

# Data types

**Data Science in a Box**  
[datasciencebox.org](http://datasciencebox.org)

Modified by Tyler George



# Why should you care about data types?



# Example: Cat lovers

A survey asked respondents their name and number of cats. The instructions said to enter the number of cats as a numerical value.

```
cat_lovers <- read_csv("data/cat-lovers.csv")
```

```
## # A tibble: 60 x 3
##   name      number_of_cats handedness
##   <chr>        <chr>       <chr>
## 1 Bernice Warren 0           left
## 2 Woodrow Stone 0           left
## 3 Willie Bass  1           left
## 4 Tyrone Estrada 3          left
## 5 Alex Daniels 3           left
## 6 Jane Bates  2           left
## # ... with 54 more rows
```



# Oh why won't you work?!

```
cat_lovers %>%  
  summarise(mean_cats = mean(number_of_cats))
```

```
## Warning in mean.default(number_of_cats): argument is not numeric  
## or logical: returning NA  
  
## # A tibble: 1 x 1  
##   mean_cats  
##       <dbl>  
## 1       NA
```



mean {base}

R Documentation

# Arithmetic Mean

## Description

Generic function for the (trimmed) arithmetic mean.

## Usage

```
mean(x, ...)

## Default S3 method:
mean(x, trim = 0, na.rm = FALSE, ...)
```

## Arguments

- x An R object. Currently there are methods for numeric/logical vectors and [date](#), [date-time](#) and [time interval](#) objects. Complex vectors are allowed for trim = 0, only.
- trim the fraction (0 to 0.5) of observations to be trimmed from each end of x before the mean is computed. Values of trim outside that range are taken as the nearest endpoint.
- na.rm a logical value indicating whether NA values should be stripped before the computation proceeds.
- ... further arguments passed to or from other methods.



# Oh why won't you still work??!!

```
cat_lovers %>%  
  summarise(mean_cats = mean(number_of_cats, na.rm = TRUE))
```

```
## Warning in mean.default(number_of_cats, na.rm = TRUE): argument  
## is not numeric or logical: returning NA  
  
## # A tibble: 1 x 1  
##   mean_cats  
##       <dbl>  
## 1       NA
```



# Take a breath and look at your data

What is the type of the number\_of\_cats variable?

```
glimpse(cat_lovers)
```

```
## Rows: 60
## Columns: 3
## $ name      <chr> "Bernice Warren", "Woodrow Stone", "Will~
## $ number_of_cats <chr> "0", "0", "1", "3", "3", "2", "1", "1", ~
## $ handedness <chr> "left", "left", "left", "left", "left", ~
```



# Let's take another look

Show 10 ▾ entries

Search:

	name	number_of_cats	handedness
1	Bernice Warren	0	left
2	Woodrow Stone	0	left
3	Willie Bass	1	left
4	Tyrone Estrada	3	left
5	Alex Daniels	3	left
6	Jane Bates	2	left
7	Latoya Simpson	1	left
8	Darin Woods	1	left
9	Agnes Cobb	0	left
10	Tabitha Grant	0	left

Showing 1 to 10 of 60 entries

Previous

1

2

3

4

5

6

Next



# Sometimes you might need to babysit your respondents

```
cat_lovers %>%  
  mutate(number_of_cats = case_when(  
    name == "Ginger Clark" ~ 2,  
    name == "Doug Bass" ~ 3,  
    TRUE ~ as.numeric(number_of_cats))  
) %>%  
  summarise(mean_cats = mean(number_of_cats))
```

```
## Warning in eval_tidy(pair$rhs, env = default_env): NAs introduced  
## by coercion
```

```
## # A tibble: 1 x 1  
##   mean_cats  
##       <dbl>  
## 1     0.833
```



# Always you need to respect data types

```
cat_lovers %>%  
  mutate(  
    number_of_cats = case_when(  
      name == "Ginger Clark" ~ "2",  
      name == "Doug Bass" ~ "3",  
      TRUE ~ number_of_cats  
    ),  
    number_of_cats = as.numeric(number_of_cats)  
  ) %>%  
  summarise(mean_cats = mean(number_of_cats))
```

```
## # A tibble: 1 x 1  
##   mean_cats  
##     <dbl>  
## 1     0.833
```



# Now that we know what we're doing...

```
cat_lovers <- cat_lovers %>%
  mutate(
    number_of_cats = case_when(
      name == "Ginger Clark" ~ "2",
      name == "Doug Bass"     ~ "3",
      TRUE                  ~ number_of_cats
    ),
    number_of_cats = as.numeric(number_of_cats)
  )
```



# Moral of the story

- If your data does not behave how you expect it to, type coercion upon reading in the data might be the reason.
- Go in and investigate your data, apply the fix, *save your data*, live happily ever after.



now that we have a good motivation for learning about data types in R

let's learn about data types in R!



# Data types



# Data types in R

- **logical**
- **double**
- **integer**
- **character**
- and some more, but we won't be focusing on those



# Logical & character

**logical** - boolean values TRUE and FALSE

```
typeof(TRUE)
```

```
## [1] "logical"
```

**character** - character strings

```
typeof("hello")
```

```
## [1] "character"
```



# Double & integer

**double** - floating point numerical values  
(default numerical type)

```
typeof(1.335)
```

```
## [1] "double"
```

```
typeof(7)
```

```
## [1] "double"
```

**integer** - integer numerical values  
(indicated with an L)

```
typeof(7L)
```

```
## [1] "integer"
```

```
typeof(1:3)
```

```
## [1] "integer"
```



# Concatenation

Vectors can be constructed using the `c()` function.

```
c(1, 2, 3)
```

```
## [1] 1 2 3
```

```
c("Hello", "World!")
```

```
## [1] "Hello"  "World!"
```

```
c(c("hi", "hello"), c("bye", "jello"))
```

```
## [1] "hi"     "hello"   "bye"    "jello"
```



# Converting between types

with intention...

```
x <- 1:3  
x
```

```
## [1] 1 2 3
```

```
typeof(x)
```

```
## [1] "integer"
```



# Converting between types

with intention...

```
x <- 1:3  
x
```

```
## [1] 1 2 3
```

```
typeof(x)
```

```
## [1] "integer"
```

```
y <- as.character(x)  
y
```

```
## [1] "1" "2" "3"
```

```
typeof(y)
```

```
## [1] "character"
```



# Converting between types

with intention...

```
x <- c(TRUE, FALSE)  
x
```

```
## [1] TRUE FALSE
```

```
typeof(x)
```

```
## [1] "logical"
```



# Converting between types

with intention...

```
x <- c(TRUE, FALSE)  
x
```

```
## [1] TRUE FALSE
```

```
typeof(x)
```

```
## [1] "logical"
```

```
y <- as.numeric(x)  
y
```

```
## [1] 1 0
```

```
typeof(y)
```

```
## [1] "double"
```



# Converting between types

without intention...

R will happily convert between various types without complaint when different types of data are concatenated in a vector, and that's not always a great thing!

```
c(1, "Hello")
```

```
## [1] "1"      "Hello"
```

```
c(FALSE, 3L)
```

```
## [1] 0 3
```



# Converting between types

without intention...

R will happily convert between various types without complaint when different types of data are concatenated in a vector, and that's not always a great thing!

```
c(1, "Hello")
```

```
## [1] "1"      "Hello"
```

```
c(1.2, 3L)
```

```
## [1] 1.2 3.0
```

```
c(FALSE, 3L)
```

```
## [1] 0 3
```

```
c(2L, "two")
```

```
## [1] "2"    "two"
```



# Explicit vs. implicit coercion

Let's give formal names to what we've seen so far:



# Explicit vs. implicit coercion

Let's give formal names to what we've seen so far:

- **Explicit coercion** is when you call a function like `as.logical()`, `as.numeric()`, `as.integer()`, `as.double()`, or `as.character()`



# Explicit vs. implicit coercion

Let's give formal names to what we've seen so far:

- **Explicit coercion** is when you call a function like `as.logical()`, `as.numeric()`, `as.integer()`, `as.double()`, or `as.character()`
- **Implicit coercion** happens when you use a vector in a specific context that expects a certain type of vector



# Your turn!

- RStudio > AE 05 - Hotels + Data types > open type-coercion.Rmd and knit.
- What is the type of the given vectors? First, guess. Then, try it out in R. If your guess was correct, great! If not, discuss why they have that type.



# Your turn!

- RStudio > AE 05 - Hotels + Data types > open type-coercion.Rmd and knit.
- What is the type of the given vectors? First, guess. Then, try it out in R. If your guess was correct, great! If not, discuss why they have that type.

**Example:** Suppose we want to know the type of `c(1, "a")`. First, I'd look at:

```
typeof(1)
```

```
## [1] "double"
```

and make a guess based on these. Then finally I'd check:

```
typeof(c(1, "a"))
```

```
## [1] "character"
```

```
typeof("a")
```

```
## [1] "character"
```



# Special values



# Special values

- NA: Not available
- NaN: Not a number
- Inf: Positive infinity
- -Inf: Negative infinity



# Special values

- NA: Not available
- NaN: Not a number
- Inf: Positive infinity
- -Inf: Negative infinity

```
pi / 0
```

```
## [1] Inf
```

```
0 / 0
```

```
## [1] NaN
```

```
1/0 - 1/0
```

```
## [1] NaN
```

```
1/0 + 1/0
```

```
## [1] Inf
```



# NAs are special s

```
x <- c(1, 2, 3, 4, NA)
```

```
mean(x)
```

```
## [1] NA
```

```
mean(x, na.rm = TRUE)
```

```
## [1] 2.5
```

```
summary(x)
```

```
##    Min. 1st Qu. Median    Mean 3rd Qu.    Max.    NA's
##    1.00    1.75    2.50    2.50    3.25    4.00     1
```

# NAs are logical

R uses NA to represent missing values in its data structures.

```
typeof(NA)
```

```
## [1] "logical"
```



# Mental model for NAs

- Unlike NaN, NAs are genuinely unknown values
- But that doesn't mean they can't function in a logical way
- Let's think about why NAs are logical...



# Mental model for NAs

- Unlike NaN, NAs are genuinely unknown values
- But that doesn't mean they can't function in a logical way
- Let's think about why NAs are logical...

Why do the following give different answers?

```
# TRUE or NA  
TRUE | NA
```

```
## [1] TRUE
```

```
# FALSE or NA  
FALSE | NA
```

```
## [1] NA
```

→ See next slide for answers...



- NA is unknown, so it could be TRUE or FALSE

- TRUE | NA

```
TRUE | TRUE # if NA was TRUE
```

```
## [1] TRUE
```

```
TRUE | FALSE # if NA was FALSE
```

```
## [1] TRUE
```

- FALSE | NA

```
FALSE | TRUE # if NA was TRUE
```

```
## [1] TRUE
```

```
FALSE | FALSE # if NA was FALSE
```

```
## [1] FALSE
```

- Doesn't make sense for mathematical operations
- Makes sense in the context of missing data



# Data classes



# Data classes

We talked about *types* so far, next we'll introduce the concept of *classes*

- Vectors are like Lego building blocks



# Data classes

We talked about *types* so far, next we'll introduce the concept of *classes*

- Vectors are like Lego building blocks
- We stick them together to build more complicated constructs, e.g. *representations of data*



# Data classes

We talked about *types* so far, next we'll introduce the concept of *classes*

- Vectors are like Lego building blocks
- We stick them together to build more complicated constructs, e.g. *representations of data*
- The **class** attribute relates to the S3 class of an object which determines its behaviour
  - You don't need to worry about what S3 classes really mean, but you can read more about it [here](#) if you're curious



# Data classes

We talked about *types* so far, next we'll introduce the concept of *classes*

- Vectors are like Lego building blocks
- We stick them together to build more complicated constructs, e.g. *representations of data*
- The **class** attribute relates to the S3 class of an object which determines its behaviour
  - You don't need to worry about what S3 classes really mean, but you can read more about it [here](#) if you're curious
- Examples: factors, dates, and data frames



# Factors

R uses factors to handle categorical variables, variables that have a fixed and known set of possible values

```
x <- factor(c("BS", "MS", "PhD", "MS"))
x
```

```
## [1] BS  MS  PhD MS
## Levels: BS MS PhD
```



# Factors

R uses factors to handle categorical variables, variables that have a fixed and known set of possible values

```
x <- factor(c("BS", "MS", "PhD", "MS"))
x
```

```
## [1] BS  MS  PhD MS
## Levels: BS MS PhD
```

```
typeof(x)
```

```
## [1] "integer"
```

```
class(x)
```

```
## [1] "factor"
```



# More on factors

We can think of factors like character (level labels) and an integer (level numbers) glued together

```
glimpse(x)
```

```
## Factor w/ 3 levels "BS","MS","PhD": 1 2 3 2
```

```
as.integer(x)
```

```
## [1] 1 2 3 2
```



# Dates

```
y <- as.Date("2020-01-01")  
y
```

```
## [1] "2020-01-01"
```

```
typeof(y)
```

```
## [1] "double"
```

```
class(y)
```

```
## [1] "Date"
```



