FE8828 Programming Web Applications in Finance - Session 4 -

Data Visualization and EDA Shiny/3: Advanced

Dr. Yang Ye yy@runchee.com

Oct 8, 2020

Section 1

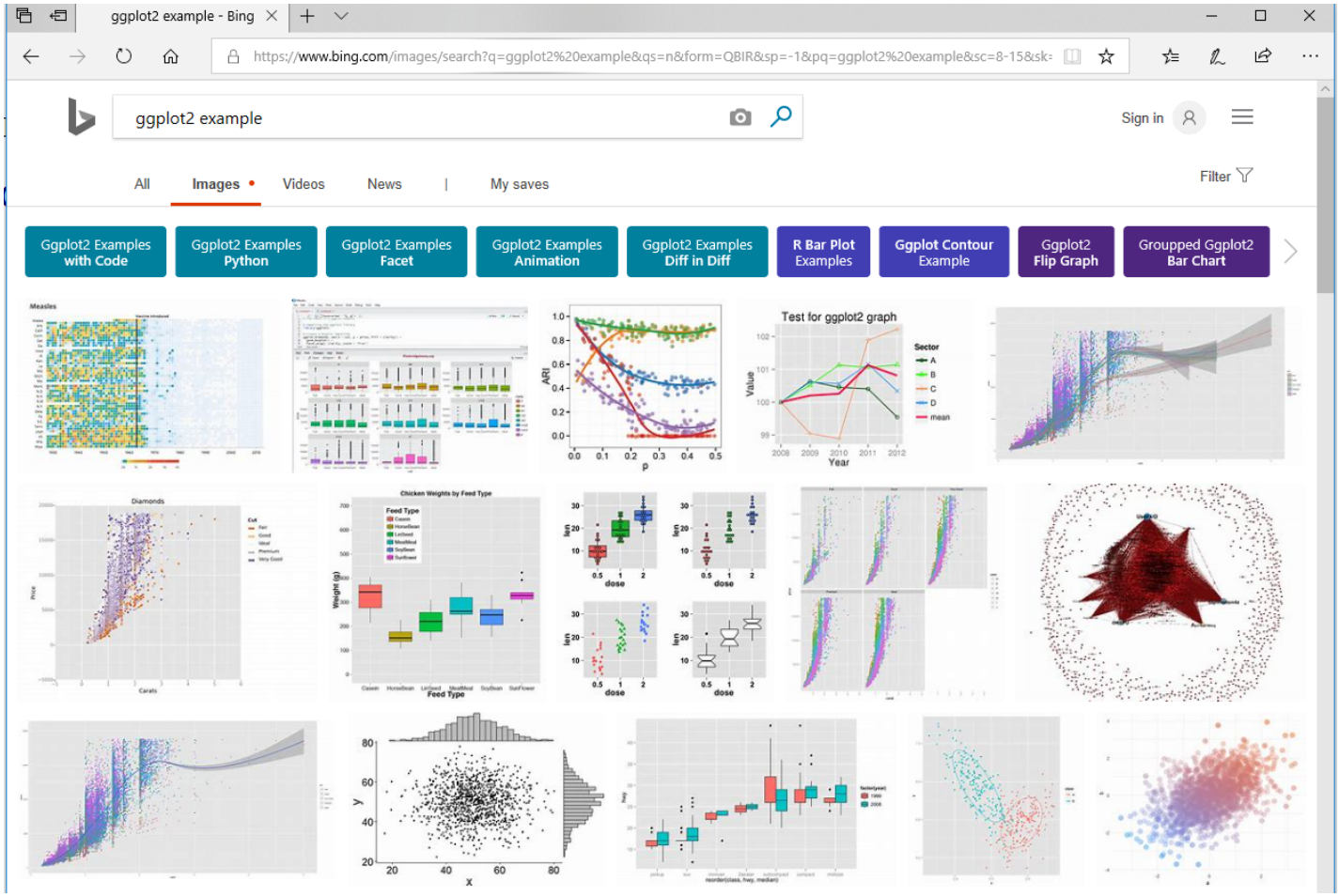
Lecture 8: Data Visualization and EDA

Lecture 8: Data Visualization and EDA

- `library(ggplot)`, author Hadley Wickham. First release on June 10, 2007.
- gg stands for "Grammar of Graphics"

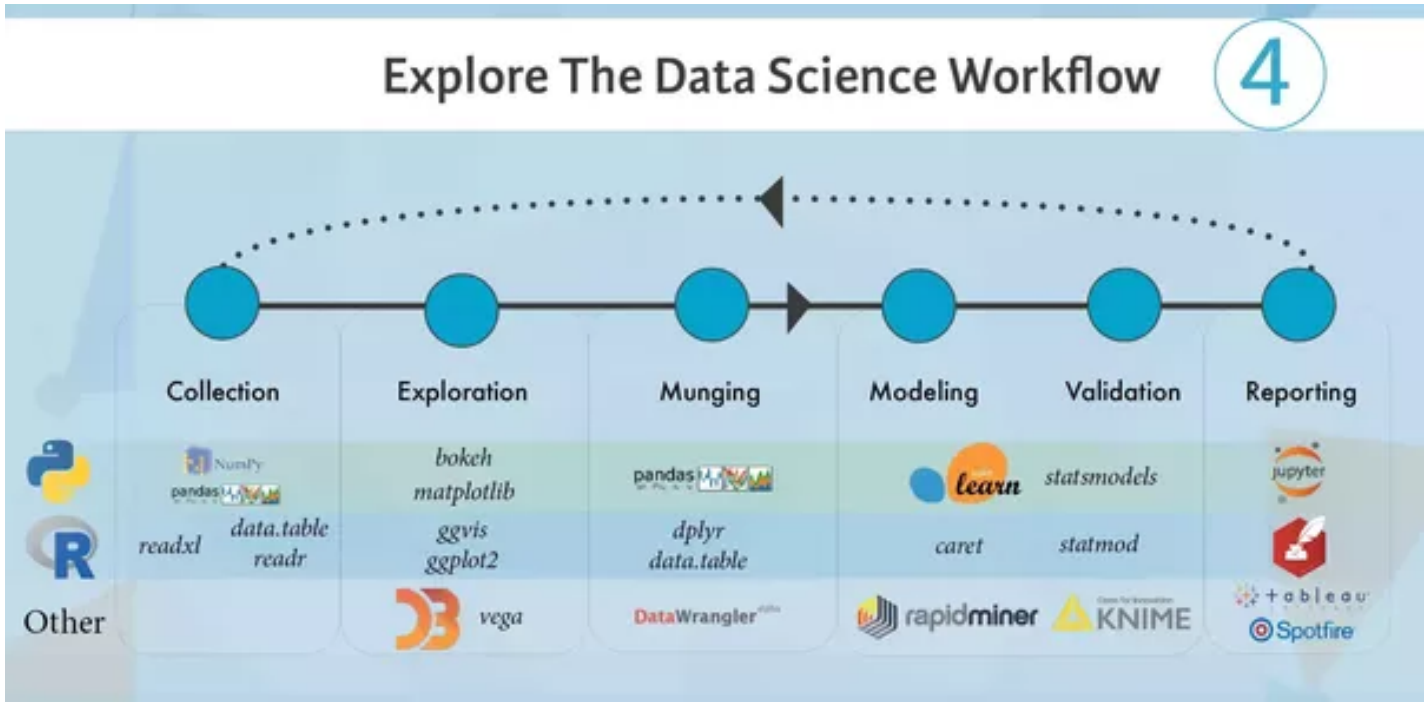
https://qz.com/1007328/all-hail-ggplot2-the-code-powering-all-those-excellent-charts-is-10-years-old/

ggplot2 Examples



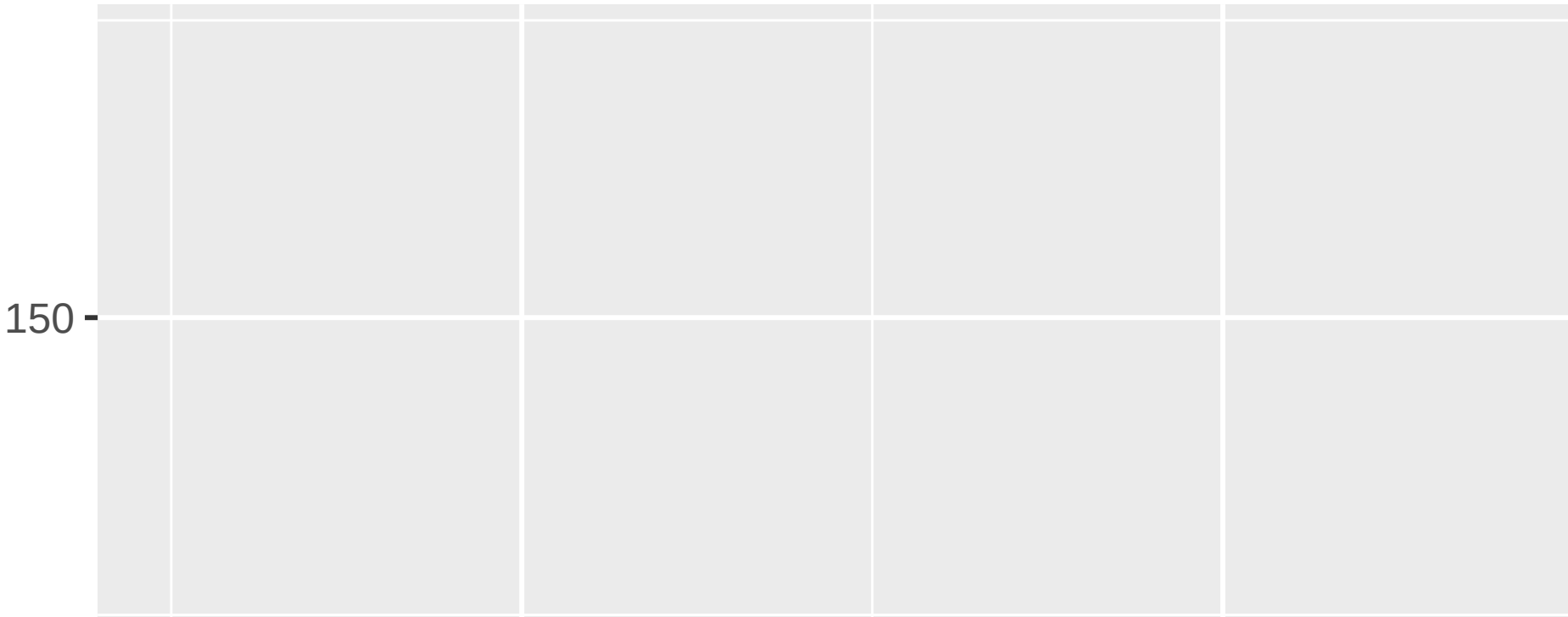
Why do we need ggplot?

It's part of the exploration of the data via visualization.



ggplot system

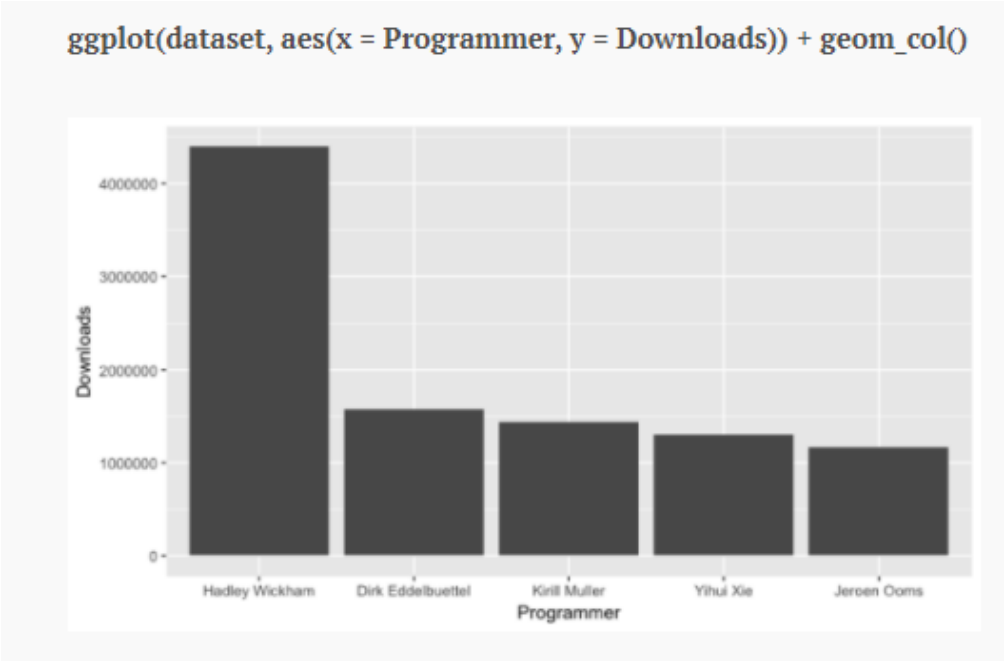
```
# library(ggplot2)
ggplot(airquality, aes(Temp, Ozone)) +
        geom_point() +
        geom_smooth(method = "loess", se = FALSE)
## `geom_smooth()` using formula 'y ~ x'
## Warning: Removed 37 rows containing non-finite values (stat_smoo
## Warning: Removed 37 rows containing missing values (geom_point).
```



Syntax of ggplot

- Definition of data + Definitions of layers

```
ggplot(data = <DATA>, ...) +
  <GEOM_FUNCTION>(mapping = aes(<MAPPINGS>))
```



Syntax of ggplot

- For example, `geom_point()` is for points, `geom_line()` is for line, `geom_smooth()` for smoothed line.
- The definition of data will pass down to the layers. But layers can have its own data.
- Put the + sign in the end of the line, not the beginning of the line.

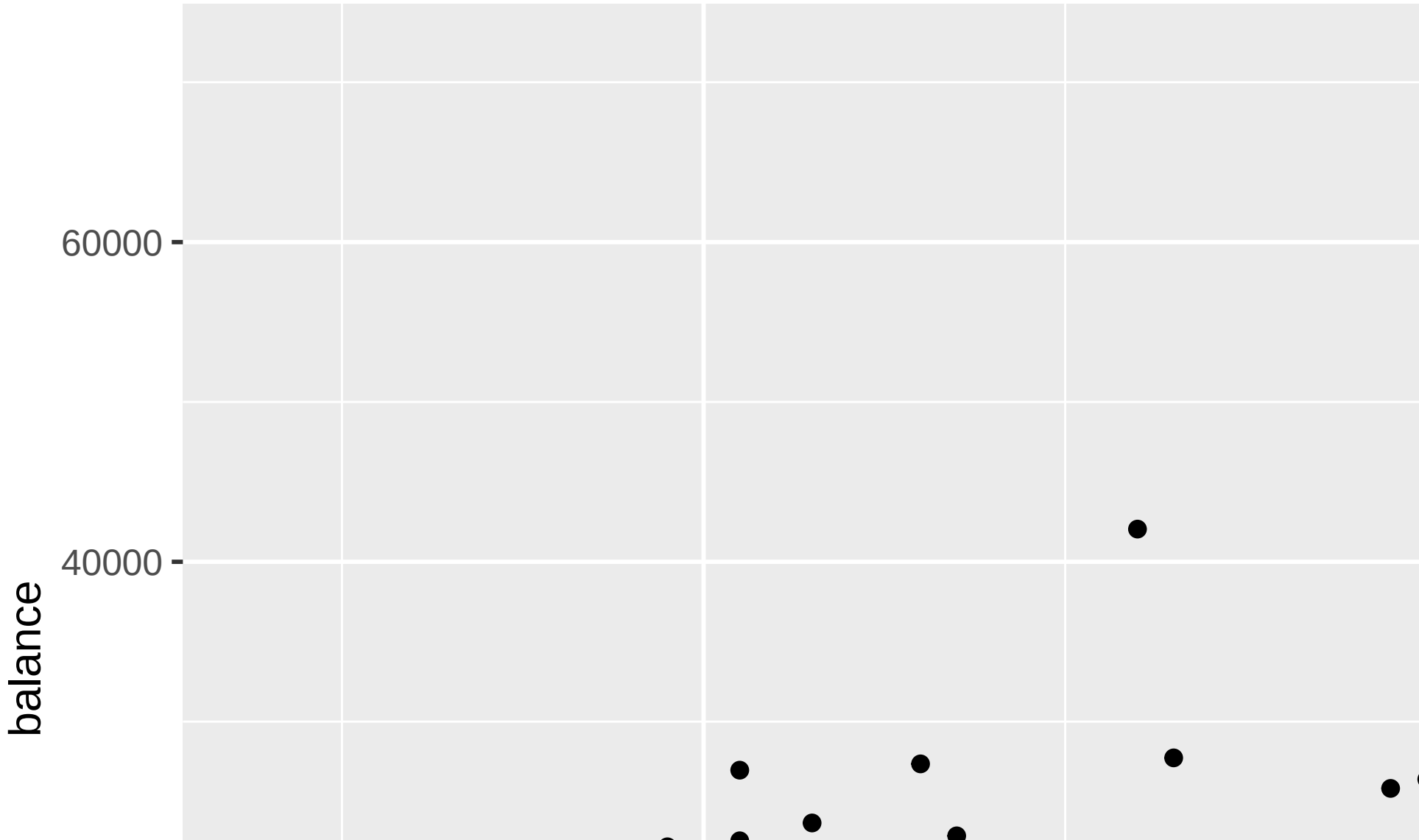
```
ggplot(data = d1, ...) +
  geom_point() + # this would get data = d1
  geom_point(data = d2, ...) # this would get data = d2
```

- Below would result in error

```
ggplot(data = <DATA>, ...)
+ <GEOM_FUNCTION>(mapping = aes(<MAPPINGS>))
```

Simply plot x and y

```
ggplot(bank, aes(age, balance)) + geom_point()
```



aes and aes_string

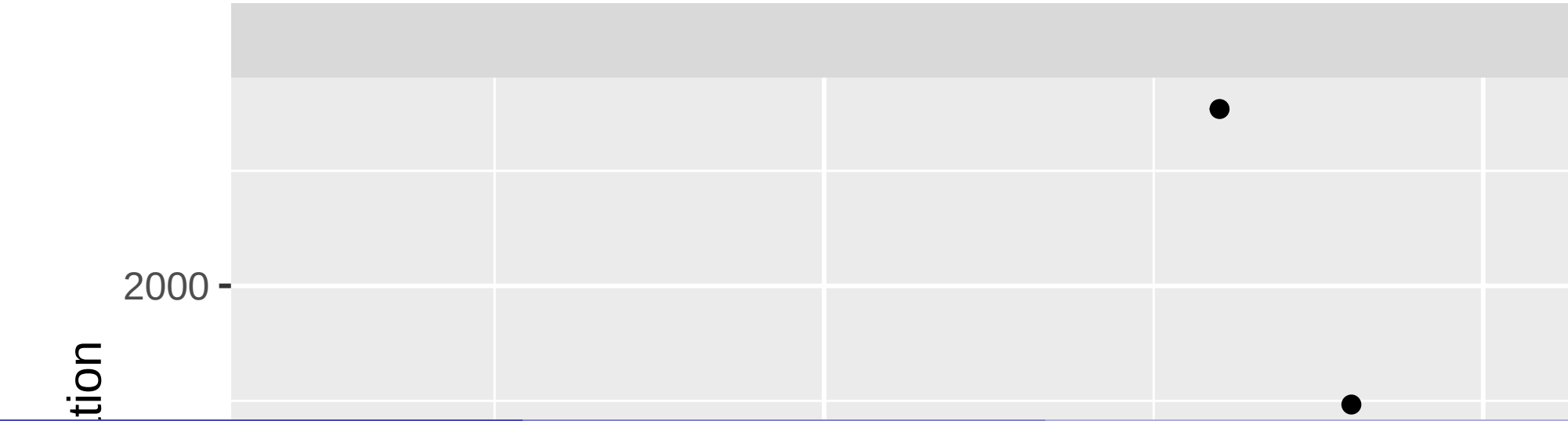
If you don't know the column name, use `aes_string` to pass variable name as string/character.

```
ggplot(bank, aes_string("age", "balance", color = "job")) + geom_poi
```

Example of `aes_string`

```
# in a Shiny app
selectInput("column", "col", choices = c("balance", "duration"))
plot_bank("age", input$col)
```

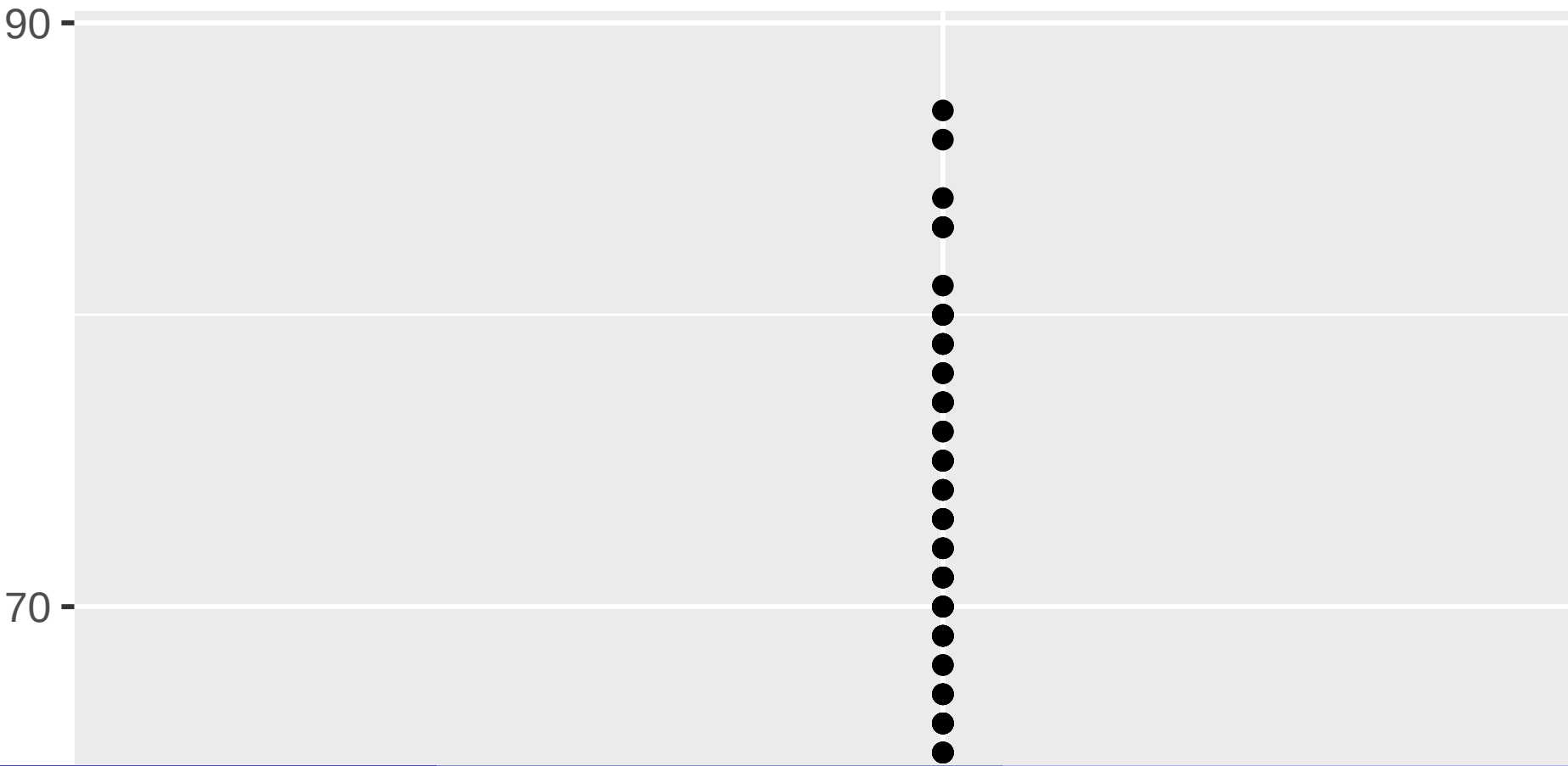
```
## `geom_smooth()` using formula 'y ~ x'
```



For non-numeric data: default and age

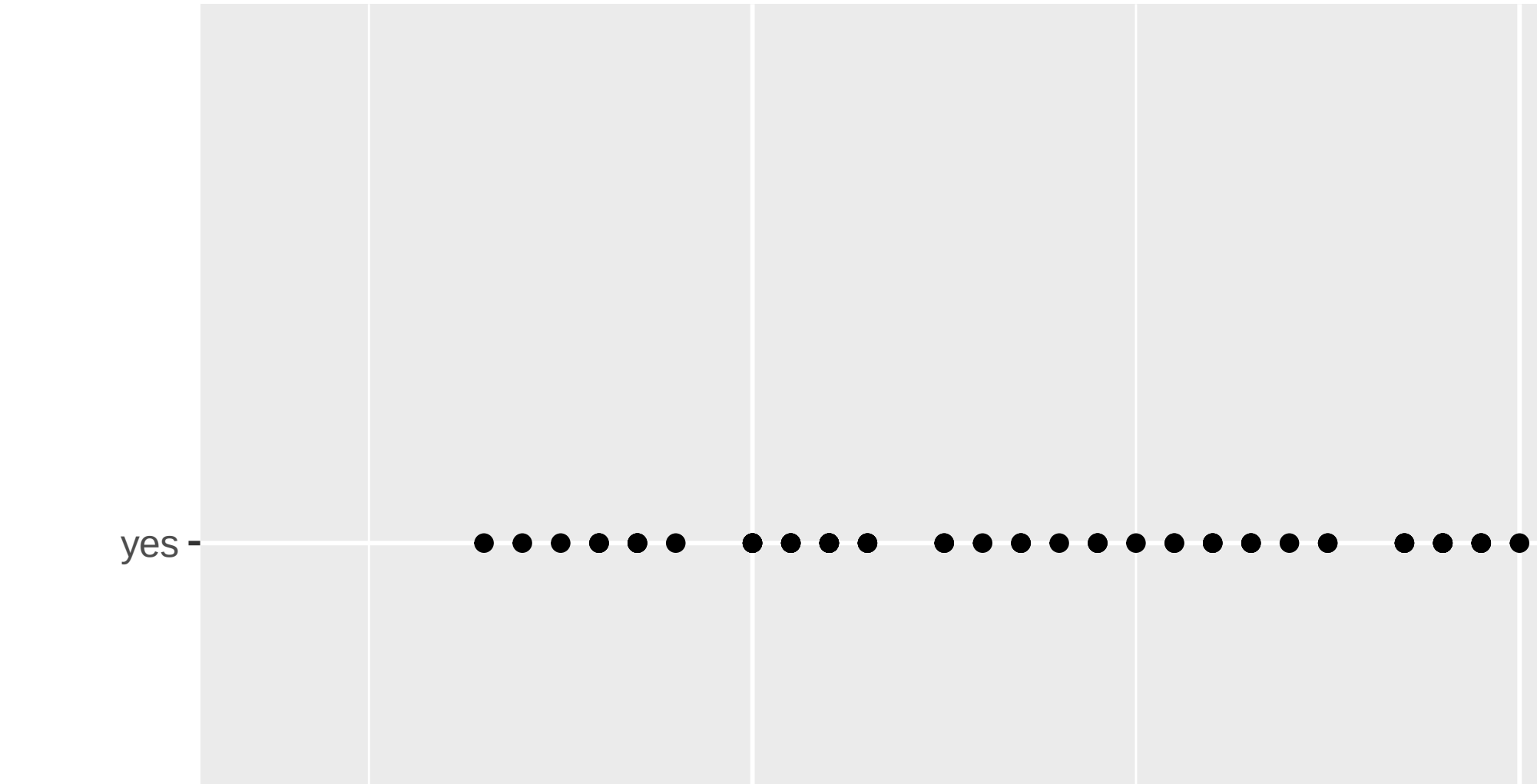
```
ggplot(bank, aes(default, age)) + geom_point()
ggplot(bank, aes(age, default)) + geom_point()
ggplot(bank, aes(job, age)) + geom_point()
```

```
ggplot(bank, aes(default, age)) + geom_point()
```



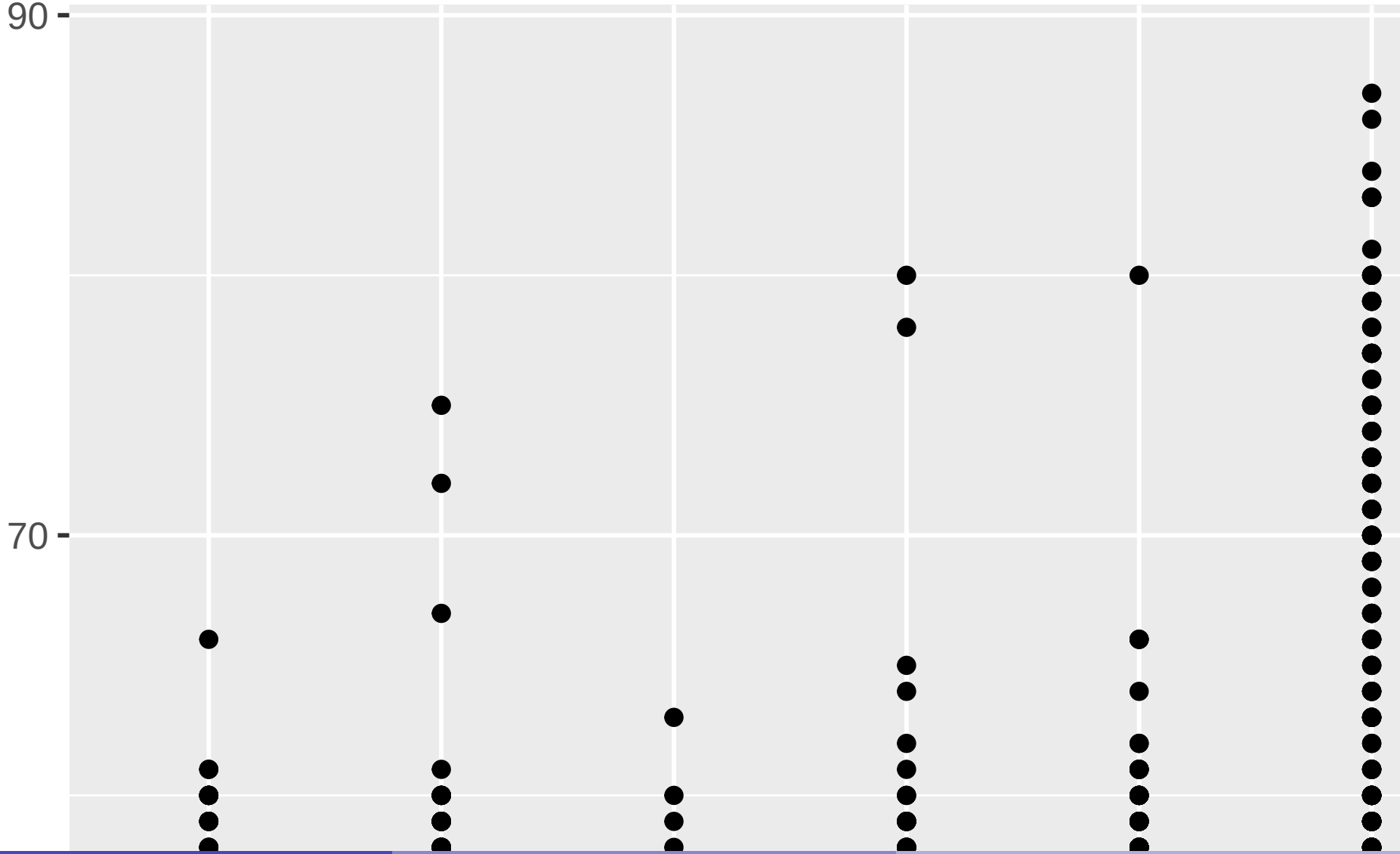
For non-numeric data: age and default (make it landscape)

```
ggplot(bank, aes(age, default)) + geom_point()
```



For non-numeric data: job and age

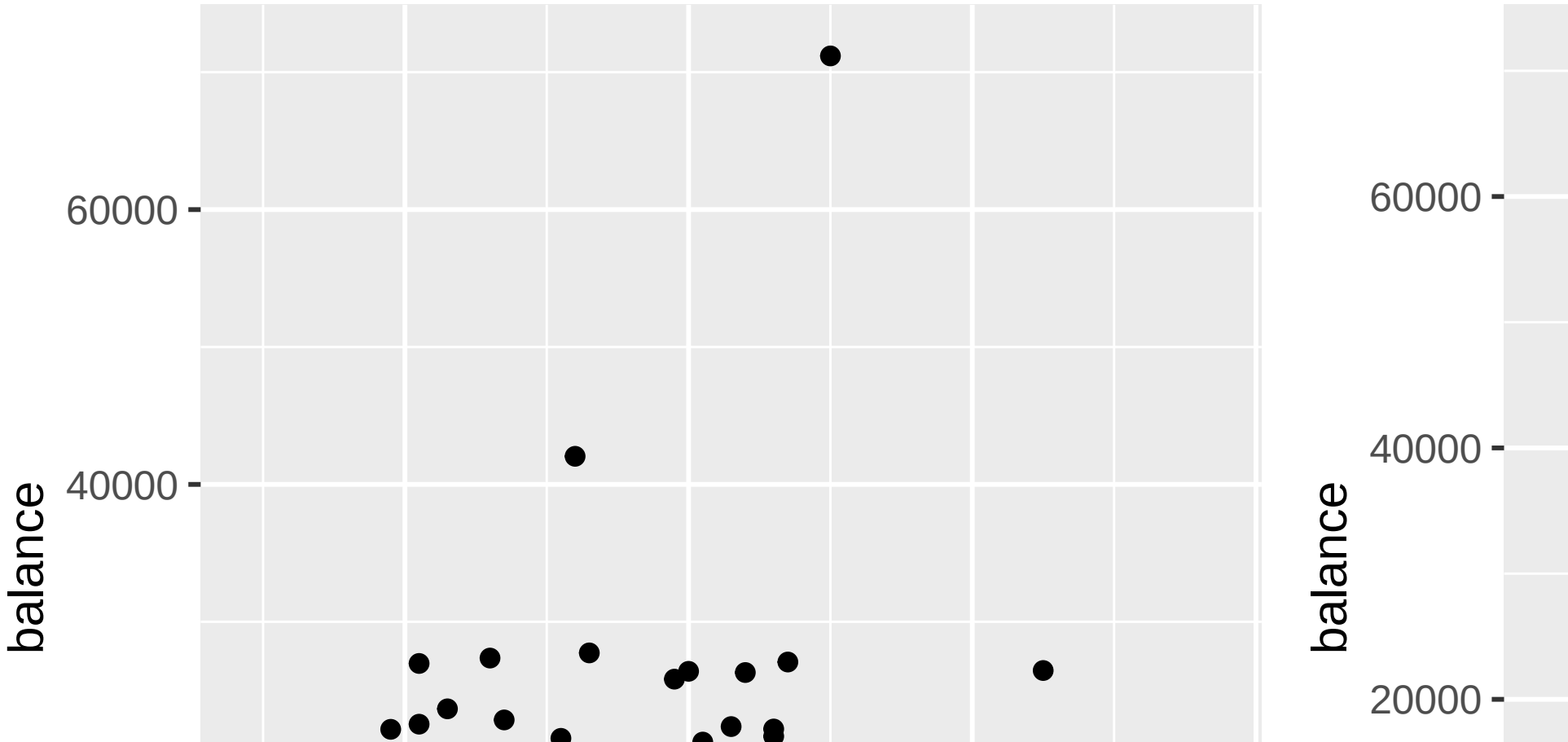
```
ggplot(bank, aes(job, age)) + geom_point()
```



Add 2nd geometry

```
ggplot(bank, aes(age, balance)) + geom_point() + geom_smooth()
ggplot(bank, aes(age, balance, color = job)) + geom_point() + geom_s
```

