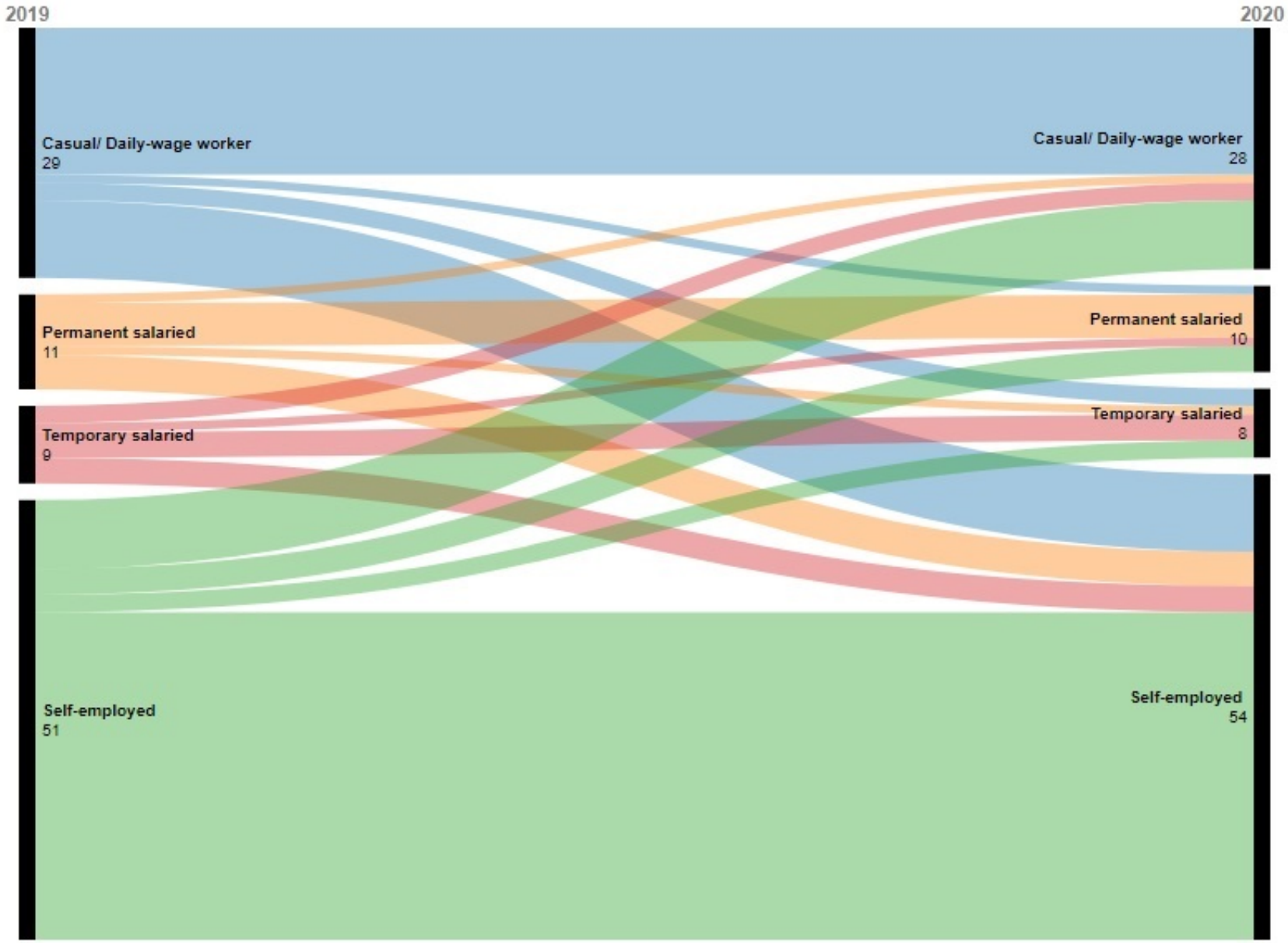
Load data

We will be using the file called `ex_3`, it is the data that was used to create this matrix:

	Casual/daily wage worker	Self-employed	Temporary salaried	Permanent salaried
Casual/daily wage worker	57.9	33.1	5.6	3.4
Self-employed	15.0	75.5	4.1	5.5
Temporary salaried	22.5	31.3	35.6	10.7
Permanent salaried	9.8	34.1	8.5	47.6

The graph

And to create this graph:



Alluvial Chart

This chart visualises transitions, flows, networks between different categorical variables. Each ribbon is **weighted** by some numerical value. When using dis-aggregated data, this could from frequencies, percentages etc.

Note that in this version of the data, the percentages in each category of `pre` add up to 100%, so our graph will look slightly different, as compared to the original!

```
library(ggforce)
data_1 <- read.csv("F:/Surbhi (Data Viz)/ex_3.csv")

data_alluvial <- data_1 %>%
  gather_set_data(2:1)

data_alluvial$x <-
  factor(data_alluvial$x,
         levels = c("pre", "post"),
         labels = c(1, 2))
```

Alluvial Plot Step 1: Data

The setup

```
plot_3 <- data_alluvial %>%
  ggplot(aes(x, id = id,
             split = y,
             value = percentage))
plot_3
```

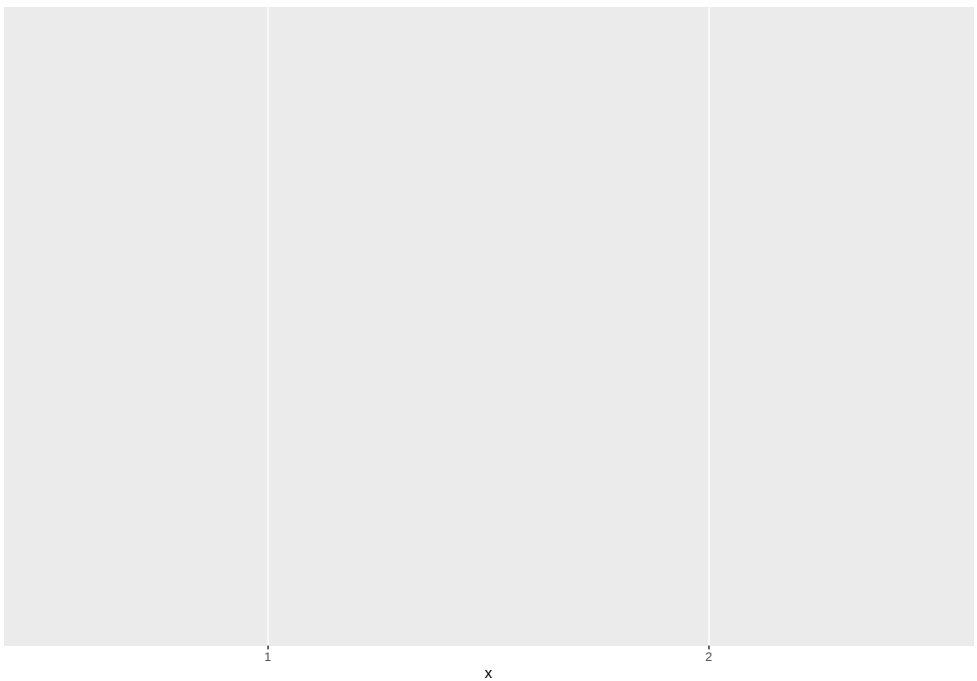


Figure 10: Step 1

Alluvial Plot Step 2: Mapping

```
plot_3 <- plot_3 +
  geom_parallel_sets(aes(fill = pre,
                         alpha=0.5),
                         strength = .4,
                         show.legend = F)
plot_3
```