# More on dates

We can think of dates like an integer (the number of days since the origin, 1 Jan 1970) and an integer (the origin) glued together

```
as.integer(y)
```

```
## [1] 18262
```

```
as.integer(y) / 365 # roughly 50 yrs
```

```
## [1] 50.03288
```



# Data frames

We can think of data frames like vectors of equal length glued together

```
df <- data.frame(x = 1:2, y = 3:4)
```

```
##   x y  
## 1 1 3  
## 2 2 4
```

```
typeof(df)
```

```
## [1] "list"
```

```
class(df)
```

```
## [1] "data.frame"
```



# Lists

Lists are a generic vector container vectors of any type can go in them

```
l <- list(  
  x = 1:4,  
  y = c("hi", "hello", "jello"),  
  z = c(TRUE, FALSE)  
)  
l
```

```
## $x  
## [1] 1 2 3 4  
##  
## $y  
## [1] "hi"    "hello" "jello"  
##  
## $z  
## [1] TRUE FALSE
```



# Lists and data frames

- A data frame is a special list containing vectors of equal length
- When we use the `pull()` function, we extract a vector from the data frame

```
df
```

```
##   x y
## 1 1 3
## 2 2 4
```

```
df %>%
  pull(y)
```

```
## [1] 3 4
```



# Working with factors



# Read data in as character strings

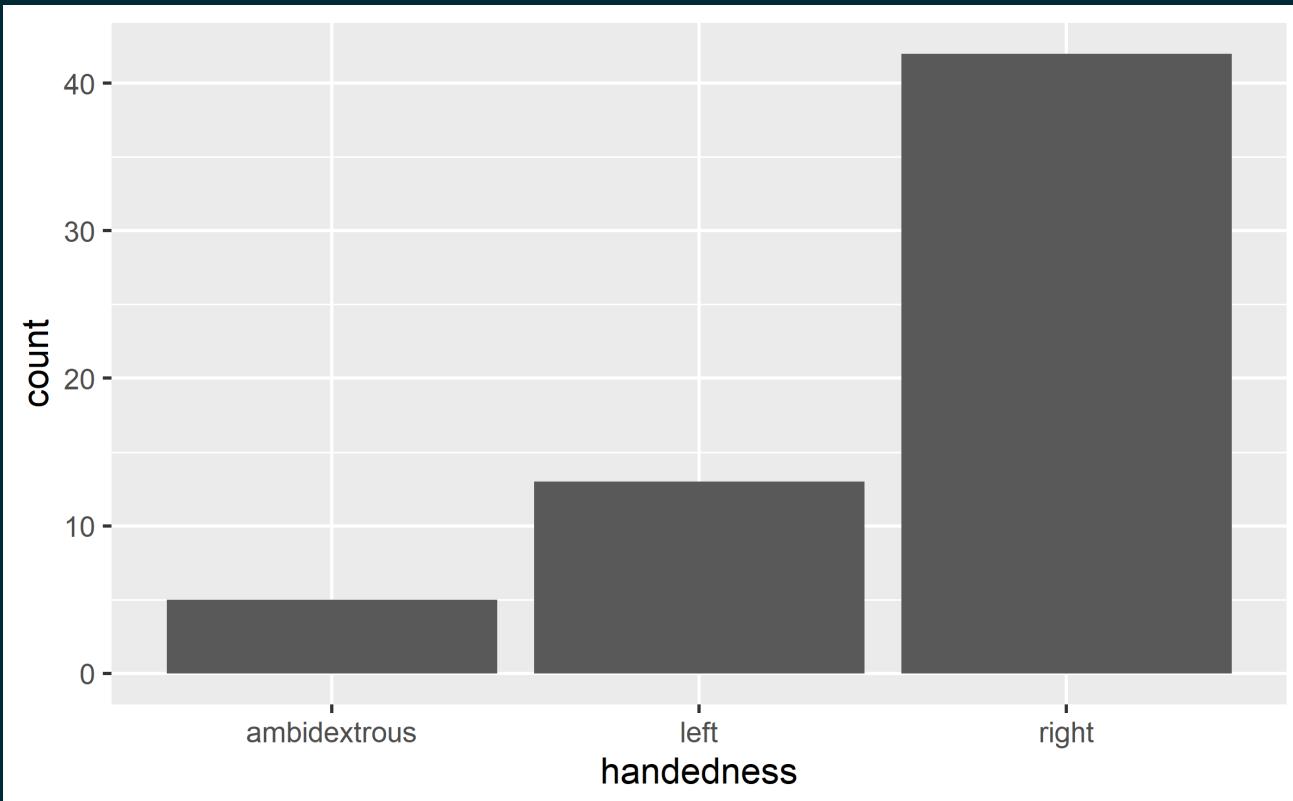
```
glimpse(cat_lovers)
```

```
## Rows: 60
## Columns: 3
## $ name      <chr> "Bernice Warren", "Woodrow Stone", "Will~
## $ number_of_cats <chr> "0", "0", "1", "3", "3", "2", "1", "1", ~
## $ handedness <chr> "left", "left", "left", "left", "left", ~
```



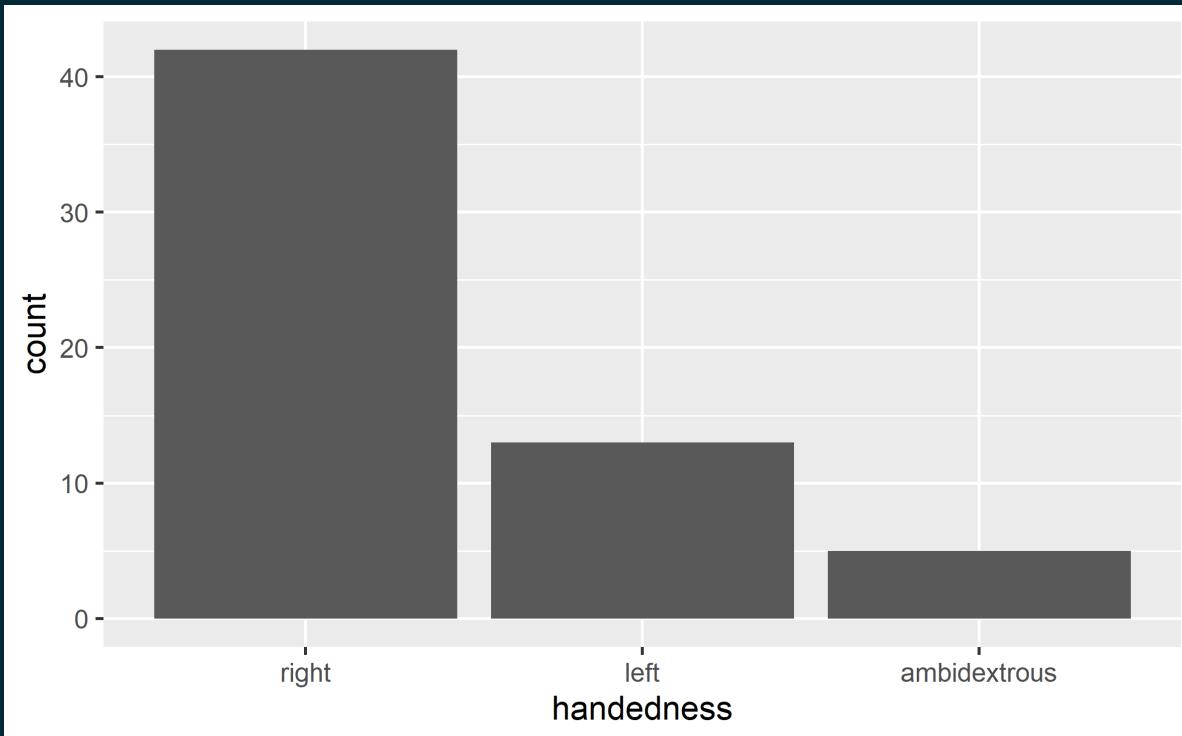
# But coerce when plotting

```
ggplot(cat_lovers, mapping = aes(x = handedness)) +  
  geom_bar()
```



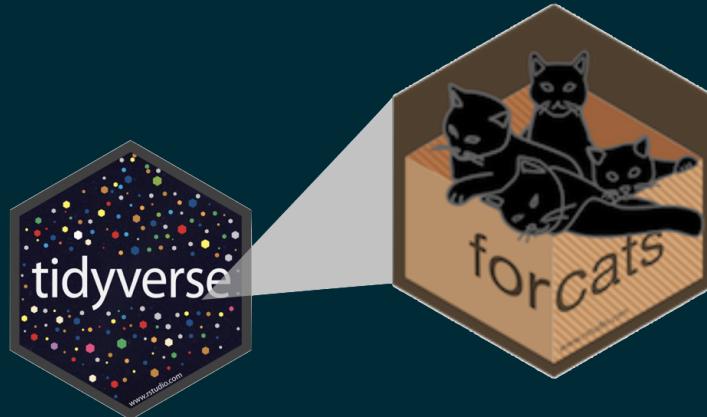
# Use forcats to manipulate factors

```
cat_lovers %>%  
  mutate(handedness = fct_infreq(handedness)) %>%  
  ggplot(mapping = aes(x = handedness)) +  
  geom_bar()
```



# Come for the functionality

... stay for the logo



- Factors are useful when you have true categorical data and you want to override the ordering of character vectors to improve display
- They are also useful in modeling scenarios
- The **forcats** package provides a suite of useful tools that solve common problems with factors

# Your turn!

- RStudio > AE 05 - Hotels + Data types > hotels-forcats.Rmd > knit
- Recreate the x-axis of the following plot.
- **Stretch goal:** Recreate the y-axis.



# Working with dates



# Make a date



- **lubridate** is the tidyverse-friendly package that makes dealing with dates a little easier
- It's not one of the *core* tidyverse packages, hence it's installed with `install.packages("tidyverse")` but it's not loaded with it, and needs to be explicitly loaded with `library(lubridate)`

we're just going to scratch the surface of working with dates in R here...



Calculate and visualise the number of bookings on any given arrival date.

```
hotels %>%  
  select(starts_with("arrival_"))  
  
## # A tibble: 119,390 x 4  
##   arrival_date_year arrival_date_month arrival_date_week_number  
##   <dbl> <chr>                      <dbl>  
## 1 2015 July                  27  
## 2 2015 July                  27  
## 3 2015 July                  27  
## 4 2015 July                  27  
## 5 2015 July                  27  
## 6 2015 July                  27  
## # ... with 119,384 more rows, and 1 more variable:  
## #   arrival_date_day_of_month <dbl>
```



# Step 1. Construct dates

```
library(glue)

hotels %>%
  mutate(
    arrival_date = glue("{arrival_date_year} {arrival_date_month} {arrival_date_day_of_month}")
  ) %>%
  relocate(arrival_date)
```

```
## # A tibble: 119,390 x 33
##   arrival_date hotel      is_canceled lead_time arrival_date_ye~
##   <glue>        <chr>          <dbl>       <dbl>           <dbl>
## 1 2015 July 1  Resort Hot~      0         342           2015
## 2 2015 July 1  Resort Hot~      0         737           2015
## 3 2015 July 1  Resort Hot~      0           7           2015
## 4 2015 July 1  Resort Hot~      0          13           2015
...
...
```



# Step 2. Count bookings per date

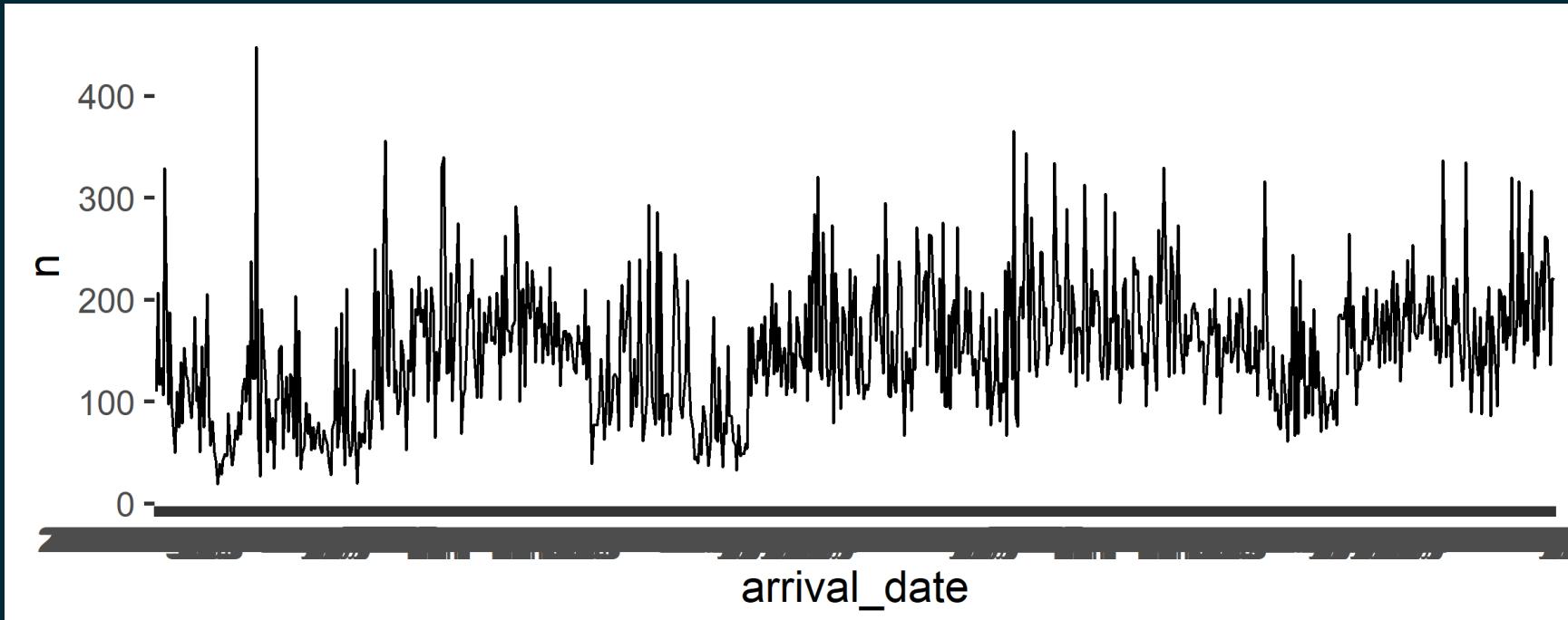
```
hotels %>%
  mutate(arrival_date = glue("{arrival_date_year} {arrival_date_month} {arrival_date_day_of_month}")
  count(arrival_date)
```

```
## # A tibble: 793 x 2
##   arrival_date     n
##   <glue>     <int>
## 1 2015 August 1    110
## 2 2015 August 10   207
## 3 2015 August 11   117
## 4 2015 August 12   133
## 5 2015 August 13   107
## 6 2015 August 14   329
## # ... with 787 more rows
```