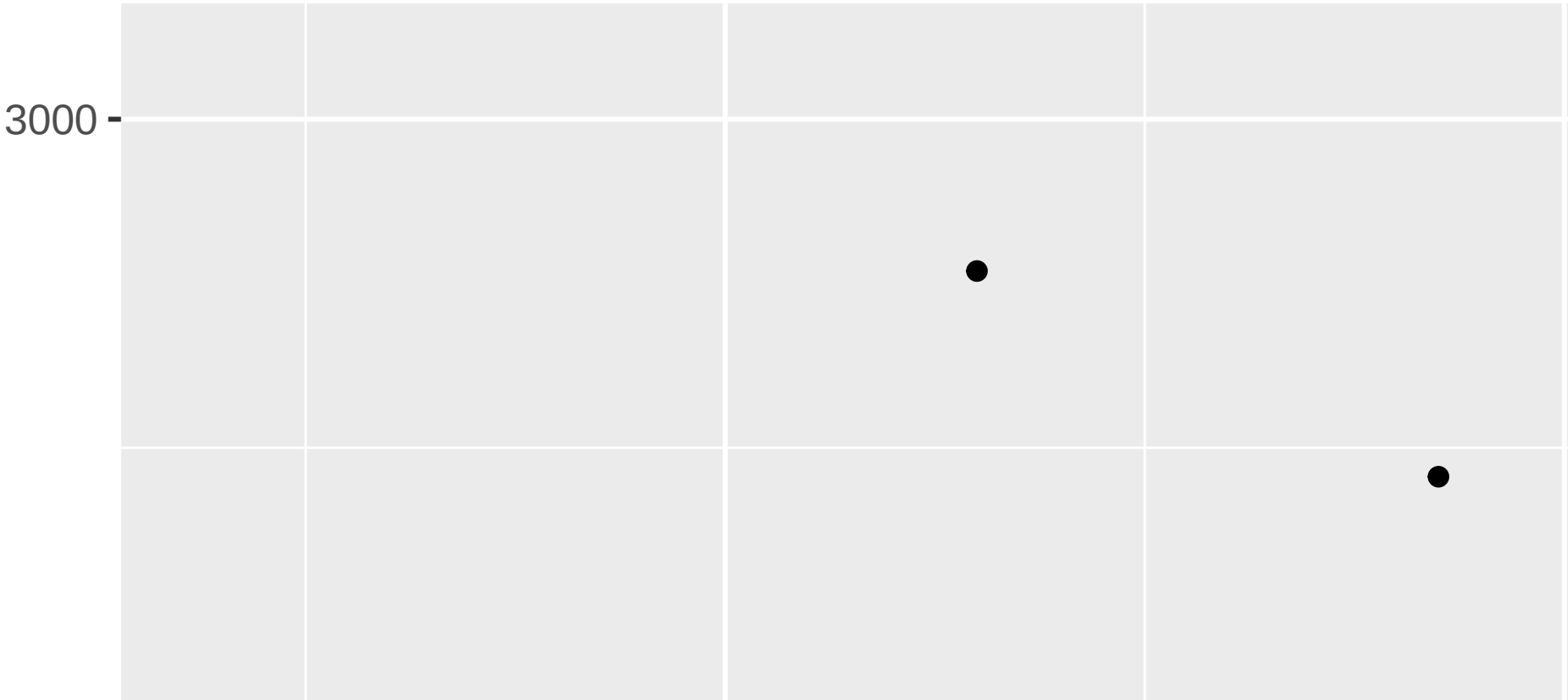
```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = '
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Pass aes down

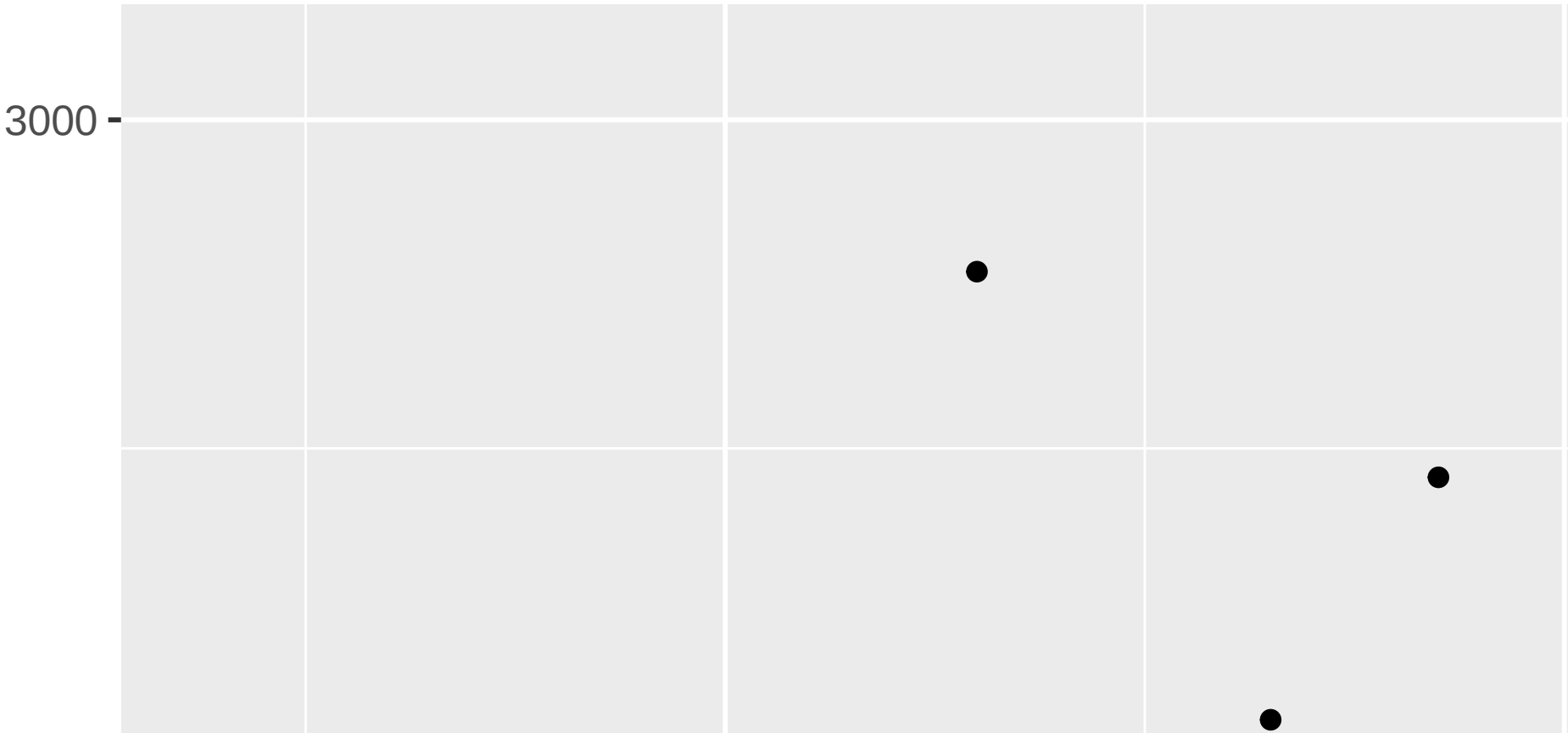
`geom_*` functions has a default parameter of `inherit.aes = T`.

```
ggplot(bank, aes(x = age, y = duration)) +
geom_smooth() + # same as geom_smooth(aes(x = age, y = duration))
geom_point() # same as geom_point(aes(x = age, y = duration))
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs =
```



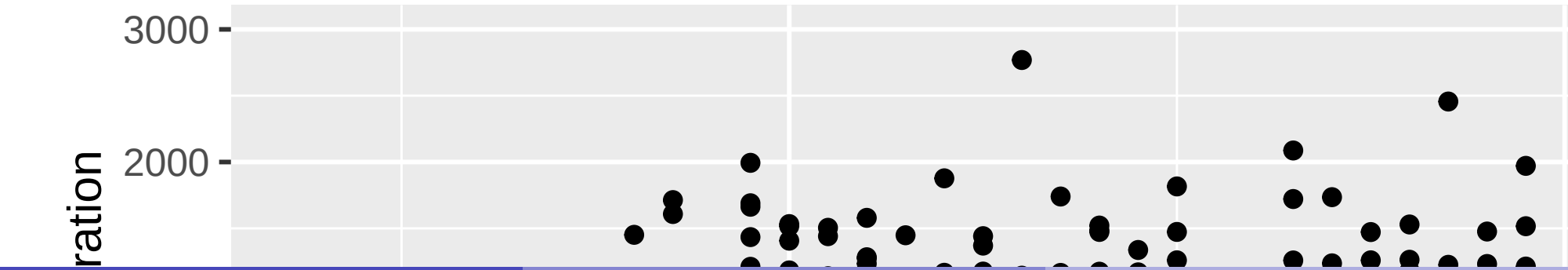
Pass aes down

```
# This is equivalent to below. But this is a bit repeating.
ggplot(bank) +
geom_point(aes(x = age, y = duration)) +
geom_smooth(aes(x = age, y = duration))
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs =
```



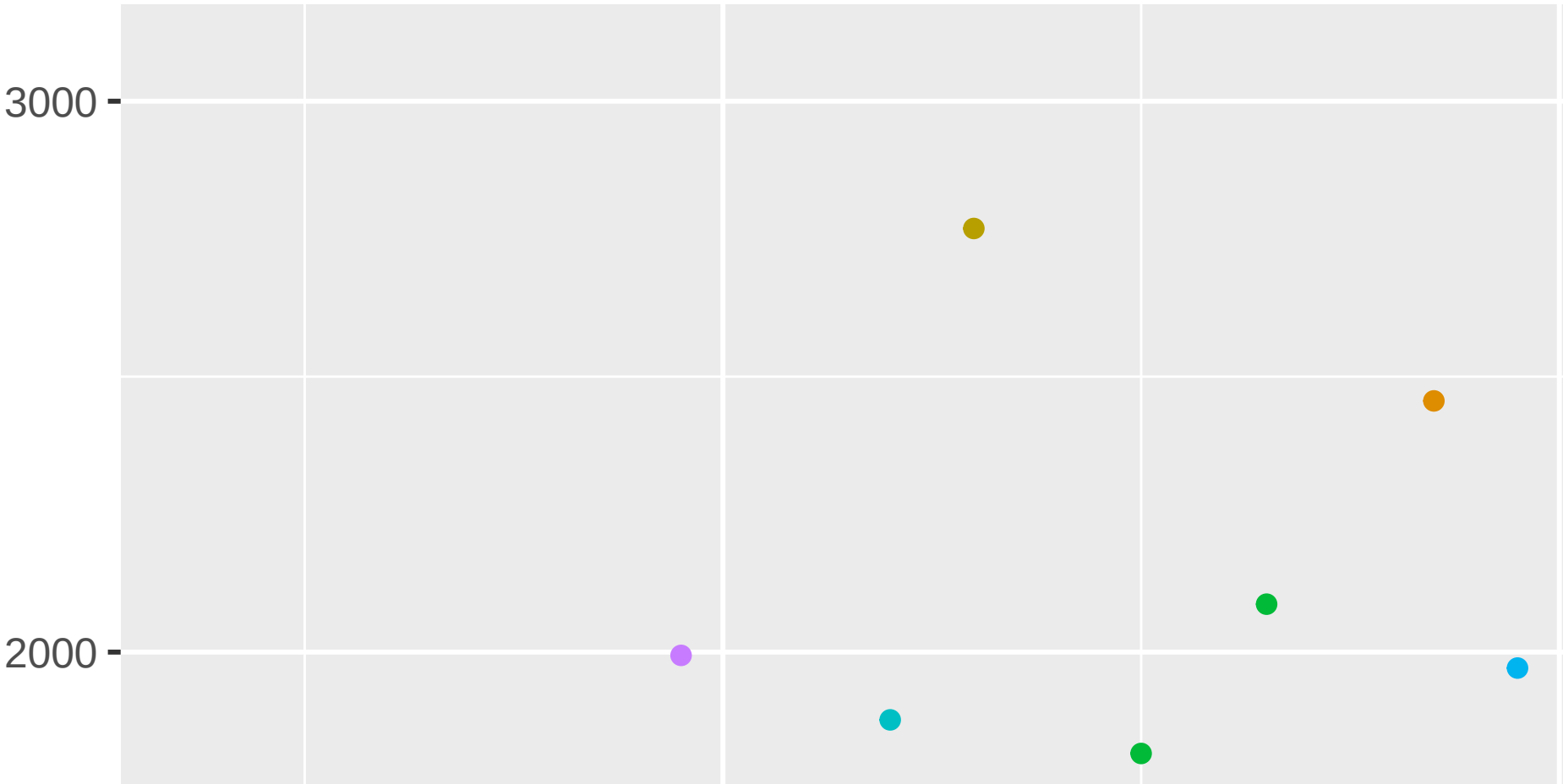
Pass aes down

```
# But repeating is useful sometimes.
# we can do specify different data and aes for different geom_* func
ggplot(bank) +
geom_point(aes(x = age, y = duration)) +
geom_smooth(data = dplyr::filter(bank, job == "entrepreneur"),
            aes(x = age, y = duration), color = "green") +
geom_smooth(data = dplyr::filter(bank, job == "blue-collar"),
            aes(x = age, y = duration), color = "blue") +
geom_smooth(data = dplyr::filter(bank, job != "entrepreneur"),
            aes(x = age, y = duration), color = "orange")
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs =
```



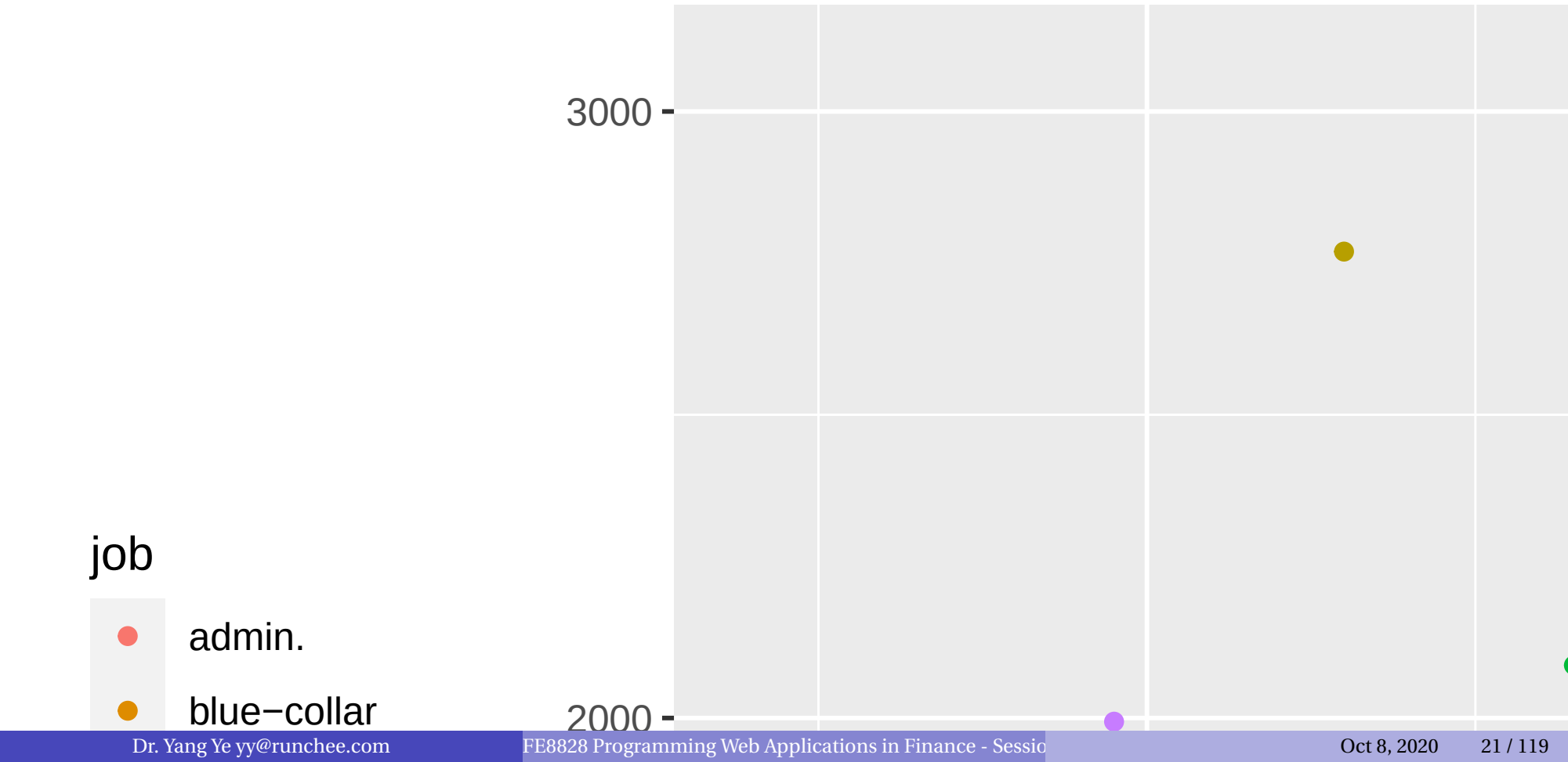
Adjustment: legend bottom

```
# adjust legend position
ggplot(bank, aes(x = age, y = duration, color = job)) +
geom_point() +
theme(legend.position="bottom")
```



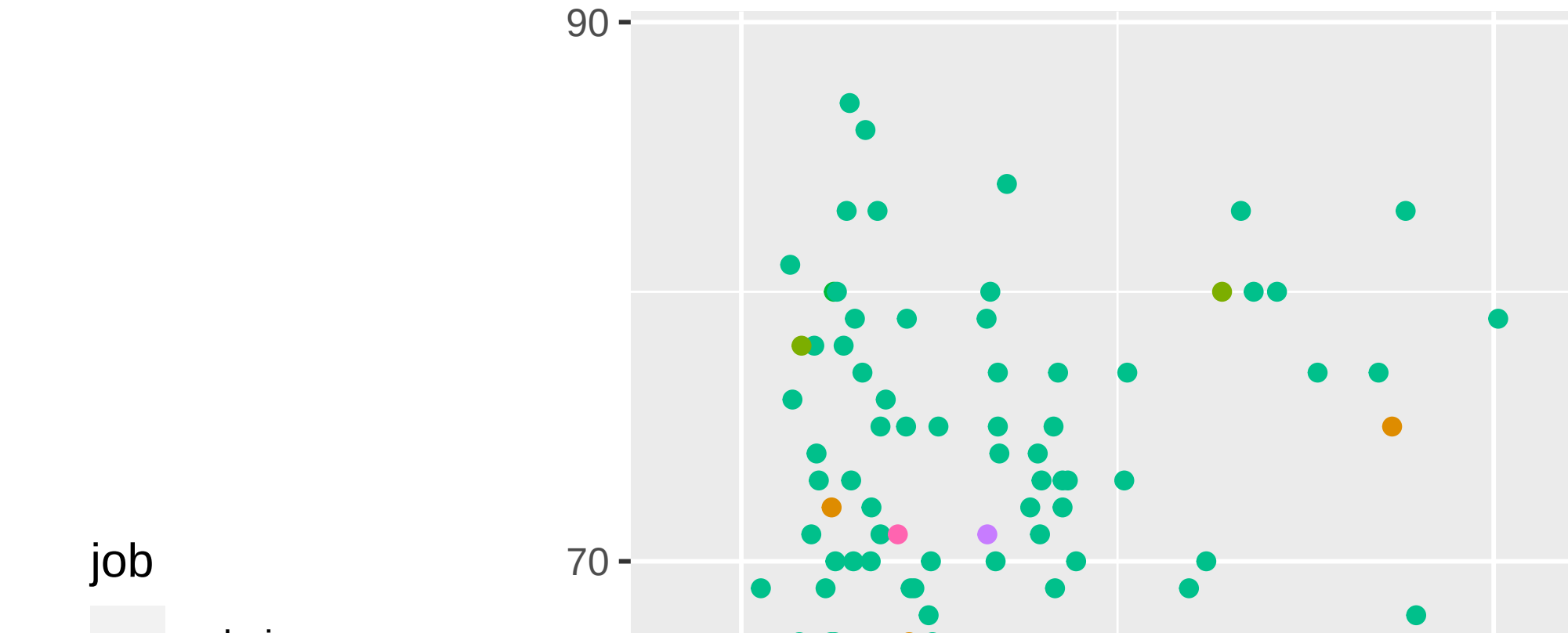
Adjustment: legend left

```
# legend to the left
ggplot(bank, aes(x = age, y = duration, color = job)) +
geom_point() +
theme(legend.position="left")
```



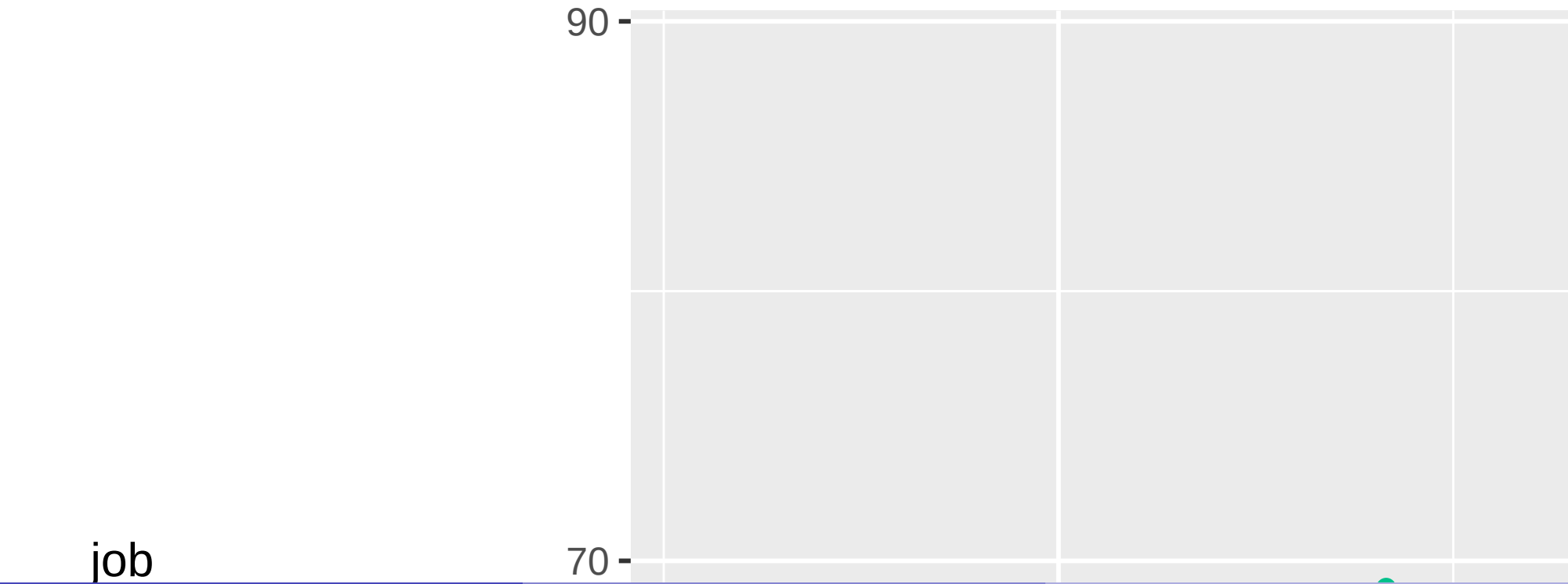
Adjustment: coordinate flip

```
# Flip the x and y axis
# Different feeling?
ggplot(bank, aes(x = age, y = duration, color = job)) +
geom_point() +
theme(legend.position="left") +
coord_flip()
```



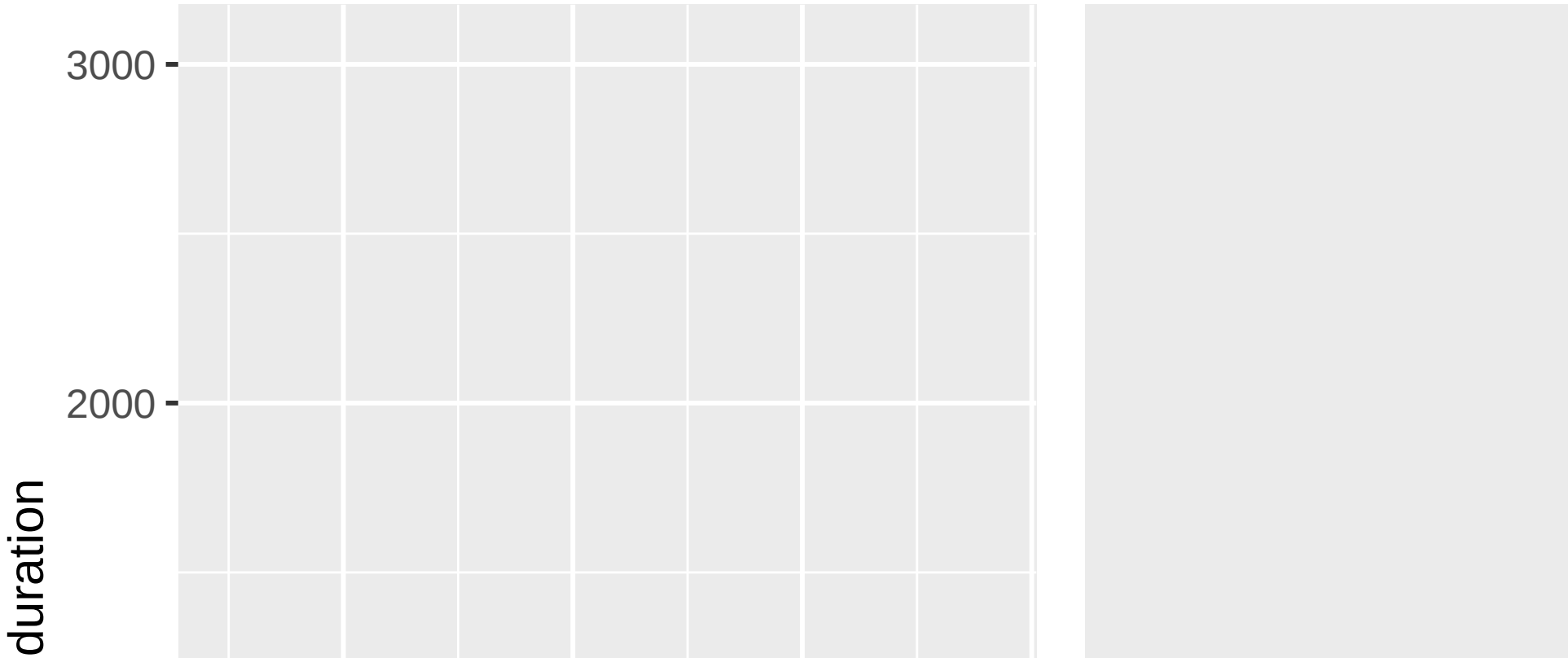
Adjustment: log scale

```
# Make y as log scaled.
# Note that before flip, x is y, so we use scale_y_log10()
ggplot(bank, aes(x = age, y = duration, color = job)) +
geom_point() +
theme(legend.position="left") +
coord_flip() +
scale_y_log10()
```



Each + is a layer

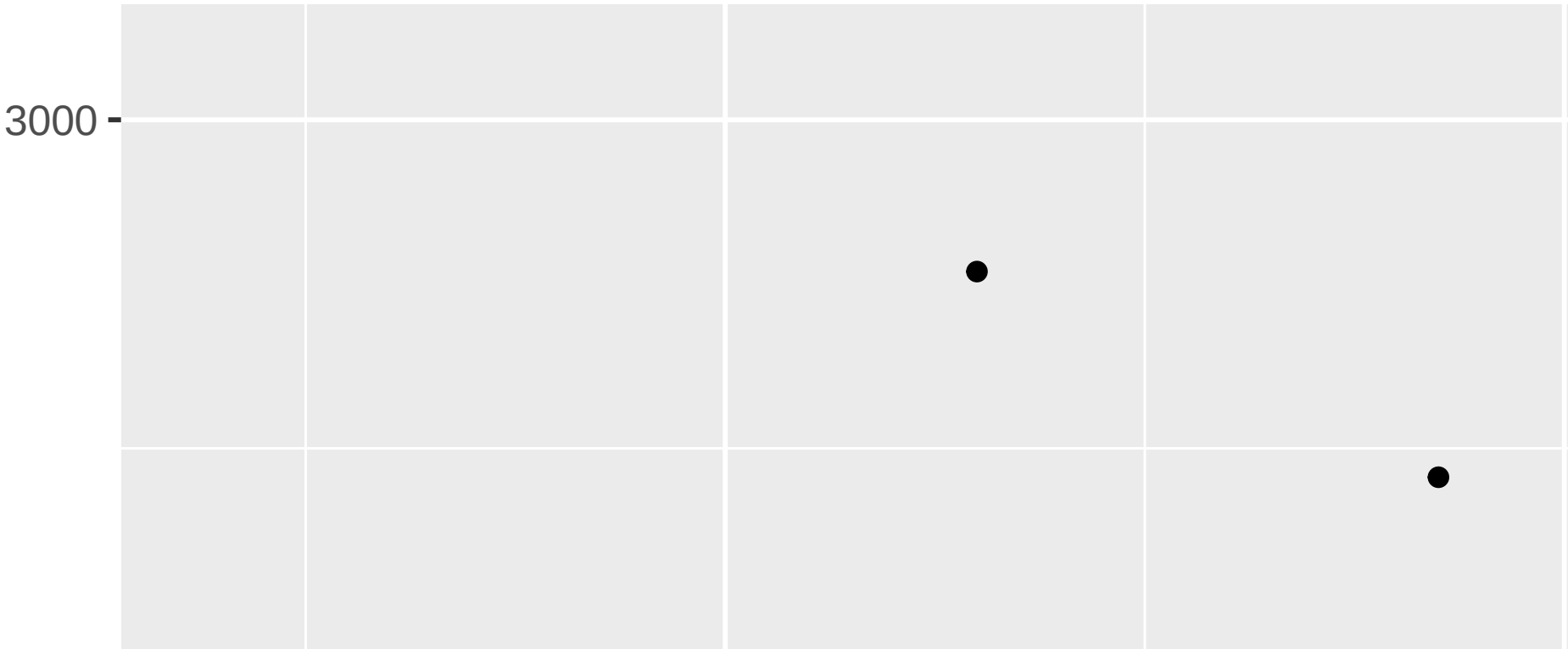
```
# Nearly empty chart.
g <- ggplot(bank, aes(x = age, y = duration))
# This is almost empty
g <- ggplot(bank)
# This is really empty.
g <- ggplot()
```



Combine g with layers

```
ggplot(bank, aes(x = age, y = duration)) +
  geom_point() + geom_smooth()
```

```
# This is equivalent to above
g <- ggplot(bank, aes(x = age, y = duration))
g + geom_point() + geom_smooth()
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs =
```



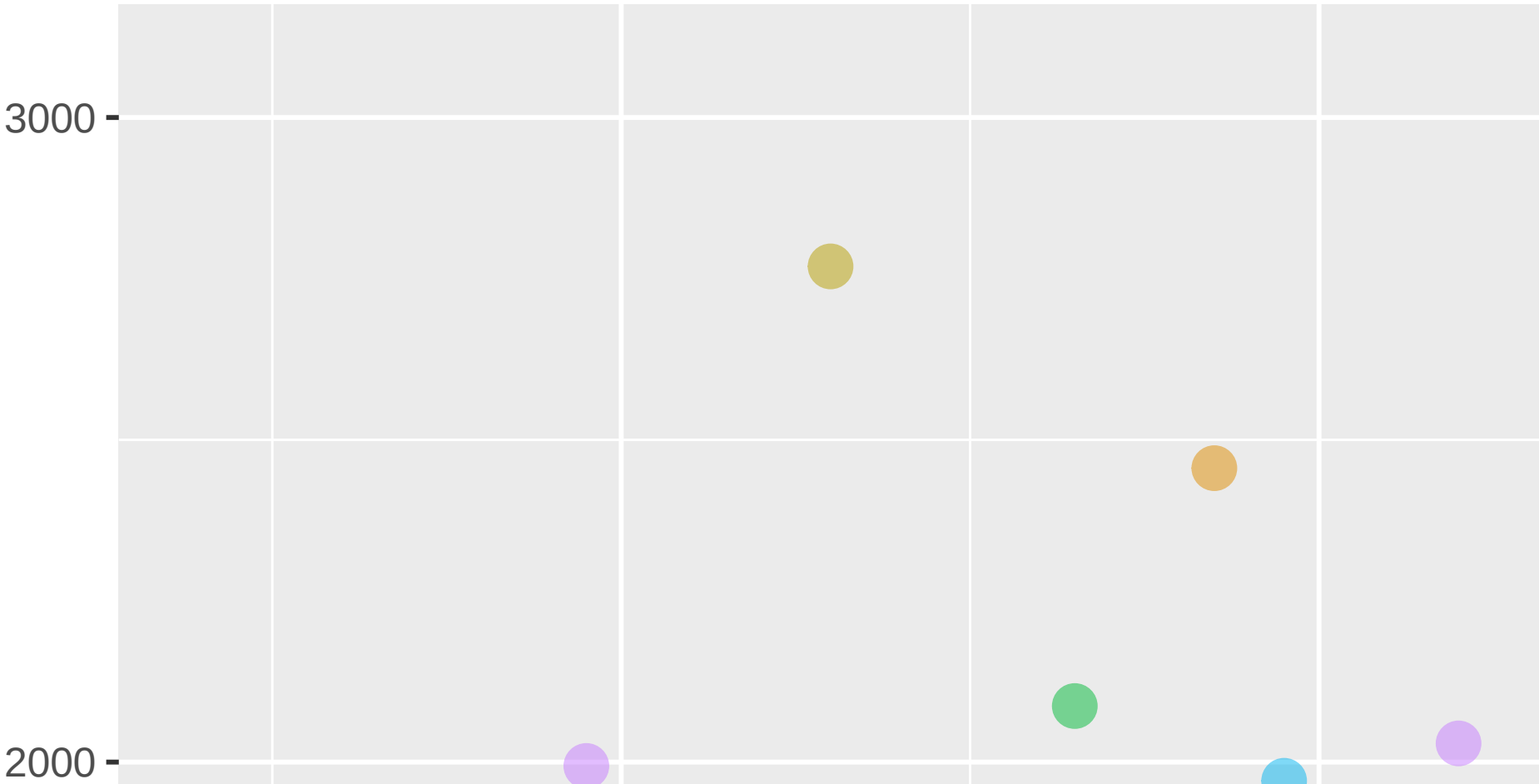
g can be re-used.

- g can be re-used. It's good to be used when we want to exploratory data and try to plot many figures.
- Fixed a few variables in `g <- ggplot(data, aes(...))`.
- Use `g + geom_XXX()` to find the best representation for the relationship.

```
g + geom_point() + geom_smooth(method = "lm") + facet_grid(. ~
g + geom_point(color = "steelblue", size = 4, alpha = 1/2)
g + geom_point(aes(color = job), size = 4, alpha = 1/2)
g + geom_point() + geom_point(aes(color = job), size = 4, alph
```

g: mix and match: which figure?