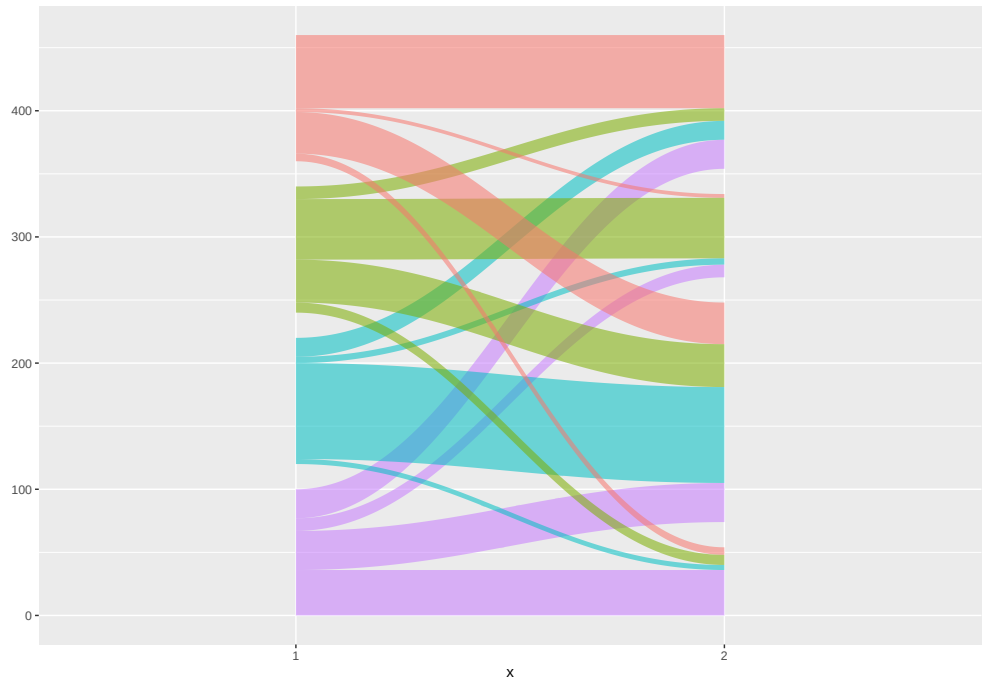


Figure 11: Step 2

Alluvial Plot Step 3: More mapping

```
plot_3 <- plot_3 +
  geom_parallel_sets_labels(angle = 0) +
  geom_parallel_sets_axes(axis.width = 0.01)
plot_3
```

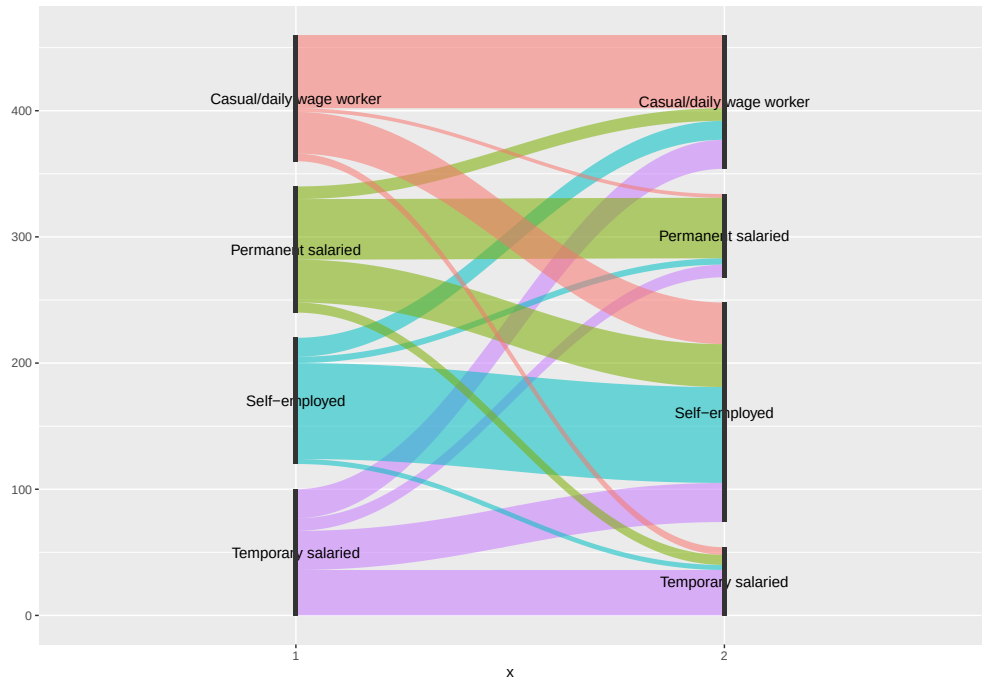


Figure 12: Step 3

Alluvial Plot Step 4: More mapping

```
plot_3 <- plot_3 +
  scale_x_discrete(labels = c('2019', '2020'),
                   position = "top") + theme_void()
plot_3
```

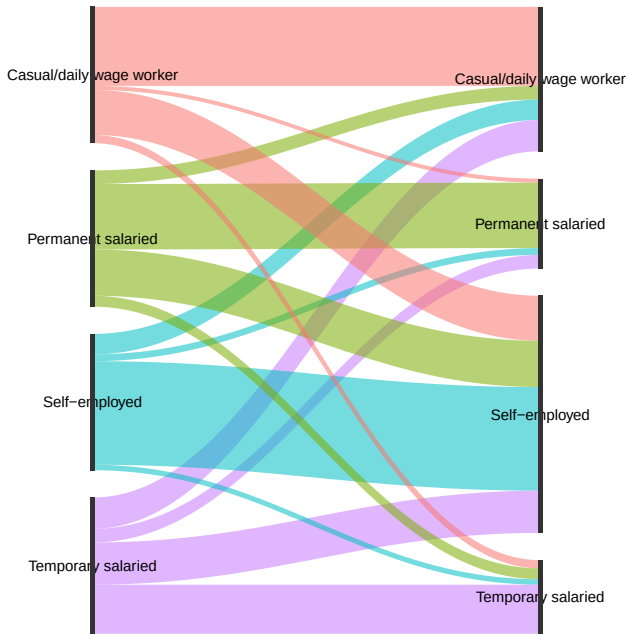


Figure 13: Step 4

Alluvial Plot Step 5: Edit the theme a bit

```
plot_3 <- plot_3 +
  theme(axis.text.x =
          element_text(
            size = 15,
            face = "bold"),
        plot.margin =
          unit(c(1,1,1,1), "cm"))
```

Alluvial Plot: Next Steps

The following extensions are remaining are:

- Adding **percentage labels**
- **Aligning** the labels

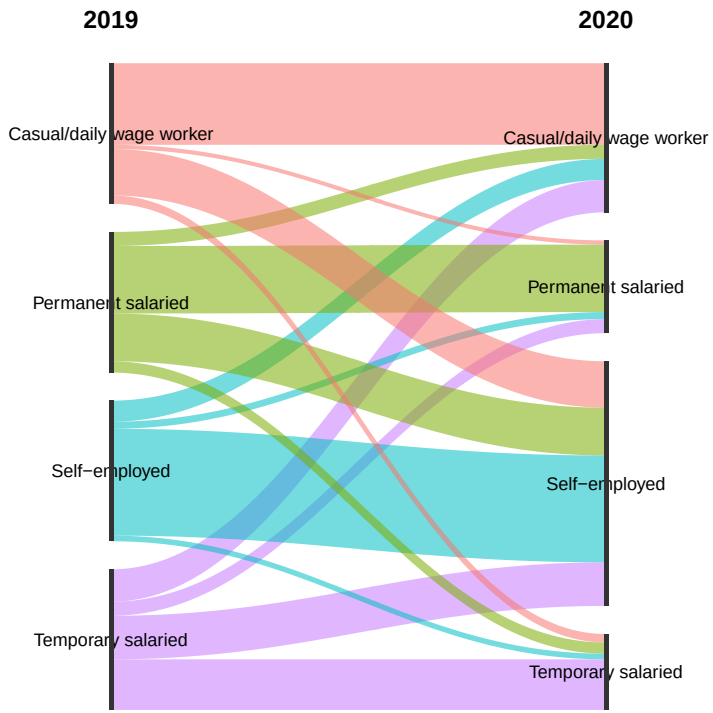


Figure 14: Step 6

Visualisation 4

Load the data

We will use the dataset called `map_data`, to make a dumbbell plot. This is a subset, we will share the full data, later on, if needed.