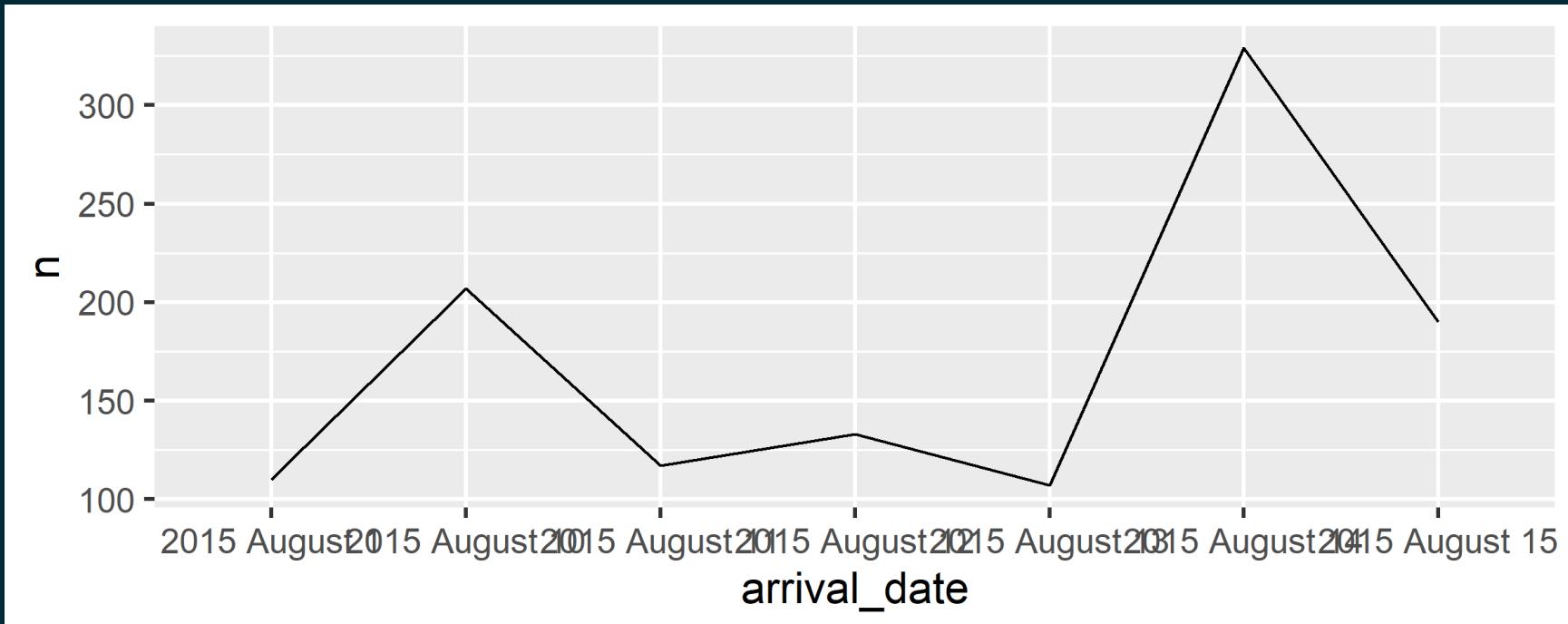
# Step 3. Visualise bookings per date

```
hotels %>%
  mutate(arrival_date = glue("{arrival_date_year} {arrival_date_month} {arrival_date_day_of_month}")
count(arrival_date) %>%
ggplot(aes(x = arrival_date, y = n, group = 1)) +
  geom_line()
```



zooming in a bit...

Why does the plot start with August when we know our data start in July? And why does 10 August come after 1 August?



# Step 1. *REVISED* Construct dates "as dates"

```
library(lubridate)

hotels %>%
  mutate(
    arrival_date = ymd(glue("{arrival_date_year} {arrival_date_month} {arrival_date_day_of_month}"))
  ) %>%
  relocate(arrival_date)

## # A tibble: 119,390 x 33
##   arrival_date hotel      is_canceled lead_time arrival_date_ye~
##   <date>        <chr>          <dbl>       <dbl>           <dbl>
## 1 2015-07-01  Resort Hot~      0          342            2015
## 2 2015-07-01  Resort Hot~      0          737            2015
## 3 2015-07-01  Resort Hot~      0             7            2015
## 4 2015-07-01  Resort Hot~      0            13            2015
## # ...
```



# Step 2. Count bookings per date

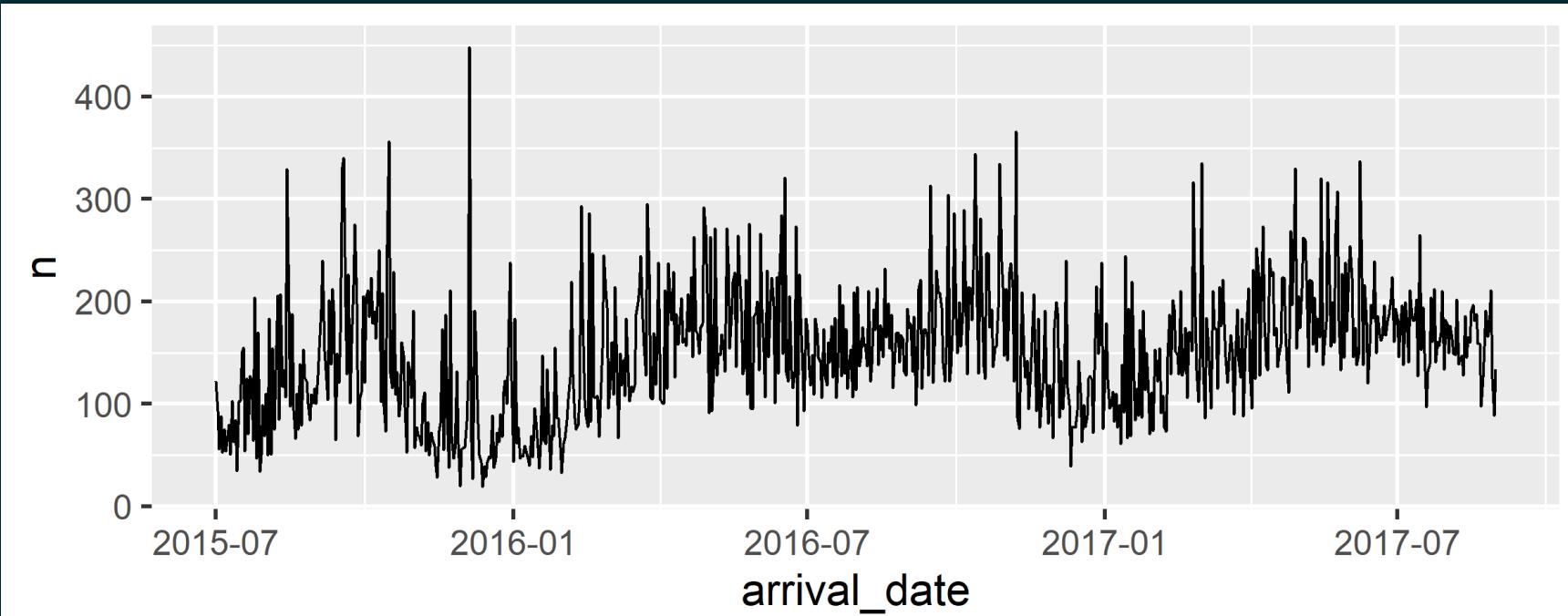
```
hotels %>%  
  mutate(arrival_date = ymd(glue("{arrival_date_year} {arrival_date_month} {arrival_date_day_of_r  
count(arrival_date)
```

```
## # A tibble: 793 x 2  
##   arrival_date     n  
##   <date>     <int>  
## 1 2015-07-01    122  
## 2 2015-07-02     93  
## 3 2015-07-03     56  
## 4 2015-07-04     88  
## 5 2015-07-05     53  
## 6 2015-07-06     75  
## # ... with 787 more rows
```



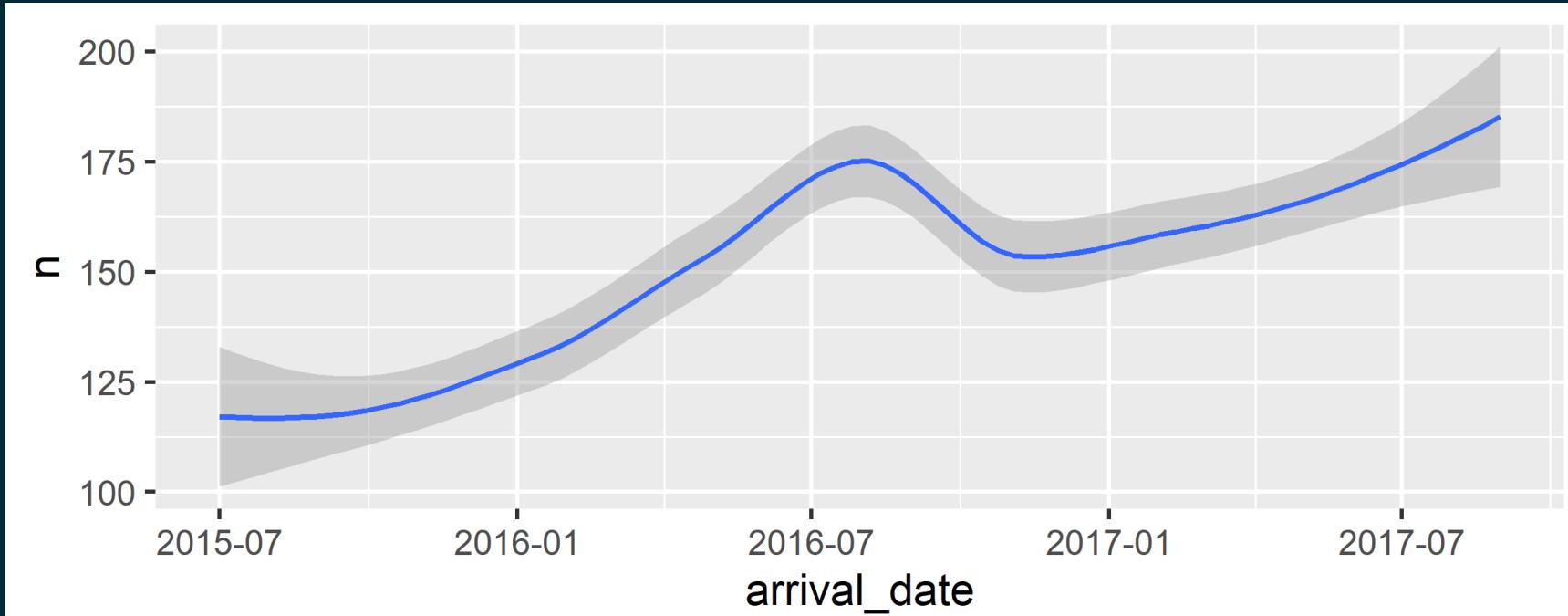
# Step 3a. Visualise bookings per date

```
hotels %>%
  mutate(arrival_date = ymd(glue("{arrival_date_year} {arrival_date_month} {arrival_date_day_of_r
count(arrival_date) %>%
ggplot(aes(x = arrival_date, y = n, group = 1)) +
geom_line()
```



# Step 3b. Visualise using a smooth curve

```
hotels %>%  
  mutate(arrival_date = ymd(glue("{arrival_date_year} {arrival_date_month} {arrival_date_day_of_r  
count(arrival_date) %>%  
  ggplot(aes(x = arrival_date, y = n, group = 1)) +  
  geom_smooth()
```



# Reading rectangular data into R





# readr

- `read_csv()` - comma delimited files
- `read_csv2()` - semicolon separated files (common in countries where , is used as the decimal place)
- `read_tsv()` - tab delimited files
- `read_delim()` - reads in files with any delimiter
- `read_fwf()` - fixed width files
- ...