```
g + geom_point() + geom_smooth(method = "lm") + facet_grid(. ~ job)
g + geom_point(color = "steelblue", size = 4, alpha = 1/2)
g + geom_point(aes(color = job), size = 4, alpha = 1/2)
g + geom_point() + geom_point(aes(color = job), size = 4, alpha = 1/
```

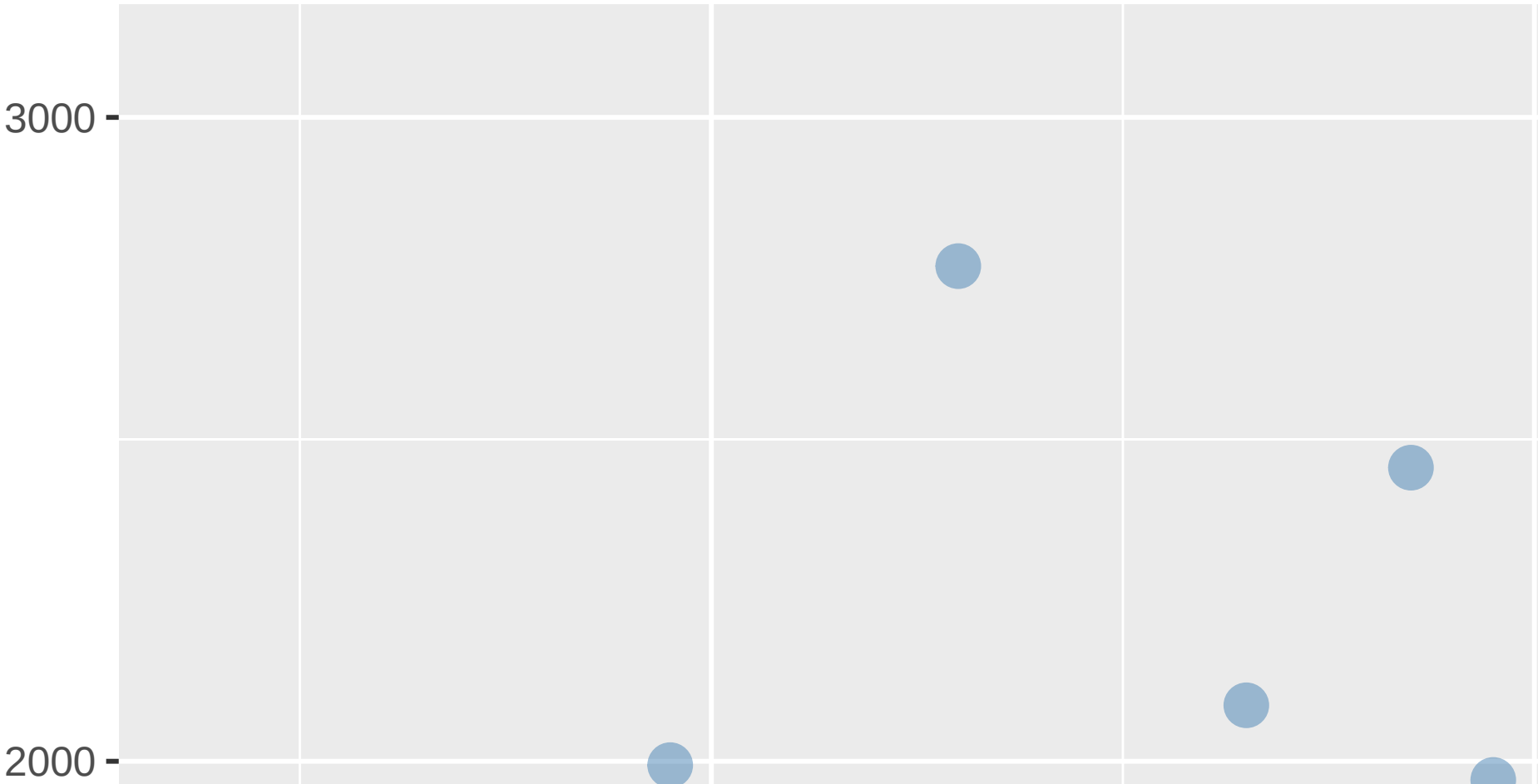


g: mix and match: which figure?

```
g + geom_point() + geom_smooth(method = "lm") + facet_grid(. ~ job)
g + geom_point(color = "steelblue", size = 4, alpha = 1/2)
g + geom_point(aes(color = job), size = 4, alpha = 1/2)
g + geom_point() + geom_point(aes(color = job), size = 4, alpha = 1/
```

g: mix and match: which figure?

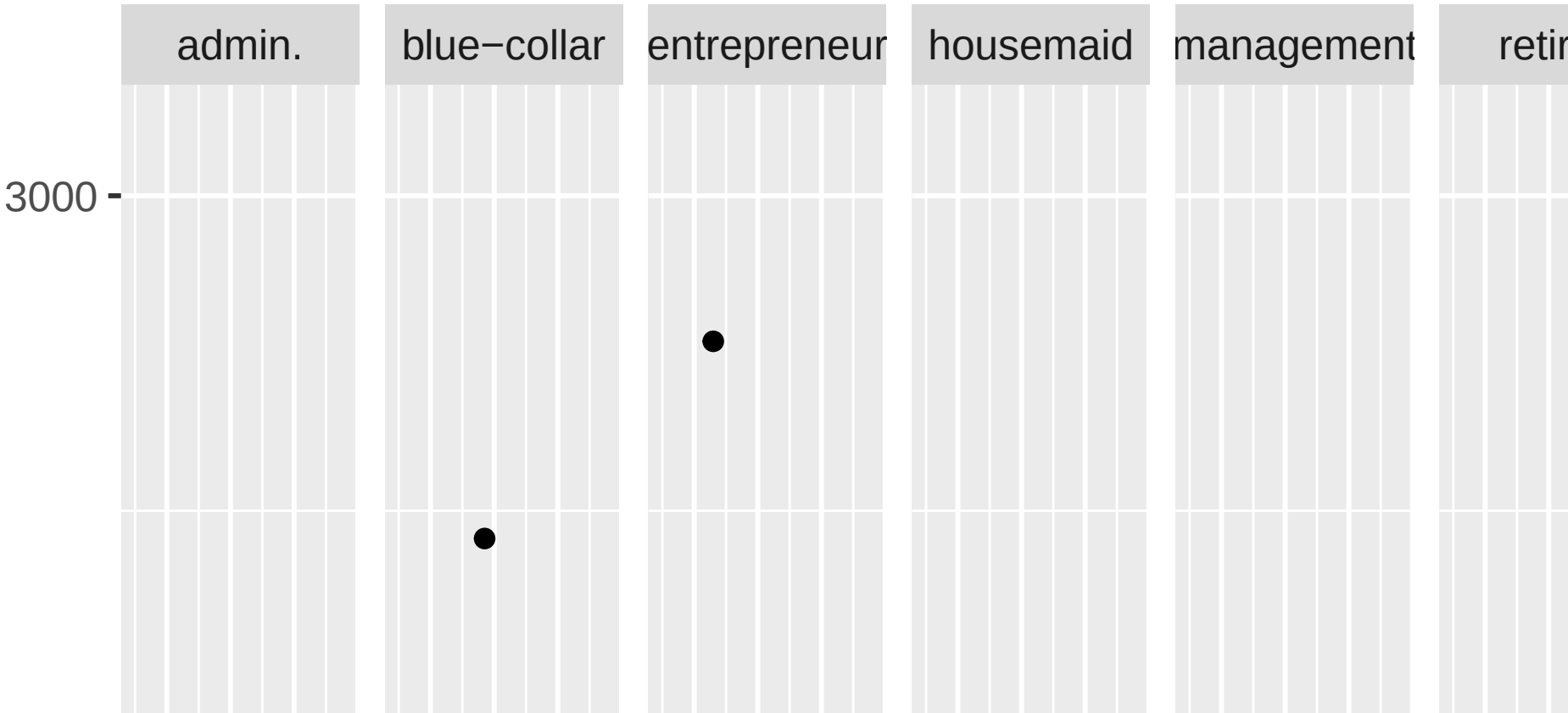
```
g + geom_point() + geom_smooth(method = "lm") + facet_grid(. ~ job)
g + geom_point(color = "steelblue", size = 4, alpha = 1/2)
g + geom_point(aes(color = job), size = 4, alpha = 1/2)
g + geom_point() + geom_point(aes(color = job), size = 4, alpha = 1/
```



g: mix and match: which figure?

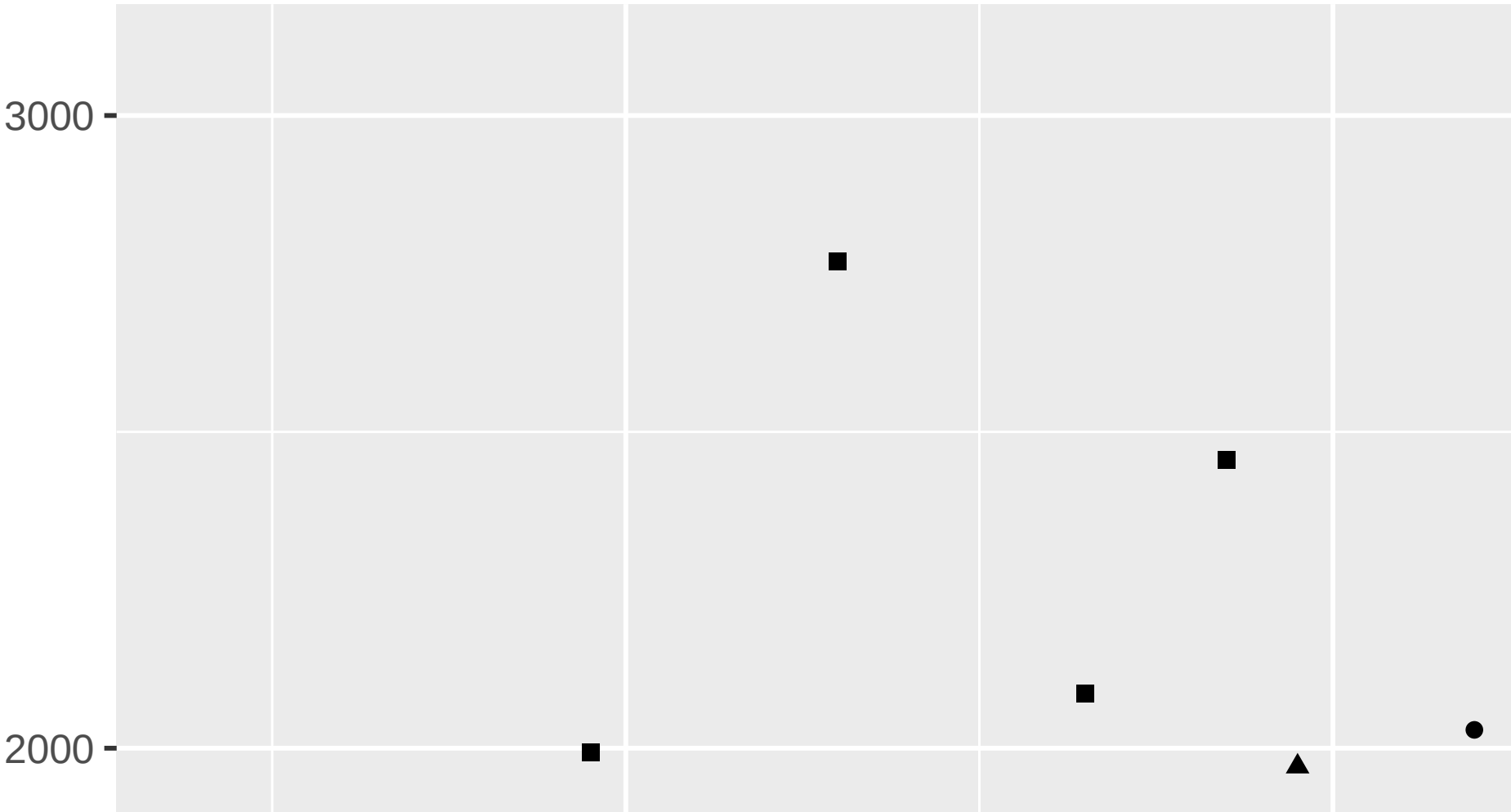
```
g + geom_point() + geom_smooth(method = "lm") + facet_grid(. ~ job)
g + geom_point(color = "steelblue", size = 4, alpha = 1/2)
g + geom_point(aes(color = job), size = 4, alpha = 1/2)
g + geom_point() + geom_point(aes(color = job), size = 4, alpha = 1/
```

```
## `geom_smooth()` using formula 'y ~ x'
```



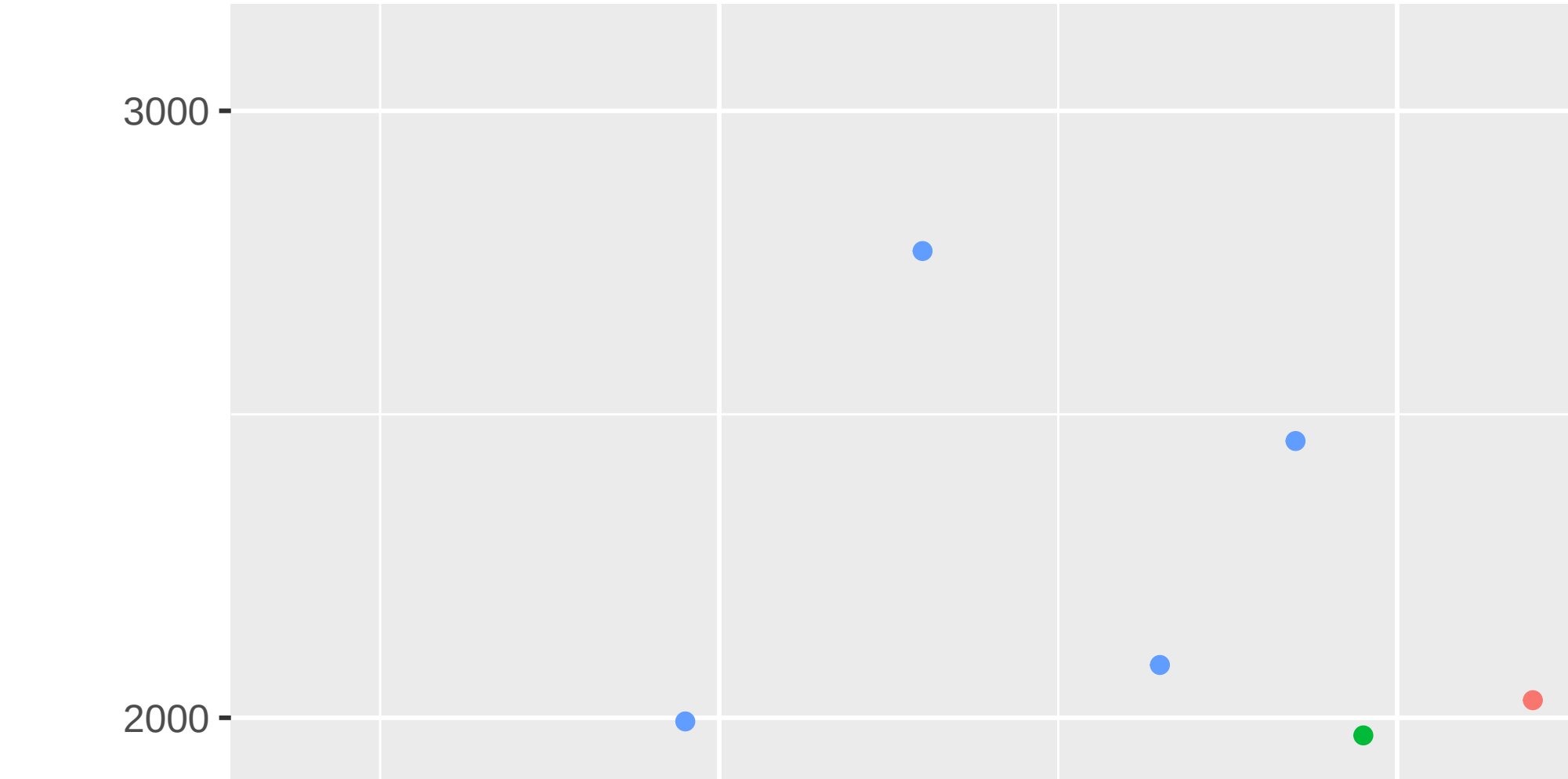
Differentiate groups - shape

```
# Use shape
ggplot(bank) +
  geom_point(aes(age, duration, shape = contact))
```



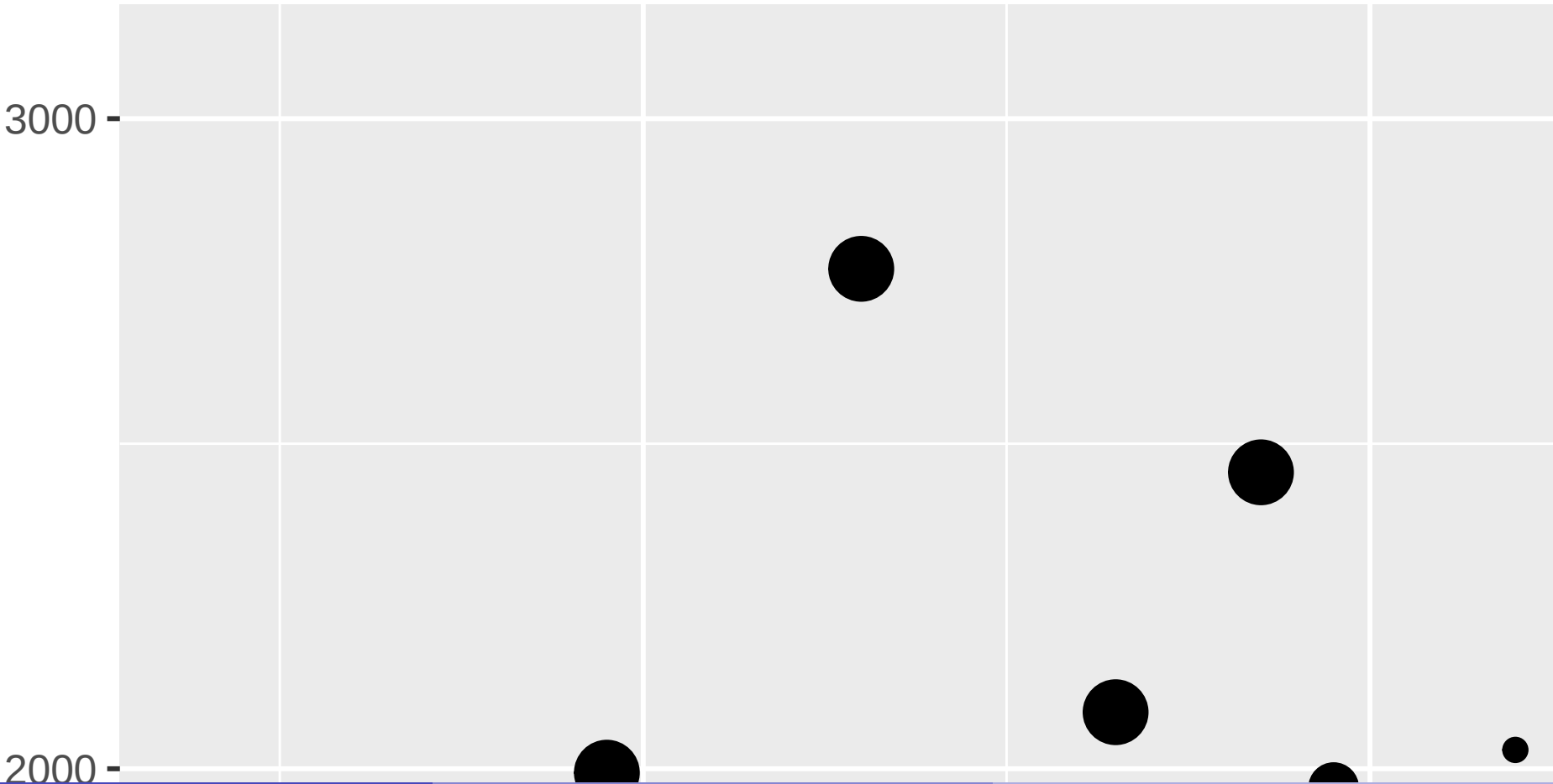
Differentiate groups - color

```
# Use color
ggplot(bank) +
  geom_point(aes(age, duration, color = contact))
```



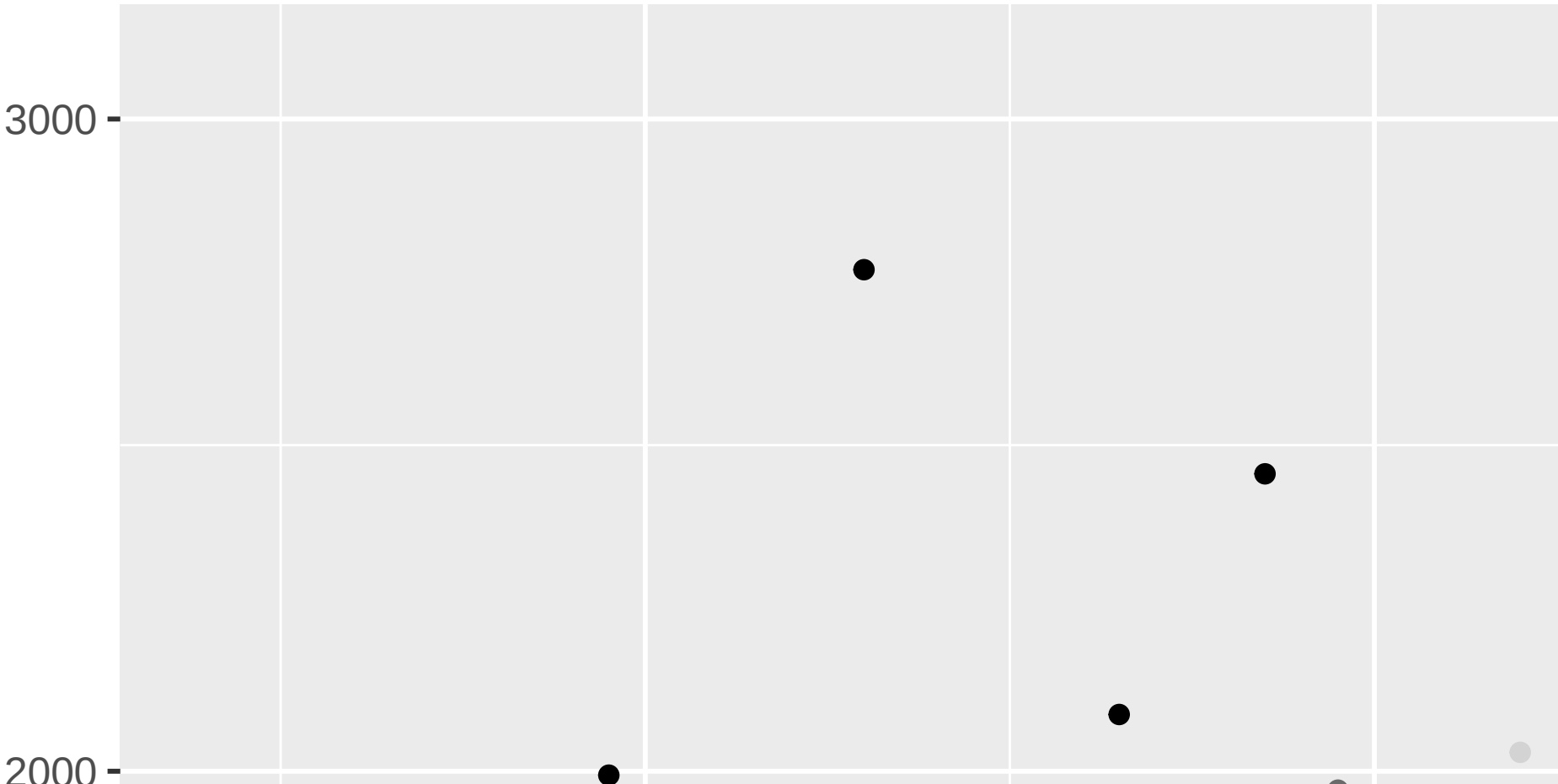
Differentiate groups - `size`

```
# Use size
ggplot(bank) +
  geom_point(aes(age, duration, size = contact))
## Warning: Using size for a discrete variable is not advised.
```



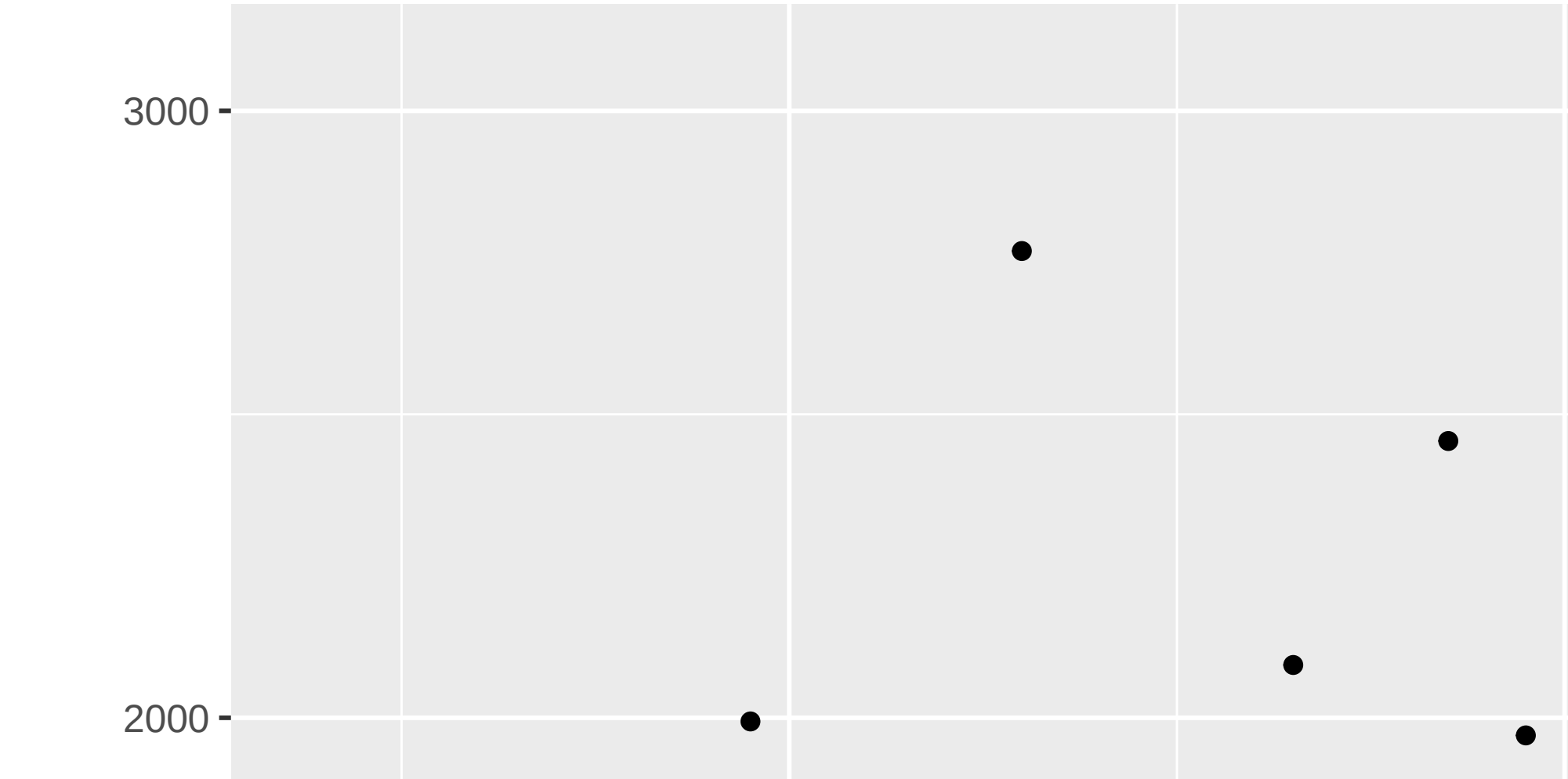
Differentiate groups - alpha

```
# Use alpha - transparency
ggplot(bank) +
  geom_point(aes(age, duration, alpha = contact))
## Warning: Using alpha for a discrete variable is not advised.
```



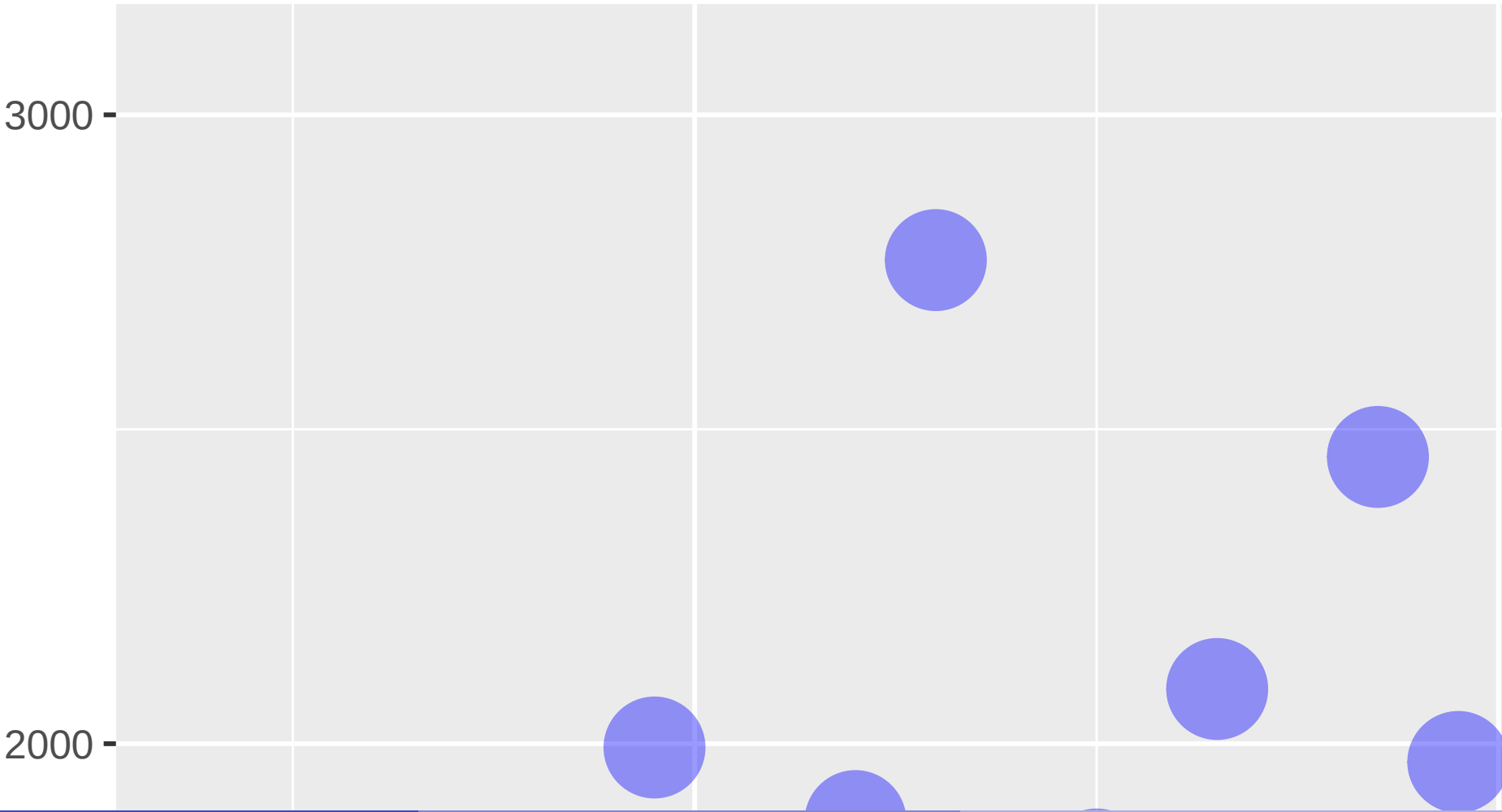
Differentiate groups - group

```
# Use group.
ggplot(bank) +
  geom_point(aes(age, duration, group = contact))
```



Enforce color, put things outside aes

```
## you can also enforce color, put things outside aes
ggplot(bank) +
  geom_point(aes(age, duration), color = "blue", size = 10, alpha =
```



What's inside bank's clients? Things to consider

- Which variables in data are categorical?
- Which variables are continuous?
- bio:
  - age
  - job
  - marital
  - education
- financial
  - default
  - balance
  - housing
  - loan

What's inside bank's clients? Things to consider 2

- communication
  - contact: celluar v.s. telephone v.s. unknown
  - day/month: maybe good to ignore?
  - duration:
  - campaign:
  - pdays:
  - previous:
  - poutcome:

Categorical/Continuous variable

- numeric -> continuous data
- string -> discrete data

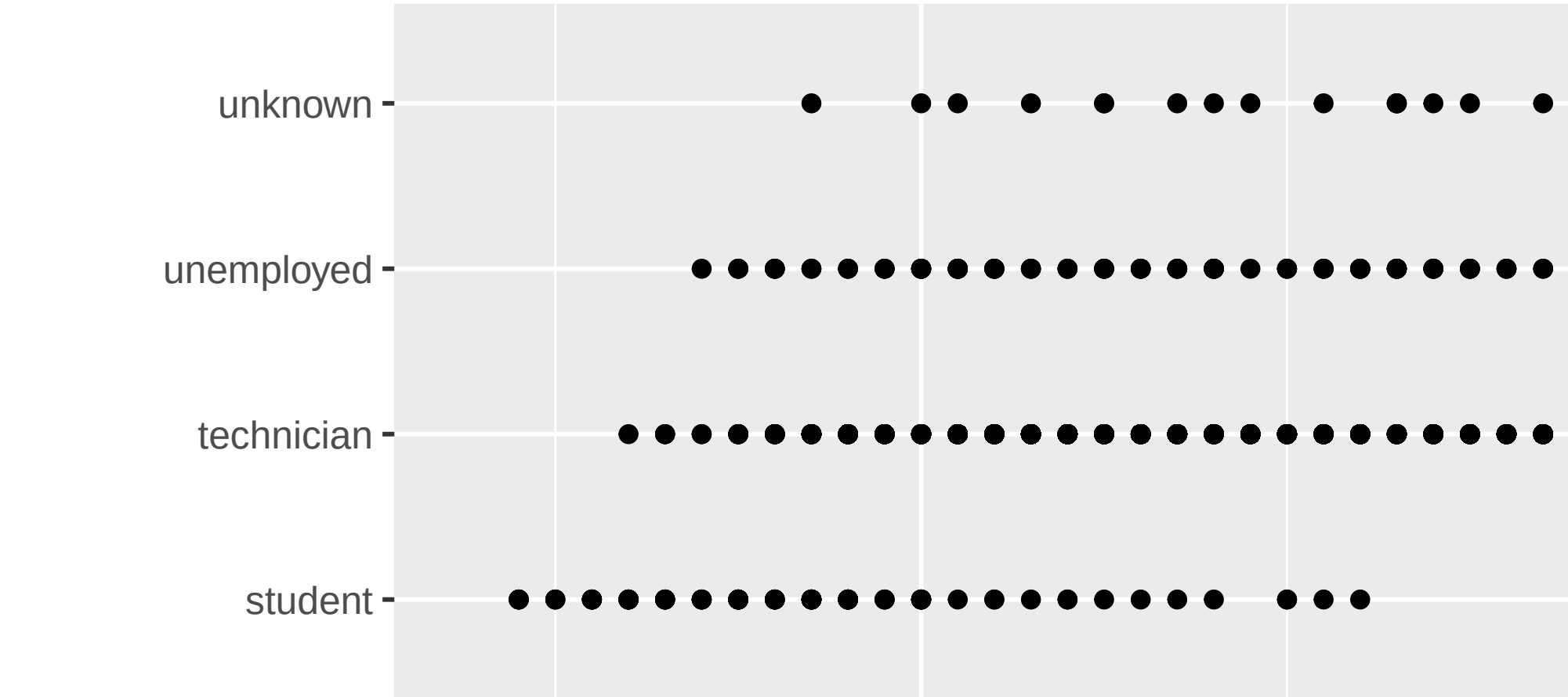
For different type of variables, the scale is also different

- Reverse a categorical (discrete) variable, we use `scale_y_discrete(limit = rev(levels(...)))`.
- Reverse a continuous numerical variable, we use `scale_x_reverse()`.

Exercise

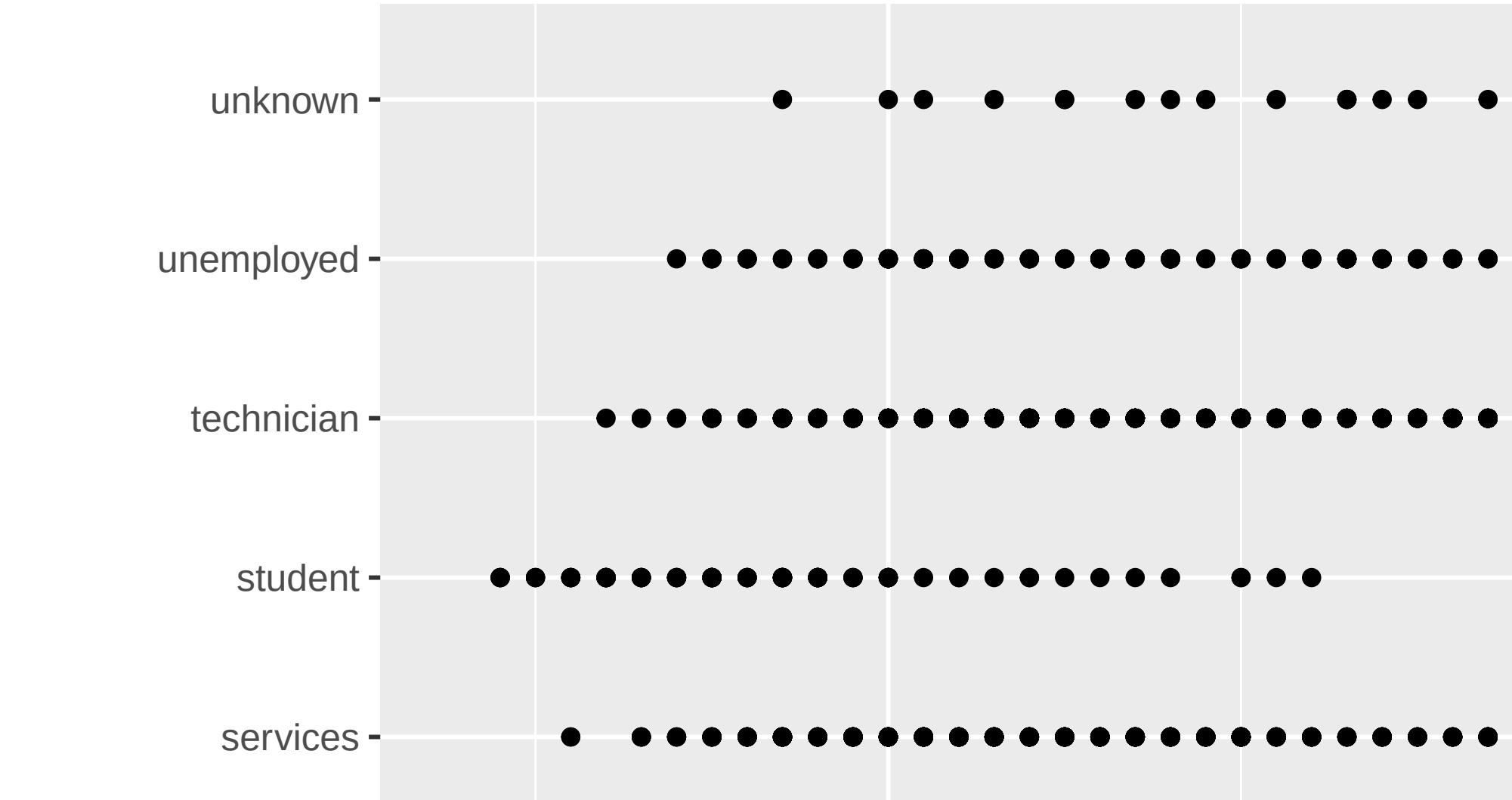
1. What does this figure mean?