1. `recovery_male` : % of male workers who lost, and recovered jobs
2. `noeffect_male` : % of male workers who didn't change job status
1. `recovery_female` : % of female workers who lost, and recovered jobs
3. `noeffect_female` : % of female workers who didn't change job status
4. `recovery_total` : % of workers who lost, and recovered jobs
5. `noeffect_total` : % of workers who didn't change job status
6. `ST_NM` : State name

Dumbbell Plot Step 1: Prepare the data

```
plot_data <- arrange(map_data,
                     recovery_male) %>%
  mutate(state.fact = factor(ST_NM,
                             levels = unique(ST_NM)))
```

Dumbbell Plot Step 2: Prepare some colours

You can just type in some colours - ‘red’, ‘seagreen’, ‘purple’, or add a hexcode. Remember, these are going in as strings, so you need to add the quotation marks!

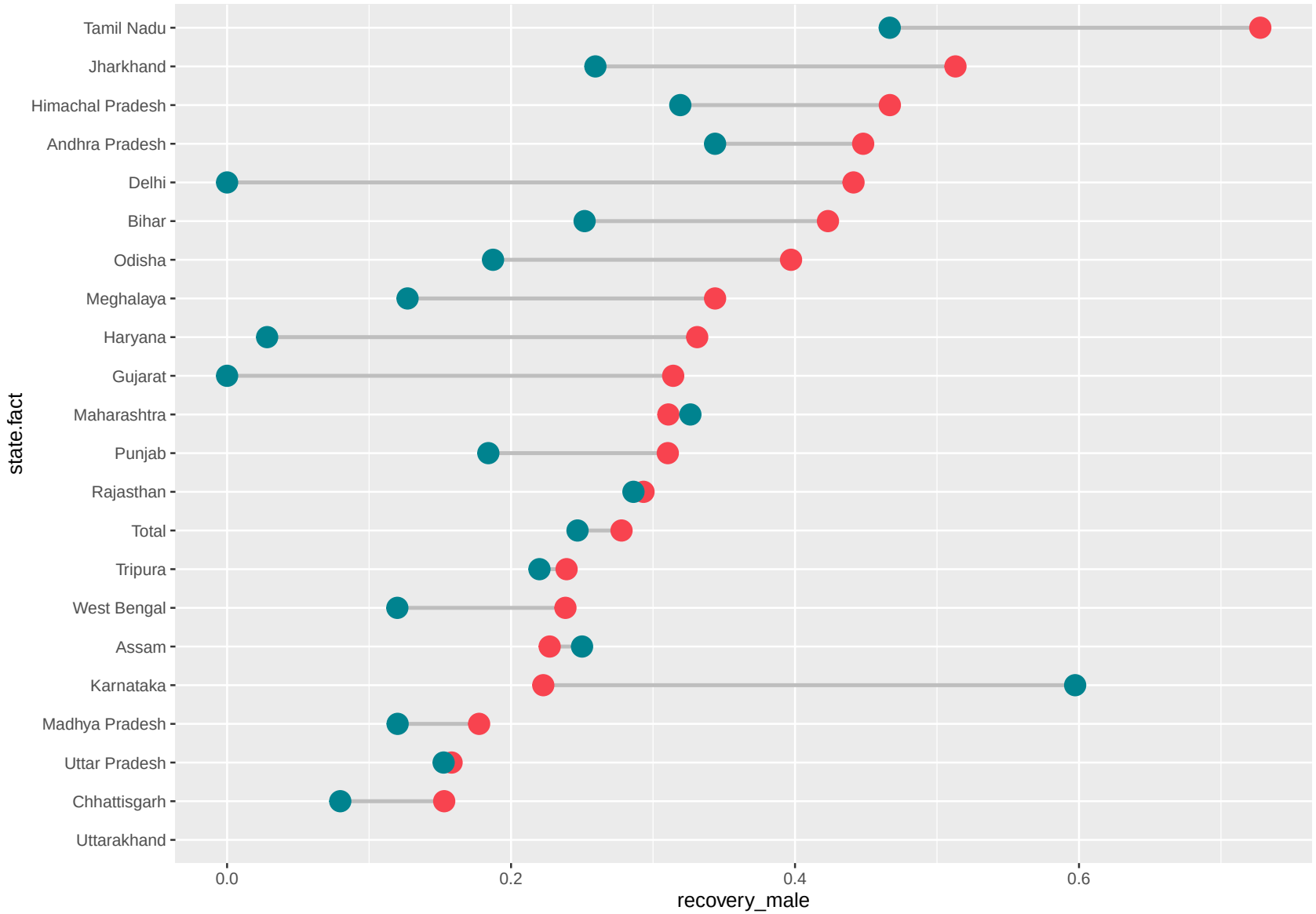
There’s a really nice colour tool that you can access here.

```
female <- "#00838f"
male <- "#f8434f"
```

Dumbbell Plot Step 3: Mapping

```
plot_4 <- ggplot() +
  geom_dumbbell(
    data = plot_data,
    aes(y = state.fact,
        x = recovery_male,
        xend = recovery_female),
    size = 1,
    color = "grey",
    alpha = 1,
    size_x = 5,
    size_xend = 5,
    colour_x = male,
    colour_xend = female)
```