# readr

- `read_csv()` - comma delimited files
- `read_csv2()` - semicolon separated files (common in countries where , is used as the decimal place)
- `read_tsv()` - tab delimited files
- `read_delim()` - reads in files with any delimiter
- `read_fwf()` - fixed width files
- ...

# readxl

- `read_excel()` - read xls orxlsx files
- ...



# Reading data

```
nobel <- read_csv(file = "data/nobel.csv")  
nobel
```

```
## # A tibble: 935 x 26  
##   id   firstname    surname    year category affiliation city  
##   <dbl> <chr>       <chr>     <dbl> <chr>      <chr>     <chr>  
## 1 1   Wilhelm Conrad Röntgen 1901 Physics  Munich Uni~ Muni~  
## 2 2   Hendrik A. Lorentz   1902 Physics  Leiden Uni~ Leid~  
## 3 3   Pieter Zeeman     1902 Physics  Amsterdam ~ Amst~  
## 4 4   Henri Becquerel  1903 Physics  École Poly~ Paris  
## 5 5   Pierre Curie     1903 Physics  École muni~ Paris  
## 6 6   Marie Curie     1903 Physics  <NA>       <NA>  
## # ... with 929 more rows, and 19 more variables: country <chr>,  
## #   born_date <date>, died_date <date>, gender <chr>,  
## #   born_city <chr>, born_country <chr>,  
## #   born_country_code <chr>, died_city <chr>,  
## #   died_country <chr>, died_country_code <chr>,  
## #   overall_motivation <chr>, share <dbl>, motivation <chr>,  
## #   born_country_original <chr>, born_city_original <chr>, ...
```



# Writing data

- Write a file

```
df <- tribble(  
  ~x, ~y,  
  1, "a",  
  2, "b",  
  3, "c"  
)  
  
write_csv(df, file = "data/df.csv")
```



# Writing data

- Write a file

```
df <- tribble(  
  ~x, ~y,  
  1, "a",  
  2, "b",  
  3, "c"  
)  
  
write_csv(df, file = "data/df.csv")
```

- Read it back in to inspect

```
read_csv("data/df.csv")  
  
## # A tibble: 3 x 2  
##       x     y  
##   <dbl> <chr>  
## 1     1     a  
## 2     2     b  
## 3     3     c
```



# Your turn!

- RStudio > AE 06 - Nobels and sales + Data import > open nobels-csv.Rmd and knit.
- Read in the nobels.csv file from the data-raw/ folder.
- Split into two (STEM and non-STEM):
  - Create a new data frame, nobel\_stem, that filters for the STEM fields (Physics, Medicine, Chemistry, and Economics).
  - Create another data frame, nobel\_nonstem, that filters for the remaining fields.
- Write out the two data frames to nobel-stem.csv and nobel-nonstem.csv, respectively, to data/.

**Hint:** Use the %in% operator when filter()ing.



# Variable names



# Data with bad names

```
edibnb_badnames <- read_csv("data/edibnb-badnames.csv")
names(edibnb_badnames)
```

```
## [1] "ID"                  "Price"
## [3] "neighbourhood"       "accommodates"
## [5] "Number of bathrooms" "Number of Bedrooms"
## [7] "n beds"               "Review Scores Rating"
## [9] "Number of reviews"   "listing_url"
```



# Data with bad names

```
edibnb_badnames <- read_csv("data/edibnb-badnames.csv")
names(edibnb_badnames)
```

```
## [1] "ID"                  "Price"
## [3] "neighbourhood"       "accommodates"
## [5] "Number of bathrooms" "Number of Bedrooms"
## [7] "n beds"               "Review Scores Rating"
## [9] "Number of reviews"   "listing_url"
```

... but R doesn't allow spaces in variable names

```
ggplot(edibnb_badnames, aes(x = Number of bathrooms, y = Price)) +
  geom_point()
```

```
## Error: <text>:1:40: unexpected symbol
## 1: ggplot(edibnb_badnames, aes(x = Number of
##                                ^
```



# Option 1 - Define column names

```
edibnb_col_names <- read_csv("data/edibnb-badnames.csv",
                               col_names = c("id", "price",
                                             "neighbourhood", "accommodates",
                                             "bathroom", "bedroom",
                                             "bed", "review_scores_rating",
                                             "n_reviews", "url"))

names(edibnb_col_names)
```

```
## [1] "id"                  "price"
## [3] "neighbourhood"       "accommodates"
## [5] "bathroom"             "bedroom"
## [7] "bed"                  "review_scores_rating"
## [9] "n_reviews"            "url"
```



# Option 2 - Format text to snake\_case

```
edibnb_clean_names <- read_csv("data/edibnb-badnames.csv") %>%  
  janitor::clean_names()  
  
names(edibnb_clean_names)
```

```
## [1] "id"                  "price"  
## [3] "neighbourhood"      "accommodates"  
## [5] "number_of_bathrooms" "number_of_bedrooms"  
## [7] "n_beds"               "review_scores_rating"  
## [9] "number_of_reviews"    "listing_url"
```



# Variable types



## Which type is x? Why?

x	y	z
1	a	hi
NA	b	hello
3	Not applicable	9999
4	d	ola
5	e	hola
.	f	whatup
7	g	wassup
8	h	sup
9	i	

```
read_csv("data/df-na.csv")
```

```
## # A tibble: 9 x 3
##   x     y           z
##   <chr> <chr>      <chr>
## 1 1     a           hi
## 2 <NA>  b           hello
## 3 3     Not applicable 9999
## 4 4     d           ola
## 5 5     e           hola
## 6 .     f           whatup
## 7 7     g           wassup
## 8 8     h           sup
## 9 9     i           <NA>
```



# Option 1. Explicit NAs

```
read_csv("data/df-na.csv",
         na = c("", "NA", ".", "9999", "Not applicable"))
```

x	y	z
1	a	hi
NA	b	hello
3	Not applicable	9999
4	d	ola
5	e	hola
.	f	whatup
7	g	wassup
8	h	sup
9	i	

```
## # A tibble: 9 x 3
##       x     y     z
##   <dbl> <chr> <chr>
## 1     1     a    hi
## 2     NA    b   hello
## 3     3    <NA> <NA>
## 4     4     d    ola
## 5     5     e    hola
## 6     NA    f   whatup
## 7     7     g   wassup
## 8     8     h    sup
## 9     9     i    <NA>
```



# Option 2. Specify column types

```
read_csv("data/df-na.csv", col_types = list(col_double(),
                                             col_character(),
                                             col_character()))

## Warning: One or more parsing issues, see `problems()` for details

## # A tibble: 9 x 3
##       x     y         z
##   <dbl> <chr>    <chr>
## 1     1     a      hi
## 2     NA    b    hello
## 3     3 Not applicable 9999
## 4     4     d     ola
## 5     5     e    hola
## 6     NA    f   whatup
## 7     7     g   wassup
## 8     8     h      sup
## 9     9     i     <NA>
```



# Column types

type function	data type
col_character()	character
col_date()	date
col_datetime()	POSIXct (date-time)
col_double()	double (numeric)
col_factor()	factor
col_guess()	let readr guess (default)
col_integer()	integer
col_logical()	logical
col_number()	numbers mixed with non-number characters
col_numeric()	double or integer
col_skip()	do not read
col_time()	time



# Wondering where you remember these from?

```
read_csv("data/df-na.csv")  
  
##  
  
Rows: 9 Columns: 3  
## [36mi [39m Building [34mu2-Jumbo03-u2-10-11-12-13.rmd [39m into [32mu2-Jumbo03-u2-10 [39m...  
  
##  
  
## [36mi [39m Building [34mu2-Jumbo03-u2-10-11-12-13.rmd [39m into [32mu2-Jumbo03-u2-10 [39m...  
  
-- Column specification -----  
## [36mi [39m Building [34mu2-Jumbo03-u2-10-11-12-13.rmd [39m into [32mu2-Jumbo03-u2-10 [39m...  
  
Delimiter: ","  
## chr (3): x, y, z  
##  
## [36mi [39m Building [34mu2-Jumbo03-u2-10-11-12-13.rmd [39m into [32mu2-Jumbo03-u2-10 [39m...
```



# Case study: Favourite foods



# Favourite foods

Student ID	Full Name	favourite.food	mealPlan	AGE	SES
1	Sunil Huffmann	Strawberry yoghurt	Lunch only	4	High
2	Barclay Lynn	French fries	Lunch only	5	Middle
3	Jayendra Lyne	N/A	Breakfast and lunch	7	Low
4	Leon Rossini	Anchovies	Lunch only	99999	Middle
5	Chidiegwu Dunkel	Pizza	Breakfast and lunch	five	High



# Favourite foods

Student ID	Full Name	favourite.food	mealPlan	AGE	SES
1	Sunil Huffmann	Strawberry yoghurt	Lunch only	4	High
2	Barclay Lynn	French fries	Lunch only	5	Middle
3	Jayendra Lyne	N/A	Breakfast and lunch	7	Low
4	Leon Rossini	Anchovies	Lunch only	99999	Middle
5	Chidiegwu Dunkel	Pizza	Breakfast and lunch	five	High

```
fav_food <- read_excel("data/favourite-food.xlsx")
```

```
fav_food
```

```
## # A tibble: 5 x 6
##   `Student ID` `Full Name`  favourite.food  mealPlan  AGE    SES
##       <dbl> <chr>        <chr>          <chr>     <chr> <chr>
## 1           1 Sunil Huffm~ Strawberry yog~ Lunch on~ 4      High
## 2           2 Barclay Lynn French fries   Lunch on~ 5      Midd~
## 3           3 Jayendra Ly~ N/A            Breakfas~ 7      Low
## 4           4 Leon Rossini Anchovies   Lunch on~ 99999  Midd~
## 5           5 Chidiegwu D~ Pizza         Breakfas~ five   High
```

# Variable names

Student ID	Full Name	favourite.food	mealPlan	AGE	SES
1	Sunil Huffmann	Strawberry yoghurt	Lunch only	4	High
2	Barclay Lynn	French fries	Lunch only	5	Middle
3	Jayendra Lyne	N/A	Breakfast and lunch	7	Low
4	Leon Rossini	Anchovies	Lunch only	99999	Middle
5	Chidiegwu Dunkel	Pizza	Breakfast and lunch	five	High

# Variable names

Student ID	Full Name	favourite.food	mealPlan	AGE	SES
1	Sunil Huffmann	Strawberry yoghurt	Lunch only	4	High
2	Barclay Lynn	French fries	Lunch only	5	Middle
3	Jayendra Lyne	N/A	Breakfast and lunch	7	Low
4	Leon Rossini	Anchovies	Lunch only	99999	Middle
5	Chidiegwu Dunkel	Pizza	Breakfast and lunch	five	High

```
fav_food <- read_excel("data/favourite-food.xlsx") %>%  
  janitor::clean_names()
```

```
fav_food
```

```
## # A tibble: 5 x 6  
##   student_id full_name  favourite_food  meal_plan    age    ses  
##       <dbl> <chr>        <chr>        <chr>      <chr> <chr>  
## 1          1 Sunil Huff~ Strawberry yogh~ Lunch only    4     High  
## 2          2 Barclay Ly~ French fries    Lunch only    5     Midd~  
## 3          3 Jayendra L~ N/A           Breakfast ~ 7     Low  
## 4          4 Leon Rossi~ Anchovies    Lunch only 99999 Midd~  
## 5          5 Chidiegwu ~ Pizza         Breakfast ~ five  High
```



# Handling NAs

Student ID	Full Name	favourite.food	mealPlan	AGE	SES
1	Sunil Huffmann	Strawberry yoghurt	Lunch only	4	High
2	Barclay Lynn	French fries	Lunch only	5	Middle
3	Jayendra Lyne	N/A	Breakfast and lunch	7	Low
4	Leon Rossini	Anchovies	Lunch only	99999	Middle
5	Chidiegwu Dunkel	Pizza	Breakfast and lunch	five	High



# Handling NAs

Student ID	Full Name	favourite.food	mealPlan	AGE	SES
1	Sunil Huffmann	Strawberry yoghurt	Lunch only	4	High
2	Barclay Lynn	French fries	Lunch only	5	Middle
3	Jayendra Lyne	N/A	Breakfast and lunch	7	Low
4	Leon Rossini	Anchovies	Lunch only	99999	Middle
5	Chidiegwu Dunkel	Pizza	Breakfast and lunch	five	High

```
fav_food <- read_excel("data/favourite-food.xlsx",
                        na = c("N/A", "99999")) %>%
  janitor::clean_names()

fav_food
```

```
## # A tibble: 5 x 6
##   student_id full_name   favourite_food   meal_plan    age    ses
##       <dbl> <chr>        <chr>          <chr>      <chr> <chr>
## 1         1 Sunil Huff~ Strawberry yogh~ Lunch only    4     High
## 2         2 Barclay Ly~ French fries    Lunch only    5     Midd~
## 3         3 Jayendra L~ <NA>           Breakfast ~ 7     Low
## 4         4 Leon Rossi~ Anchovies    Lunch only <NA>   Midd~
## 5         5 Chidiegwu ~ Pizza         Breakfast ~ five  High
```