```
ggplot(bank, aes(age, job)) +
geom_point() +
scale_y_discrete(limit = rev(levels(bank$job)))
```



Exercise

```
# y labels without sort.
ggplot(bank, aes(age, job)) + geom_point()
```



Exercise

2. I tried to plot between job, loan and age. Any better idea?

```
suppressWarnings(
  ggplot(bank, aes(age, job, color = loan)) + geom_point()
)
```

Other geoms

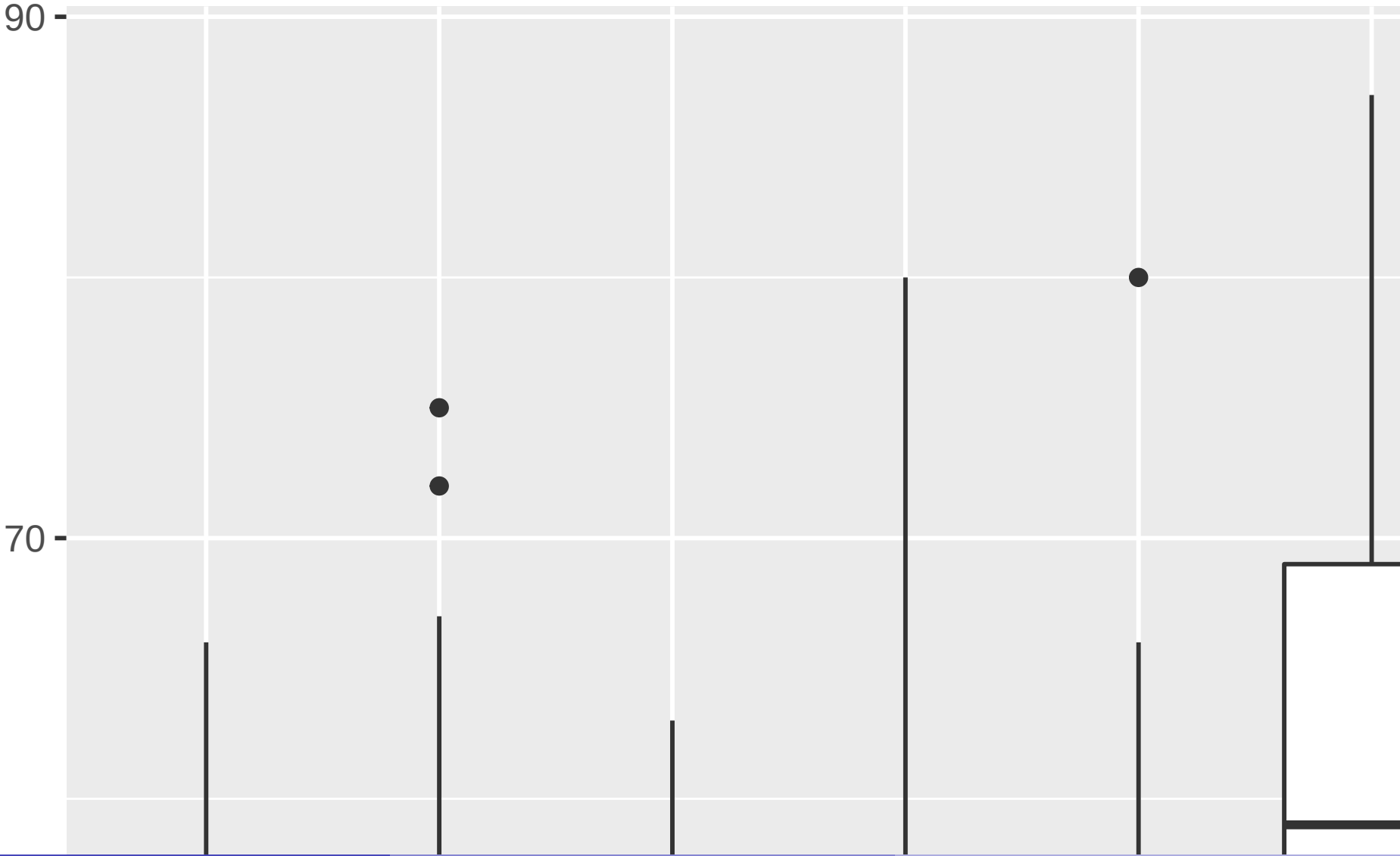
- geom_boxplot()
- geom_density()
- geom_histogram()
- geom_bar()

Boxplot: job and duration

```
ggplot(bank, aes(job, duration)) + geom_boxplot()
```

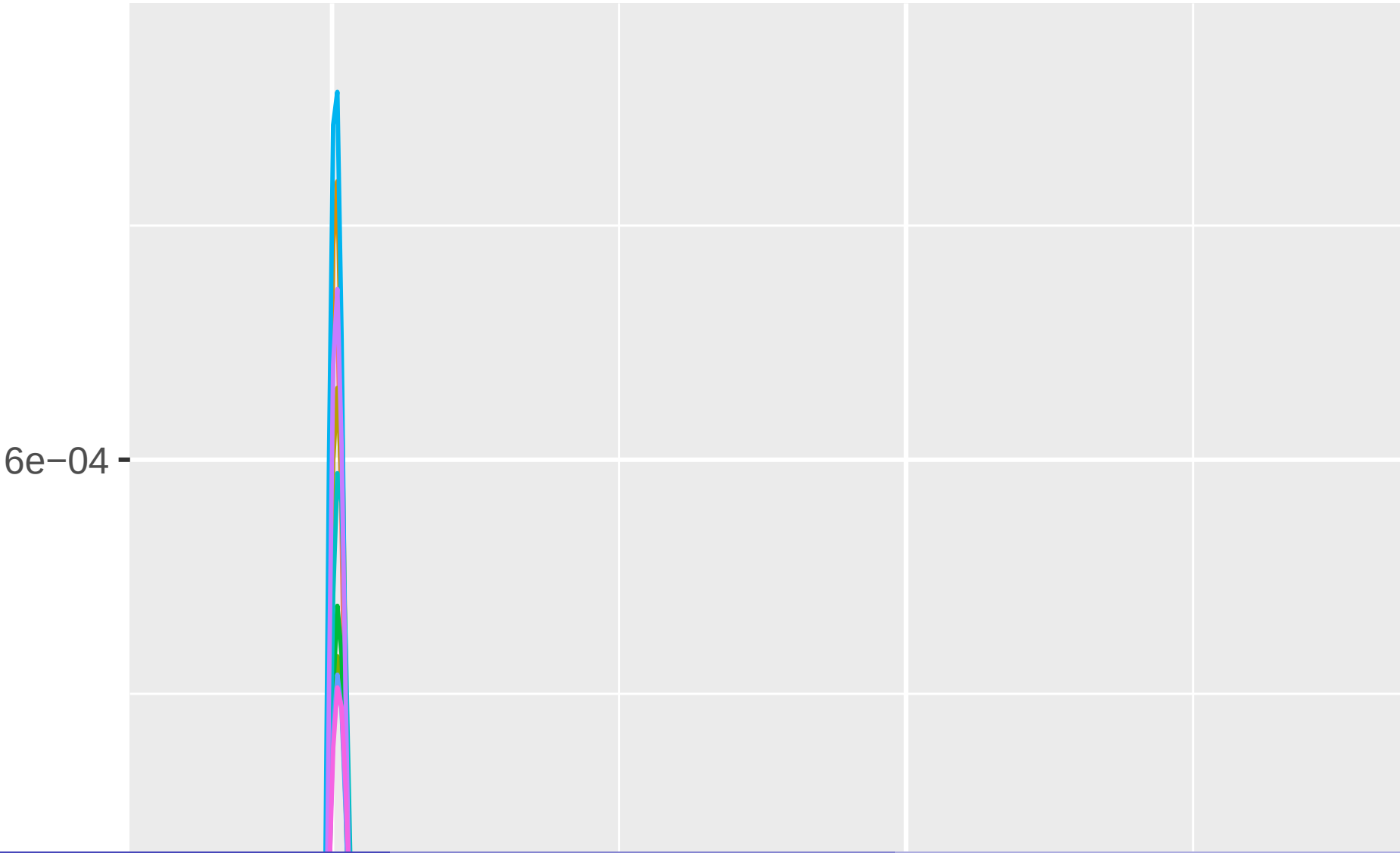
Boxplot: job and age

```
ggplot(bank, aes(job, age)) + geom_boxplot()
```



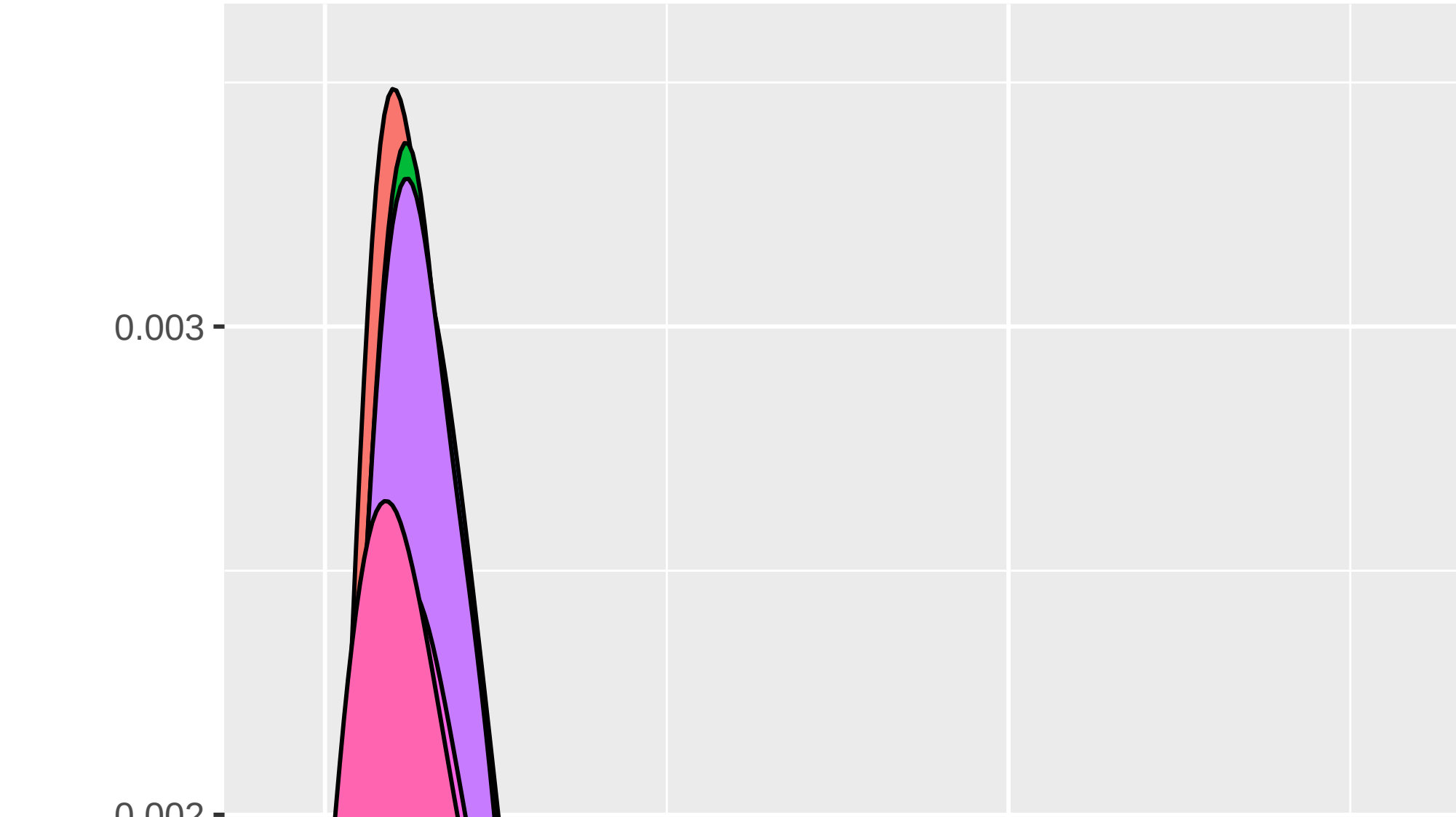
Density: balance and job

```
ggplot(bank, aes(balance, color = job)) + geom_density()
```



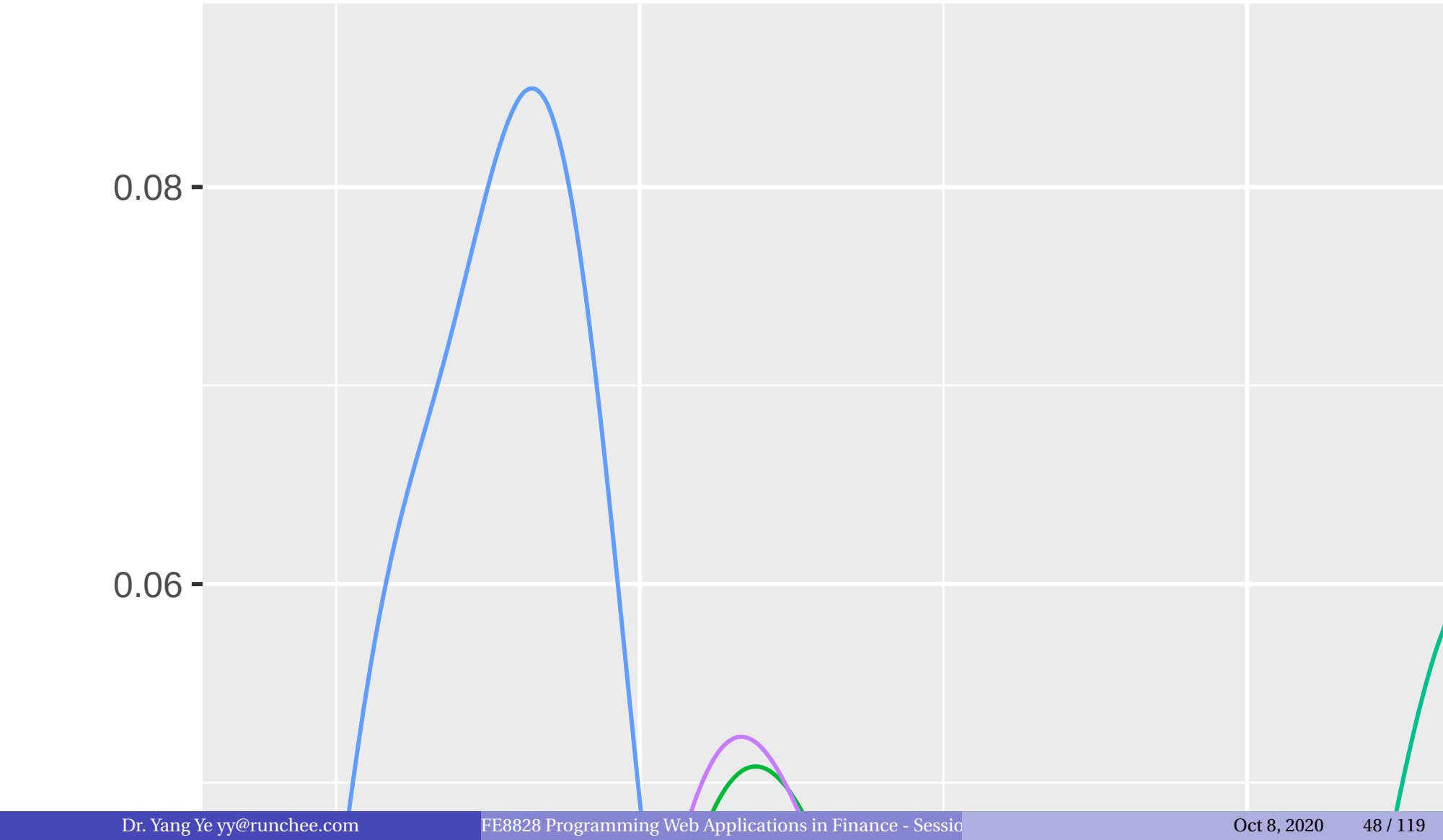
Density: balance and job with fill

```
ggplot(bank, aes(duration, fill = job)) + geom_density()
```



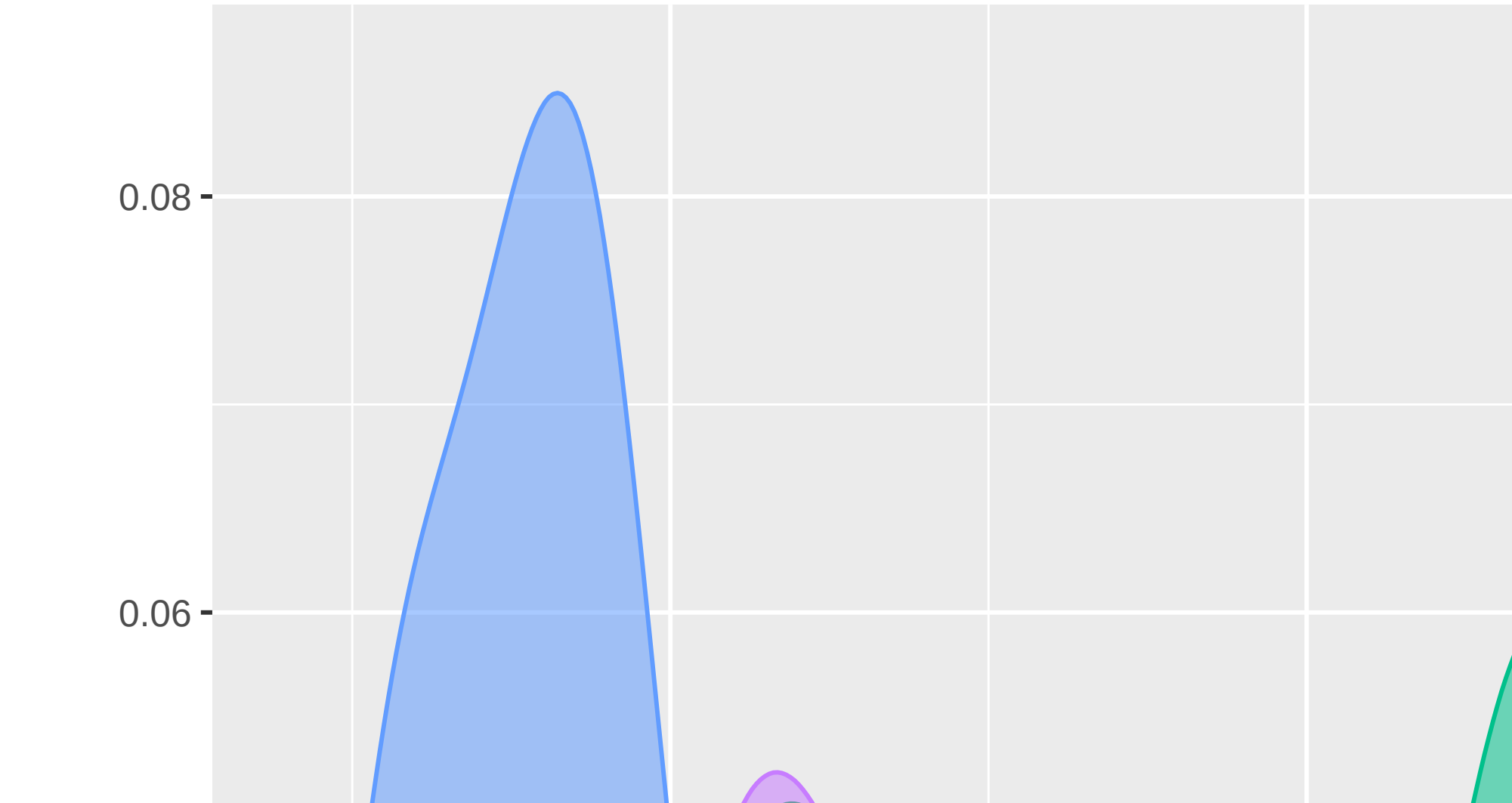
Density: age and job with alpha

```
ggplot(bank, aes(age, color = job, alpha = 0.3)) + geom_density()
```



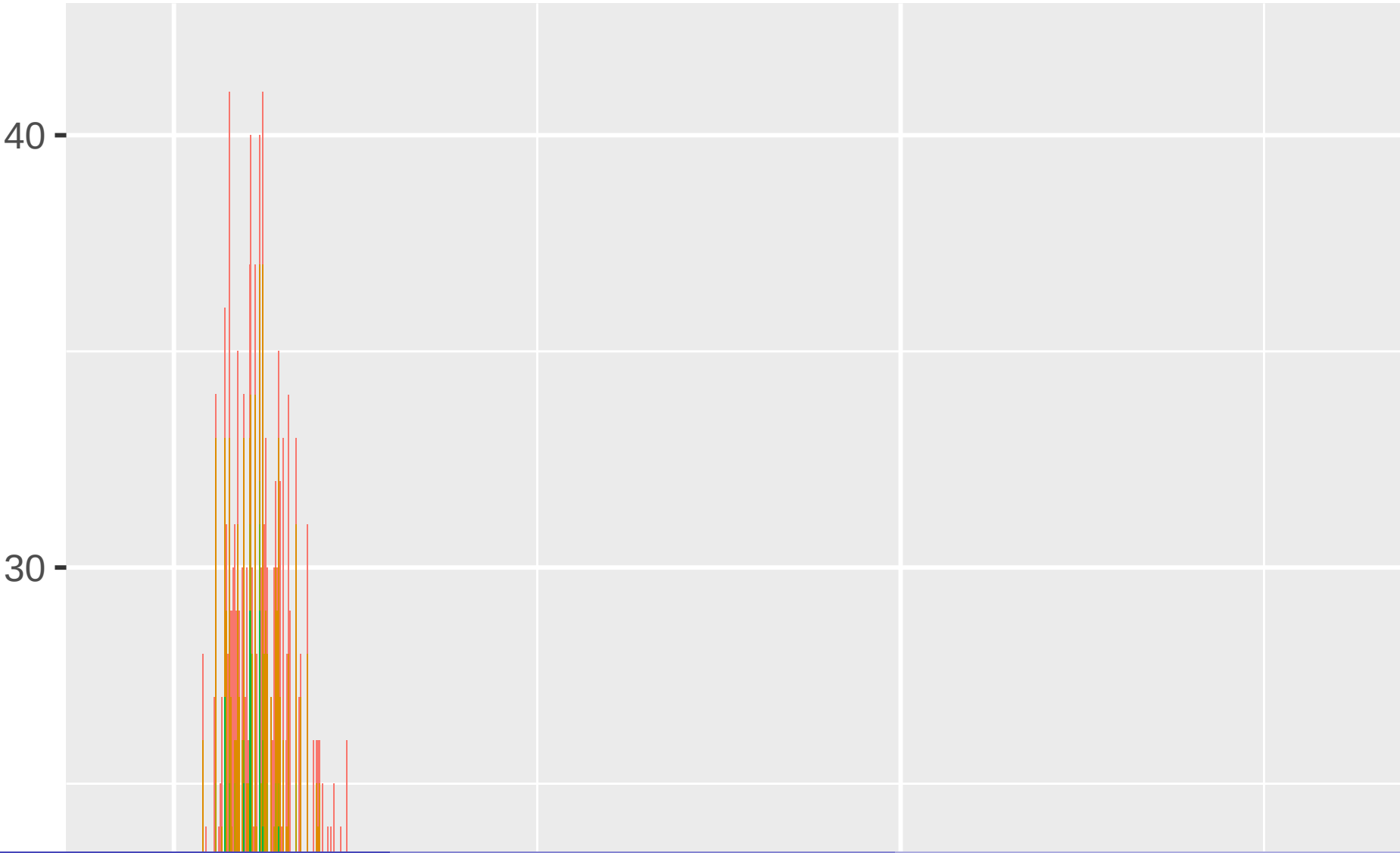
Density: age and job with alpha and fill and color

```
# Which is better?
ggplot(bank, aes(age, color = job, fill = job, alpha = 0.3)) + geom_
```



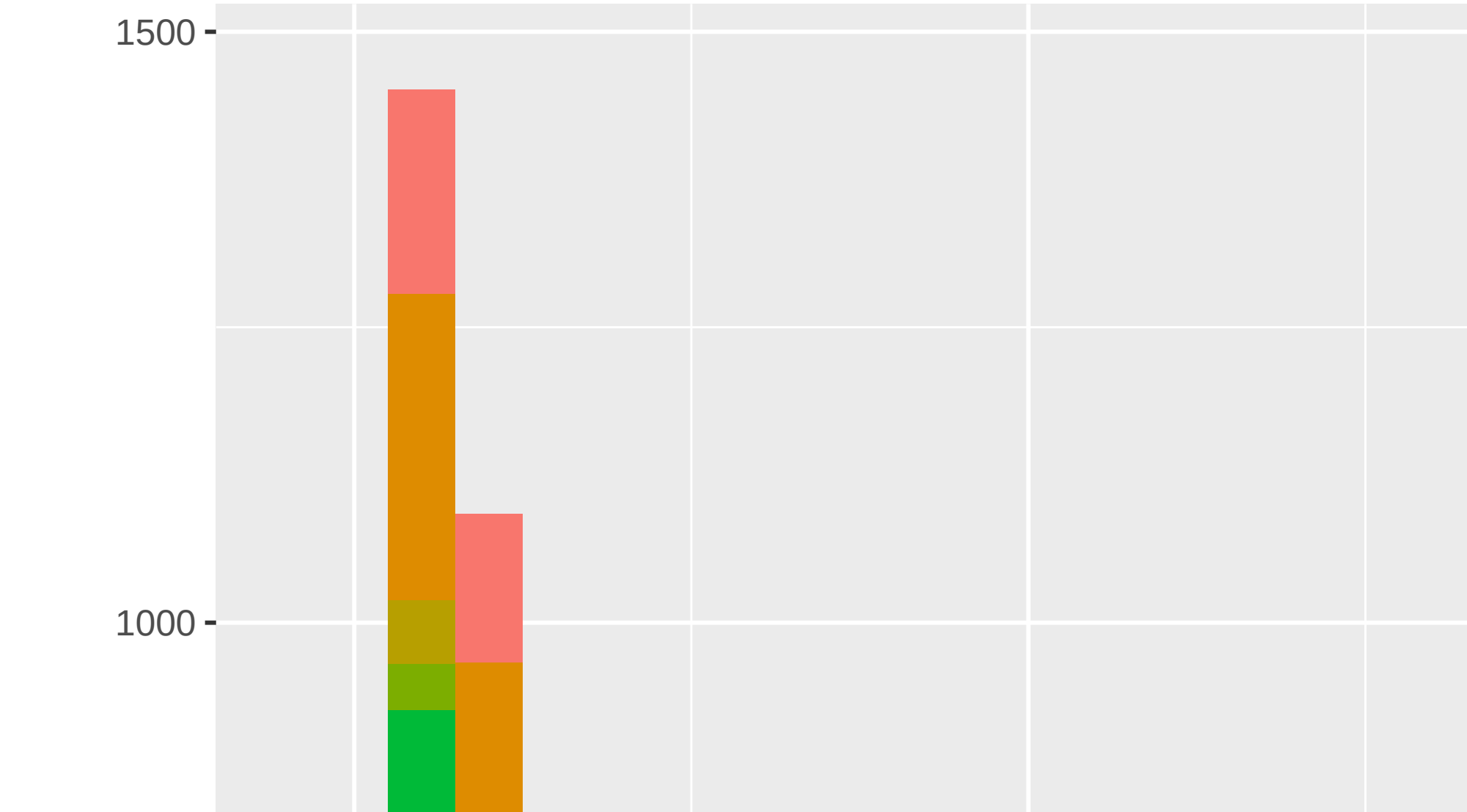
histogram: duration and job and `binwidth = 2`

```
ggplot(data = bank, mapping = aes(x = duration, fill = job)) + geom_
```



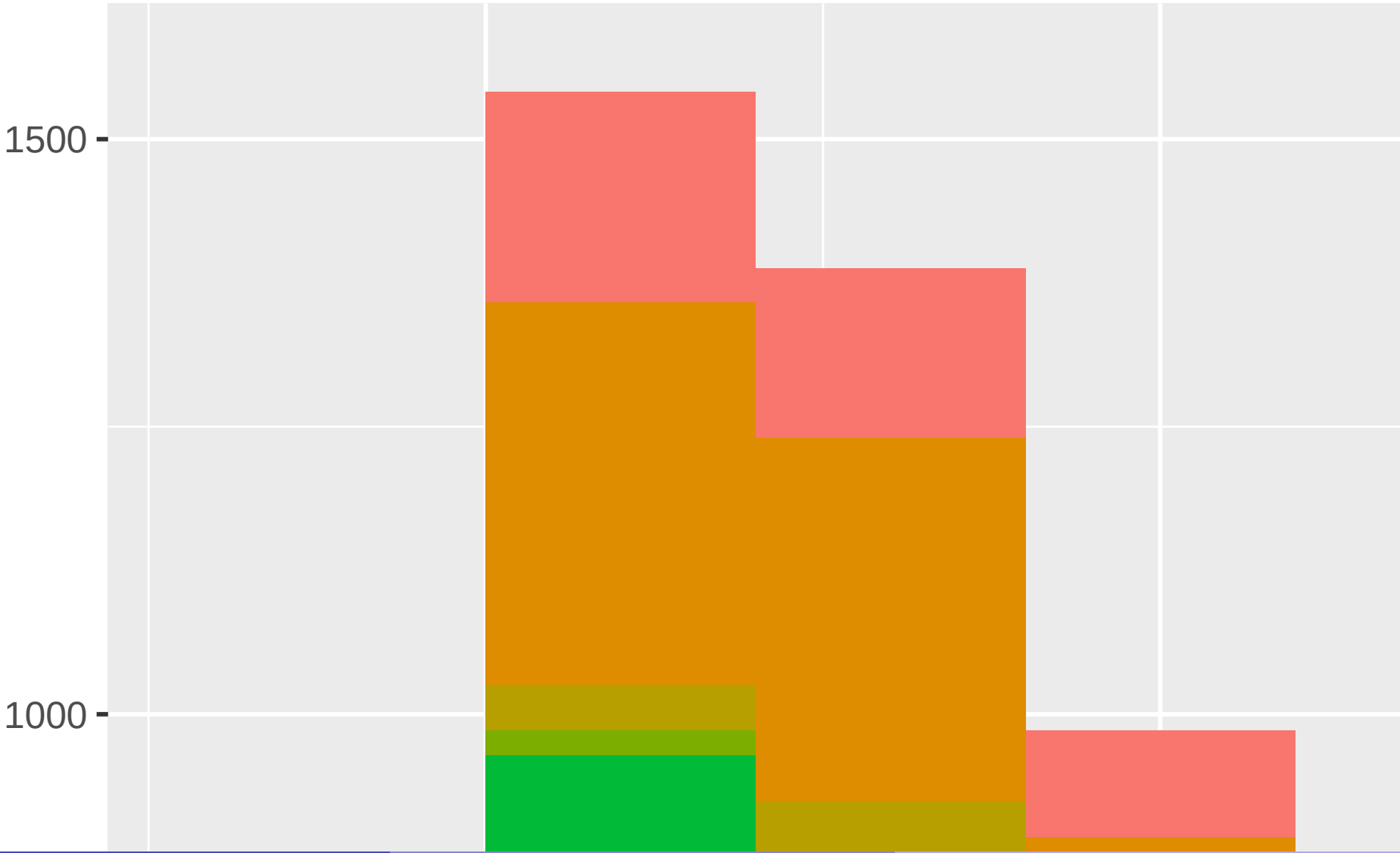
histogram: duration and job and binwidth = 100