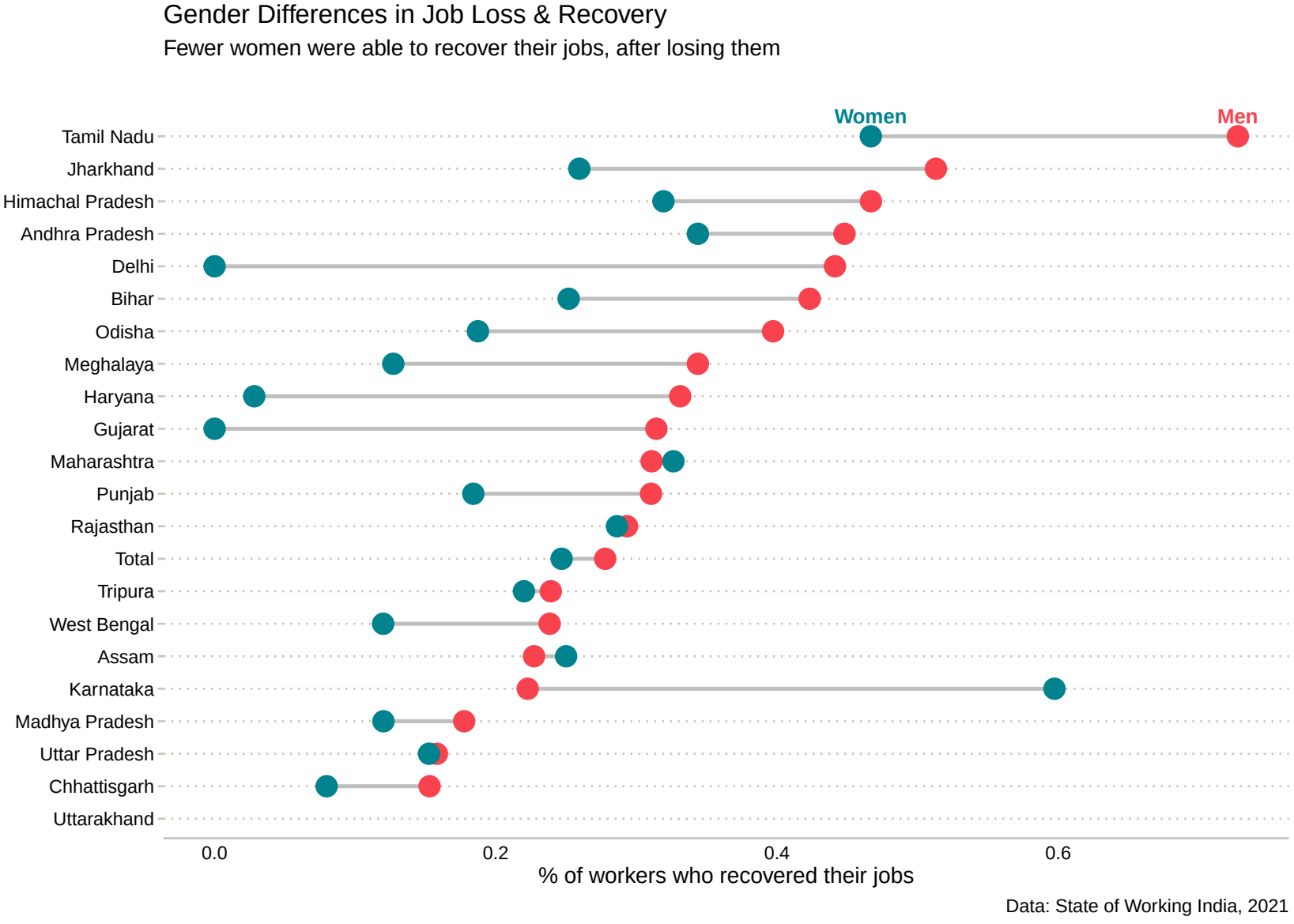
Dumbbell Plot Step 3: Mapping



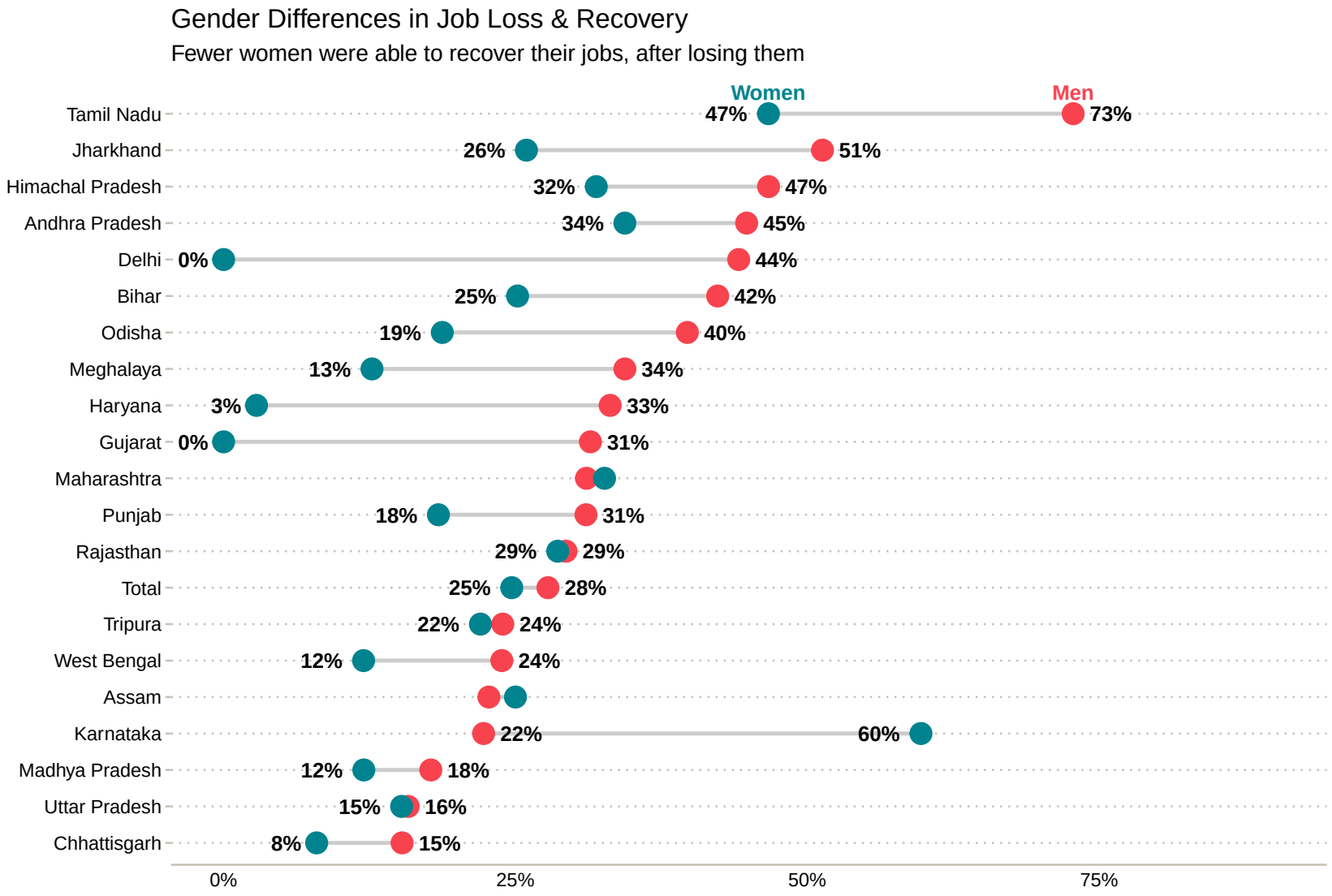
Dumbbell Plot Step 4: Add some labels

```
plot_4 <- plot_4 + geom_text(
# use this to filter the y axis; we are only adding a label on top,
#  in line with Tamil Nadu - so that it isn't cluttered
    data = filter(plot_data,
                  ST_NM == "Tamil Nadu"),
    aes(x = recovery_male,
        y = ST_NM,
        label = "Men"),
    color = male,
    vjust = -.9,
    fontface = "bold") +
# Don't forget to use your theme!
  my_theme()
```

Try to add labels wherever needed, by yourself



This is super cluttered—how do we make it better?!



Data: State of Working India, 2021

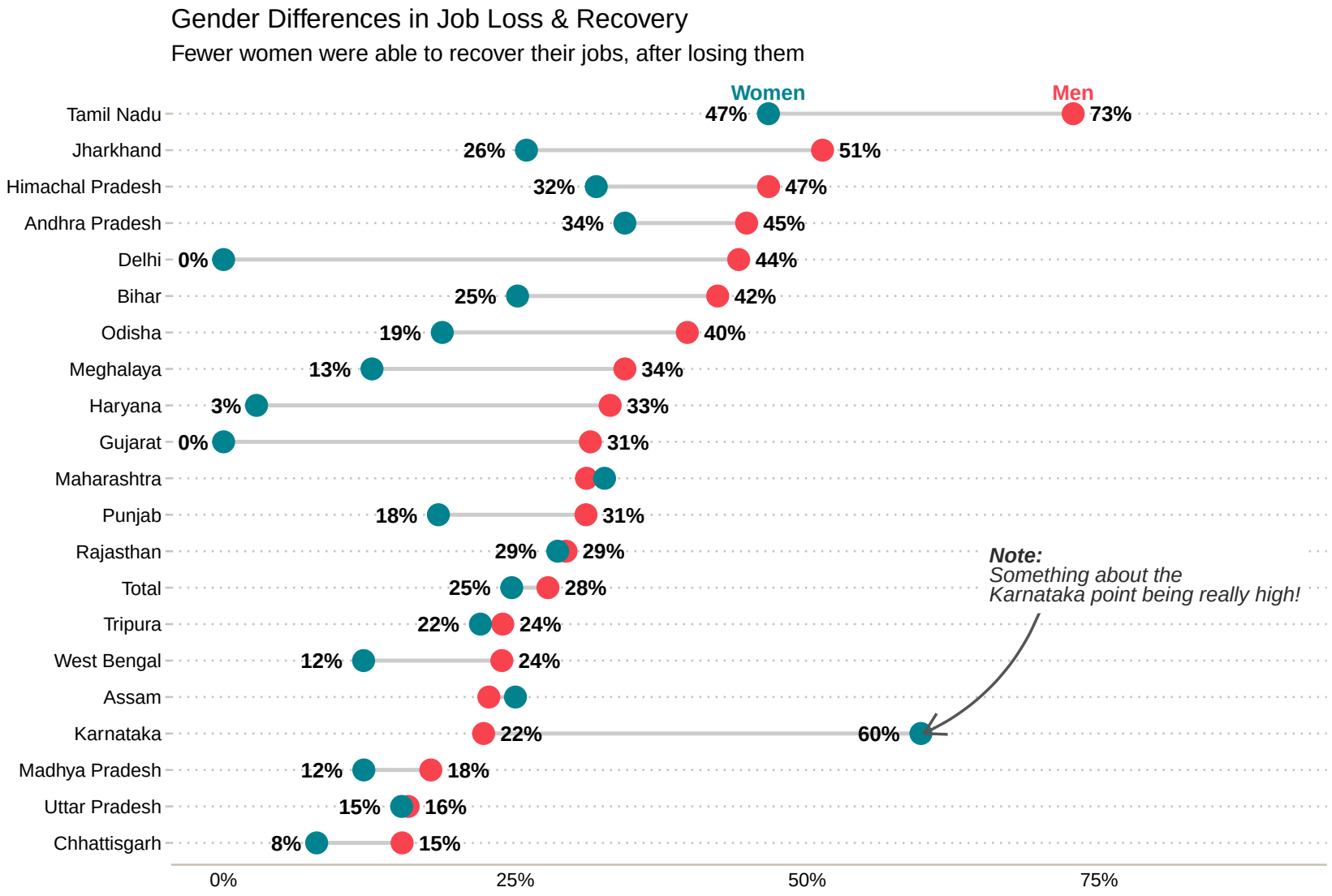
Annotation

The Karnataka point is super high—maybe we can add some explanation there, direct attention to it?