# Make age numeric

```
fav_food <- fav_food %>%
  mutate(
    age = if_else(age == "five", "5", age),
    age = as.numeric(age)
  )

glimpse(fav_food)
```

AGE	SES
4	High
5	Middle
7	Low
99999	Middle
five	High

```
## Rows: 5
## Columns: 6
## $ student_id      <dbl> 1, 2, 3, 4, 5
## $ full_name       <chr> "Sunil Huffmann", "Barclay Lynn", "Jayen~
## $ favourite_food   <chr> "Strawberry yoghurt", "French fries", NA~
## $ meal_plan        <chr> "Lunch only", "Lunch only", "Breakfast a~
## $ age              <dbl> 4, 5, 7, NA, 5
## $ ses              <chr> "High", "Middle", "Low", "Middle", "High"
```



# Socio-economic status

What order are the levels of ses listed in?

```
fav_food %>%  
  count(ses)
```

```
## # A tibble: 3 x 2  
##   ses      n  
##   <chr>  <int>  
## 1 High     2  
## 2 Low      1  
## 3 Middle   2
```

SES
4 High
5 Middle
7 Low
99 Middle
High

# Make ses factor

```
fav_food <- fav_food %>%  
  mutate(ses = fct_relevel(ses, "Low", "Middle", "High"))
```

```
fav_food %>%  
  count(ses)
```

```
## # A tibble: 3 x 2  
##   ses      n  
##   <fct>  <int>  
## 1 Low      1  
## 2 Middle   2  
## 3 High     2
```



# Putting it altogether

```
fav_food <- read_excel("data/favourite-food.xlsx", na = c("N/A", "99999")) %>%  
  janitor::clean_names() %>%  
  mutate(  
    age = if_else(age == "five", "5", age),  
    age = as.numeric(age),  
    ses = fct_relevel(ses, "Low", "Middle", "High")  
  )  
  
fav_food
```

```
## # A tibble: 5 x 6  
##   student_id full_name  favourite_food meal_plan      age ses  
##       <dbl> <chr>        <chr>        <chr>      <dbl> <fct>  
## 1          1 Sunil Huff~ Strawberry yogh~ Lunch only      4 High  
## 2          2 Barclay Ly~ French fries   Lunch only      5 Midd~  
## 3          3 Jayendra L~ <NA>           Breakfast ~      7 Low  
## 4          4 Leon Rossi~ Anchovies   Lunch only     NA Midd~  
## 5          5 Chidiegwu ~ Pizza        Breakfast ~      5 High
```



# Out and back in

```
write_csv(fav_food, file = "data/fav-food-clean.csv")  
fav_food_clean <- read_csv("data/fav-food-clean.csv")
```



## What happened to ses again?

```
fav_food_clean %>%  
  count(ses)
```

```
## # A tibble: 3 x 2  
##   ses      n  
##   <chr>  <int>  
## 1 High      2  
## 2 Low       1  
## 3 Middle    2
```



# **read\_rds() and write\_rds()**

- CSVs can be unreliable for saving interim results if there is specific variable type information you want to hold on to.
- An alternative is RDS files, you can read and write them with `read_rds()` and `write_rds()`, respectively.

```
read_rds(path)
write_rds(x, path)
```



# Out and back in, take 2

```
write_rds(fav_food, file = "data/fav-food-clean.rds")  
  
fav_food_clean <- read_rds("data/fav-food-clean.rds")  
  
fav_food_clean %>%  
  count(ses)
```

```
## # A tibble: 3 x 2  
##   ses      n  
##   <fct>  <int>  
## 1 Low      1  
## 2 Middle   2  
## 3 High     2
```



# Other types of data



# Other types of data

- **googlesheets4**: Google Sheets
- **haven**: SPSS, Stata, and SAS files
- **DBI**, along with a database specific backend (e.g. RMySQL, RSQLite, RPostgreSQL etc):  
allows you to run SQL queries against a database and return a data frame
- **jsonline**: JSON
- **xml2**: xml
- **rvest**: web scraping
- **httr**: web APIs
- **sparklyr**: data loaded into spark



# Your turn!

- RStudio > AE 06 - Nobels and sales + Data import > sales-excel.Rmd.
- Load the sales.xlsx file from the data-raw/ folder, using appropriate arguments for the read\_excel() function such that it looks like the output on the left.
- **Stretch goal:** Manipulate the sales data such that it looks like the output on the right.

```
## # A tibble: 9 x 2
##   id      n
##   <chr>  <chr>
## 1 Brand 1 n
## 2 1234   8
## 3 8721   2
## 4 1822   3
## 5 Brand 2 n
## 6 3333   1
## # ... with 3 more rows
```

```
## # A tibble: 7 x 3
##   brand    id      n
##   <chr>  <dbl> <dbl>
## 1 Brand 1 1234   8
## 2 Brand 1 8721   2
## 3 Brand 1 1822   3
## 4 Brand 2 3333   1
## 5 Brand 2 2156   3
## 6 Brand 2 3987   6
## # ... with 1 more row
```



# Case study: Religion and income



## Income distribution by religious group

% of adults who have a household income of...

Chart

Table

Share

Save Image

Religious tradition	Less than \$30,000	\$30,000-\$49,999	\$50,000-\$99,999	\$100,000 or more	Sample Size
<b>Buddhist</b>	36%	18%	32%	13%	233
<b>Catholic</b>	36%	19%	26%	19%	6,137
<b>Evangelical Protestant</b>	35%	22%	28%	14%	7,462
<b>Hindu</b>	17%	13%	34%	36%	172
<b>Historically Black Protestant</b>	53%	22%	17%	8%	1,704
<b>Jehovah's Witness</b>	48%	25%	22%	4%	208
<b>Jewish</b>	16%	15%	24%	44%	708
<b>Mainline Protestant</b>	29%	20%	28%	23%	5,208
<b>Mormon</b>	27%	20%	33%	20%	594
<b>Muslim</b>	34%	17%	29%	20%	205
<b>Orthodox Christian</b>	18%	17%	36%	29%	155
<b>Unaffiliated (religious "nones")</b>	33%	20%	26%	21%	6,790

Sample sizes and margins of error vary from subgroup to subgroup, from year to year and from state to state. You can see the sample size for the estimates in this chart on rollover or in the last column of the table. And visit [this table](#) to see approximate margins of error for a group of a given size. Readers should always bear in mind the approximate margin of error for the group they are examining when making comparisons with other groups or assessing the significance of trends over time. For full question wording, see the [survey questionnaire](#).

Source: [pewforum.org/religious-landscape-study/income-distribution](http://pewforum.org/religious-landscape-study/income-distribution), Retrieved 14 April, 2020

# Read data

```
library(readxl)
rel_inc <- read_excel("data/relig-income.xlsx")
```

```
## # A tibble: 12 x 6
##   `Religious tradition` `Less than $30,~` '$30,000-$49,99~` <dbl> <dbl>
## 1 Buddhist                0.36      0.18
## 2 Catholic                0.36      0.19
## 3 Evangelical Protestant   0.35      0.22
## 4 Hindu                   0.17      0.13
## 5 Historically Black Protestant  0.53      0.22
## 6 Jehovah's Witness        0.48      0.25
## # ... with 6 more rows, and 3 more variables:
## #   $50,000-$99,999 <dbl>, $100,000 or more <dbl>,
## #   Sample Size <dbl>
```



# Rename columns

```
rel_inc %>%
  rename(
    religion = `Religious tradition`,
    n = `Sample Size`
  )

## # A tibble: 12 x 6
##   religion      `Less than $30,~` `$30,000-$49,99~` `$50,000-$99,99~
##   <chr>          <dbl>            <dbl>            <dbl>
## 1 Buddhist       0.36             0.18             0.32
## 2 Catholic       0.36             0.19             0.26
## 3 Evangelical~  0.35             0.22             0.28
## 4 Hindu          0.17             0.13             0.34
## 5 Historically~ 0.53             0.22             0.17
## 6 Jehovah's W~  0.48             0.25             0.22
## # ... with 6 more rows, and 2 more variables:
## #   $100,000 or more <dbl>, n <dbl>
```



If we want a new variable called `income` with levels such as "Less than \$30,000", "\$30,000-\$49,999", ... etc. which function should we use?

```
## # A tibble: 48 x 4
##   religion      n income    proportion
##   <chr>     <dbl> <chr>        <dbl>
## 1 Buddhist     233 Less than $30,000 0.36
## 2 Buddhist     233 $30,000-$49,999 0.18
## 3 Buddhist     233 $50,000-$99,999 0.32
## 4 Buddhist     233 $100,000 or more 0.13
## 5 Catholic     6137 Less than $30,000 0.36
## 6 Catholic     6137 $30,000-$49,999 0.19
## 7 Catholic     6137 $50,000-$99,999 0.26
## 8 Catholic     6137 $100,000 or more 0.19
## 9 Evangelical Protestant 7462 Less than $30,000 0.35
## 10 Evangelical Protestant 7462 $30,000-$49,999 0.22
## 11 Evangelical Protestant 7462 $50,000-$99,999 0.28
## 12 Evangelical Protestant 7462 $100,000 or more 0.14
## 13 Hindu        172 Less than $30,000 0.17
## 14 Hindu        172 $30,000-$49,999 0.13
## 15 Hindu        172 $50,000-$99,999 0.34
## # ... with 33 more rows
```



# Pivot longer

```
rel_inc %>%
  rename(
    religion = `Religious tradition`,
    n = `Sample Size`
  ) %>%
  pivot_longer(
    cols = -c(religion, n),    # all but religion and n
    names_to = "income",
    values_to = "proportion"
  )
```

```
## # A tibble: 48 x 4
##   religion     n income           proportion
##   <chr>    <dbl> <chr>            <dbl>
## 1 Buddhist    233 Less than $30,000      0.36
## 2 Buddhist    233 $30,000-$49,999      0.18
## 3 Buddhist    233 $50,000-$99,999      0.32
## 4 Buddhist    233 $100,000 or more     0.13
## 5 Catholic    6137 Less than $30,000      0.36
## 6 Catholic    6137 $30,000-$49,999      0.19
## # ... with 42 more rows
```



# Calculate frequencies

```
rel_inc %>%
  rename(
    religion = `Religious tradition`,
    n = `Sample Size`
  ) %>%
  pivot_longer(
    cols = -c(religion, n),
    names_to = "income",
    values_to = "proportion"
  ) %>%
  mutate(frequency = round(proportion * n))
```

```
## # A tibble: 48 x 5
##   religion     n income           proportion frequency
##   <chr>     <dbl> <chr>             <dbl>      <dbl>
## 1 Buddhist    233 Less than $30,000    0.36       84
## 2 Buddhist    233 $30,000-$49,999    0.18       42
## 3 Buddhist    233 $50,000-$99,999    0.32       75
## 4 Buddhist    233 $100,000 or more   0.13       30
## 5 Catholic   6137 Less than $30,000    0.36      2209
## 6 Catholic   6137 $30,000-$49,999    0.19      1166
## # ... with 42 more rows
```