```
ggplot(data = bank, mapping = aes(x = duration, fill = job)) + geom_
```



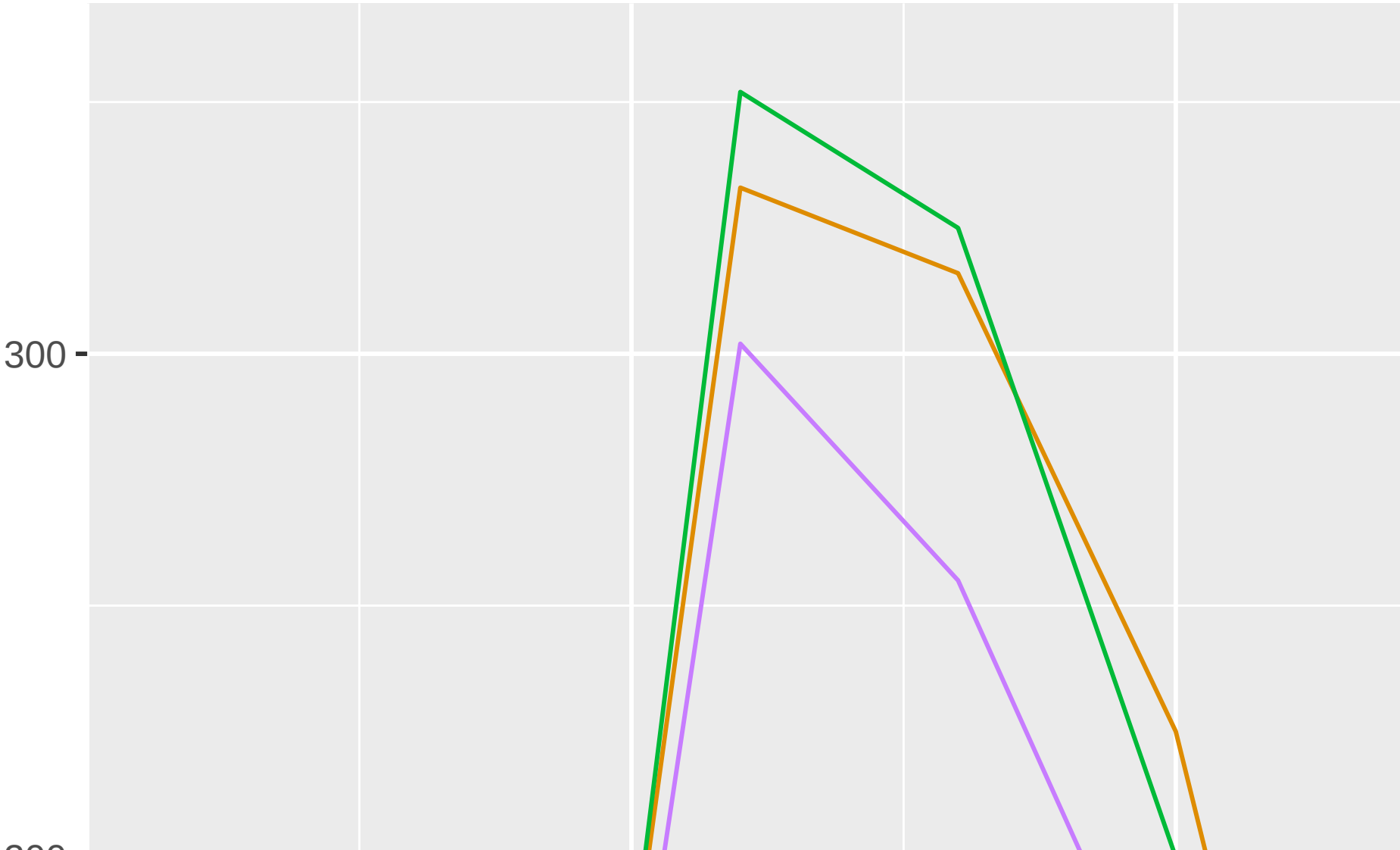
histogram: age and job and fill

```
ggplot(data = bank, mapping = aes(x = age, fill = job)) + geom_histo
```



histogram: age and job and colour

```
ggplot(data = bank, mapping = aes(x = age, colour = job)) + geom_fre
```

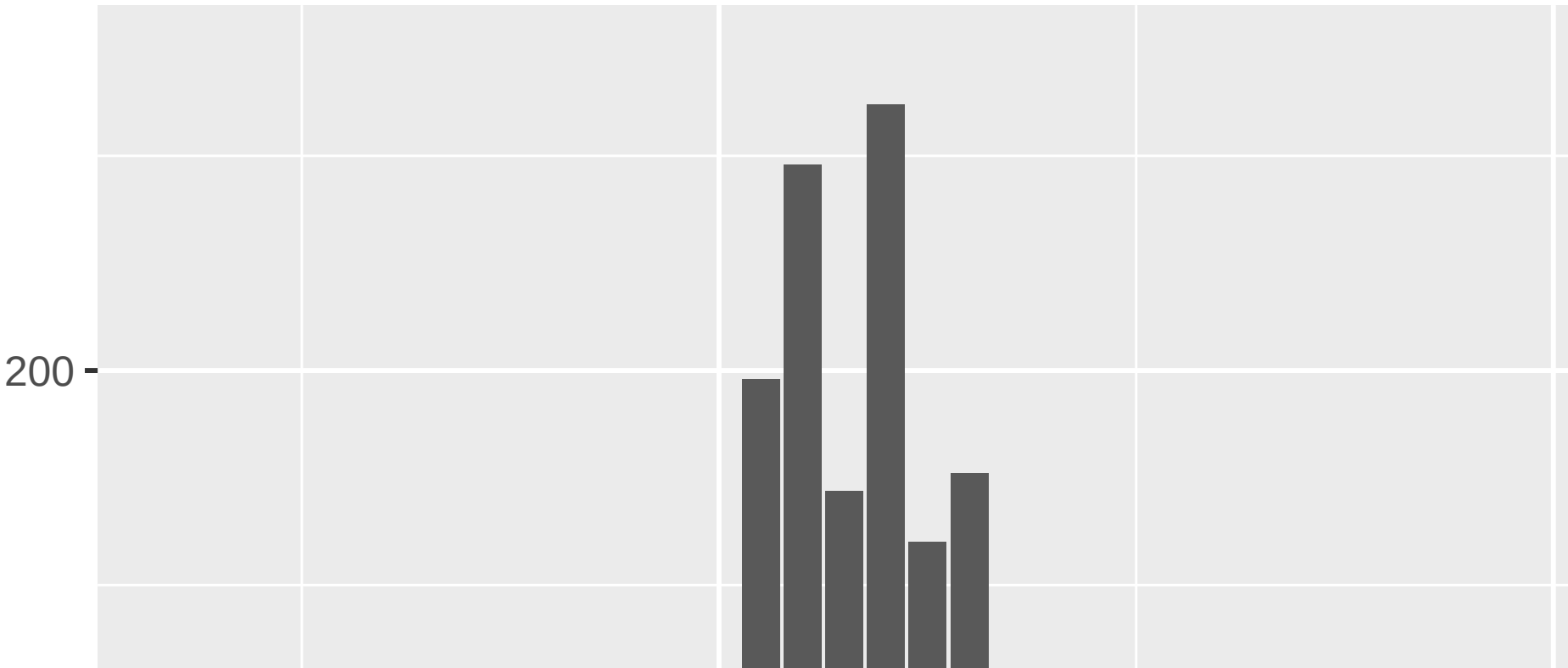


geom_bar

- bar is a statistical function: It counts.

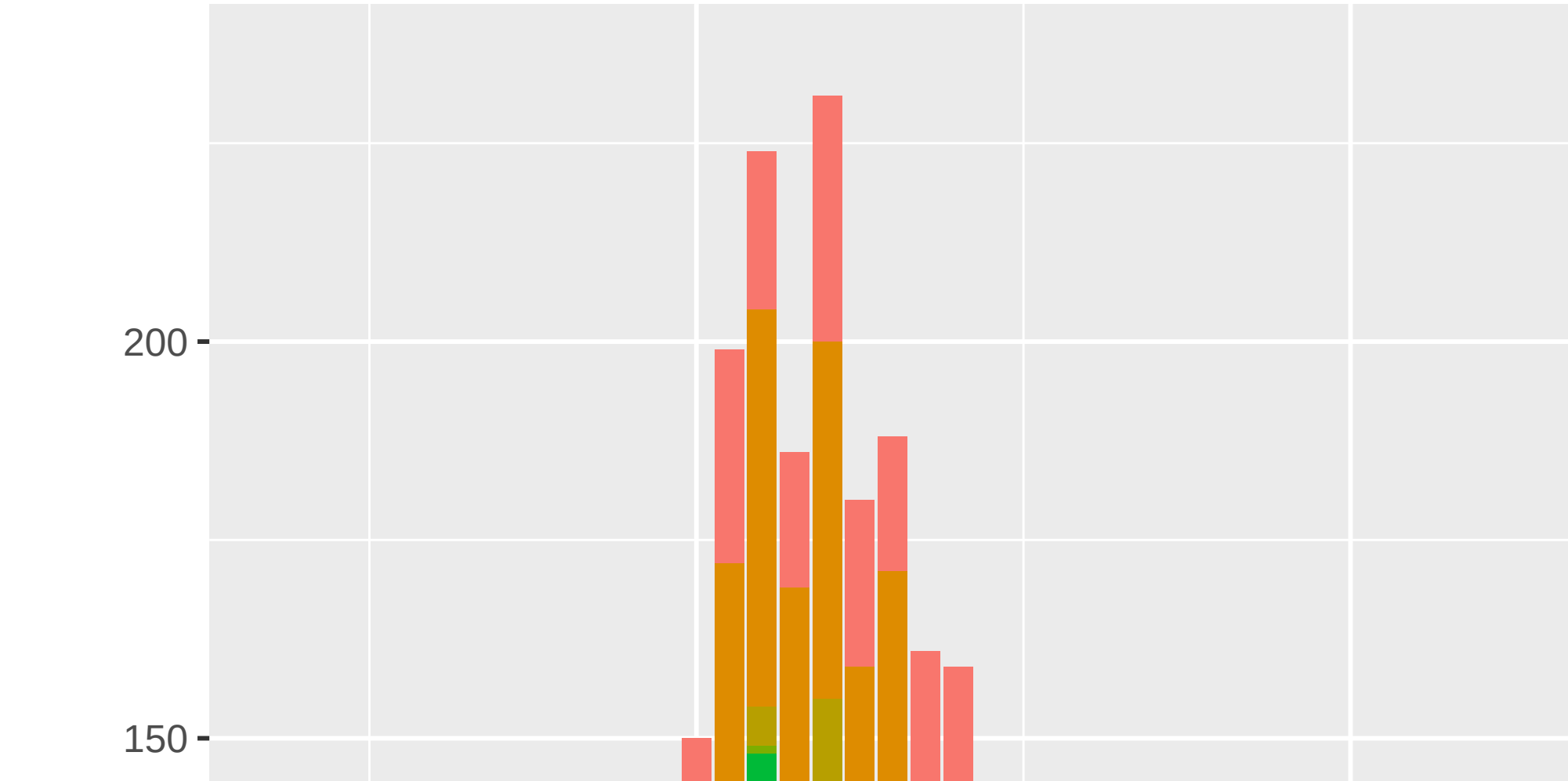
```
# First input parameter to geom_bar is mapping, so we can skip it.
ggplot(bank) + geom_bar(mapping = aes(x = age))
```

```
# We can skip mapping
ggplot(bank) + geom_bar(aes(x = age))
```



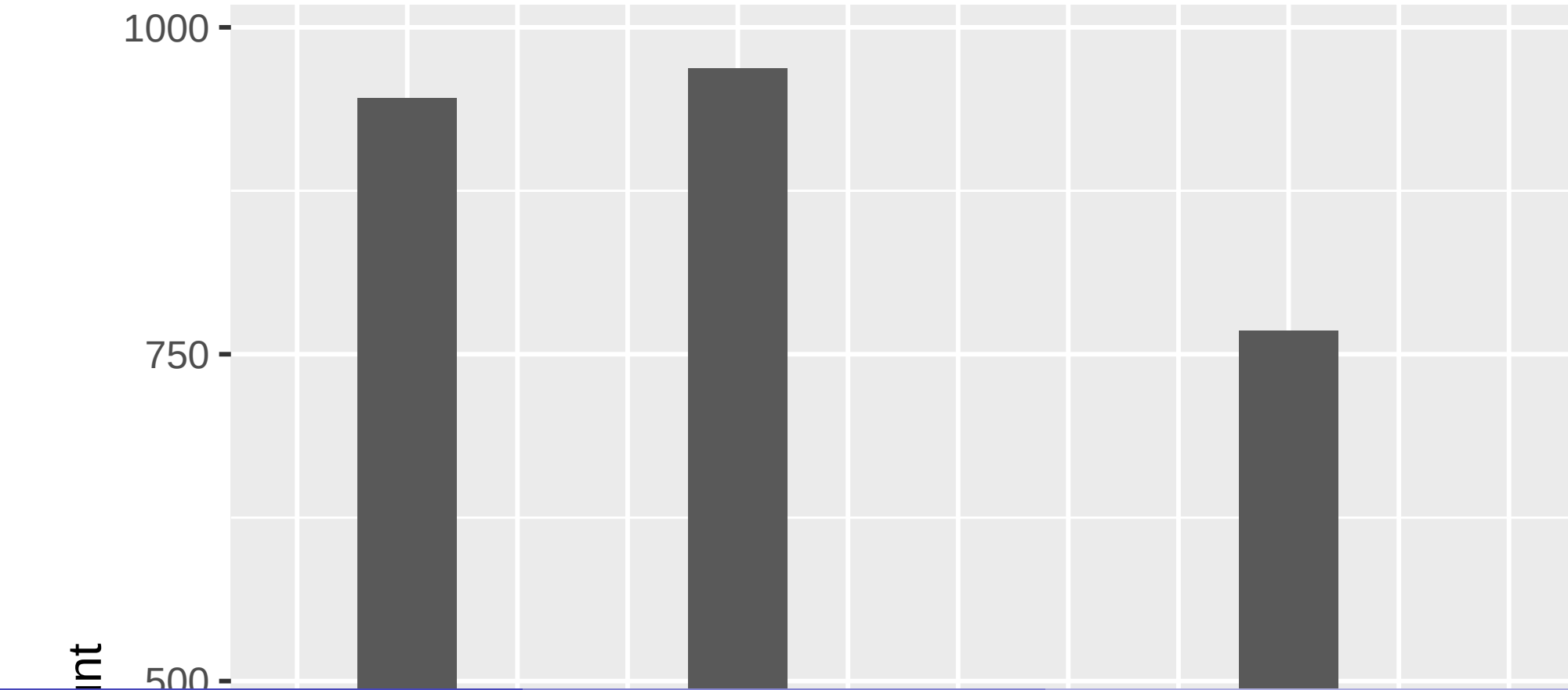
geom_bar with fill: job

```
# comparing to colour, for Bar, we better use fill
# ggplot(data = bank, ) + geom_bar(aes(x = age, colour = job))
ggplot(bank) + geom_bar(mapping = aes(x = age, fill = job))
```



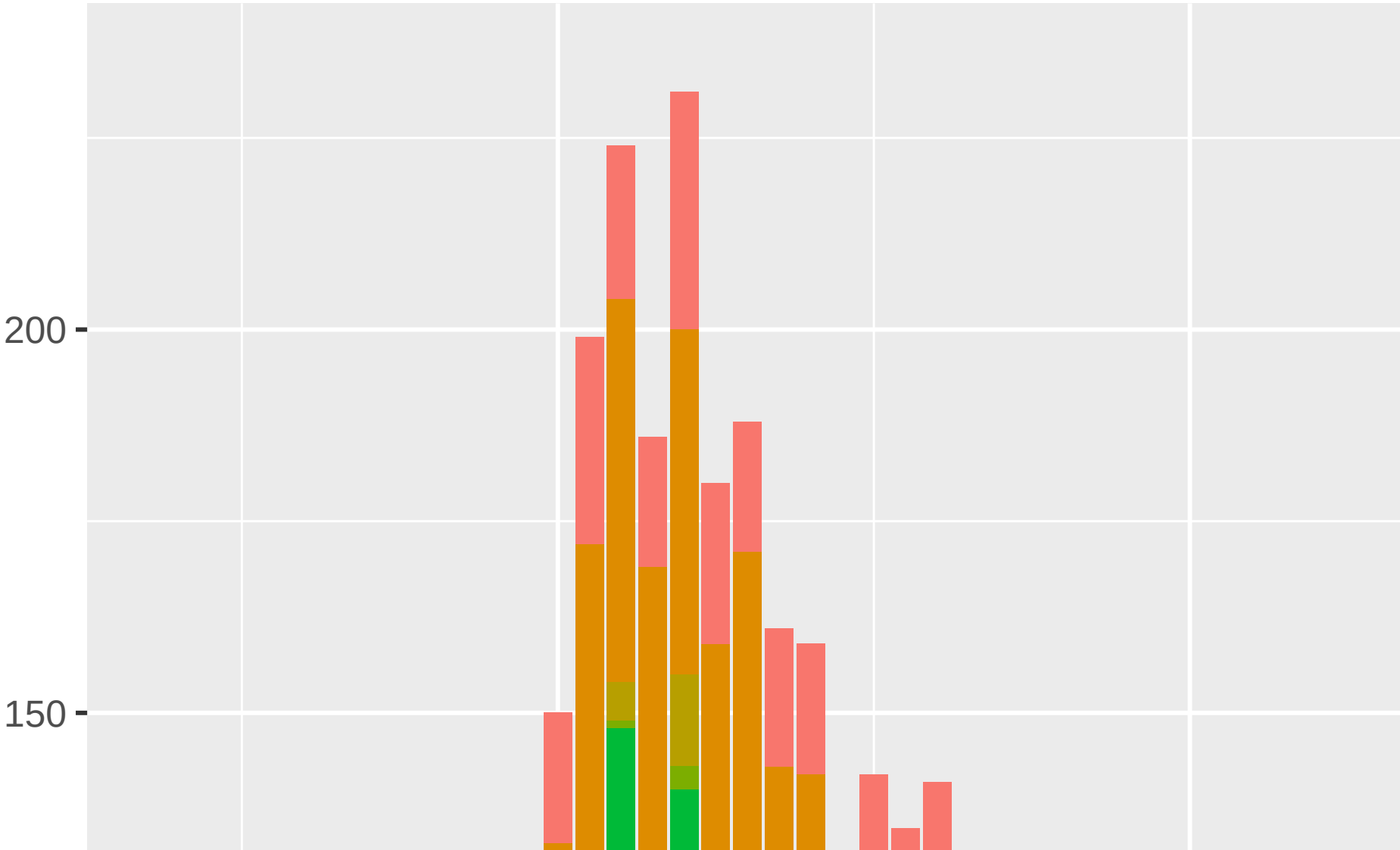
geom_bar with fill: age

```
ggplot(bank) +
  geom_bar(mapping = aes(x = job))
# Color doesn't work, because age is a continous variable.
ggplot(bank) +
  geom_bar(mapping = aes(x = job, fill = age))
```



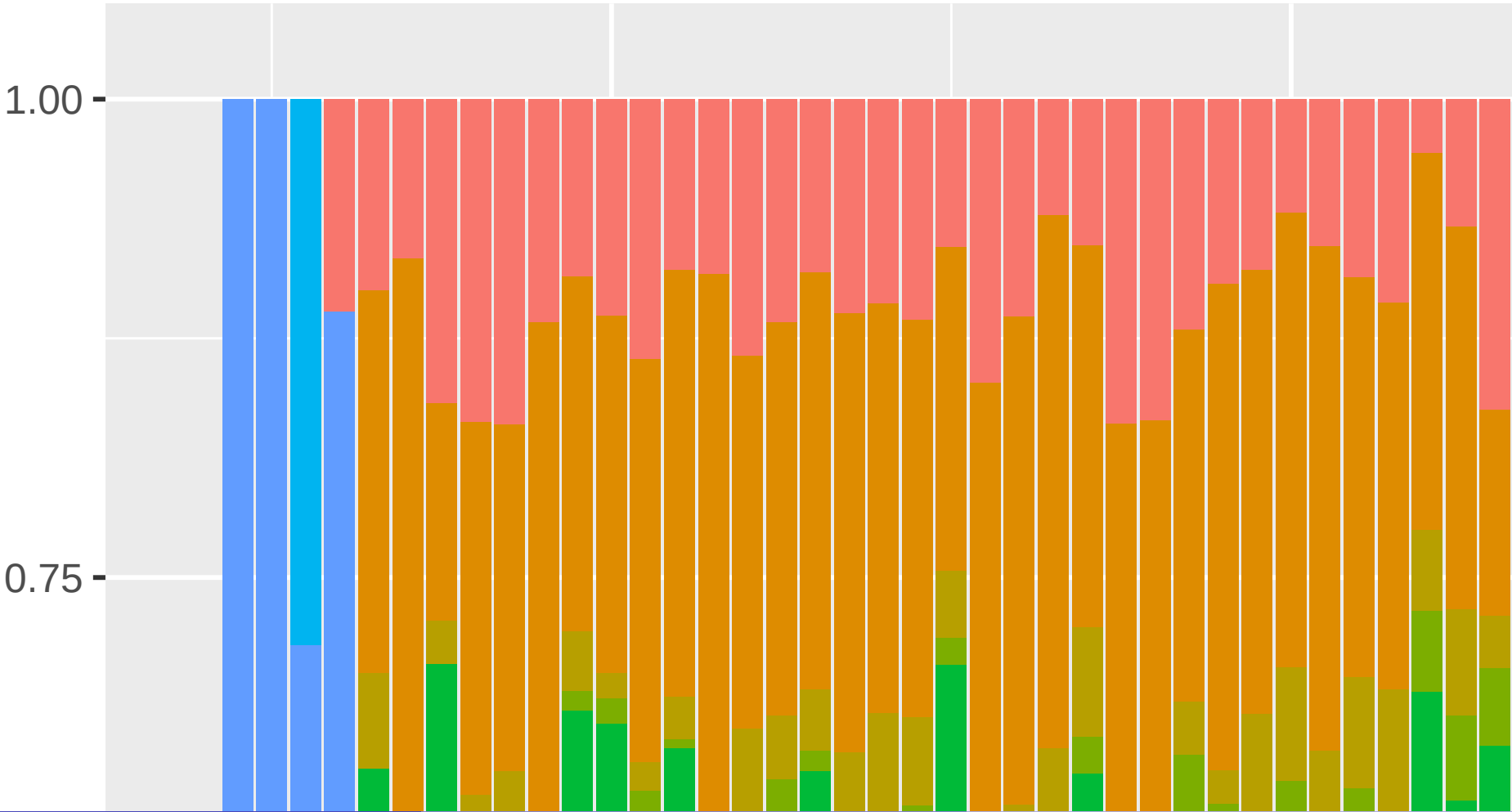
Position for bar

```
ggplot(bank) + geom_bar(mapping = aes(x = age, fill = job))
```



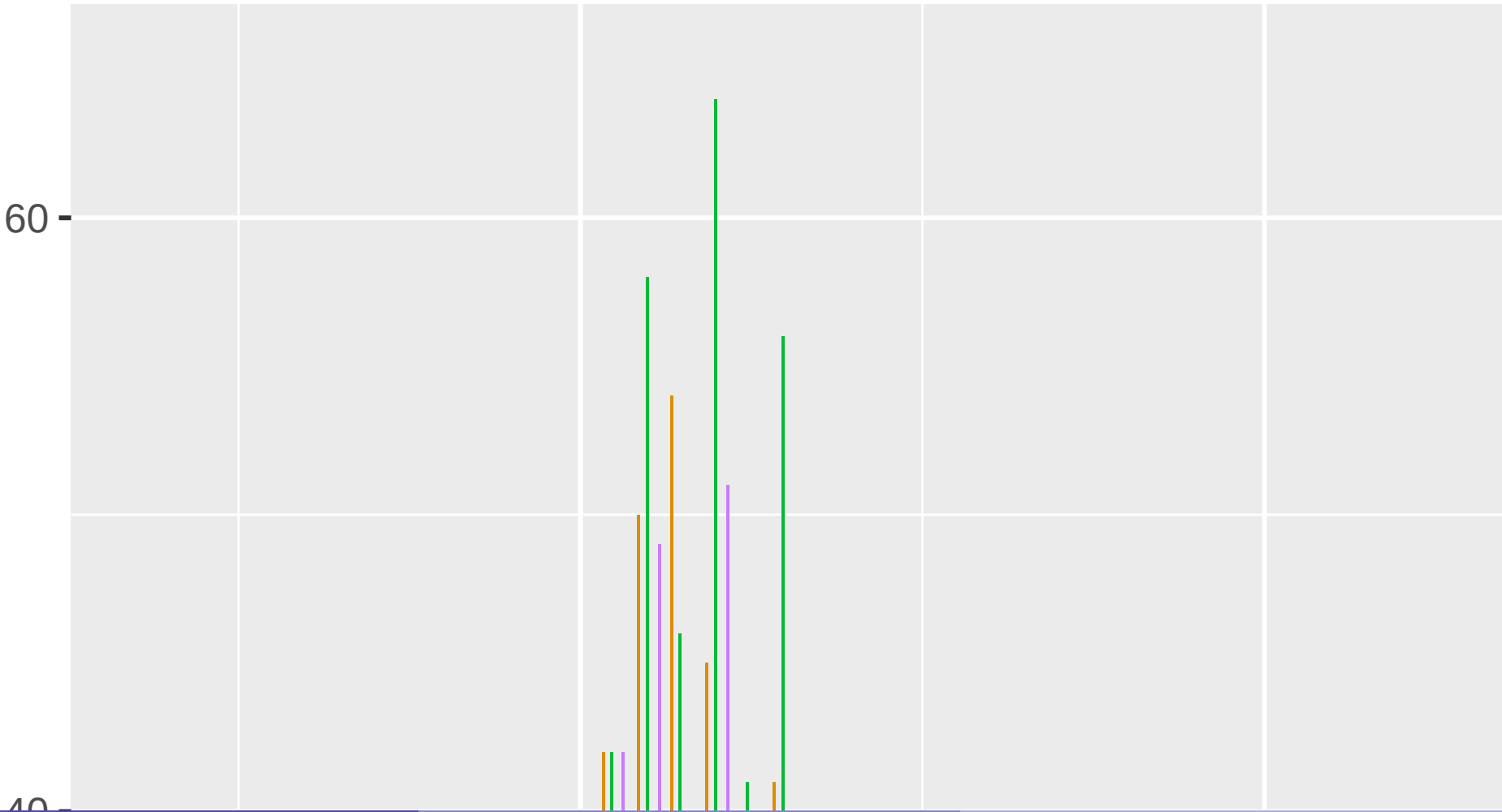
Position for bar: fill

```
# fill to 100%
ggplot(bank) + geom_bar(mapping = aes(x = age, fill = job),
                        position = "fill")
```



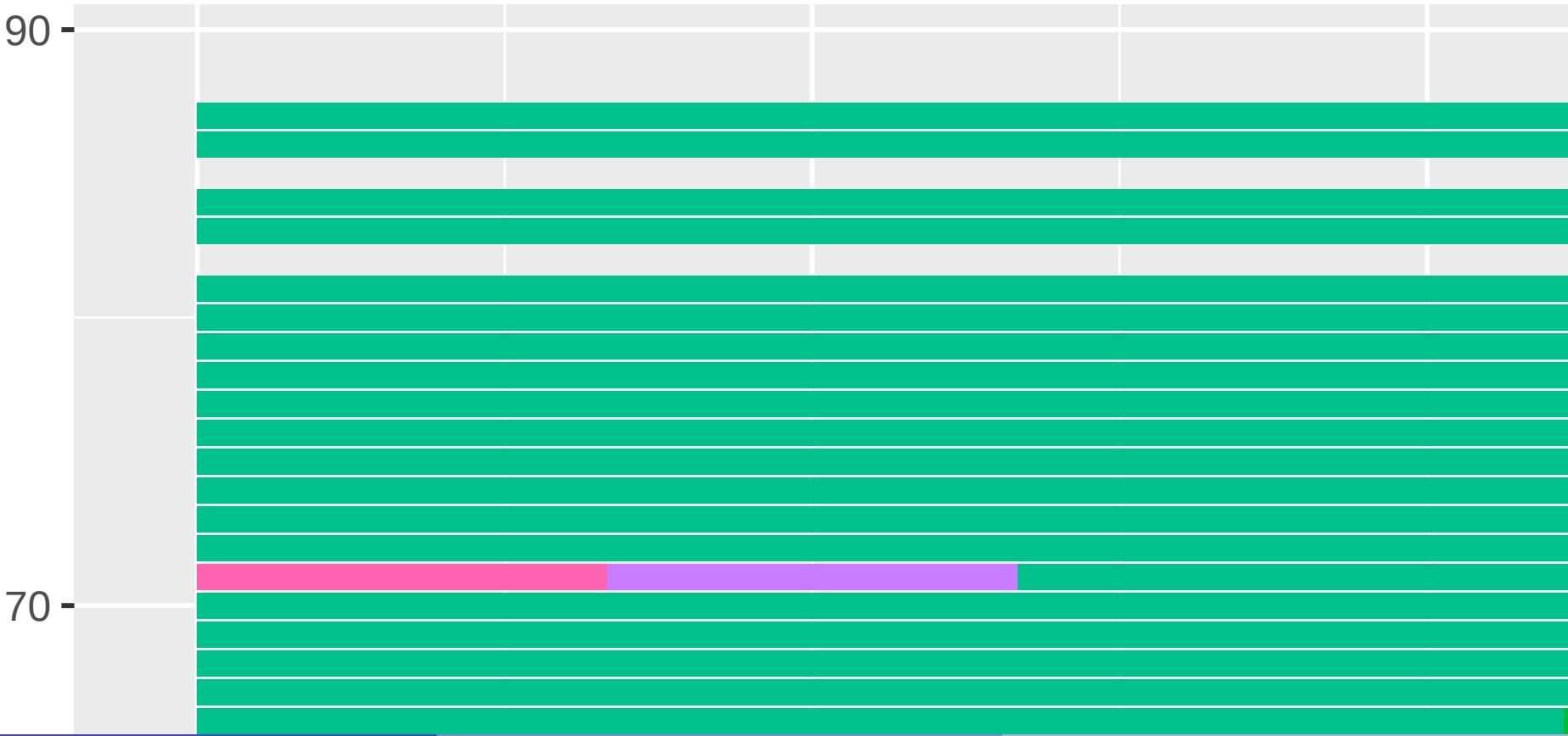
Position for bar: dodge

```
# dodge means "adaptive width of the bar"
ggplot(bank) + geom_bar(mapping = aes(x = age, fill = job),
                        position = "dodge")
```



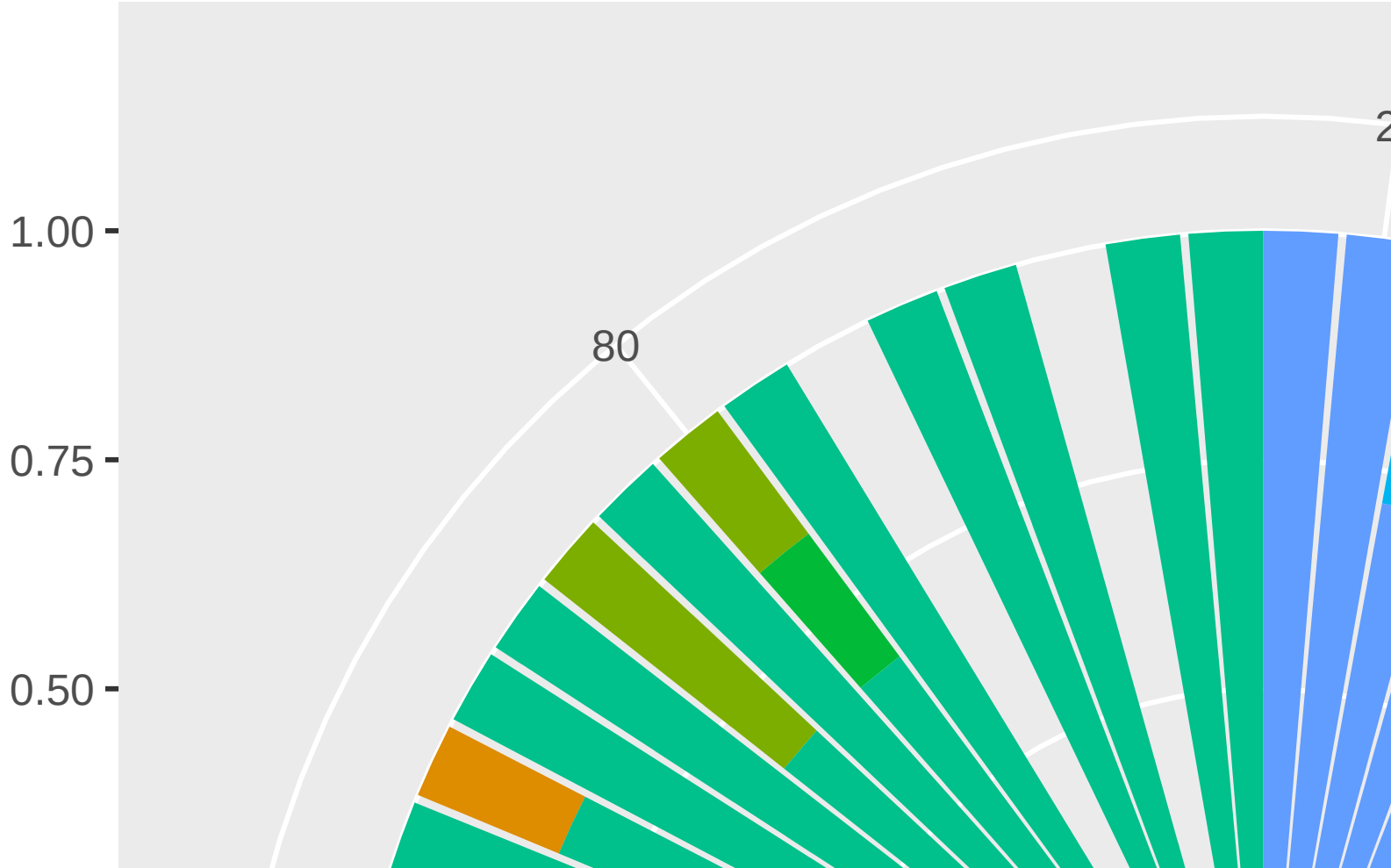
Variations: coord_flip

```
# Switch x and y axis.
# Note any adjustment on x or y axis is effective on the original na
ggplot(bank) +
geom_bar(mapping = aes(x = age, fill = job), position = "fill") +
coord_flip()
```



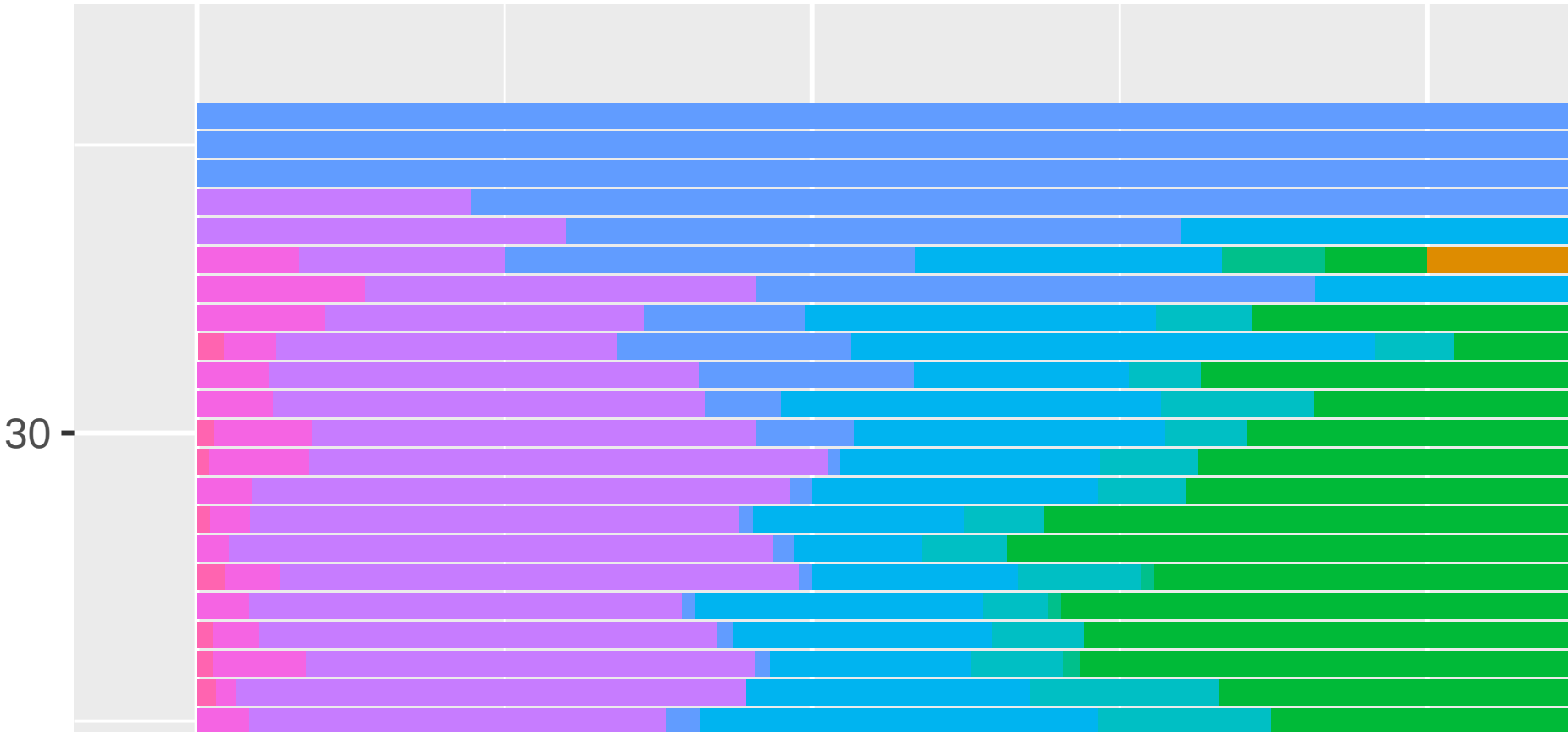
Variations: coord_polar