Again, this is done by geoms. For instance, thid adds a curved arrow

```
plot_4 <- plot_4 + geom_curve(
    aes(
      x = .7,
      y = 7.4,
      xend = .6,
      yend = 4),
    colour = "#555555",
    curvature = -0.2,
    size = .7,
# NPC stands for Normalised Parent Coordinates
    arrow = arrow(length = unit(0.03, "npc")))
```

```
plot_4 <- plot_4 +
  geom_richtext(
    aes(x = .65, y = 7.3,
# this is markdown syntax! ** word** for bold
# *word* for italic
# <br> is common to markdown, HTML etc - it adds a linebreak
label =
"***Note:***<br>*Something about the <br>
Karnataka point being really high!*"),
    lineheight = 0.8,
    colour = "#2b2b2b",
    fill = "white",
    vjust = 0,
    hjust = 0,
    label.size = NA,
    size = 4)
```



Data: State of Working India, 2021

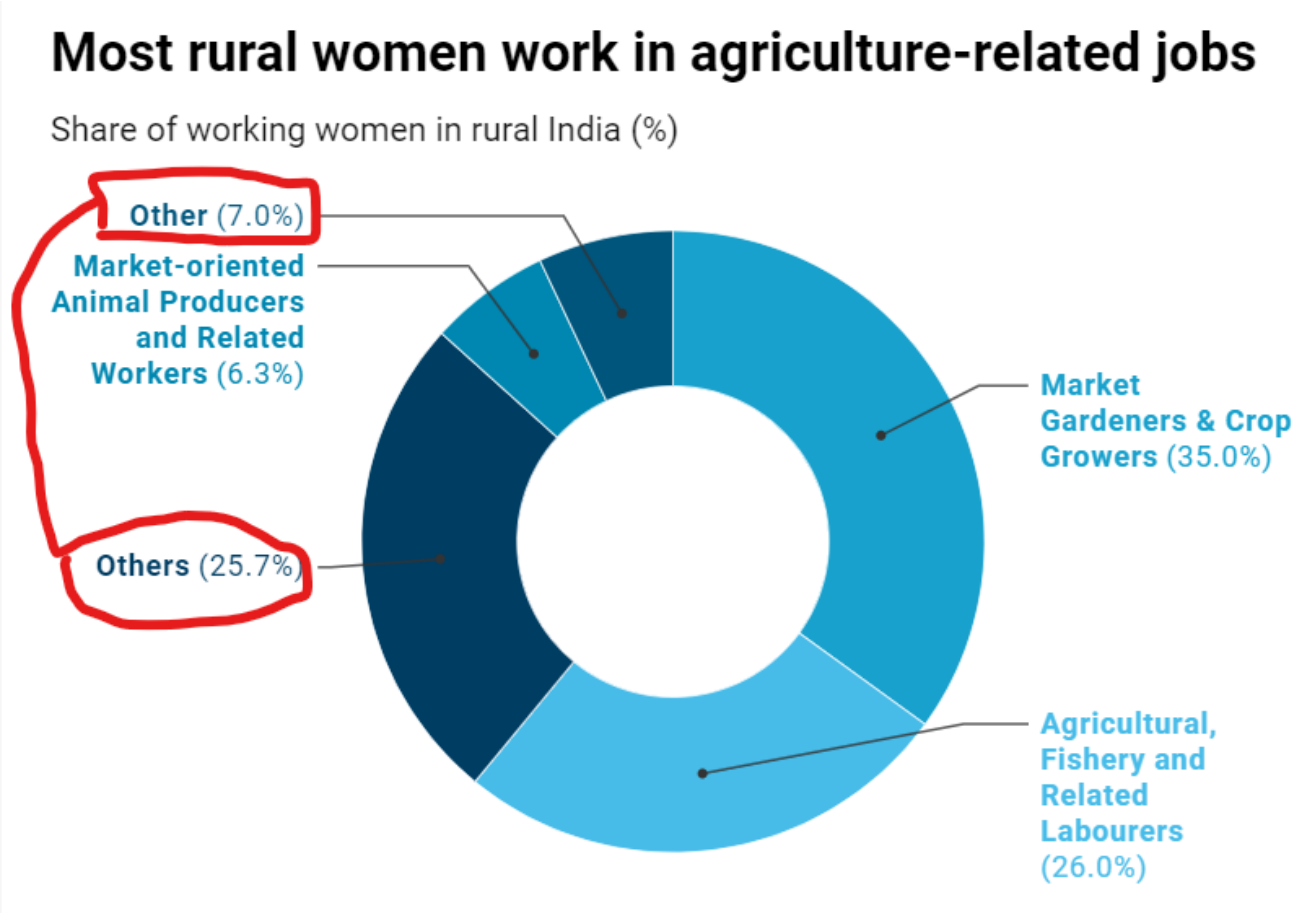
Thank You!

