Using visualizations in R to explore data

PSYC 259: Principles of Data Science

John Franchak

CDA vs. EDA

- Goals of Confirmatory Data Analysis
  - Hypothesis testing, probabilistic modeling, inference
- Goals of Exploratory Data Analysis (Tukey)
  - Understanding the patterns in the data
  - Generating hypotheses
  - Checking your assumptions about data quality
  - “To find the unexpected, to avoid being fooled, and to develop rich descriptions” (Behrens & Yu, 2003)

Why do we need EDA?

- Summarizing = a loss of information
  - If you first look at summarized data (across trials, across participants, etc.), you may miss important patterns that exist at the raw data level
- Statistics lie, so you need graphics
  - Correlations without looking at the scatterplot
  - Means without examining outliers/distribution
  - Statistical tests without checking *N*

Behrens & Yu, 2013

Today's tutorial

- Ways to explore and check data
  - `dplyr` functions like `filter()`, `summarize()`, and `count()`
  - Specialized packages: `DataExplorer`
  - Plotting distributions using `ggplot2`

Follow along from the Github repo

Last updated: 2022-02-14

Example data for today

```
glimpse(ds)
```

```
Rows: 60
Columns: 44
$ id                     <dbl> 103, 103, 104, 104, 200, 200, 201, 201, 202, 2…
$ condition              <chr> "walk", "search", "walk", "search", "walk", "s…
$ test_date              <dttm> 2019-02-07, 2019-02-07, 2019-02-20, 2019-02-2…
$ target                 <chr> "Diamond", "Diamond", "Rectangle", "Rectangle"…
$ age                    <dbl> 21, 21, 19, 19, 19, 19, 19, 19, 19, 19, 22, 22…
$ dob                    <dttm> 1997-02-24, 1997-02-24, 1999-02-24, 1999-02-2…
$ handedness             <chr> "Right", "Right", "Right", "Right", "Right", "…
$ race                   <chr> "Other: Mexican", "Other: Mexican", "Asian", "…
$ ethnicity_hispanic     <chr> "Yes", "Yes", "No", "No", "No", "No", "No", "N…
$ sex                    <chr> "Female", "Female", "Female", "Female", "Male"…
$ normal_corrected_vision <chr> "Normal", "Normal", "Normal", "Normal", "Norma…
$ temperature            <dbl> NA, NA, 75.0, 75.0, 62.0, 62.0, 64.0, 64.0, 64…
$ humidity               <chr> NA, NA, "0.27", "0.27", "0.22", "0.22", "0.36"…
$ len                    <dbl> 8023, 20152, 7092, 21344, 8606, 22818, 9448, 2…
$ gazex_std              <dbl> 15.5814, 21.7167, 29.3720, 30.4992, 26.3640, 3…
$ gazey_std              <dbl> 11.9732, 11.4963, 18.8811, 17.3218, 18.1747, 1…
$ posx_mean              <dbl> 8.69110, 5.59090, 9.24680, 2.84720, -1.32810, …
$ posx_std               <dbl> 6.1267, 10.2604, 20.4751, 23.6712, 15.9540, 18…
$ posx_speed             <dbl> 0.47016, 0.33660, 1.29310, 0.71103, 0.88678, 0…
```

Things we already know how to do

```
# Filter based on expected ranges
ds %>% select(id, age) %>% filter(age < 18 | age > 22)
```

```
# A tibble: 2 × 2
     id   age
  <dbl> <dbl>
1   206    24
2   206    24
```

```
# Filter based a set of possible values
ds %>% select(id, handedness) %>% filter(!(handedness %in% c("Right", "Left")))
```

```
# A tibble: 2 × 2
     id handedness
  <dbl> <chr>
1   221 <NA>
2   221 <NA>
```

Things we already know how to do

```
# Get summary stats
ds %>% summarize(age_min = min(age), mean_age = mean(age), max_age = max(age))
```

```
# A tibble: 1 × 3
  age_min mean_age max_age
    <dbl>    <dbl>   <dbl>
1      NA       NA      NA
```

```
ds %>% drop_na(age) %>% summarize(age_min = min(age), mean_age = mean(age), max_age = max(ag
```

```
# A tibble: 1 × 3
  age_min mean_age max_age
    <dbl>    <dbl>   <dbl>
1      18     19.9      24
```

```
summary(ds$age) #Also a nice option
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
  18.00   19.00   20.00   19.86   21.00   24.00       2
```

Things we already know how to do

```
# Check the number of participant ids
length(unique(ds$id)) # Should be 30
```

```
[1] 30
```

```
# Count
count(ds, sex)
```