```
ggplot(bank) +
geom_bar(mapping = aes(x = age, fill = job), position = "fill") +
coord_polar()
```



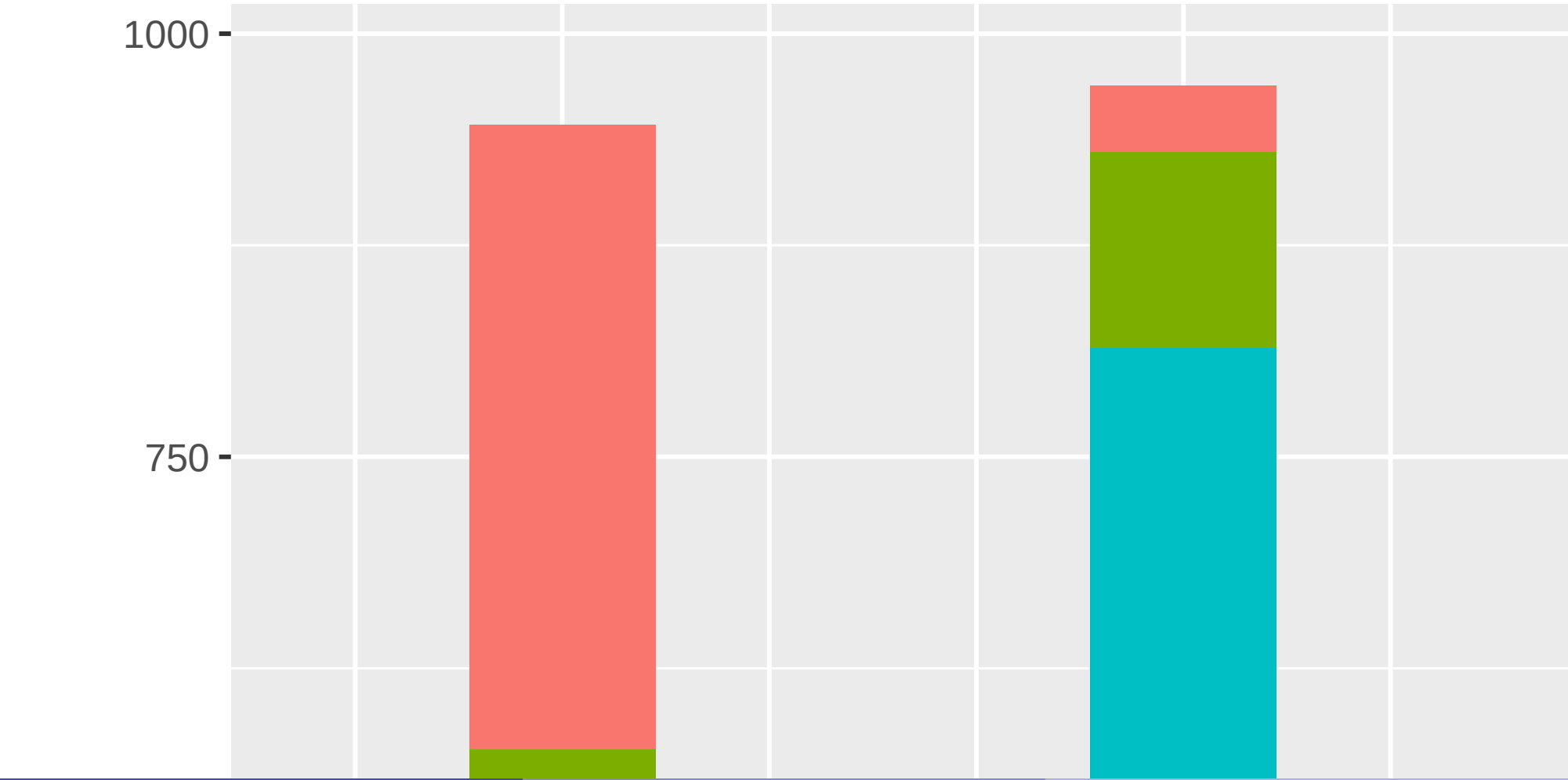
Variations: scale_x_reverse

```
# scale_x_reverse works on continous variable (numeric, date, etc.).
ggplot(bank) +
geom_bar(mapping = aes(x = age, fill = job), position = "fill") +
coord_flip() +
scale_x_reverse()
```



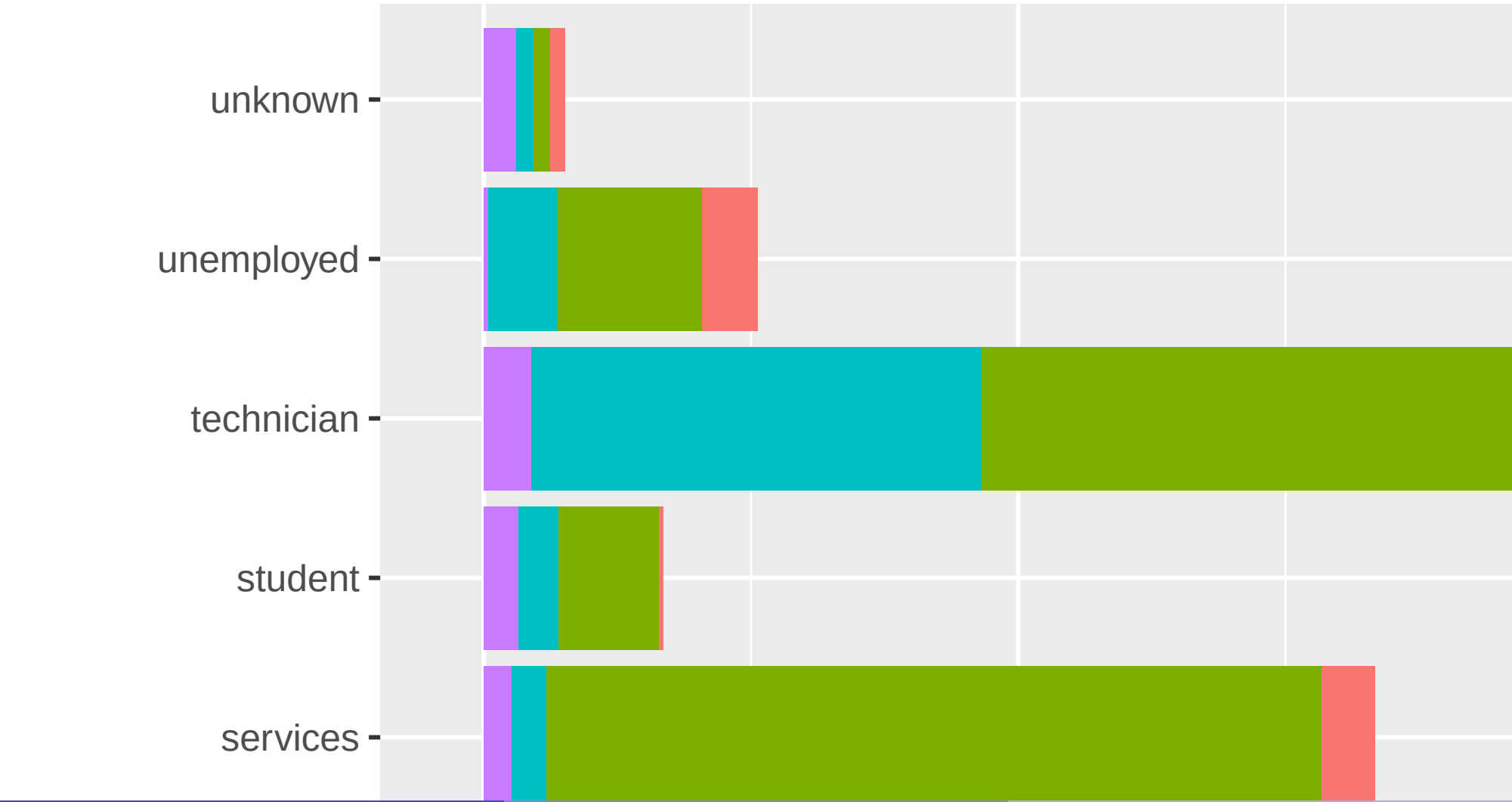
geom_bar: better serves for categorical data

```
ggplot(data = bank, mapping = aes(x = job, fill = education)) +
  geom_bar() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



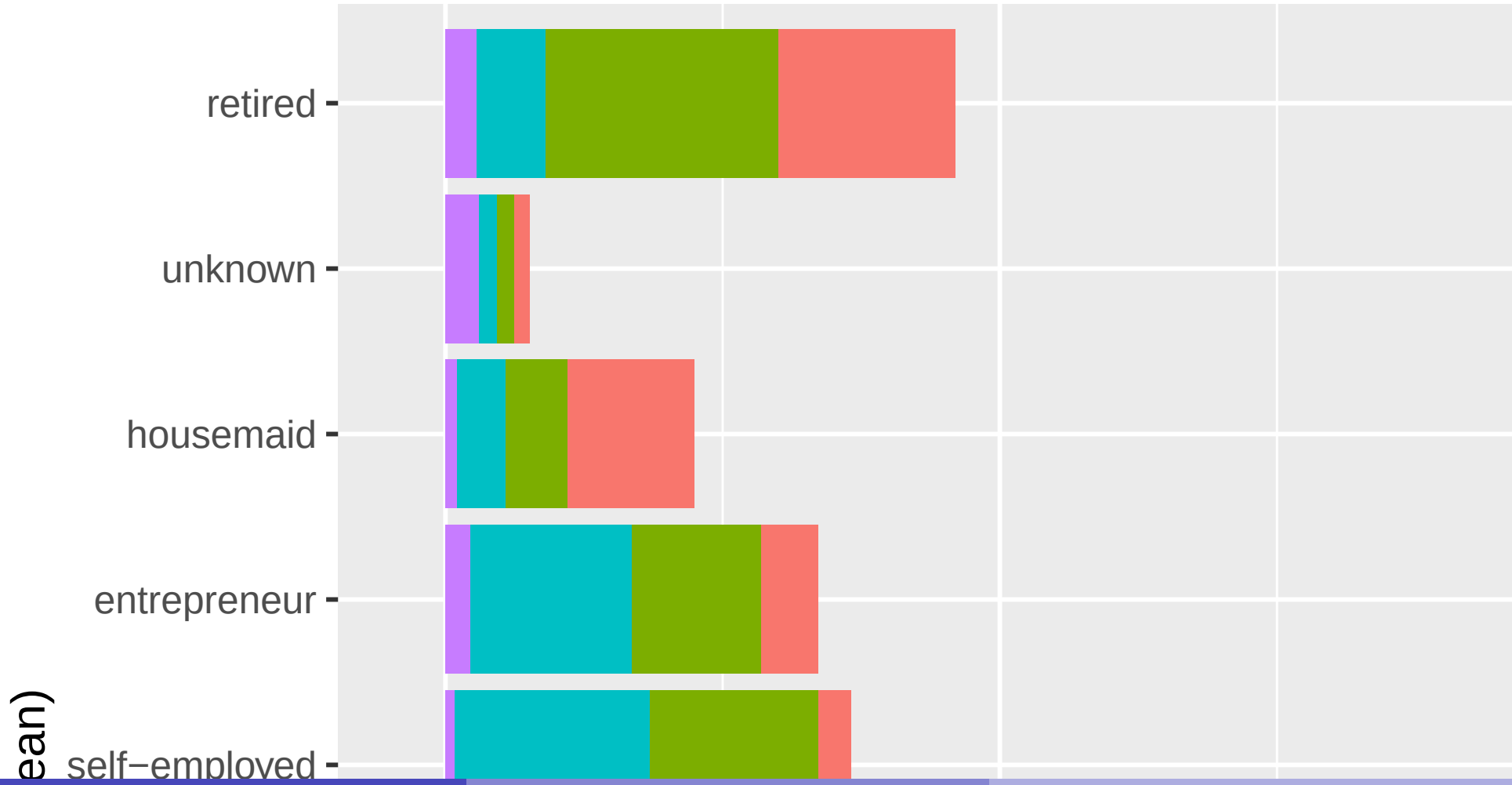
geom_bar: coord_flip

```
ggplot(data = bank, mapping = aes(x = job, fill = education)) +
  geom_bar() + coord_flip()
```



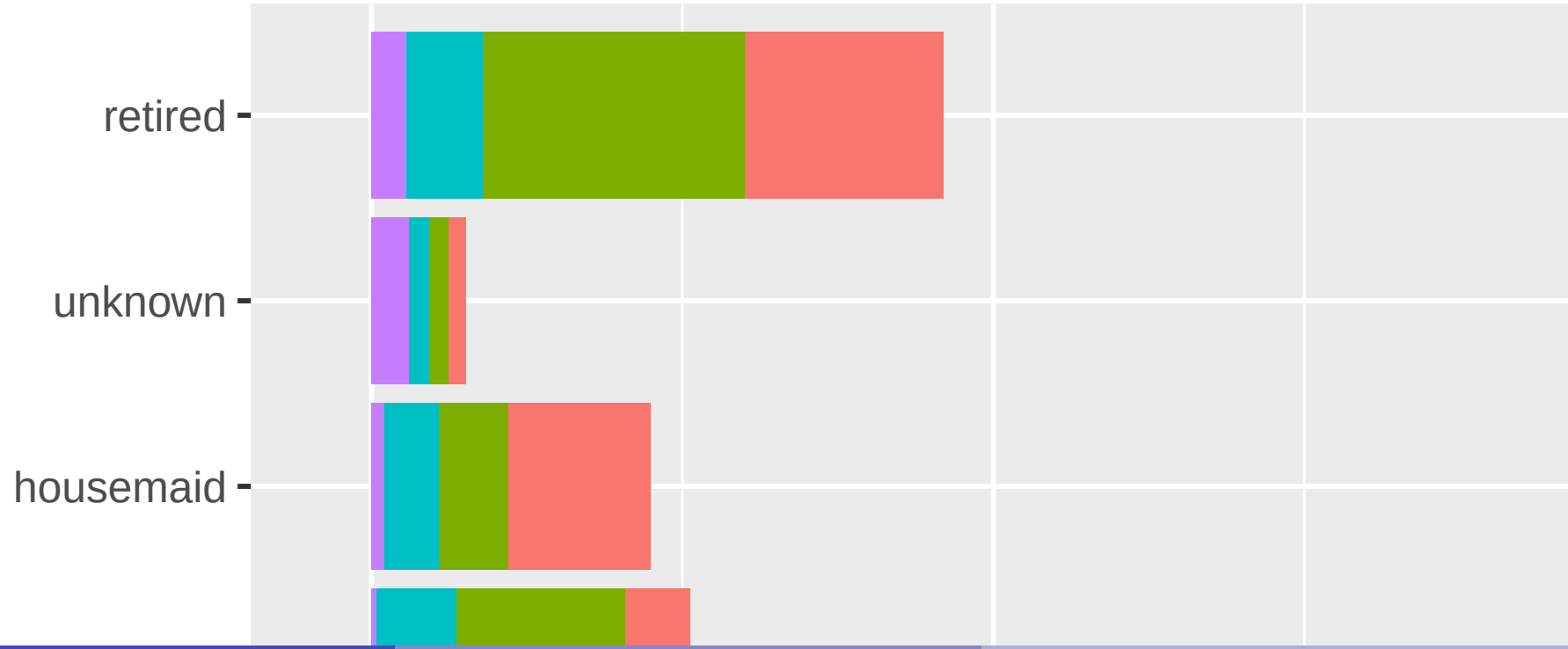
geom_bar: sort job by mean age

```
ggplot(data = bank, mapping = aes(x = reorder(job, age, FUN = mean),
       fill = education)) +
  geom_bar() + coord_flip()
```



geom_bar: sort job by alphabetical order

```
# If we want to order job according to alphabetical order.
# use rev(levels(...))
ggplot(data = bank, mapping = aes(x = reorder(job, age, FUN = median
       fill = education)) +
  geom_bar() +
  scale_x_discrete(limit = rev(levels(bank$job))) +
  coord_flip()
```

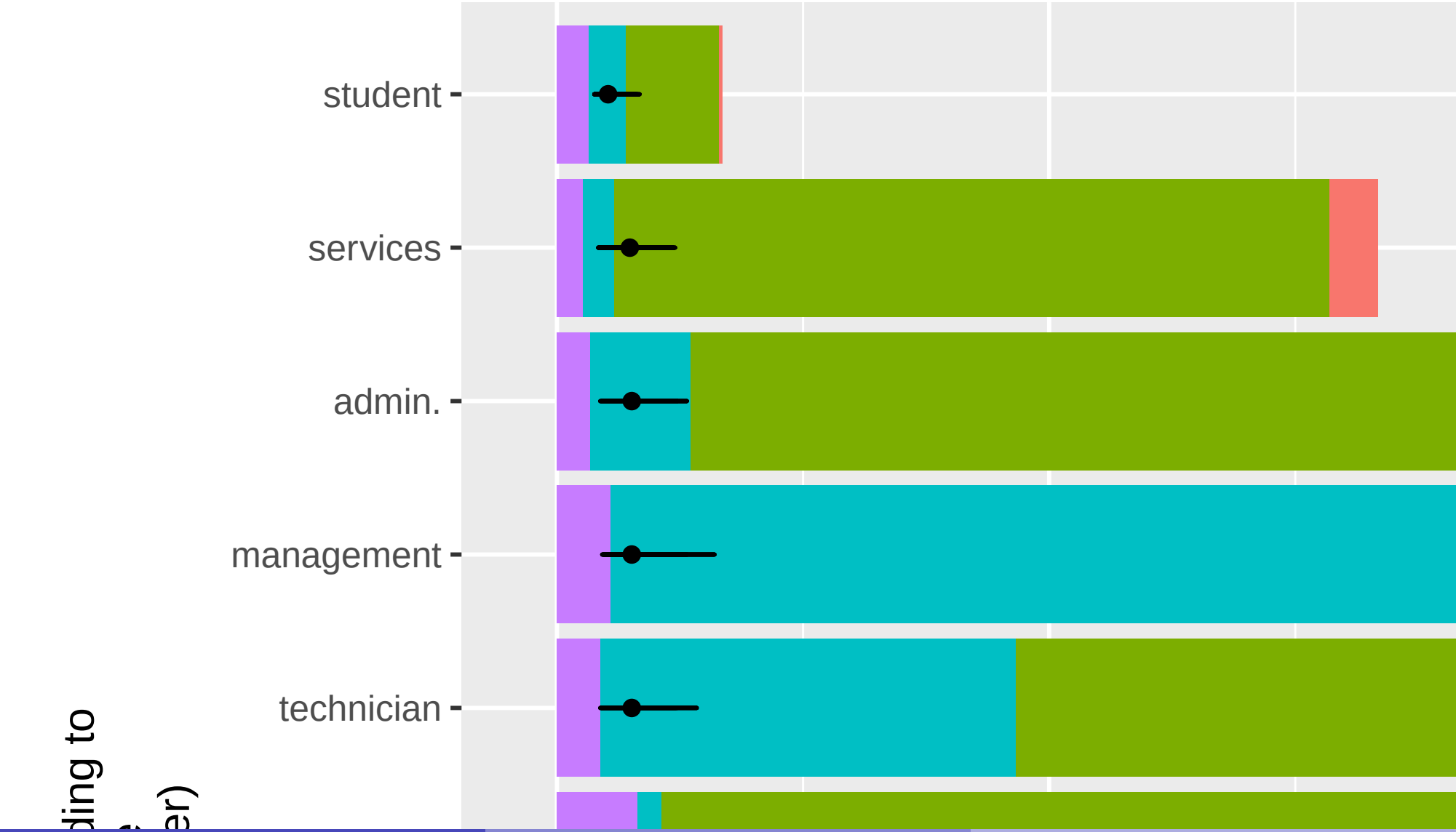


Bar with composite data

```
ggplot(data = bank, mapping = aes(x = reorder(job, age, FUN = median
       fill = education)) +
  # layer 1
  geom_bar() +
  # If we want to sort the job acccording to median age
  scale_x_discrete(limit =
    rev(levels(reorder(bank$job, bank$age, FUN = median)))) +
  # And also add age range and median age.
  geom_line(aes(x = job, y = age)) +
  geom_point(data = group_by(bank, job) %>%
               summarize(age = median(age)) %>% ungroup,
             aes(x = job, y = age), inherit.aes = FALSE) +
  xlab("Job sorted according to\nMedian age\n(Top - younger)") +
  coord_flip()
## `summarise()` ungrouping output (override with `.groups` argumen
```

Bar with composite data: plot

```
## `summarise()` ungrouping output (override with `.groups` argument
```



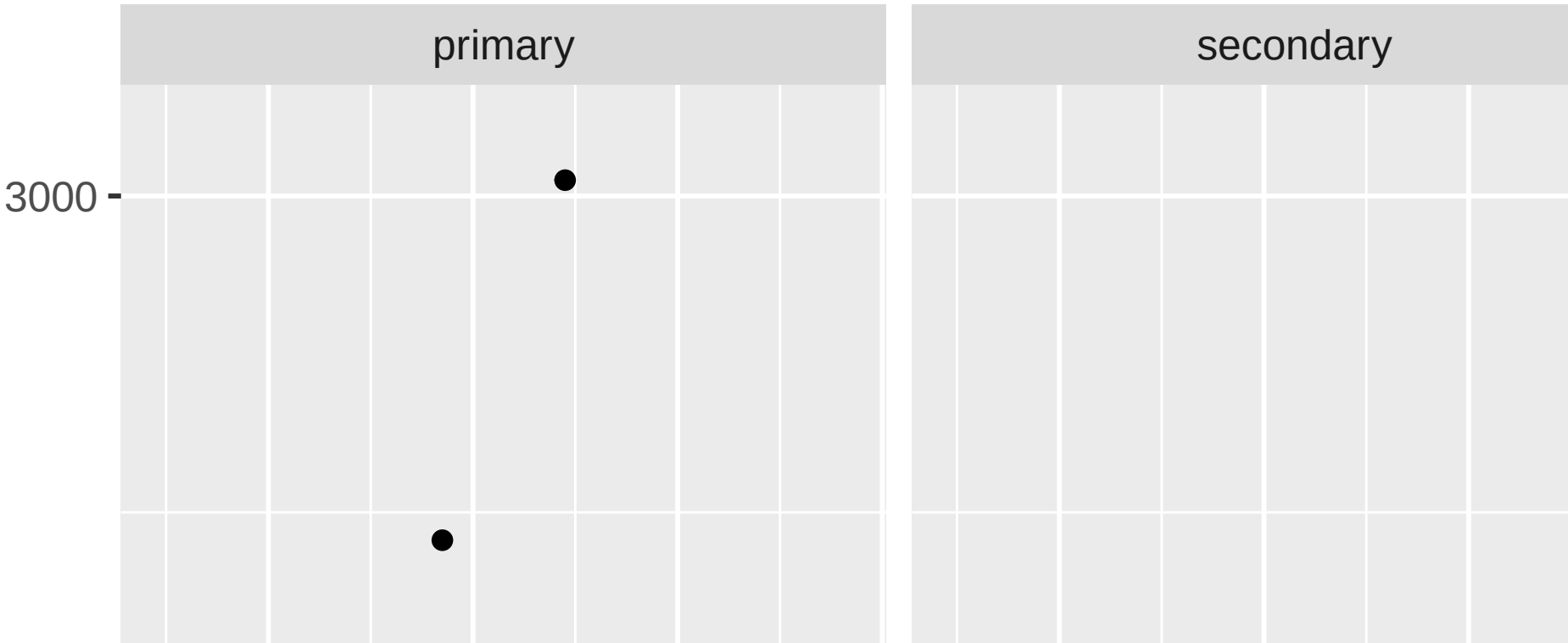
Data with statistical

```
ggplot(data = bank) +
  stat_summary(
    mapping = aes(x = age, y = balance),
    fun.ymin = min,
    fun.ymax = max,
    fun.y = median
  )
## Warning: `fun.y` is deprecated. Use `fun` instead.
## Warning: `fun.ymin` is deprecated. Use `fun.min` instead.
## Warning: `fun.ymax` is deprecated. Use `fun.max` instead.
```

Facets

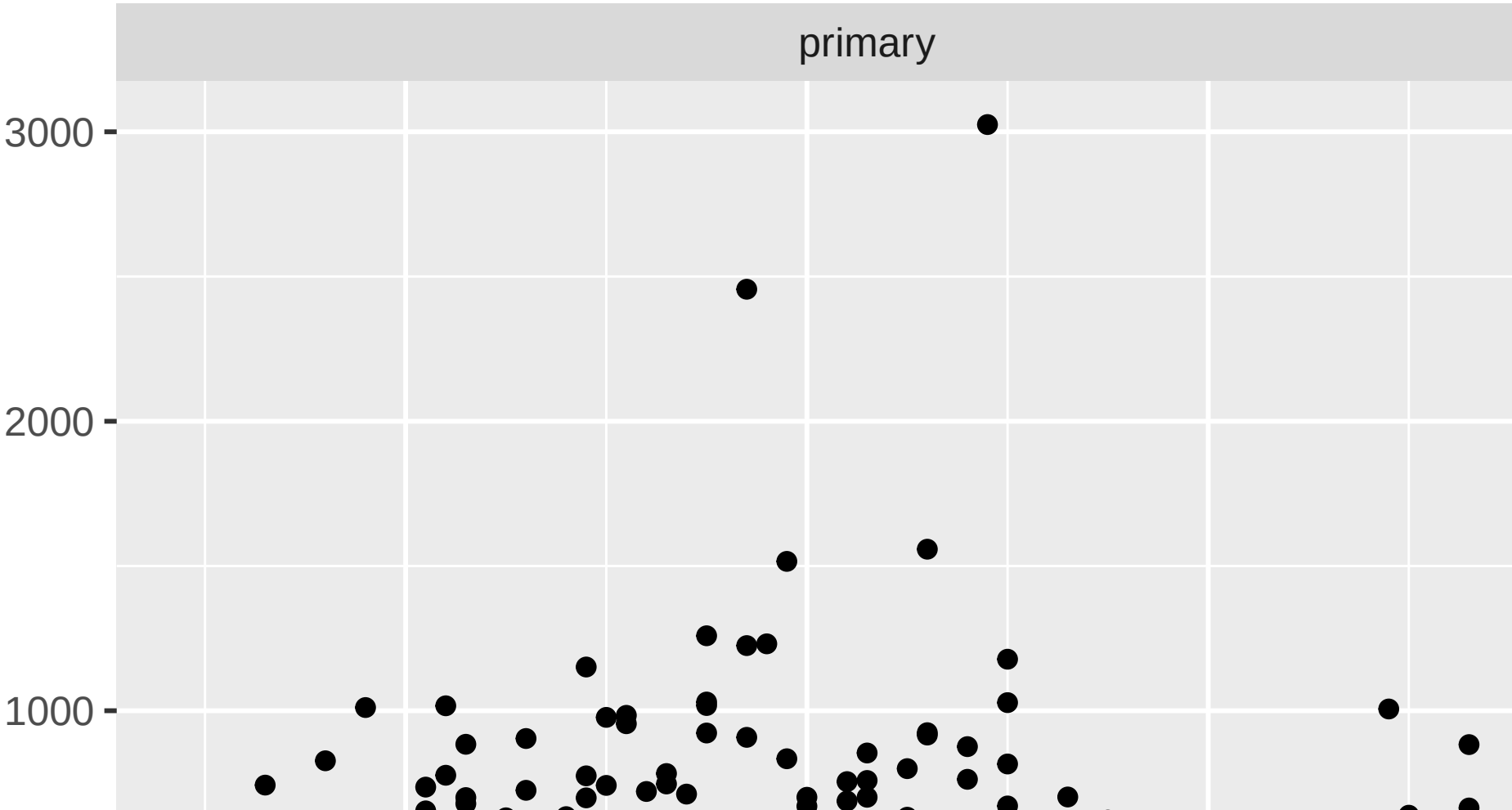
- Create invidual figures.
- `facet_grid`: basic
- `facet_wrap`: you can control number of rows and cols

```
ggplot(data = bank) +
  geom_point(mapping = aes(x = age, y = duration)) +
  facet_grid( ~ education)
```



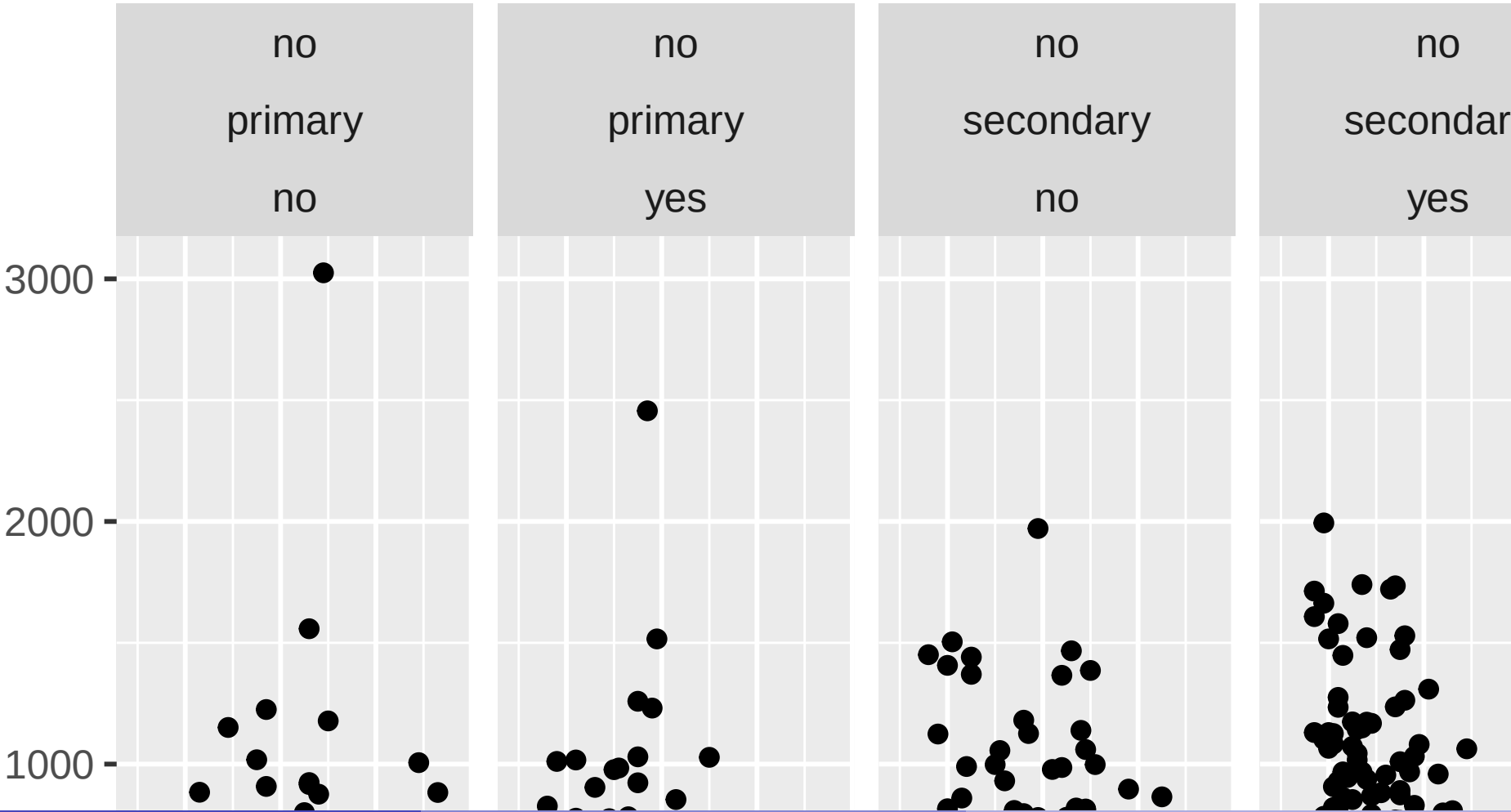
Facets - facet_wrap

```
ggplot(data = bank) +
  geom_point(mapping = aes(x = age, y = duration)) +
  facet_wrap(~ education, nrow = 2)
```



Facets - facet_wrap multi-dimension

```
ggplot(data = bank) +
  geom_point(mapping = aes(x = age, y = duration)) +
  facet_wrap(loan ~ education ~ housing, nrow = 2)
```

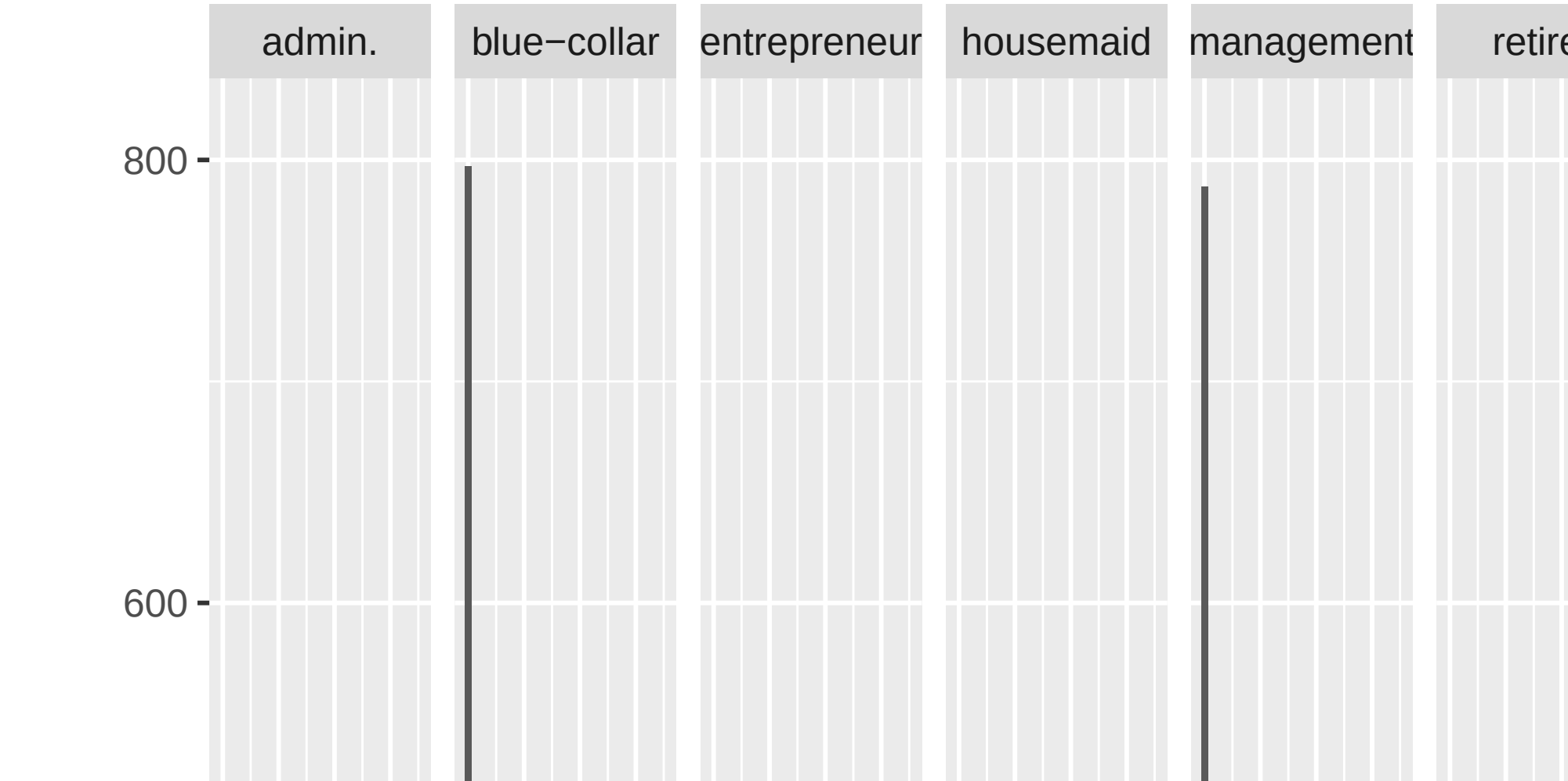


Facets - finding the best

```
# doesn't look great because we have so many jobs.
ggplot(bank, aes(pdays)) + geom_histogram() + facet_grid(job ~ .)
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth
```