Data Visualisation Hall of Shame

Piecharts are bad

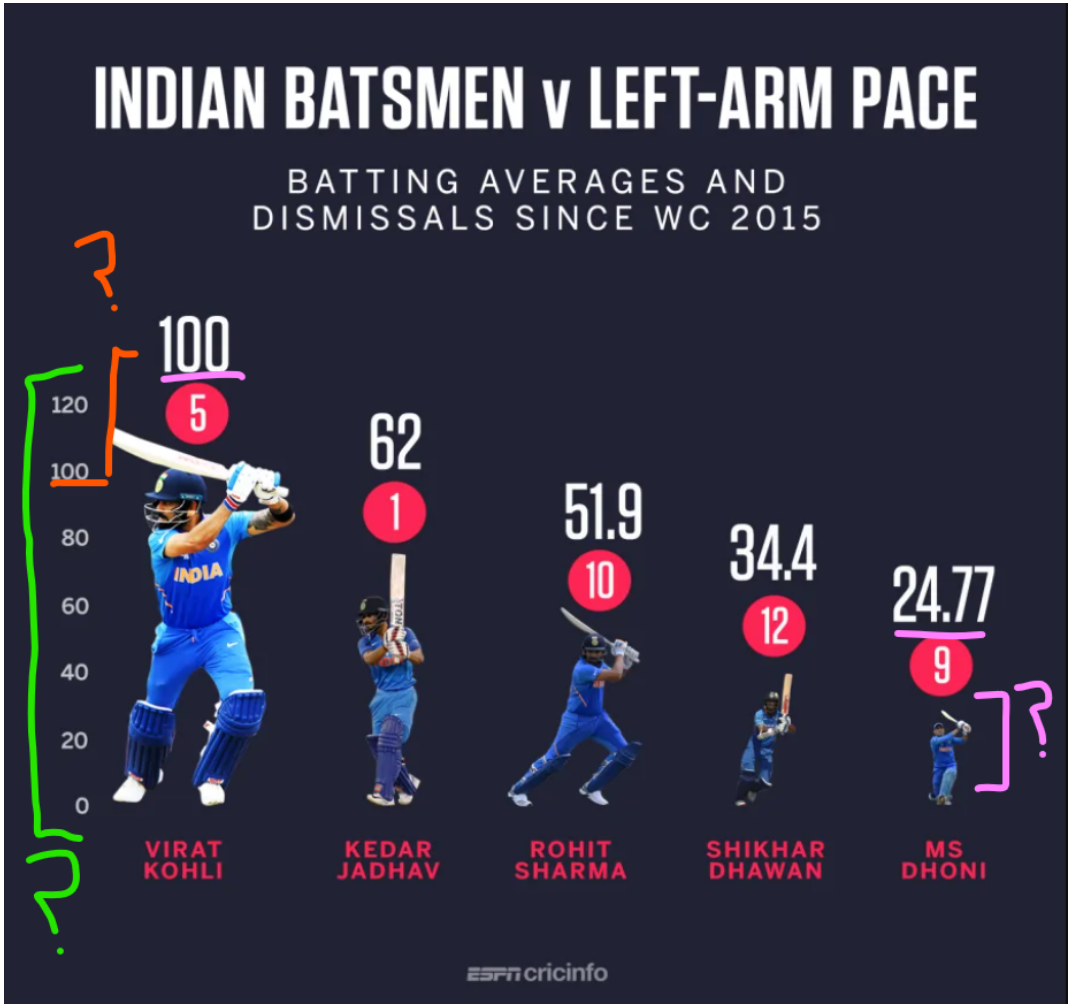
Source: Times now

Piecharts with poor colours, are really bad

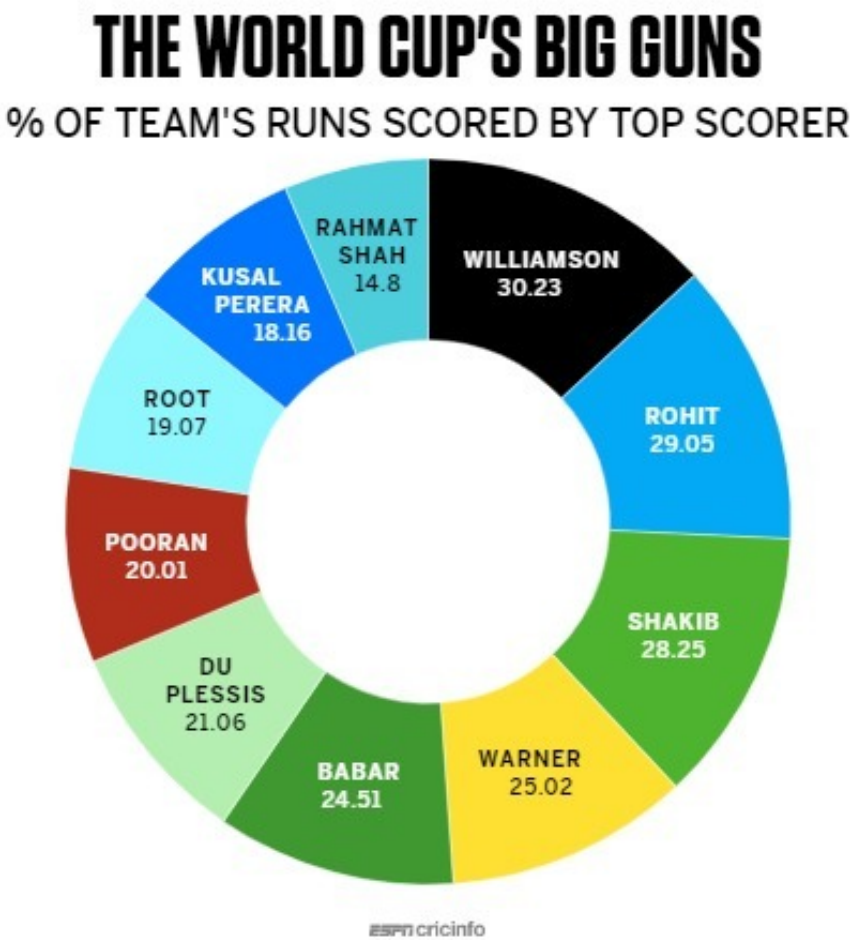


Source: Rukmini S

CricInfo is literally the worst

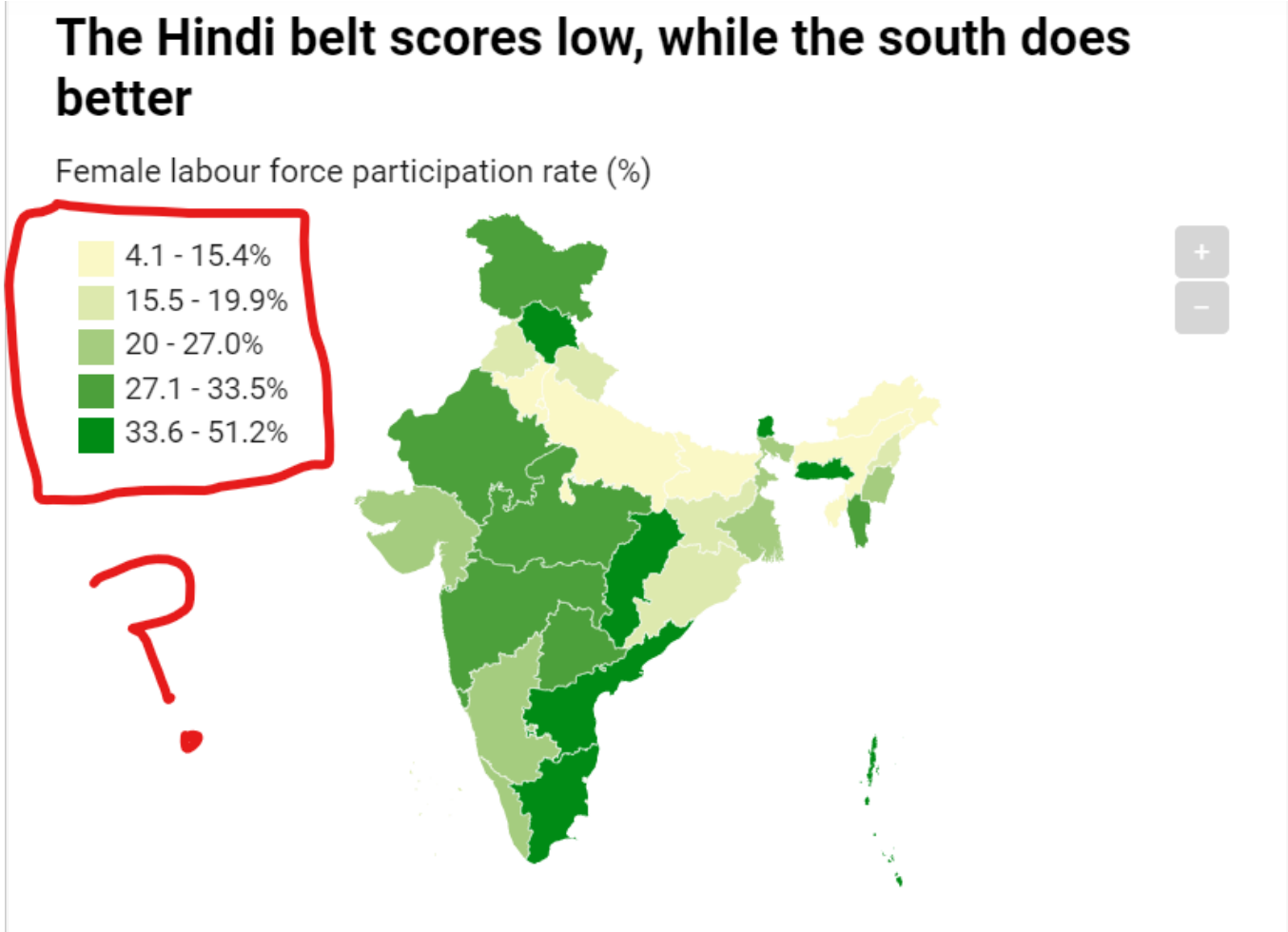


Source: CricInfo



Source: CricInfo

The scale?