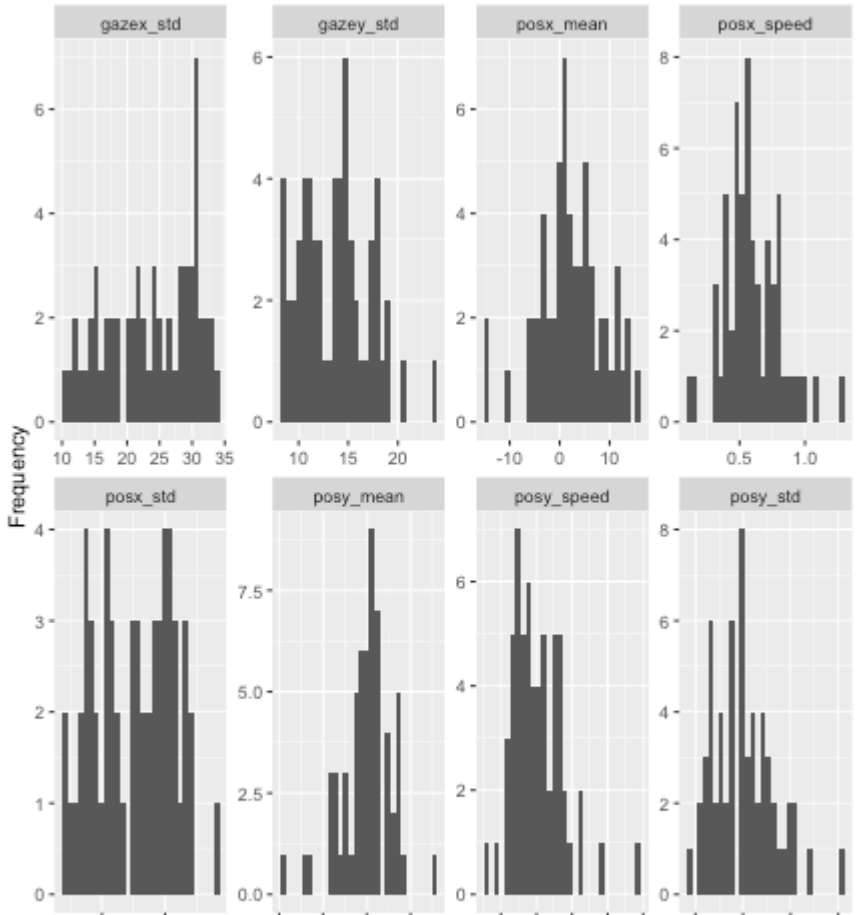
```
# A tibble: 4 × 2
  sex        n
  <chr>  <int>
1 female     2
2 Female    22
3 male      10
4 Male      26
```

Other ways to explore data

- `DataExplorer` package
  - Quickly generate EDA plots for every column in your dataset
  - `plot_histogram` and `plot_density` for continuous variables
  - `plot_bar` for categorical
  - `plot_boxplot` for detecting outliers
  - `plot_missing` for missingness
  - `plot_correlation` for between-column correlations at a glance
  - `create_report` for everything
- What it's not for: making publication-ready plots

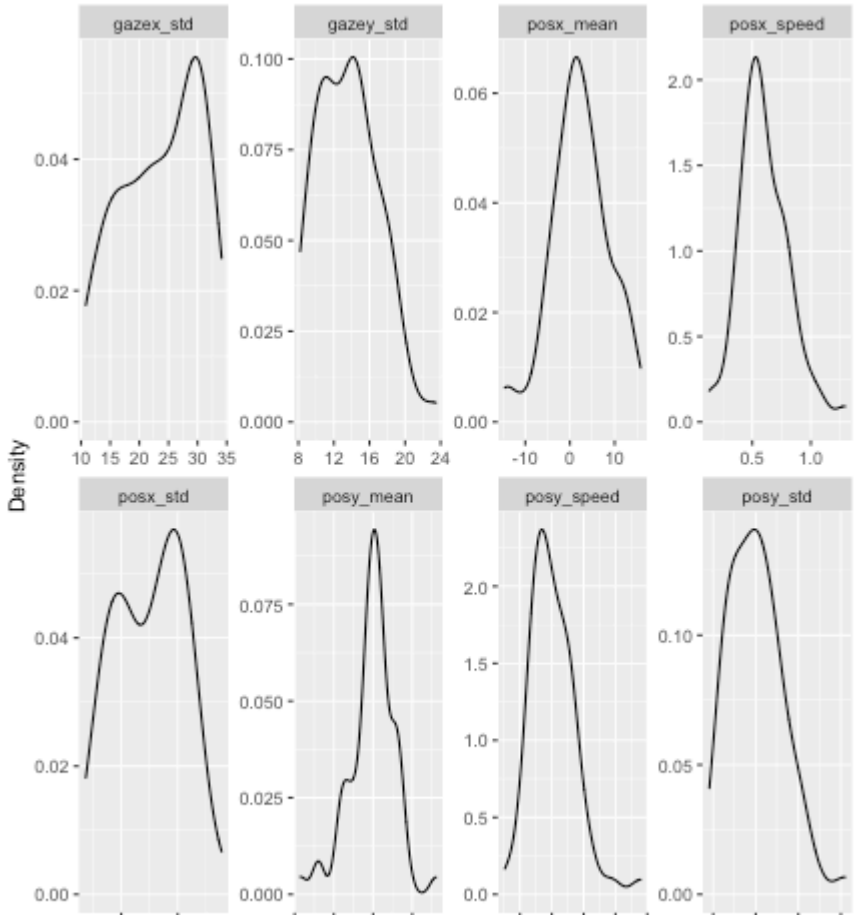
Data Explorer: Histograms

```
ds %>% select(condition, gazex_std:posy_speed) %>% plot_histogram()
```