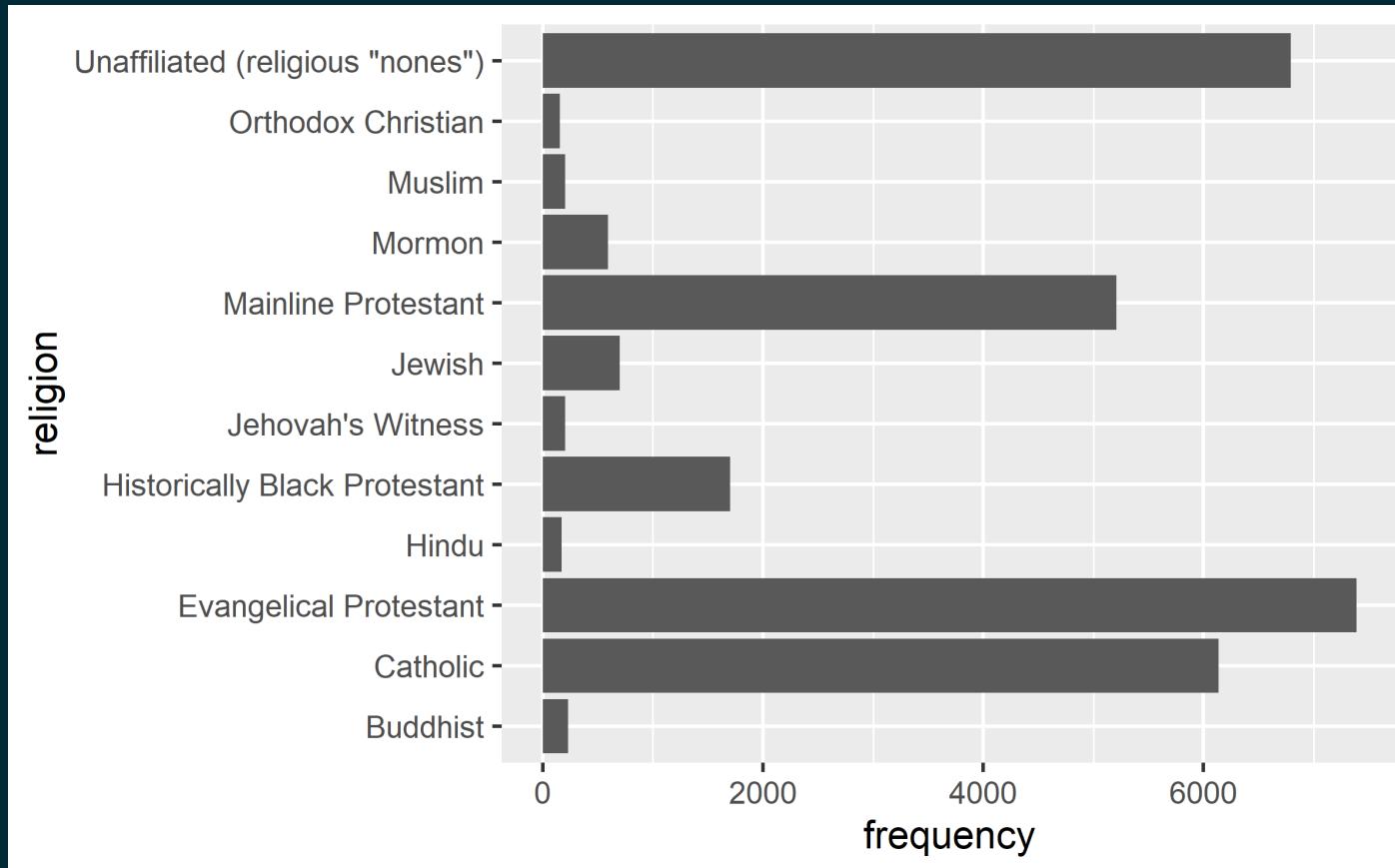
# Save data

```
rel_inc_long <- rel_inc %>%
  rename(
    religion = `Religious tradition`,
    n = `Sample Size`
  ) %>%
  pivot_longer(
    cols = -c(religion, n),
    names_to = "income",
    values_to = "proportion"
  ) %>%
  mutate(frequency = round(proportion * n))
```



# Barplot

```
ggplot(rel_inc_long, aes(y = religion, x = frequency)) +  
  geom_col()
```



# Recode religion

Recode Plot

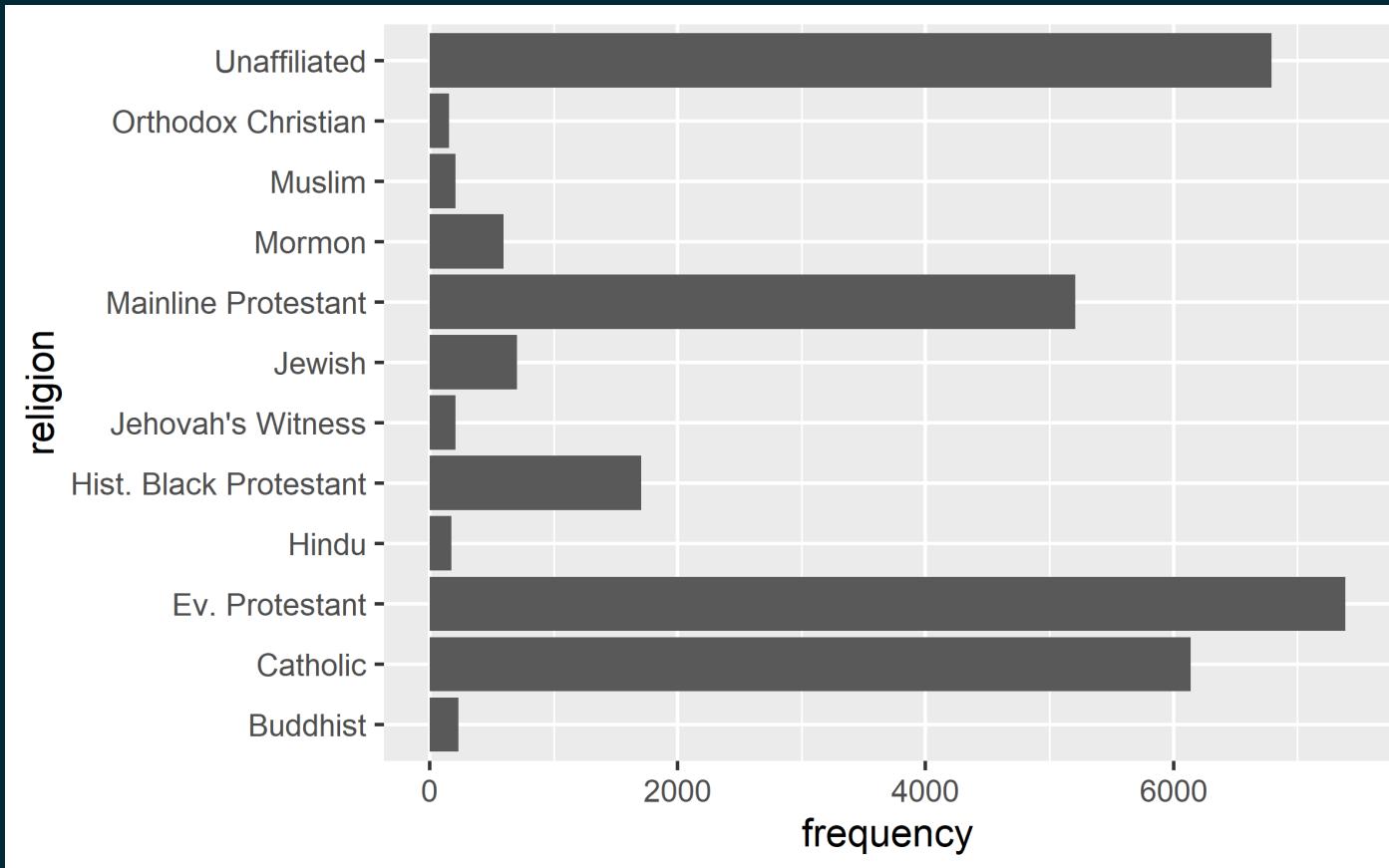
```
rel_inc_long <- rel_inc_long %>%  
  mutate(religion = case_when(  
    religion == "Evangelical Protestant"      ~ "Ev. Protestant",  
    religion == "Historically Black Protestant" ~ "Hist. Black Protestant",  
    religion == 'Unaffiliated (religious "nones")' ~ "Unaffiliated",  
    TRUE                                         ~ religion  
)
```



# Recode religion

Recode

Plot



# Reverse religion order

Recode      Plot

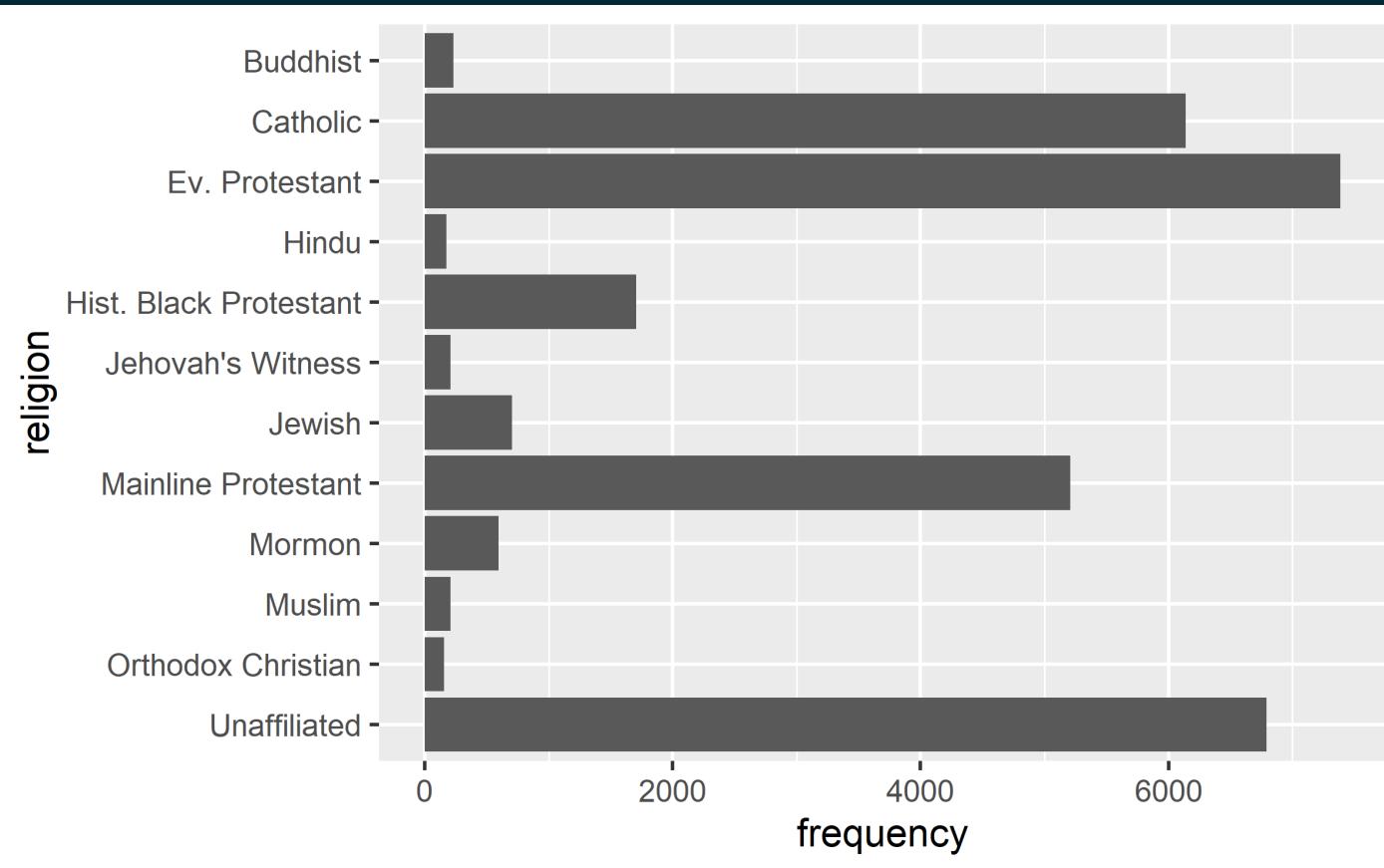
```
rel_inc_long <- rel_inc_long %>%  
  mutate(religion = fct_rev(religion))
```