Source: Rukmini S

Extra Slides

These are specified using `linetype="name"`

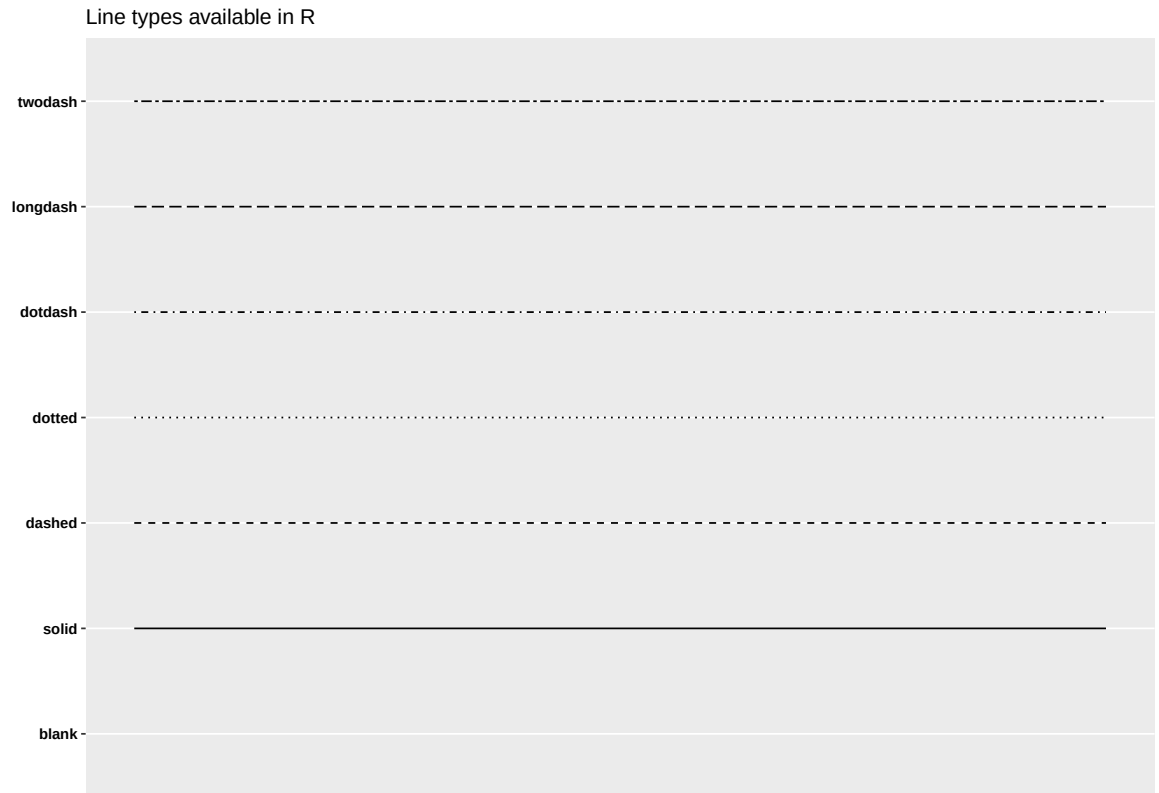


Figure 15: Line Types: Each name is an option added to linetype=

Shapes

These are specified using `shape=number`

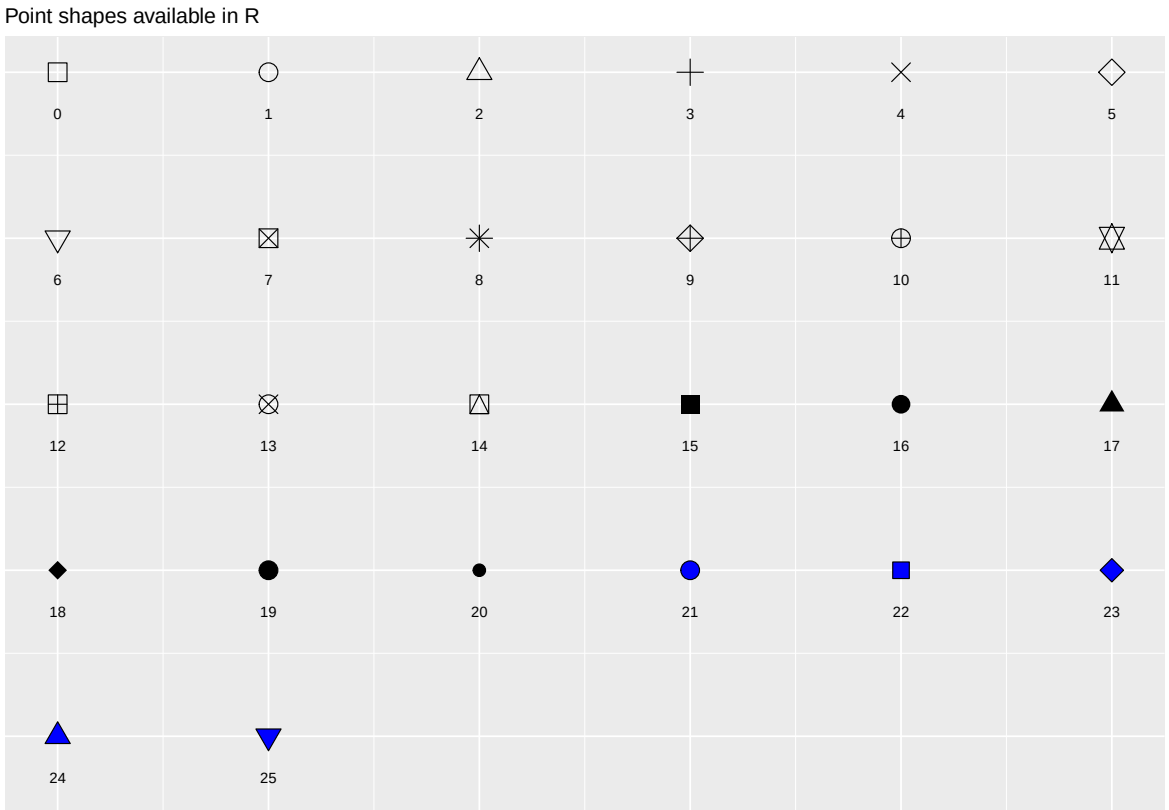


Figure 16: Shape Types: Each number is an option added to shape=

Font Family

These are specified using `family="name"`

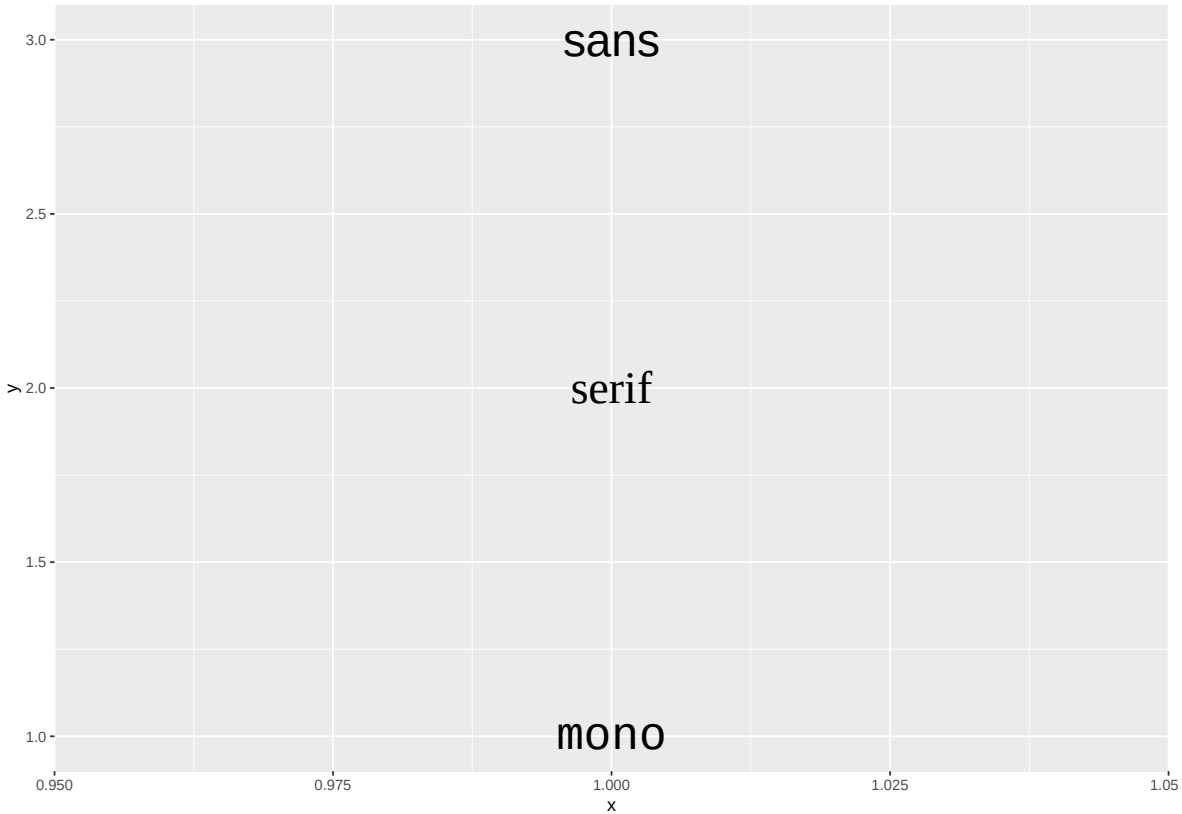


Figure 17: Font Types: Each nname is an option added to family=

Font Face

These are specified using `face="name"`

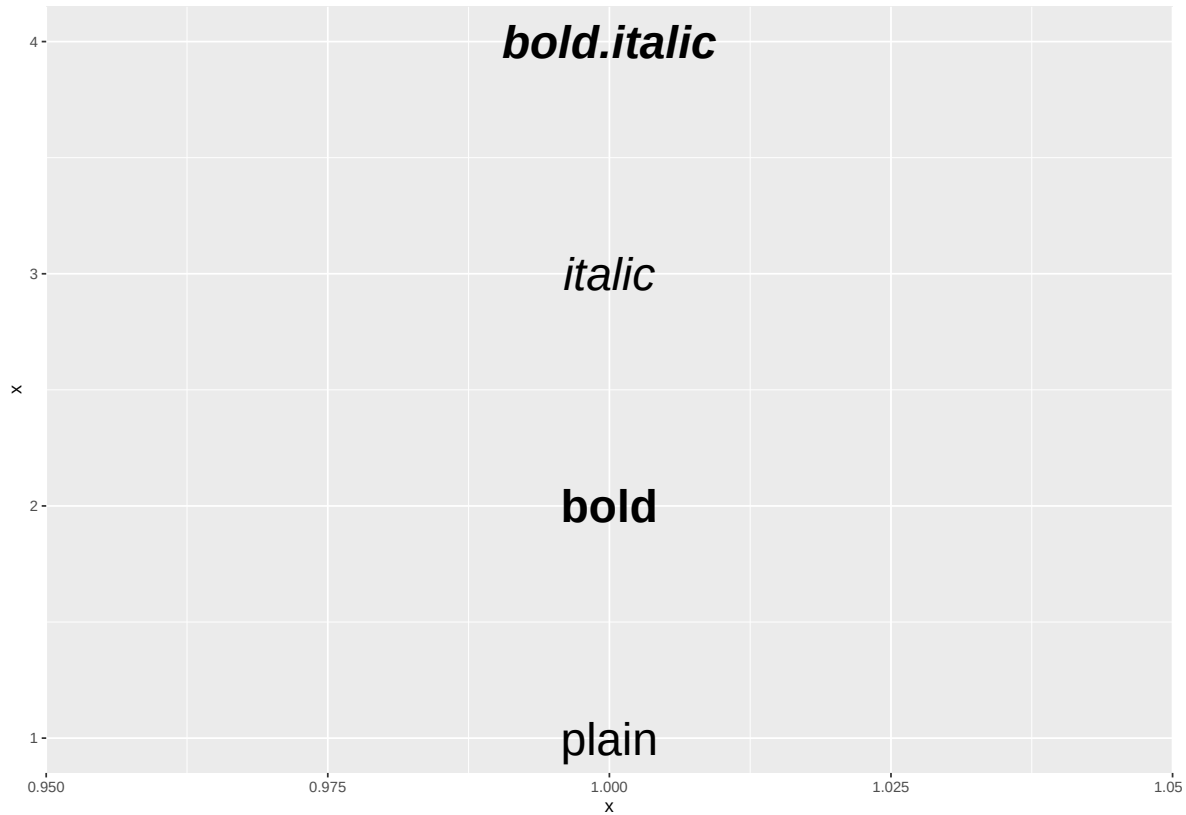


Figure 18: Font Types: Each name is an option added to face=