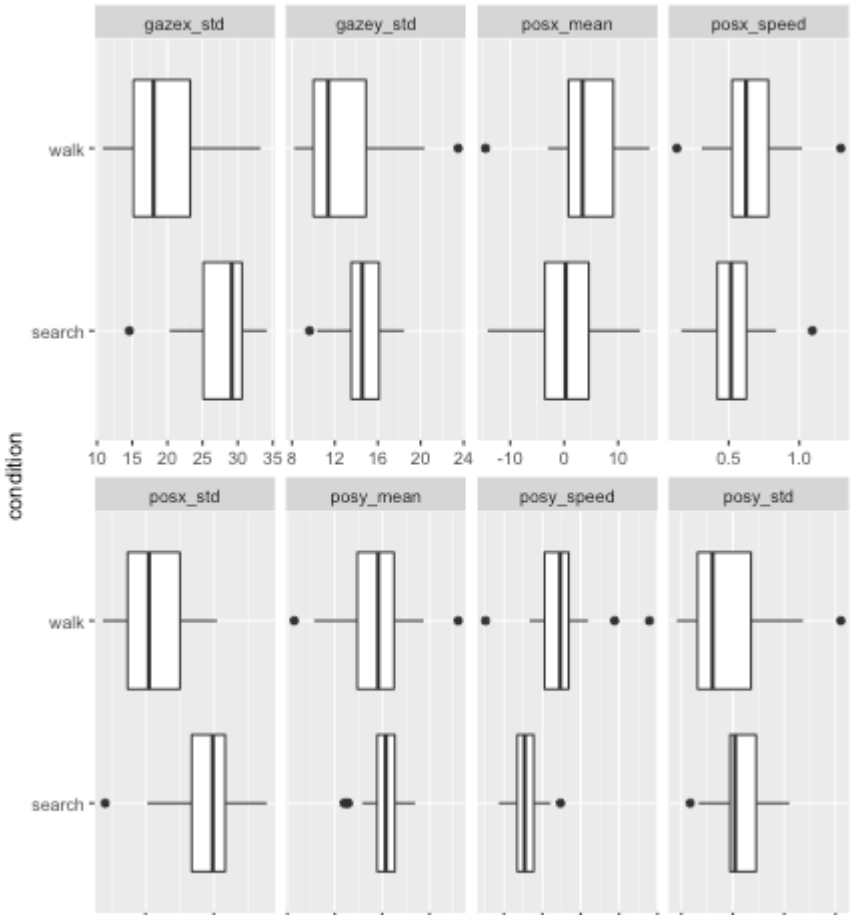
Data Explorer: Density plots

```
ds %>% select(condition, gazex_std:posy_speed) %>% plot_density()
```