Facets - finding the best. still trying.

```
# Not a good choice, neither
ggplot(bank, aes(pdays)) + geom_histogram() + facet_grid(. ~ job)
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth
```

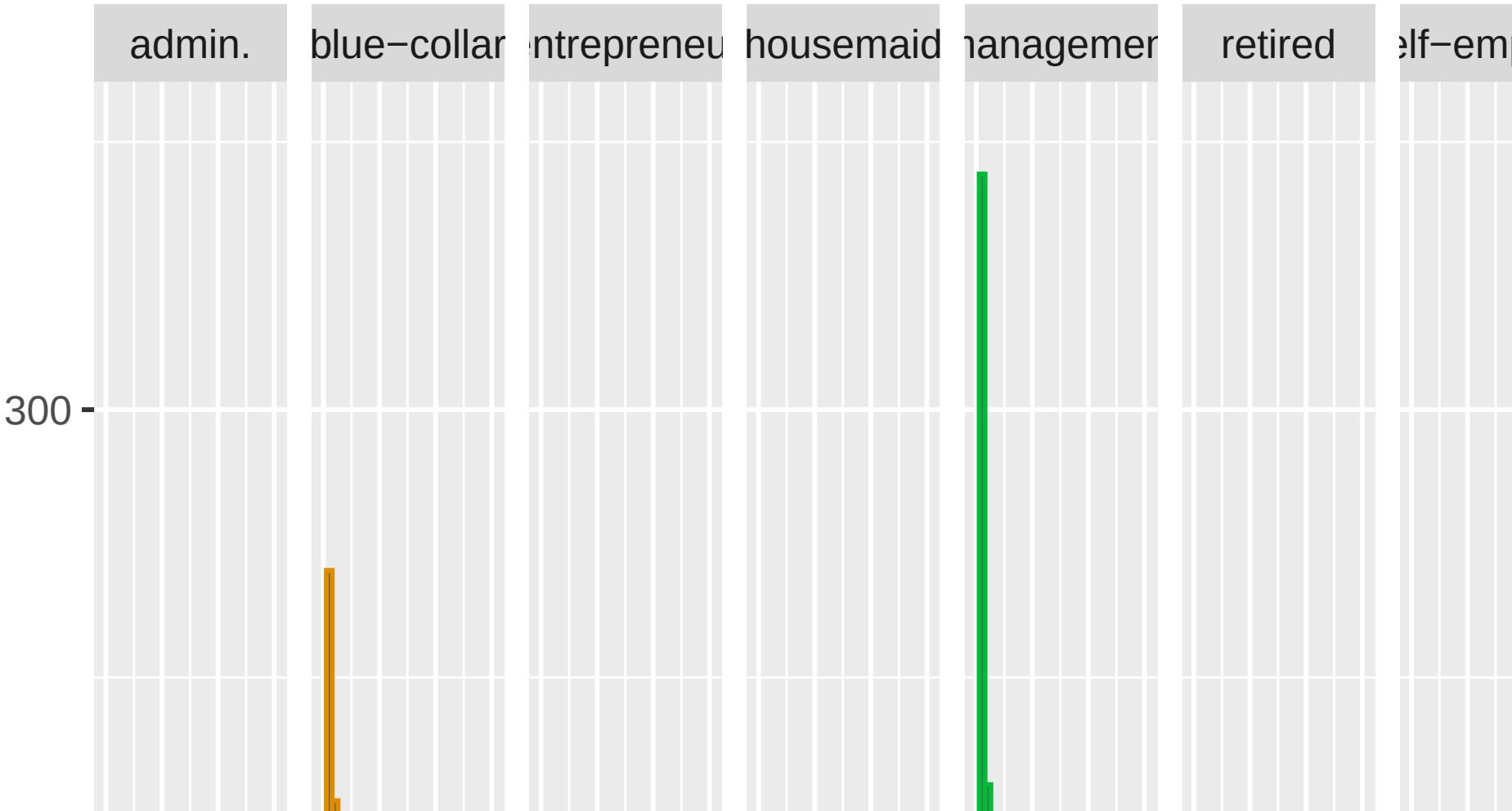


Facets - finding the best. better

```
# Can we do better?
ggplot(bank, aes(campaign)) + geom_histogram() + facet_grid(. ~ job)
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`
```

acets - finding the best. try another

```
ggplot(bank, aes(duration)) + geom_histogram(aes(color = job)) +
  facet_grid(. ~ job)
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth
```

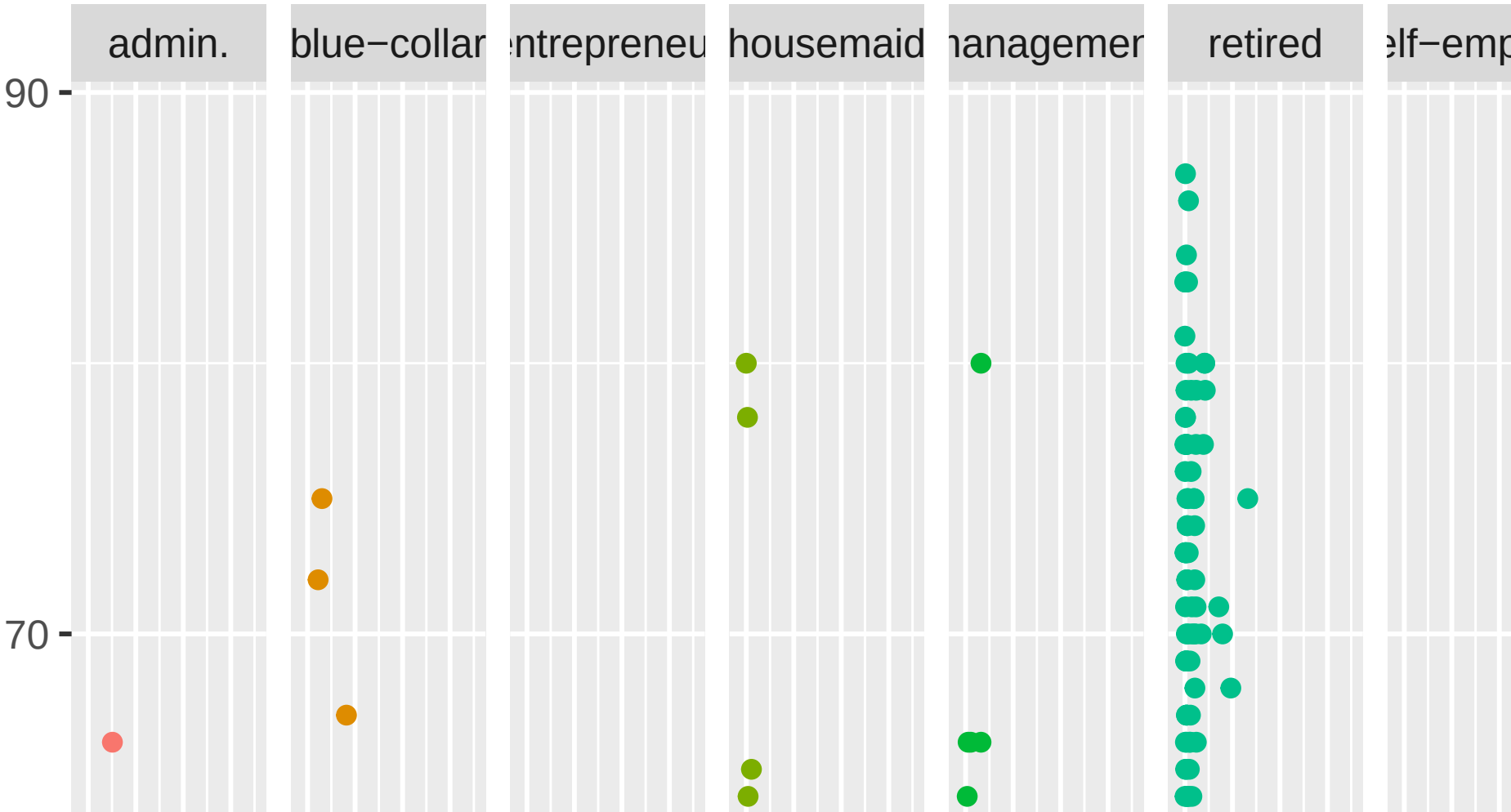


Facets - finding the best. Try points

```
# facet with points is good
ggplot(bank, aes(balance, age)) + geom_point() + facet_grid(. ~ job)
```

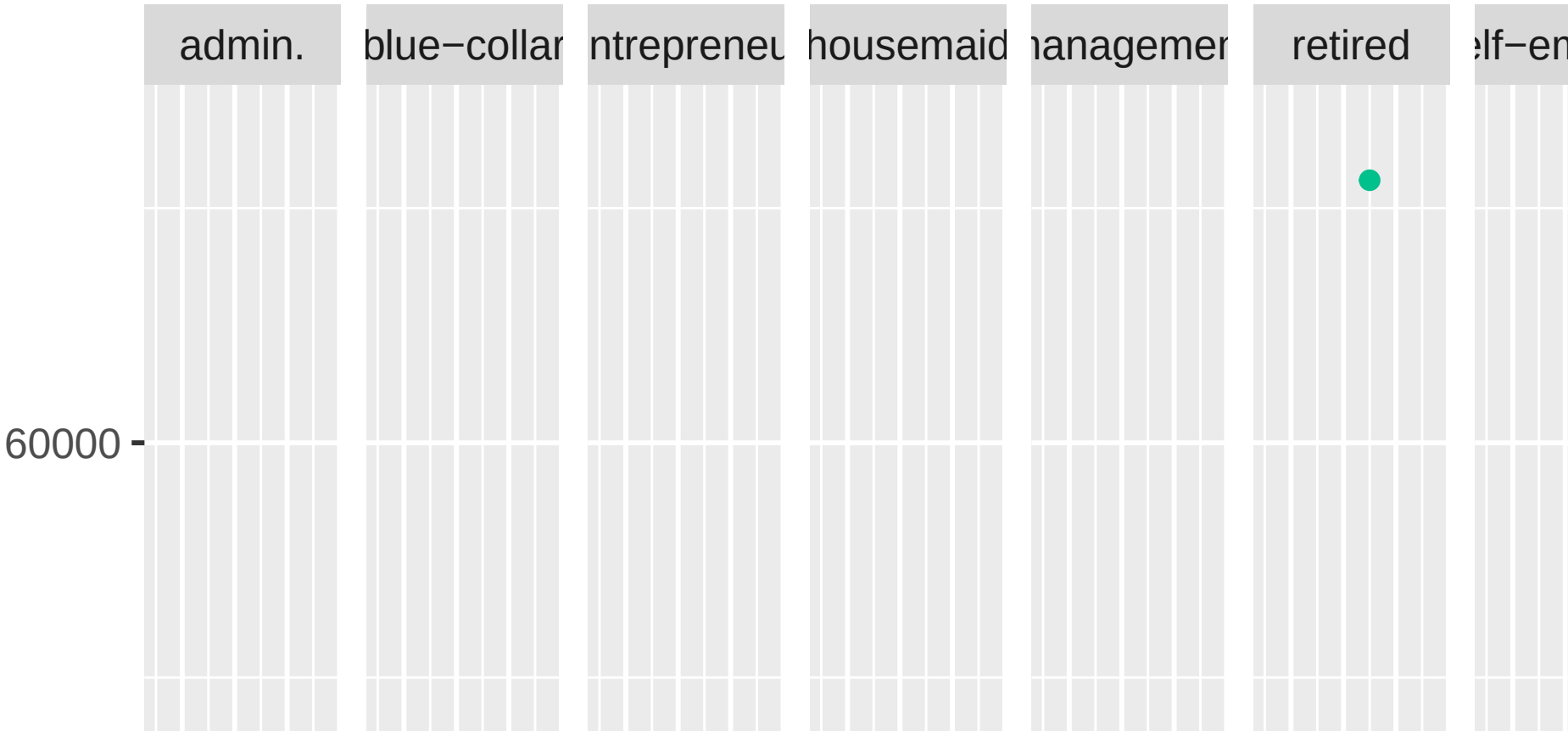
Facets - finding the best. Better with color

```
# do better
ggplot(bank, aes(balance, age)) + geom_point(aes(color = job)) +
  facet_grid(. ~ job)
```



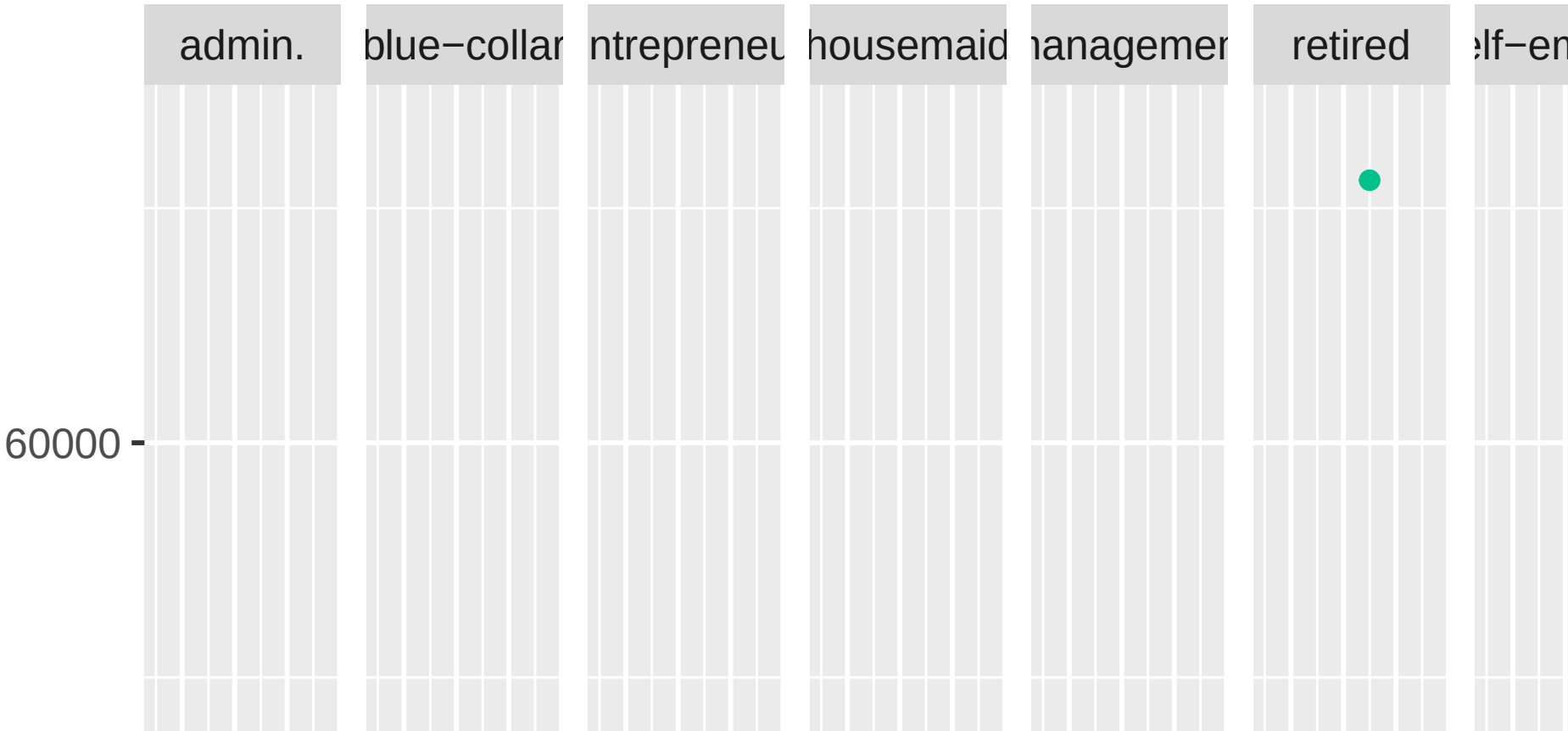
Facets - finding the best. Try Points

```
# Can we apply points between age and balance?
ggplot(bank, aes(age, balance, color = job)) +
  geom_point() + geom_smooth() +
  facet_grid(. ~ job)
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



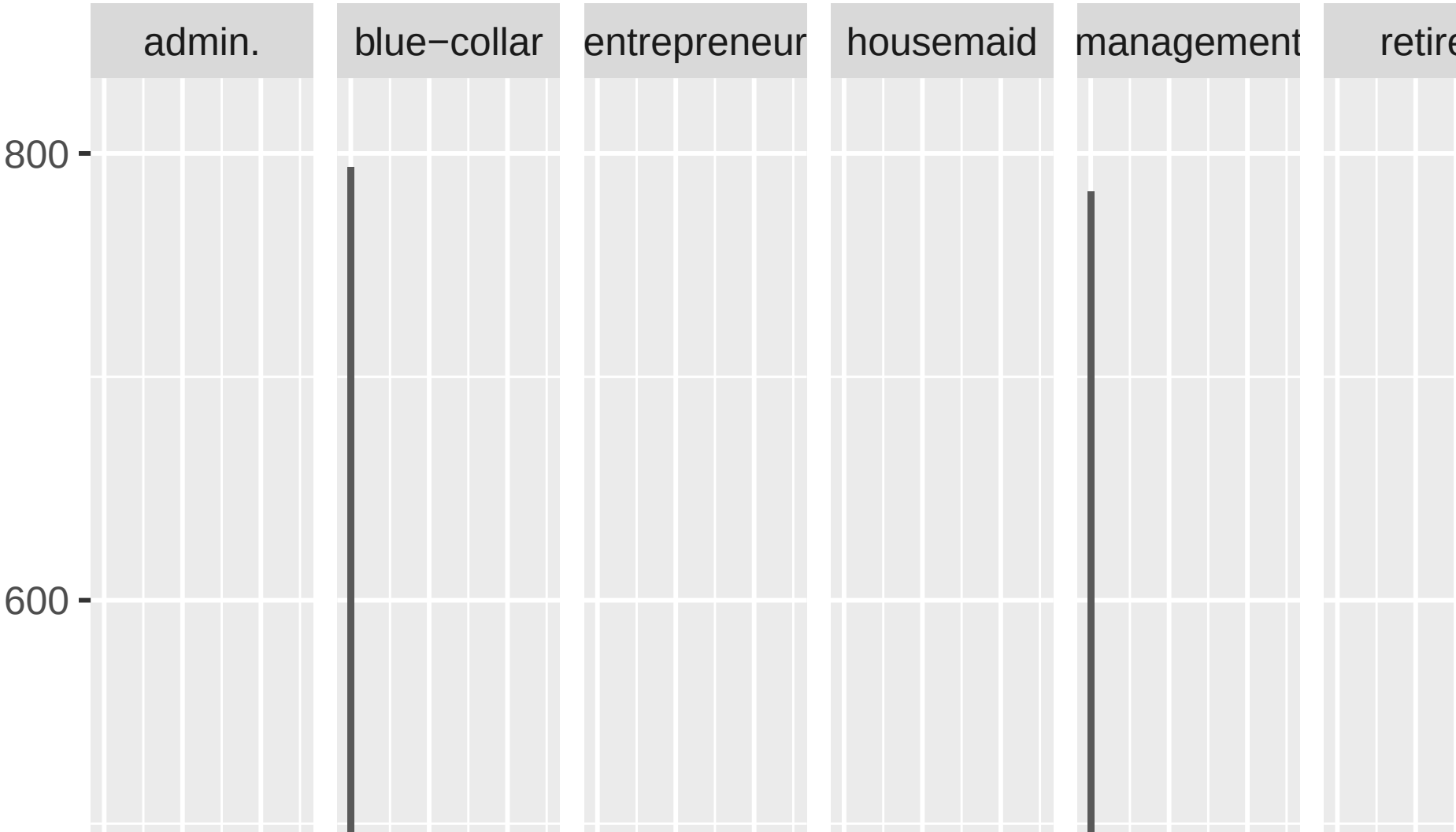
Facets - finding the best. Try Smoothie

```
# Smooth line is mixed with points
ggplot(bank, aes(age, balance)) + geom_point(aes(color = job)) +
  geom_smooth() +
  facet_grid(. ~ job)
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



With facets or without facets?

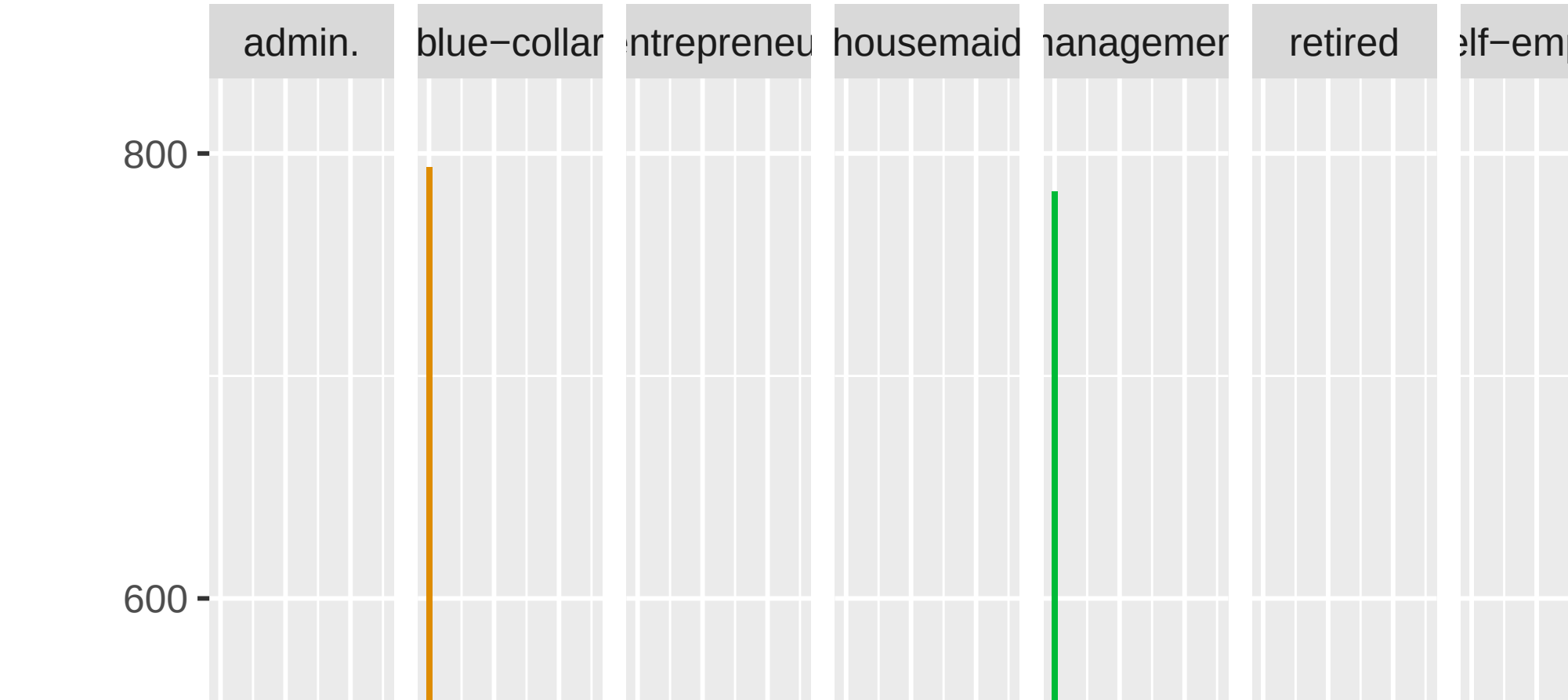
```
ggplot(bank, aes(previous)) + geom_histogram() + facet_grid(. ~ job)
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`
```



With facets or without facets?

facets with color

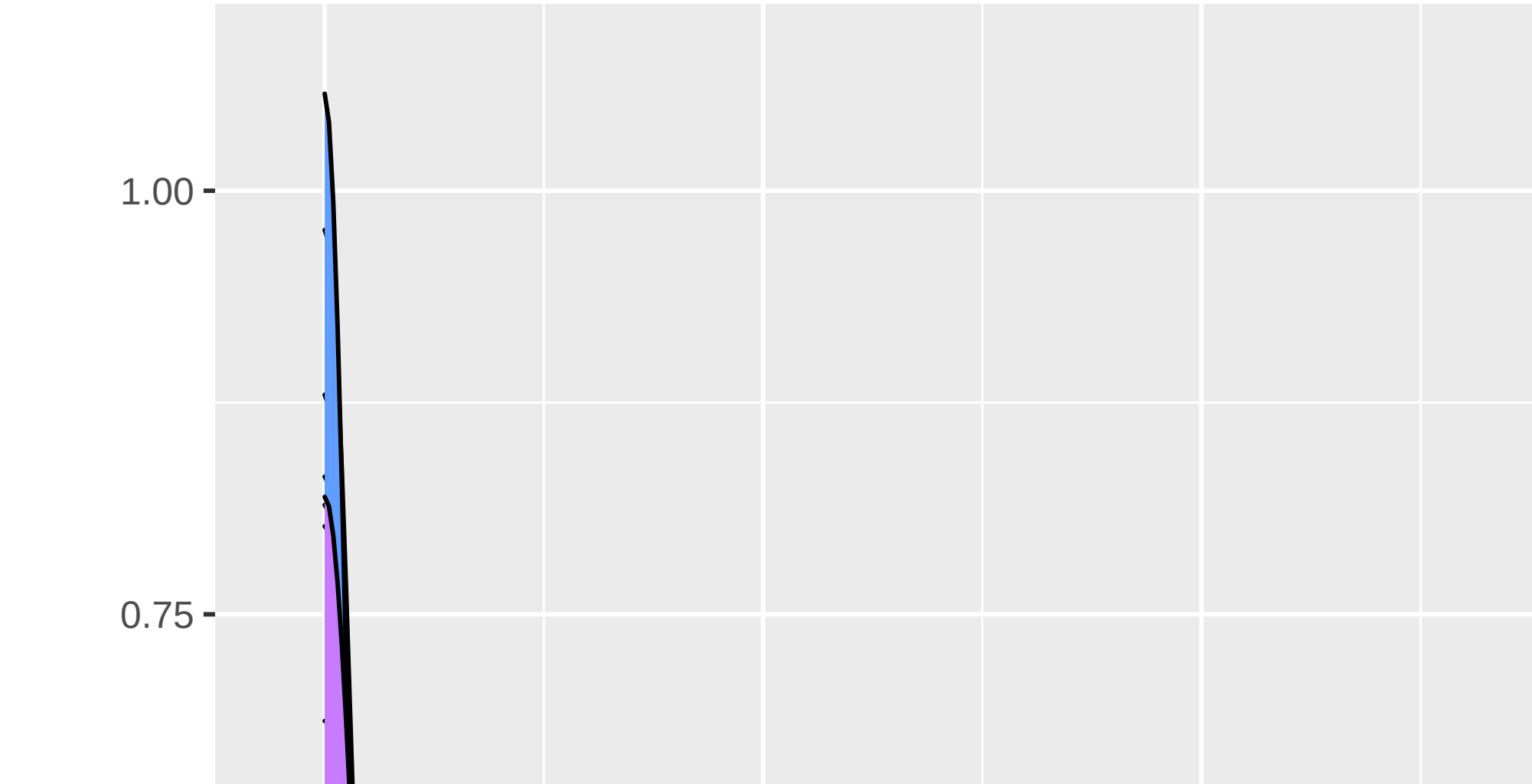
```
ggplot(bank, aes(previous)) + geom_histogram(aes(fill = job)) +
  facet_grid(. ~ job)
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth
```



With facets or without facets?

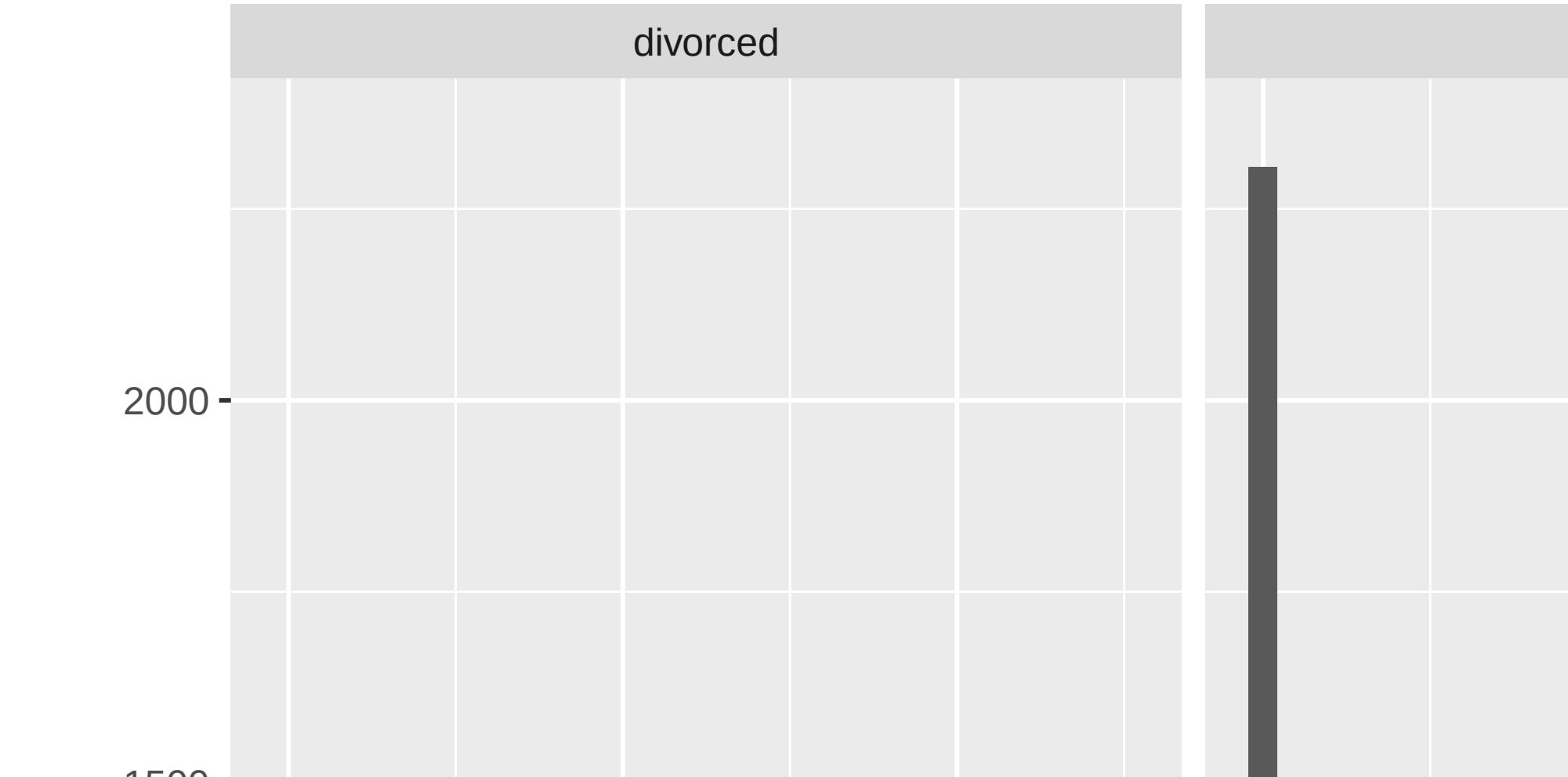
Use density/color on one figure

```
ggplot(bank, aes(previous)) + geom_density(aes(fill = job))
```



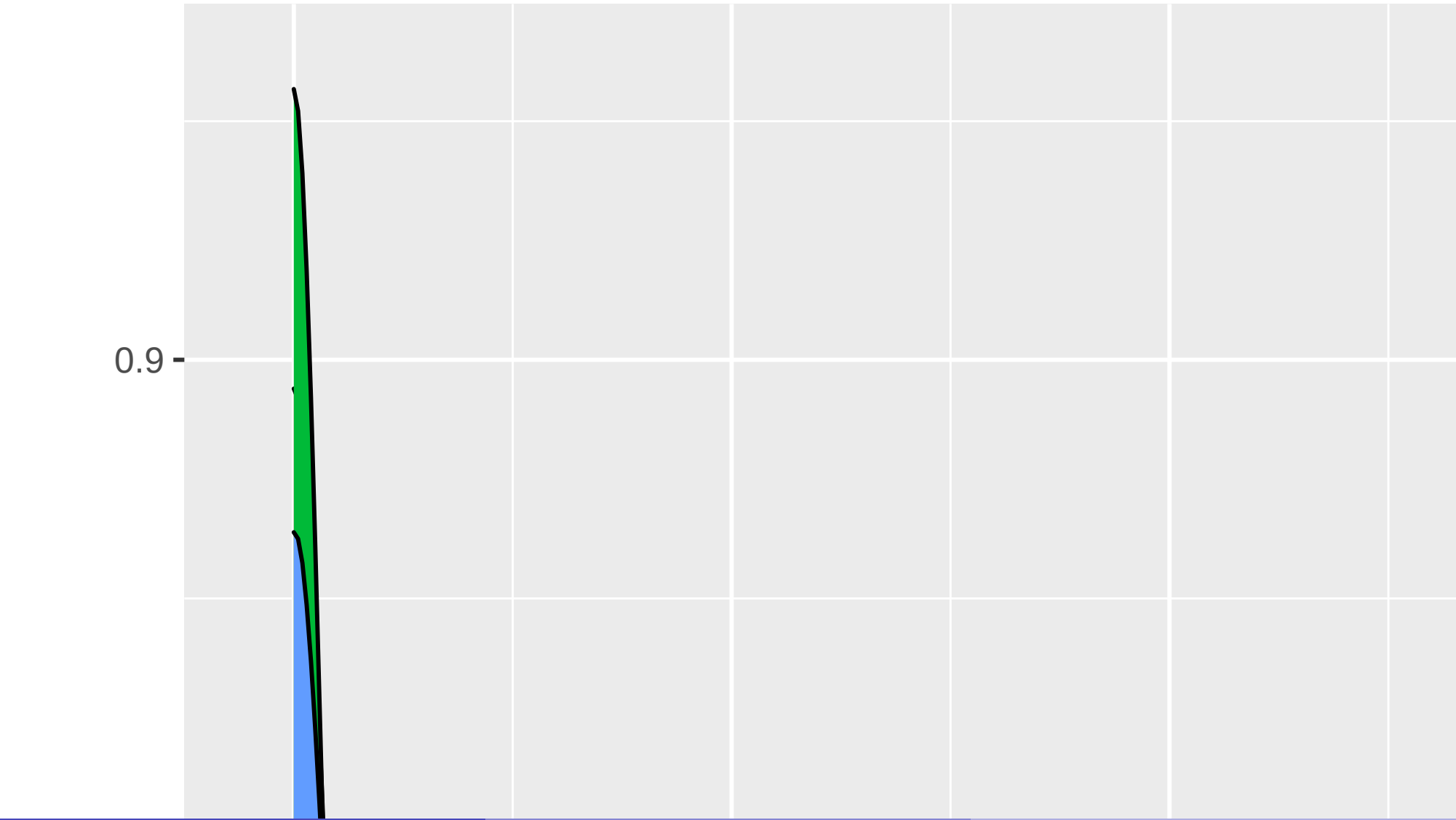
With facets or without facets: Case 2

```
ggplot(bank, aes(previous)) + geom_histogram() +
  facet_grid(. ~ marital)
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth
```