# Reverse religion order

Recode

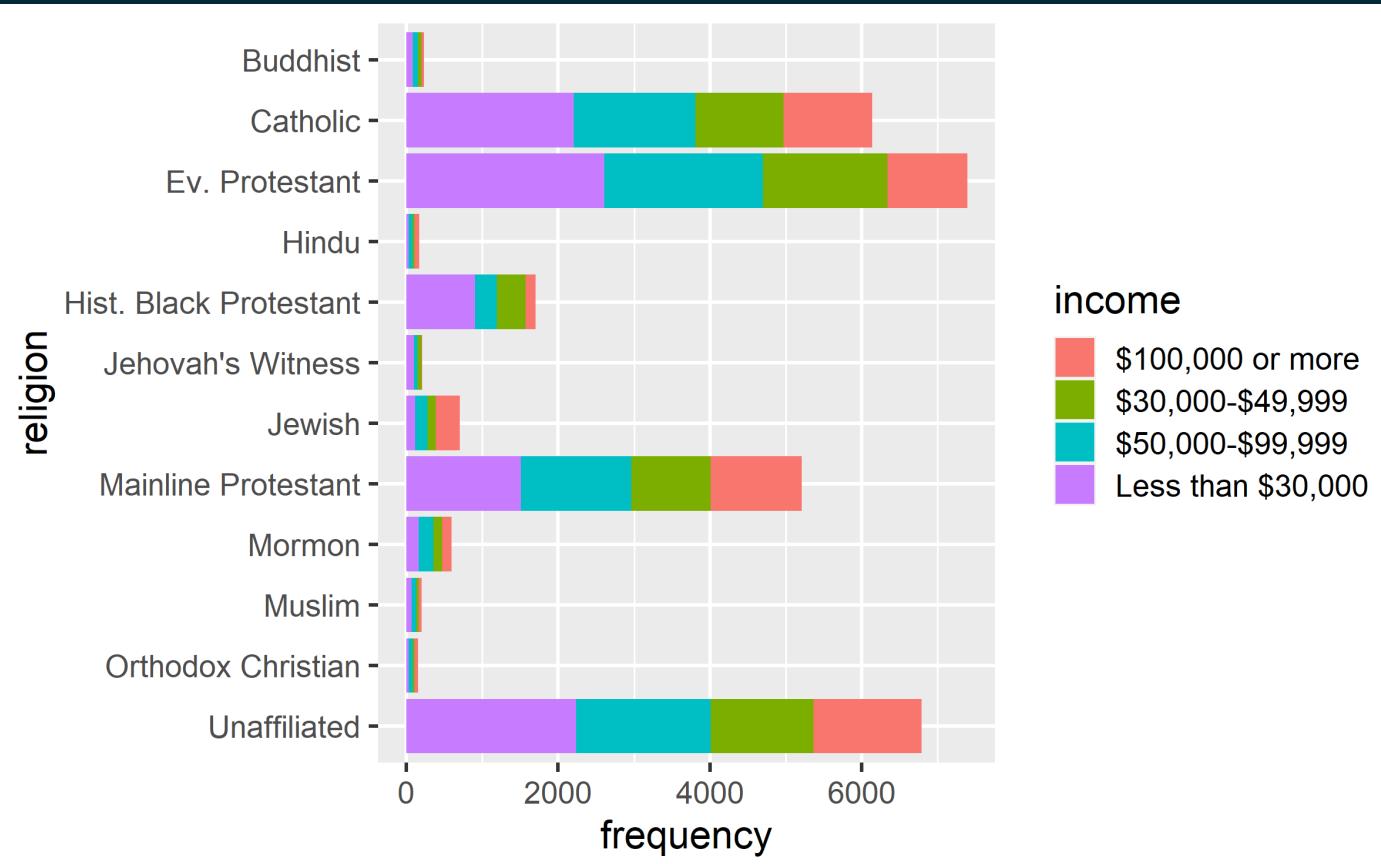
Plot



# Add income

Plot

Code



# Add income

---

Plot

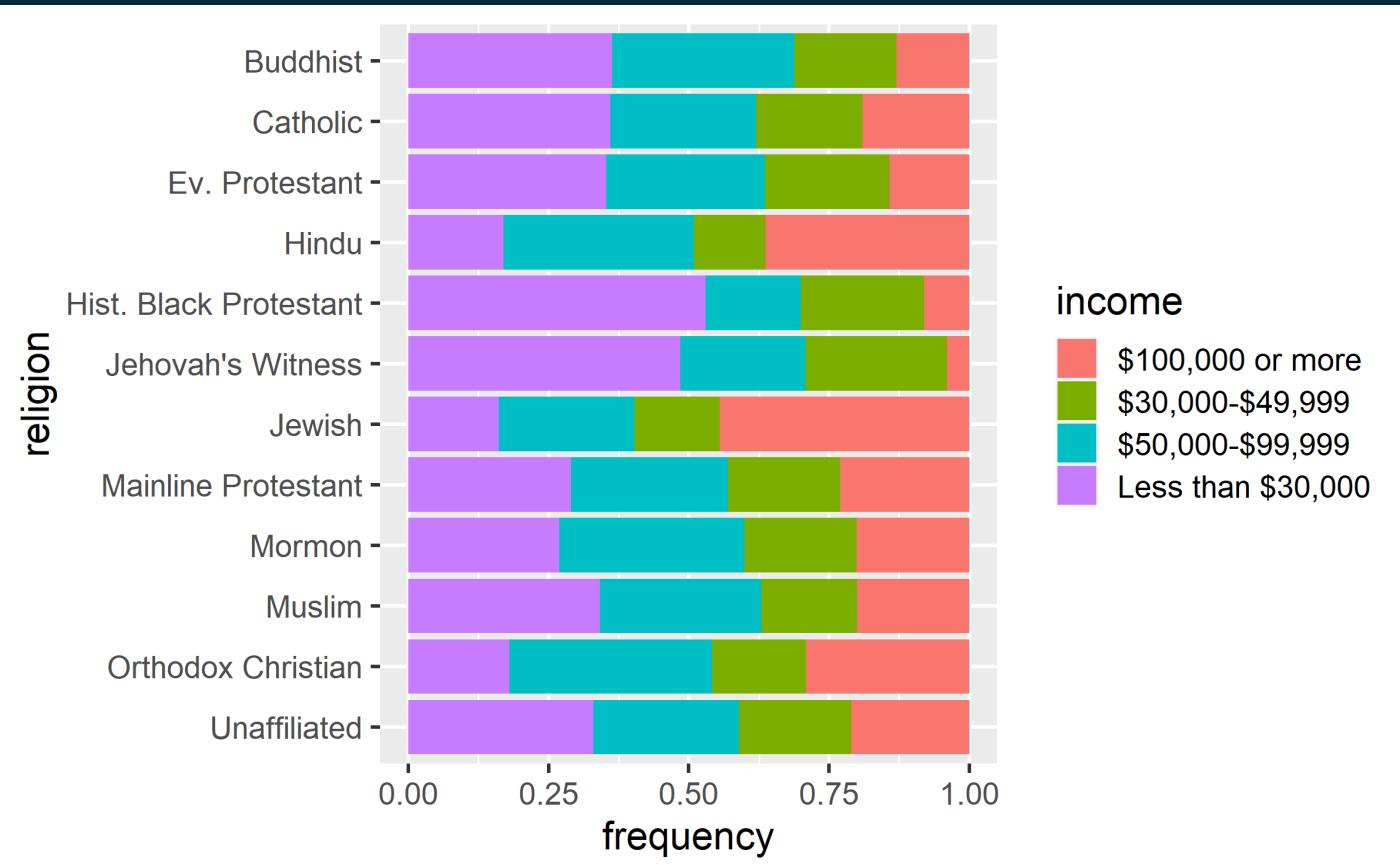
Code

```
ggplot(rel_inc_long, aes(y = religion, x = frequency, fill = income)) +  
  geom_col()
```



# Fill bars

Plot    Code



# Fill bars

---

Plot

Code

---

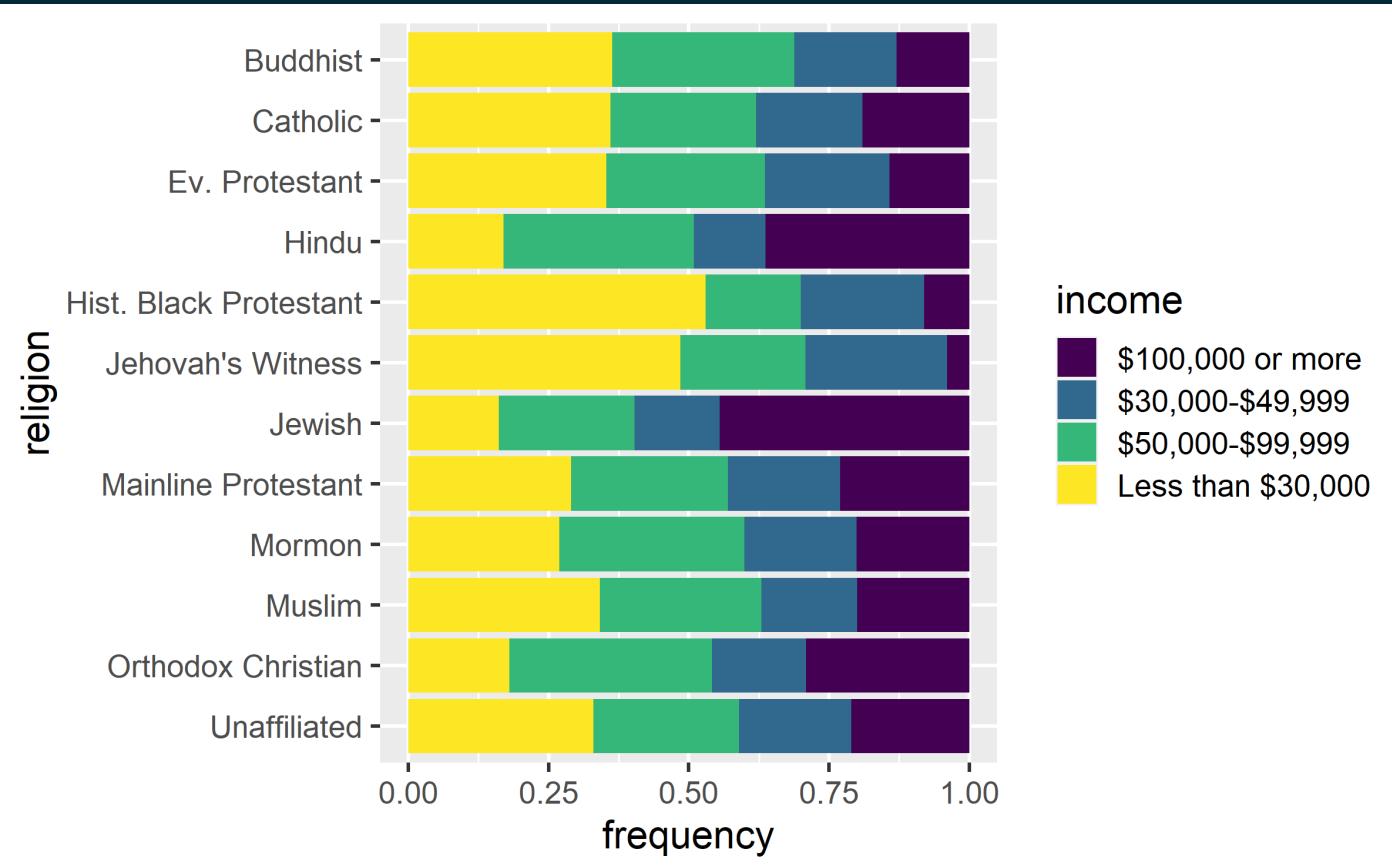
```
ggplot(rel_inc_long, aes(y = religion, x = frequency, fill = income)) +  
  geom_col(position = "fill")
```



# Change colors

Plot

Code



# Change colors

---

Plot

Code

---

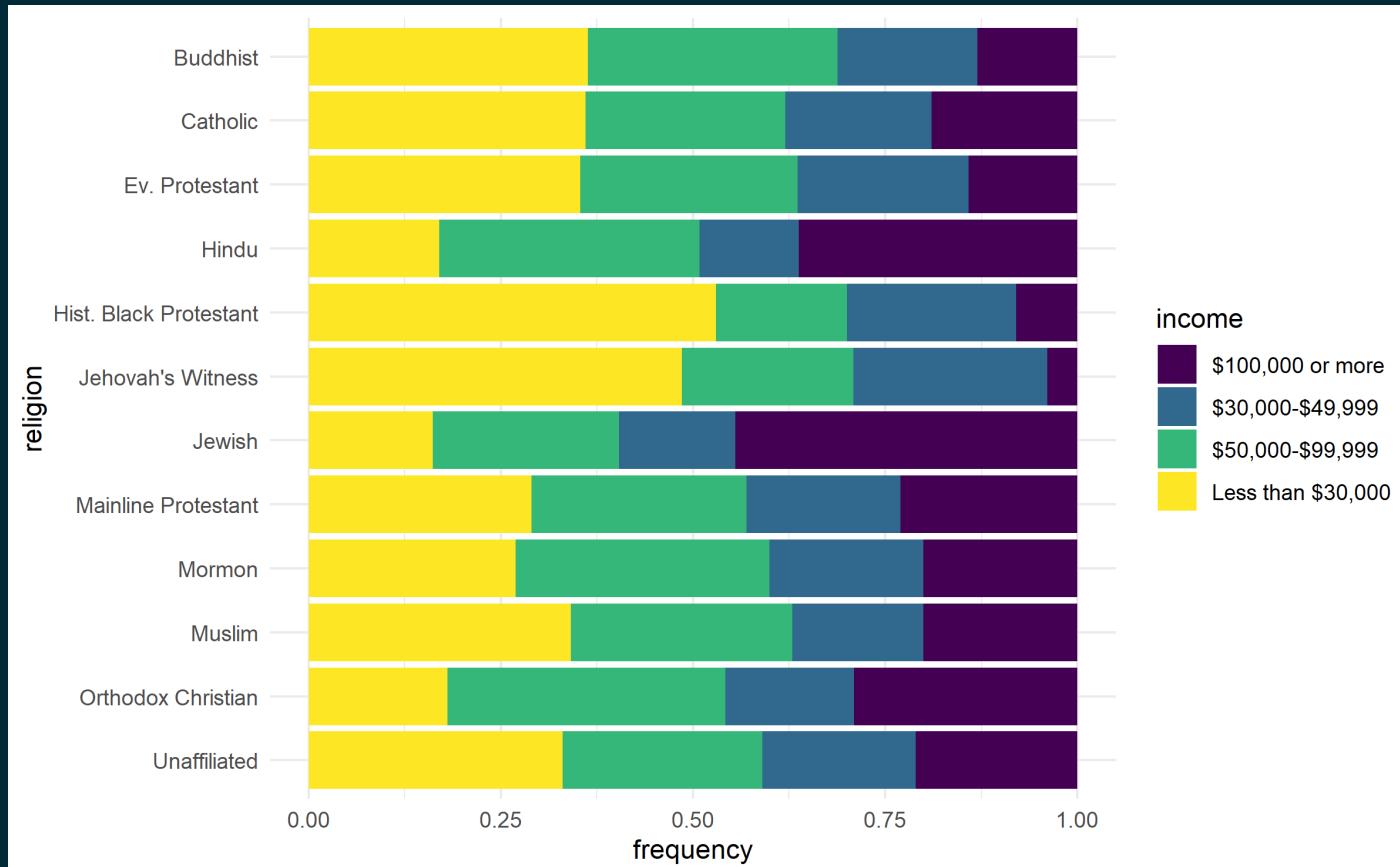
```
ggplot(rel_inc_long, aes(y = religion, x = frequency, fill = income)) +  
  geom_col(position = "fill") +  
  scale_fill_viridis_d()
```