Vertical adjustment (vjust) & Horizontal Adjustment (hjust)

These are specified using `vjust=c(x, y)` . $x, y = [0, 1]$

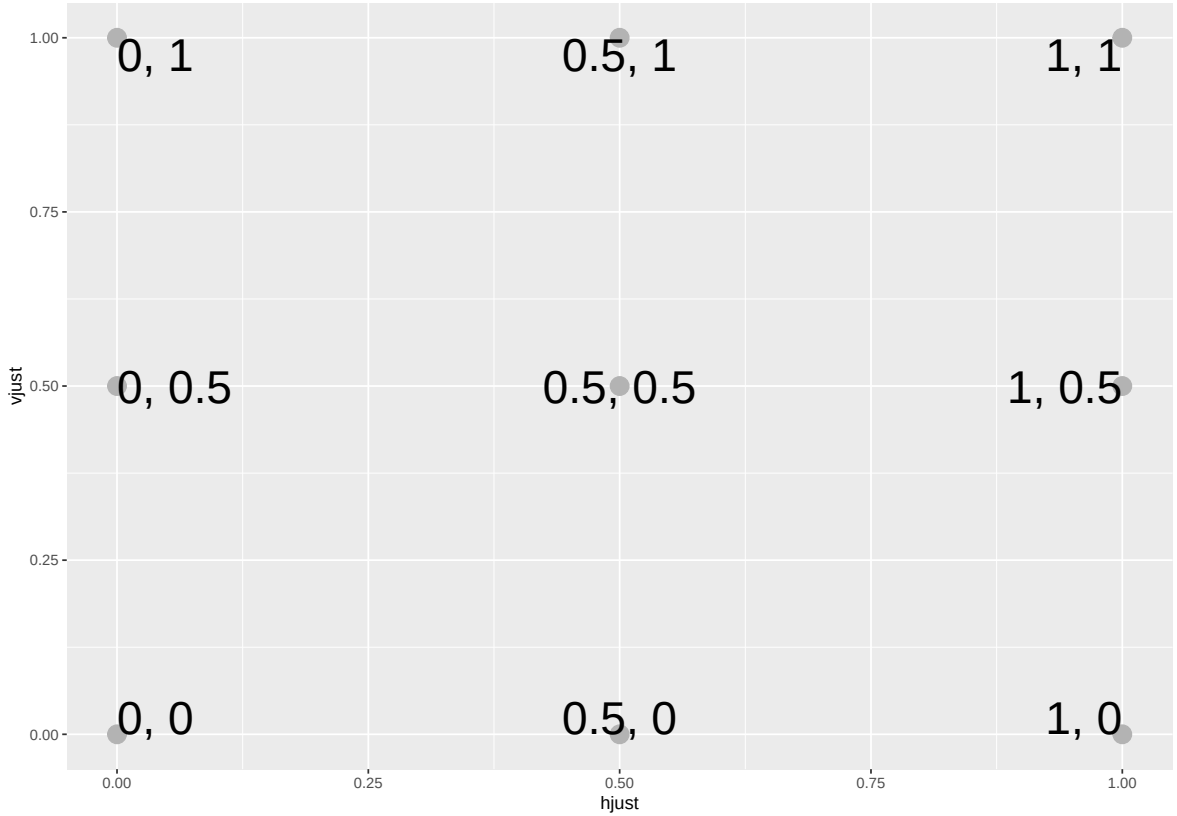


Figure 19: These are the coordinates added to vjust and hjust

Extra Extra Slides (For Nerds)

Some great resources to get great at ggplot

1. Under the hood of ggplot2 graphics in R
2. Infographics In R
3. GGplot Cookbook
4. Beautiful plotting in R: A ggplot2 cheatsheet
5. Ggplot Gallery
6. Links to cool stories
7. Best of #TidyTuesday
8. xkcd Plots With R

Some really talented artists/organisations/books/General Inspiration

1. Jasmine Mithani
2. Mona Chalabi
3. Allison Horst
4. Jim Vallandingham
5. Nadiah Bremer
6. BBC's Data Journalism Team
7. Dear Data