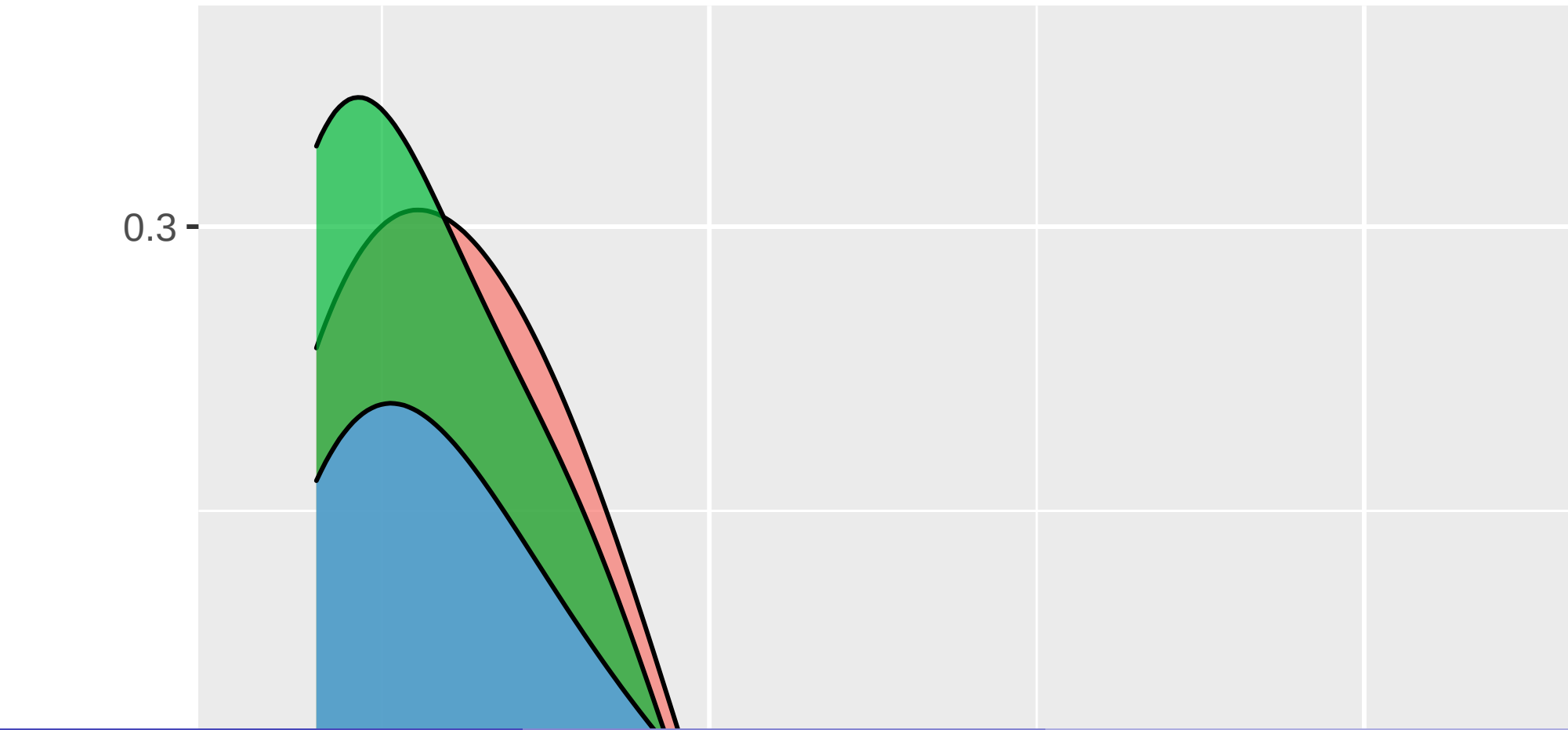
With facets or without facets: `geom_density` and `fill`

```
ggplot(bank, aes(previous)) + geom_density(aes(fill = marital))
```



With facets or without facets: xlim

```
ggplot(bank, aes(previous)) +
  geom_density(aes(fill = marital), alpha = 0.7) +
  xlim(1, 10)
## Warning: Removed 3725 rows containing non-finite values (stat_den
```

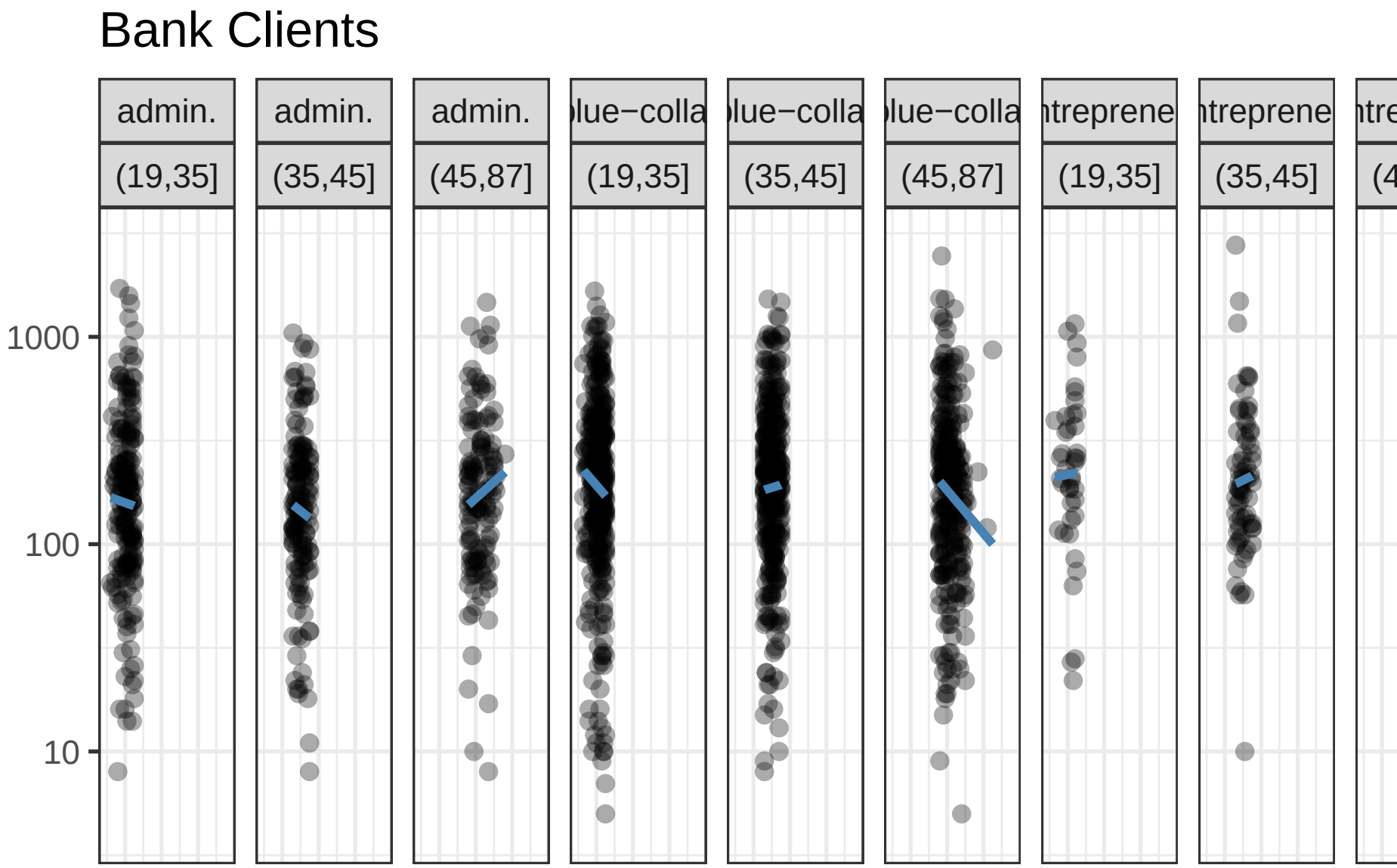


Facets in full power

```
# Levels gives more control to the layer and style.
cutpoints <- quantile(bank$age, seq(0, 1, length = 4), na.rm = TRUE)
# The age_group variable is now a categorical factor variable conta
# 3 levels, indicating the ranges of age.
bank$age_group <- cut(bank$age, cutpoints)
levels(bank$age_group)
## [1] "(19,35]" "(35,45]" "(45,87]"
# Use facet_wrap to specify nrow/ncol.
ggplot(bank, aes(age, duration)) +
  geom_point(alpha = 1/3) +
  facet_wrap(job ~ age_group, nrow = 2) +
  geom_smooth(method="lm", se=FALSE, col="steelblue") +
  theme_bw(base_size = 10) +
  labs(x = "age", y = expression("log " * Duration)) +
  scale_y_log10() +
  labs(title = "Bank Clients")
## `geom_smooth()` using formula 'y ~ x'
```

Facets in full power: plot

```
## `geom_smooth()` using formula 'y ~ x'
```



Add theme

default theme is theme_gray()

```
g <- ggplot(bank, aes(x = age, y = log10(duration)))
g + geom_point(aes(color = job), size = 4, alpha = 1/2) + theme_bw()
g + geom_point(aes(color = job), size = 4, alpha = 1/2) + theme_voi

g + geom_point(aes(color = job), size = 4, alpha = 1/2) + theme_mini
    labs(title = "Duration is longer with age",
         subtitle = "some random plot",
         caption = "from MFE") +
    labs(x = "age", y = expression("log " * Duration))
```

Add theme: result

ggthemes

- package ggthemes provides many other themes.

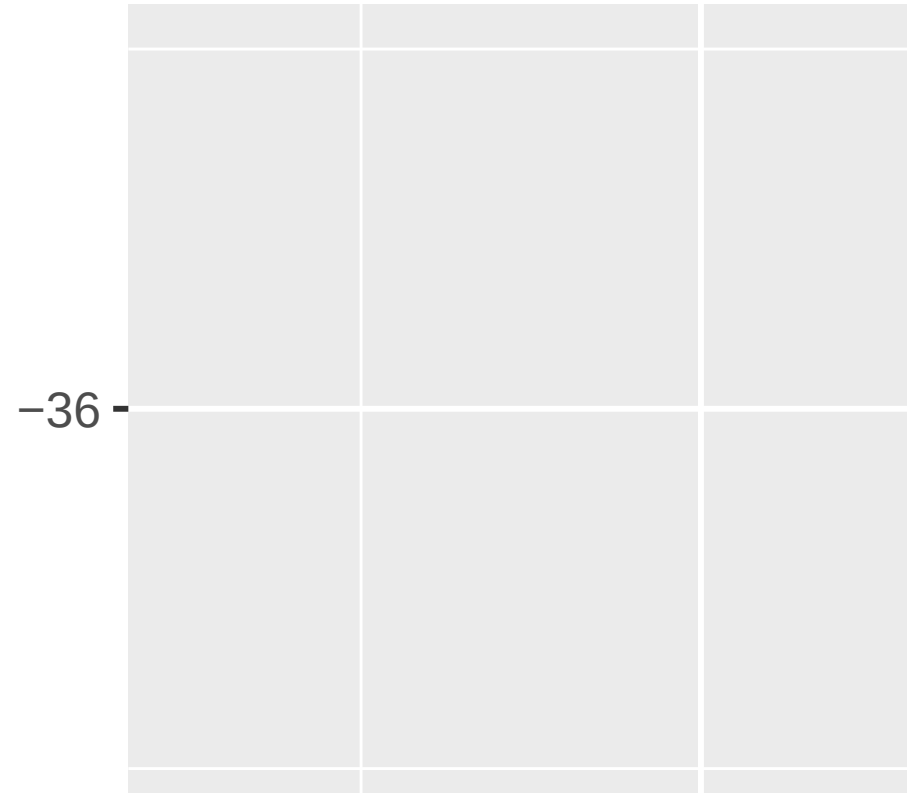
```
library(ggthemes)
## [1] "theme_base" "theme_calc"
## [3] "theme_economist" "theme_economist_white"
## [5] "theme_excel" "theme_few"
## [7] "theme_fivethirtyeight" "theme_foundation"
## [9] "theme_gdocs" "theme_hc"
## [11] "theme_igray" "theme_map"
## [13] "theme_pander" "theme_par"
## [15] "theme_solarized" "theme_solarized_2"
## [17] "theme_solid" "theme_stata"
## [19] "theme_tufte" "theme_wsj"
```

ggplot summary

```
ggplot(data = <DATA>) +
  <GEOM_FUNCTION>(
     mapping = aes(<MAPPINGS>),
     stat = <STAT>,
     position = <POSITION>
  ) +
  <COORDINATE_FUNCTION> +
  <FACET_FUNCTION>
```

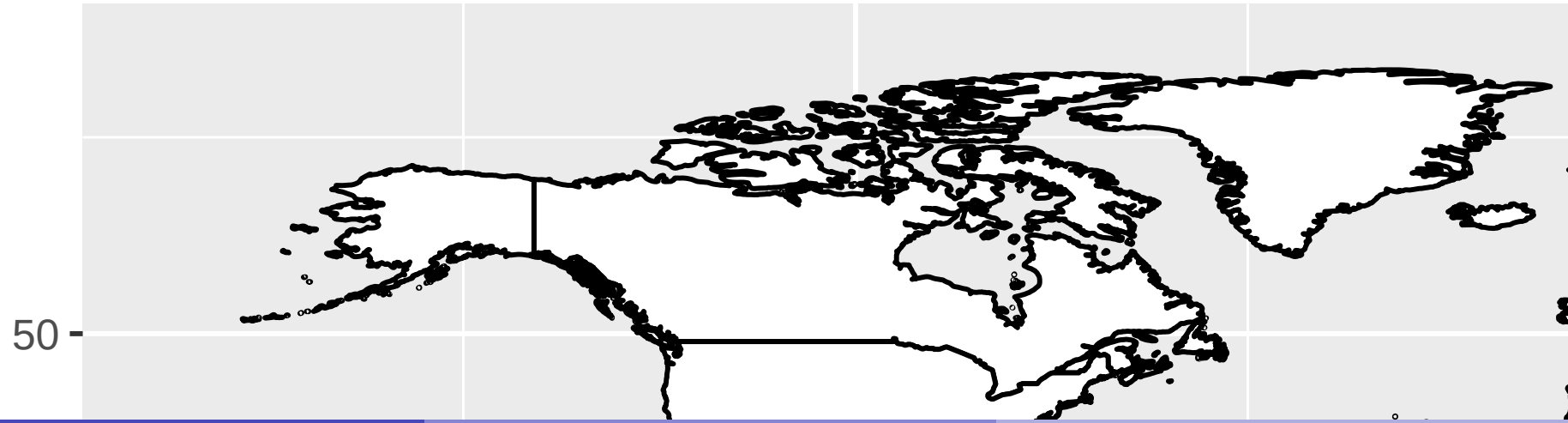
ggplot: one more thing: nz

```
# install.packages("maps")
library(maps)
nz <- map_data("nz")
ggplot(nz, aes(long, lat, group = group)) +
  geom_polygon(fill = "white", colour = "black") +
  coord_quickmap()
```



ggplot: one more thing - world

```
world <- map_data("world")
ggplot(world, aes(long, lat, group = group)) +
  geom_polygon(fill = "white", colour = "black") +
  coord_quickmap()
```



Section 2

Assignment

Assignment

- In an R Markdown document (output: html), produce 10 data insights.
  - Each with a figure and a data story.
  - You can re-use about existing examples, up to 5.

Section 3

Lecture 9: Shiny/3: Advanced

Review: Display output with `render*()` functions

- `render*()` arguments are code used to build and rebuild object
- `render*()` function re-runs the code with every change in the input

Review: render/output pair

- Static table from df, mat, etc.s
  - renderTable()/tableOutput()
- Interactive table from data frame, matrix or other table-like structure
  - renderDataTable()/dataTableOutput()
- Plot
  - renderPlot()/plotOutput()
- Get continous output
  - renderPrint()/verbatimTextOutput() or textOutput()
- Get last result
  - renderText()verbatimTextOutput() or textOutput()
- Customized UI elements
  - uiOutput()/renderUI()

Review: `render*`

- Allow binding of one output to multiple inputs

```
output$hist <- renderPlot({
  hist(data())
})

output$stat <- renderPlot({
  summary(data())
})
```

Review: observeEvent

- Allow binding of multiple outputs to multiple inputs.
- Use of `isolate` to *peek* the value not to react to its change every time.

```
actionButton(inputId = "go", label = "Click me")

observeEvent(input$go, {
  # Use of isolate to *peek* the value not to react to it.
  num_input <- isolate(input$num_input)

  output$plot1 <- renderPlot({
    # if we use input$num_input here, we build a direct reacti
    # between output$plot1 and input$num_input. This is not wh
    plot(1:number_input, runif(num_input))
  })

  output$table1 <- renderTable({ ... })
})
```

Review: When code gets to run.

- `ui`: client. run once per user per session.
- `server`: run once per session
- code inside a reactive function runs with every input change.

Advanced Shiny

- Input: `update***Input` functions (You can update output with new content.)
- Dynamic UI to create inputs and outputs dynamically.
- Graphics output: gridExtra
- Table output: kableExtra

update***Input

- Update various input values
  - updateSelectionInput(...)
  - updateNumericInput(...)

```
# shiny-36-update.R

library(shiny)

ui <- fluidPage(
  numericInput("shock", "Shock", value = round(runif(1) * 1000), 0),
  actionButton("add", "Add"),
  checkboxGroupInput("scenarios", "Scenarios", choices = c(), select

  verbatimTextOutput("o1")
)

scenarios <- c(-100, -50, 0, 50, 100)

server <- function(input, output, session) {
  updateCheckboxGroupInput(session, "scenarios",
                           choices = scenarios,
```

`renderUI`

- Dynamicaly creation of UI (user interface) with input and outputs.
- Append new items to `tagList()`

Create dynamic output `tagList()`

```
# shiny-34-renderUI.R

library(shiny)

ui <- fluidPage(
  uiOutput("p1")
)

server <- function(input, output, session) {
  output$p1 <- renderUI({
    uiopt <- tagList()

    for (i in 1:3) {
      uiopt <- tagList(
        uiopt,
        h1(paste0("HTML t", i)),
        h1(paste0("Plot p", i)),
        plotOutput(paste0("plot", i)))
    }
```

Create dynamic output 2

You can use newly created UI immediately

```
# shiny-35-renderUI-2.R

library(shiny)

ui <- fluidPage(
  uiOutput("p1")
)

server <- function(input, output, session) {
  output$p1 <- renderUI({
    tl <- tagList(
      h1("HTML t1"),
      uiOutput("t1"),
      h1("Plot p1p1"),
      plotOutput("p1p1")
    )

    tl
  })
```

Create dynamic output 3

```
# shiny-32-renderUI.R

library(shiny)
library(knitr)
library(kableExtra)

ui <- fluidPage(
  numericInput("num", "Num", 3),
  uiOutput("p1"),
  hr(),
  tableOutput("p2")
)

server <- function(input, output, session) {
  observe({
    row_num <- input$num

    output$p1 <- renderUI({
      tagList(
        tags$h1("This is a header"),
```

Dynamic input and update***Input

- Update various input values
  - updateSelectionInput(...)
  - updateNumericInput(...)

```
# shiny-36-update.R

library(shiny)

ui <- fluidPage(
  numericInput("shock", "Shock", value = round(runif(1) * 1000), 0),
  actionButton("add", "Add"),
  checkboxGroupInput("scenarios", "Scenarios", choices = c(), select

  verbatimTextOutput("o1")
)

scenarios <- c(-100, -50, 0, 50, 100)

server <- function(input, output, session) {
  updateCheckboxGroupInput(session, "scenarios",
                           choices = scenarios,
```

Very dynamic

```
# shiny-37-createDynamic.R

library(shiny)

ui <- fluidPage(
  uiOutput("p1"),
  verbatimTextOutput("o1")
)

scenarios <- c(-100, -50, 0, 50, 100)

server <- function(input, output, session) {
  baseList <- tagList(
    numericInput("shock", "Shock", value = round(runif(1) * 1000), (
    actionButton("add", "Add")
  )

  gen_ui <- function(scenarios, values = NA) {
    output$p1 <- renderUI({
      tl <<- baseList
```

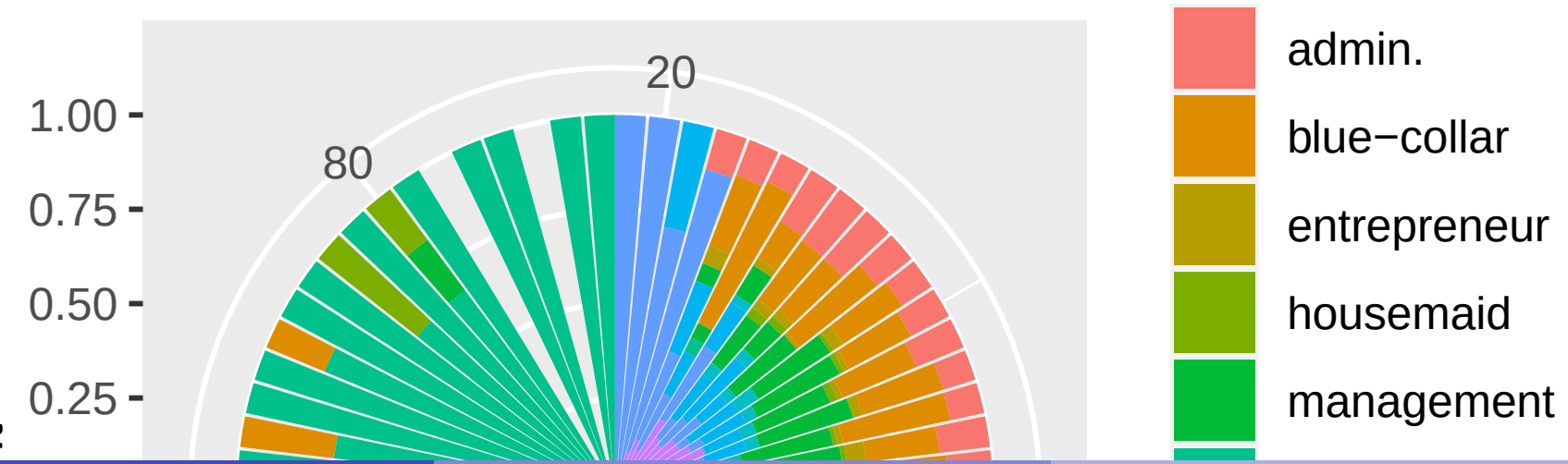
ggplot/gridExtra

If we need to generate multiple plots. ggplot has a companion package to arrange plots.

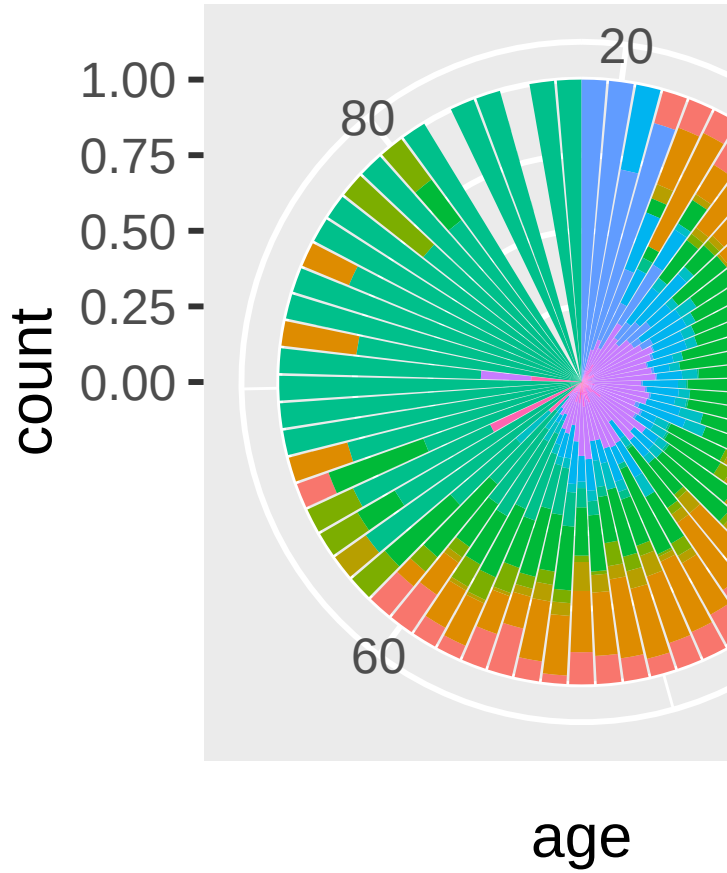
SxS: side by side

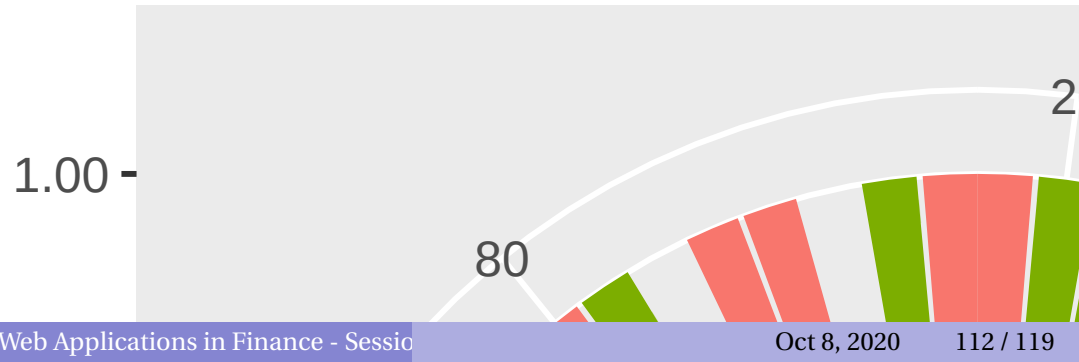
```
library(gridExtra)
p1 <- ggplot(bank) + geom_bar(mapping = aes(x = age, fill = job),
                              position = "fill") + coord_polar()
p2 <- ggplot(bank) + geom_bar(mapping = aes(x = age, fill = educatio
                              position = "fill") + coord_polar()
```

```
grid.arrange(p1, p2, ncol=2, nrow=1)
```



```
grid.arrange(p1, p2, ncol=1, nrow=2, heights = c(2,4))
```

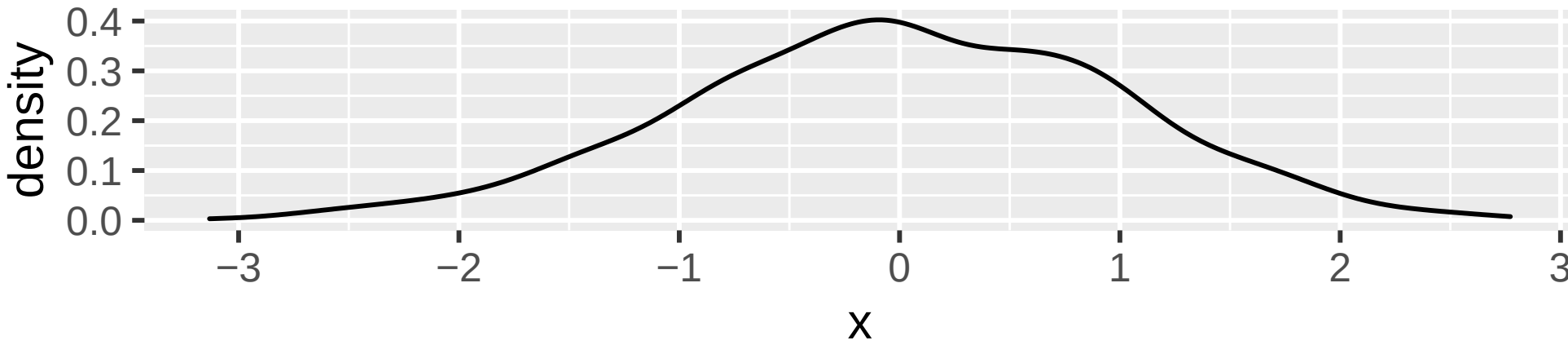


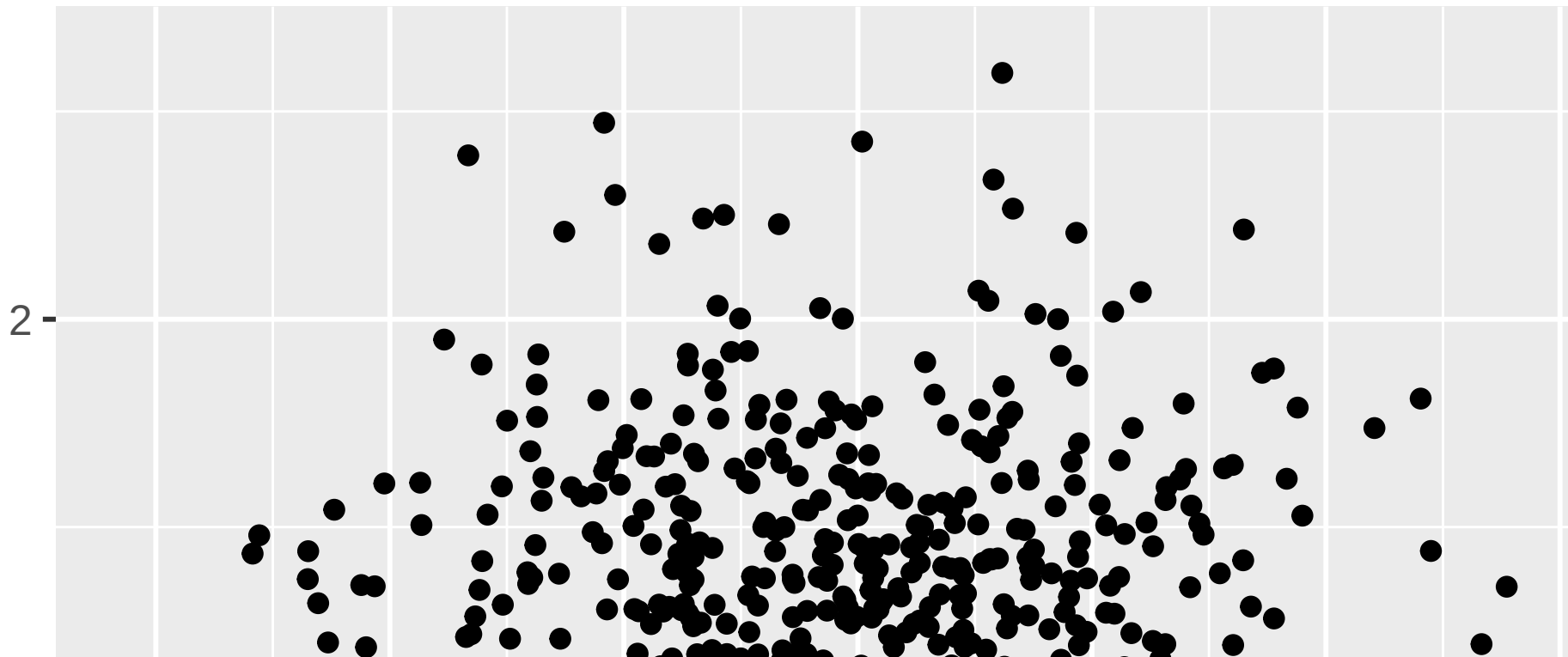


ggplot/gridExtra

```
library(tibble)
library(ggplot2)
library(gridExtra)

df <- tibble(x = rnorm(1000), y = rnorm(1000))

hist_top <- ggplot(df, aes(x = x)) + geom_density()

empty <-
  ggplot()+geom_point(aes(1,1), colour="white")+
  theme(axis.ticks=element_blank(),
        panel.background=element_blank(),
        axis.text.x=element_blank(), axis.text.y=element_blank(),
        axis.title.x=element_blank(), axis.title.y=element_blank())

scatter <- ggplot(df, aes(x = x, y = y)) + geom_point()

hist_right <- ggplot(df, aes(x = y)) + geom_density() + coord_flip()

grid.arrange(hist_top, empty, scatter, hist_right,
```

ggplot/gridExtra: result





knitr/kableExtra

kable is provided by `knitr` package. `kableExtra` enhance it with more functions. So we load both packages.

````
```{r shiny_block}
library(knitr)
library(kableExtra)

# This is HTML output
kable(df, format = "html")

# Use function() { } to output html
output$p1 <- function() {
  kable(df, format = "html")
}
```
````

kable_styling

- Get all styles from here https://cran.r-project.org/web/packages/kableExtra/vignettes/awesome_table_in_html.html
- style

```
mtcars[1:10, , drop = F] %>%
kable("html") %>%
  kable_styling(bootstrap_options = c("striped", "hover", "condensed
                font_size = 12,
                full_width = F, # True for left-to-right width
                position = "left") # if full_width == F
```

mpg

cyl

disp

hp

drat

wt

kable_styling: column_spec

```
mtcars[1:10, , drop = F] %>%
kable("html") %>%
  kable_styling(bootstrap_options = c("striped", "hover", "condensed
                font_size = 12,
                full_width = F, # True for left-to-right width
                position = "left") %>% # if full_width == FALSE
    column_spec(1, bold = T, border_right = TRUE) %>%
    column_spec(2, width = "30em", background = "yellow")
```

mpg	
cyl	
disp	
hp	
drat	
wt	
qsec	
vs	

kable_styling: row_spec

```
mtcars[1:10, , drop = F] %>%
kable("html") %>%
  kable_styling(bootstrap_options = c("striped", "hover", "condensed
                font_size = 12,
                full_width = F, # True for left-to-right width
                position = "left") %>% # if full_width == F
    column_spec(5:7, bold = TRUE) %>%
    row_spec(3:5, bold = T, color = "white", background = "#D7261E")
```

mpg

cyl

disp

hp

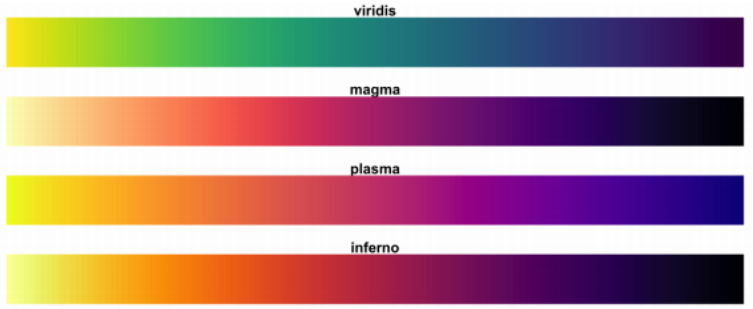
drat

wt

qsec

vs

kable_styling: cell_spec



tenor

0.1

0.25

0.5

0.75

0.9

1M

0.472

0.431