# Change theme

Plot

Code



# Change theme

---

Plot      Code

---

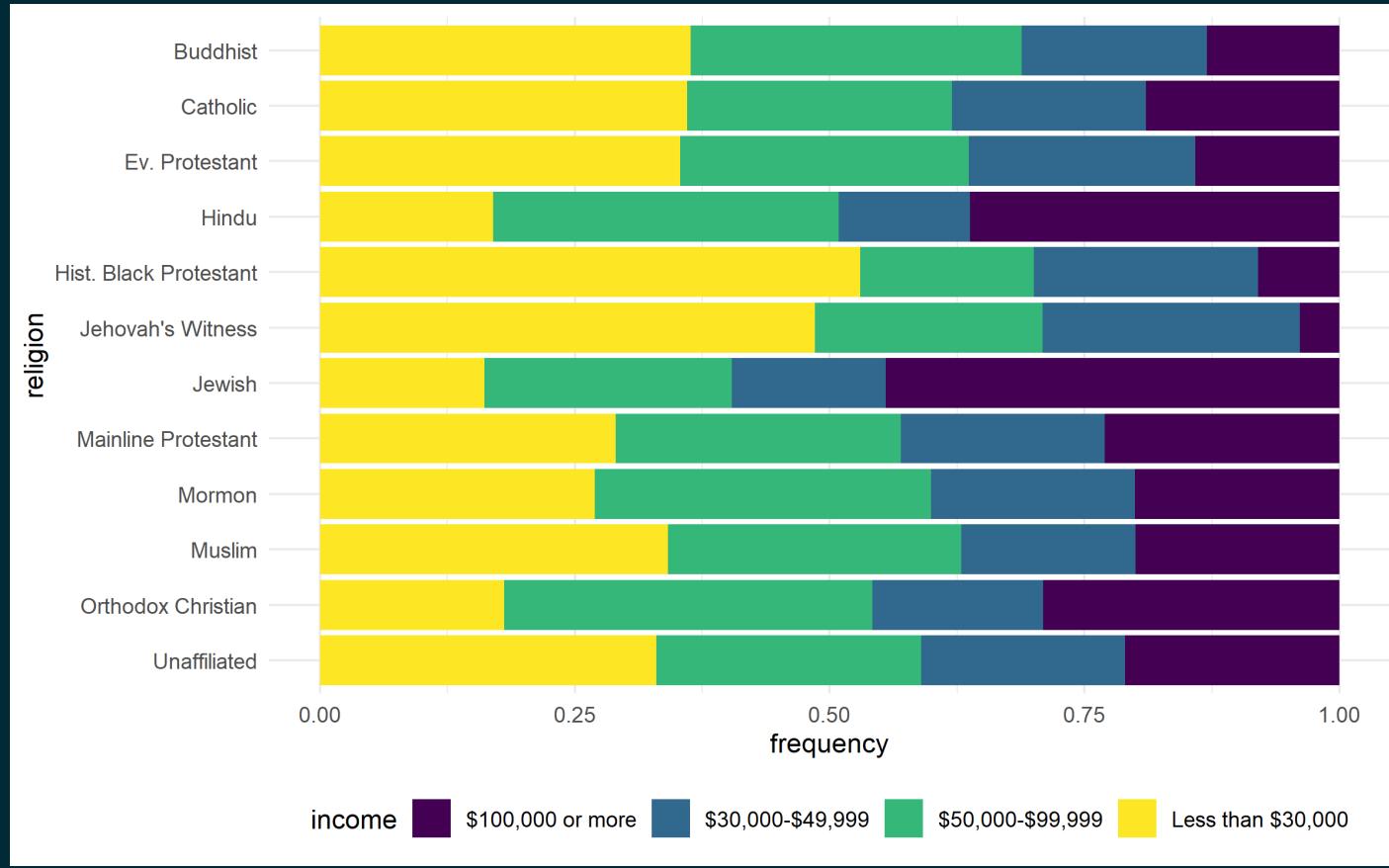
```
ggplot(rel_inc_long, aes(y = religion, x = frequency, fill = income)) +  
  geom_col(position = "fill") +  
  scale_fill_viridis_d() +  
  theme_minimal()
```



# Move legend to the bottom

Plot

Code



# Move legend to the bottom

---

Plot      Code

---

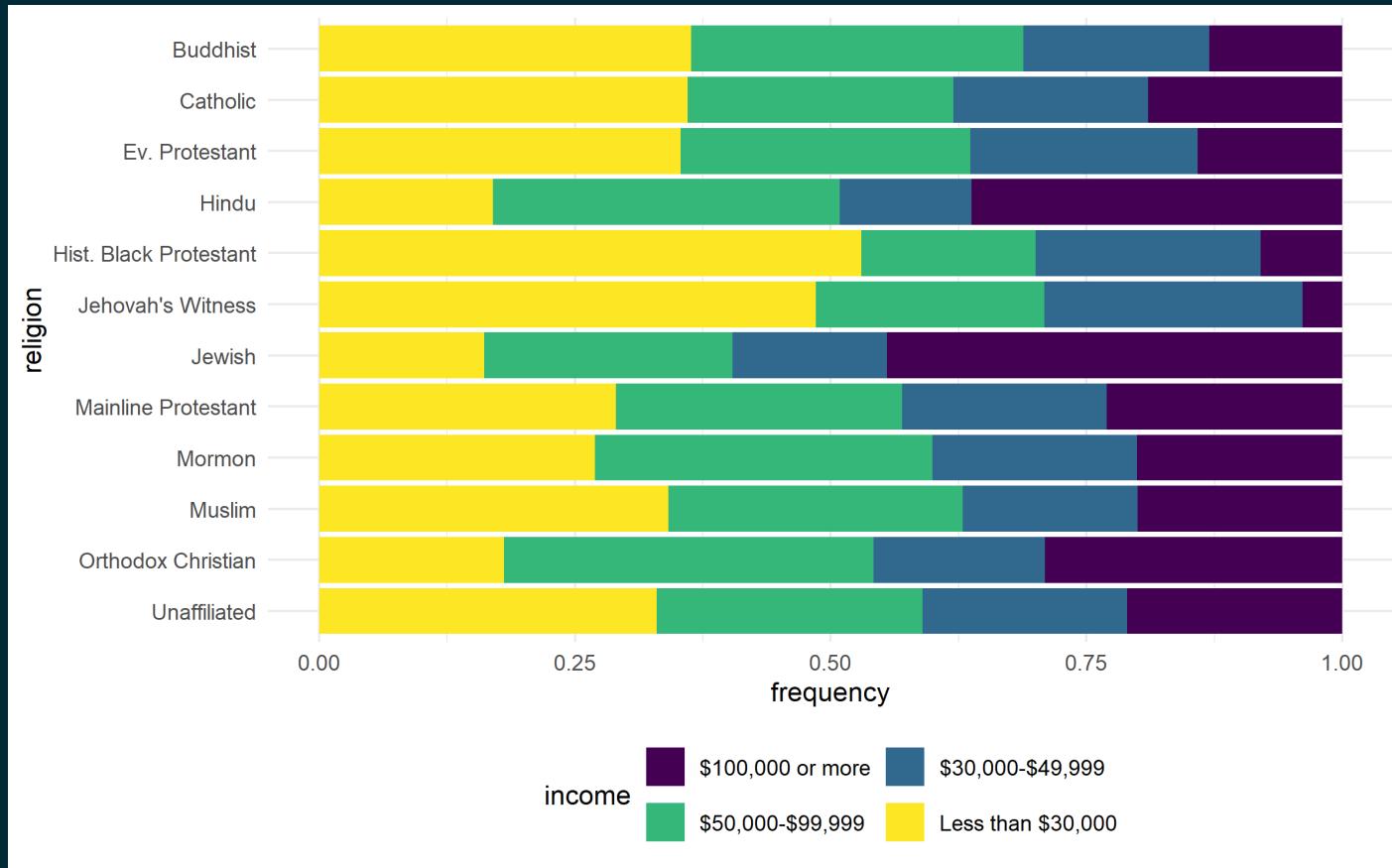
```
ggplot(rel_inc_long, aes(y = religion, x = frequency, fill = income)) +  
  geom_col(position = "fill") +  
  scale_fill_viridis_d() +  
  theme_minimal() +  
  theme(legend.position = "bottom")
```



# Legend adjustments

Plot

Code



# Legend adjustments

---

Plot      Code

---

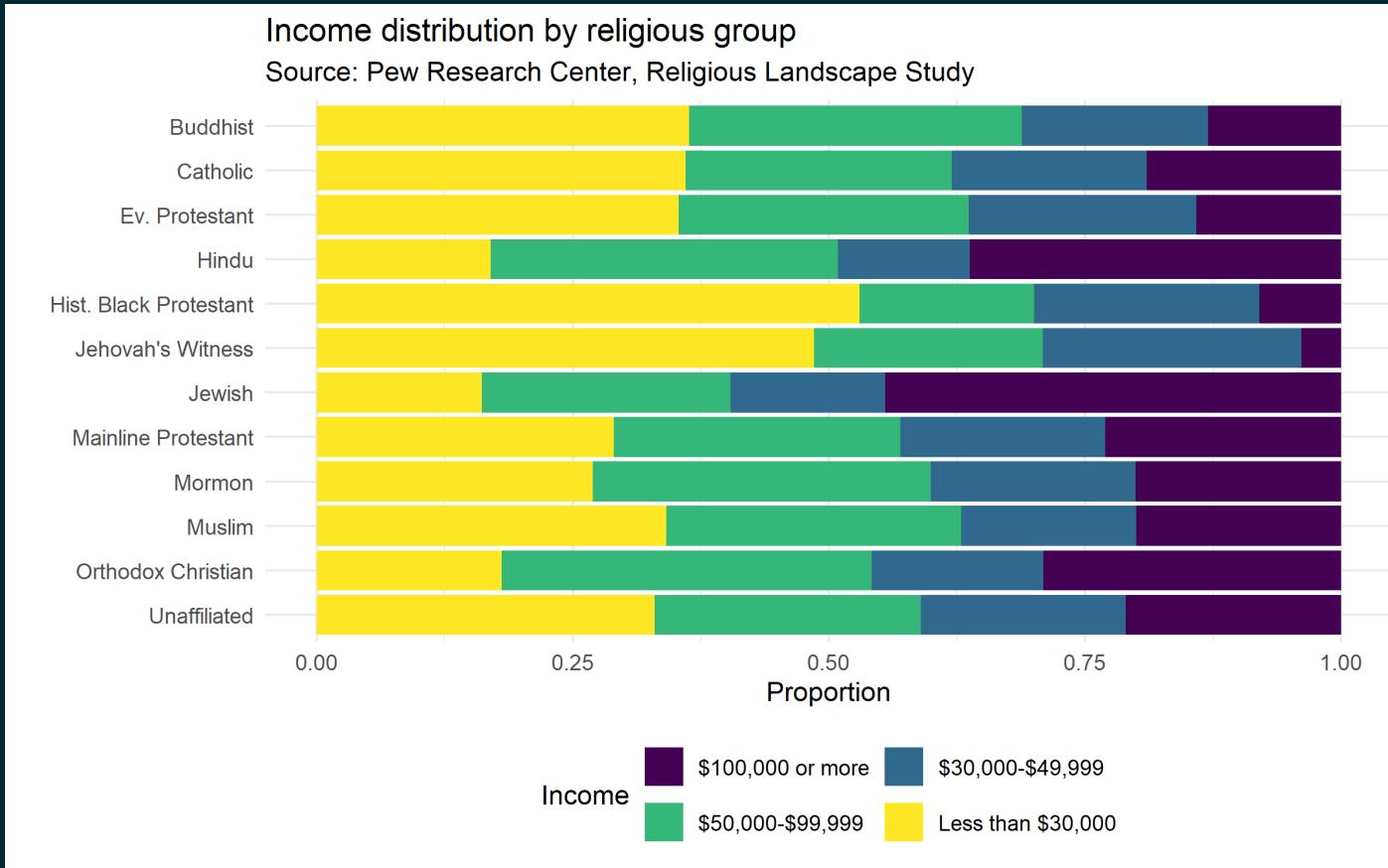
```
ggplot(rel_inc_long, aes(y = religion, x = frequency, fill = income)) +  
  geom_col(position = "fill") +  
  scale_fill_viridis_d() +  
  theme_minimal() +  
  theme(legend.position = "bottom") +  
  guides(fill = guide_legend(nrow = 2, byrow = TRUE))
```



# Fix labels

Plot

Code



# Fix labels

---

Plot      Code

---

```
ggplot(rel_inc_long, aes(y = religion, x = frequency, fill = income)) +  
  geom_col(position = "fill") +  
  scale_fill_viridis_d() +  
  theme_minimal() +  
  theme(legend.position = "bottom") +  
  guides(fill = guide_legend(nrow = 2, byrow = TRUE)) +  
  labs(  
    x = "Proportion", y = "",  
    title = "Income distribution by religious group",  
    subtitle = "Source: Pew Research Center, Religious Landscape Study",  
    fill = "Income"  
  )
```