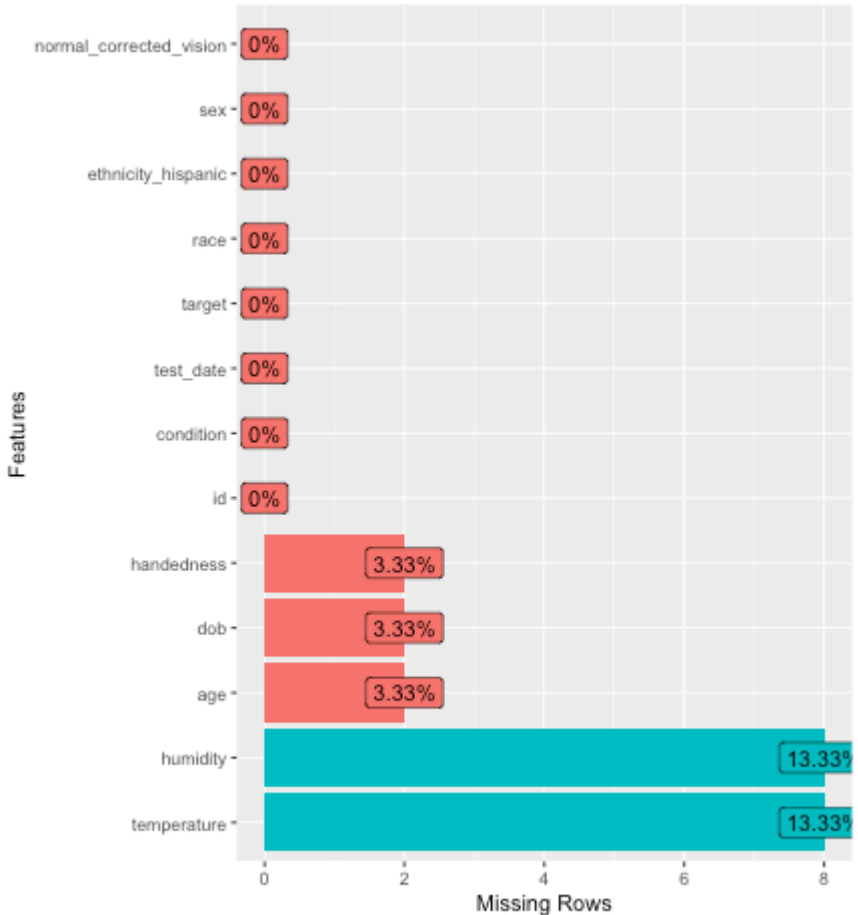
Data Explorer: Boxplots by condition

```
ds %>% select(condition, gazex_std:posy_speed) %>% plot_boxplot(by = "condition")
```



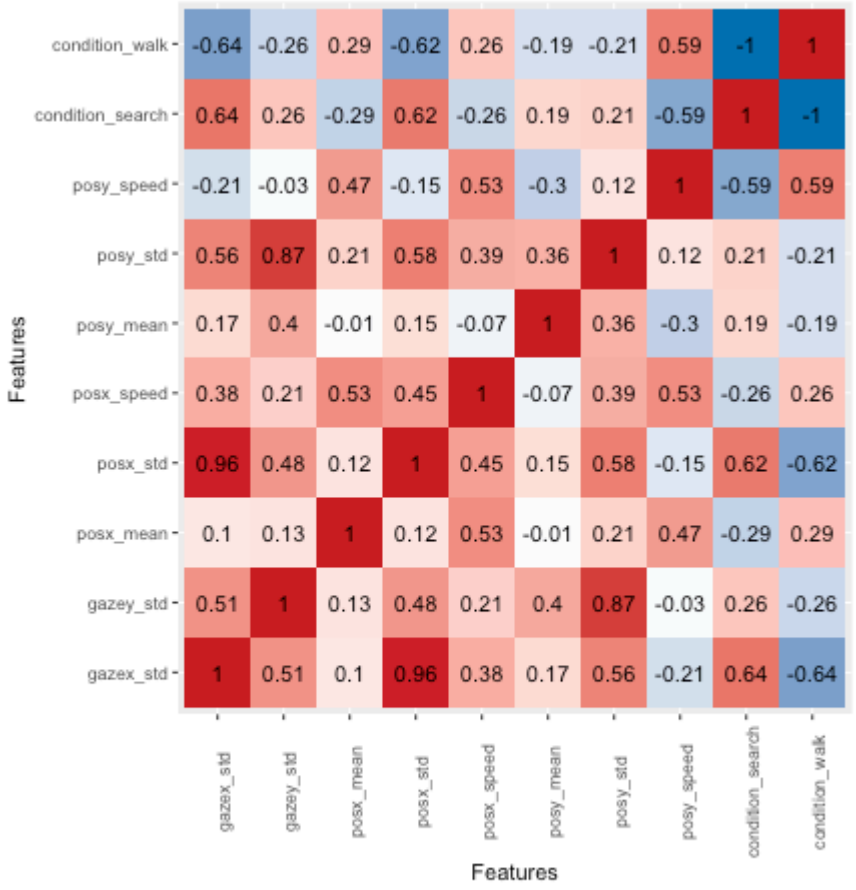
Data Explorer: Plot missing

```
ds %>% select(id:humidity) %>% plot_missing()
```



Data Explorer: Correlations

```
ds %>% select(condition, gazex_std:posy_speed) %>% plot_correlation()
```



Data Explorer: Create Report

- `create_report()` is a brute force approach, which makes every imaginable plot
- Usually better to narrow in to make sure that you're scanning plots that make sense

```
ds %>% create_report()
# Saves an html report to your working directory
```

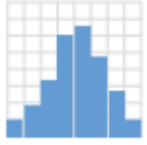
Other option: Make your own plots

- Introducing the `ggplot2` package
- Parts of a ggplot call:

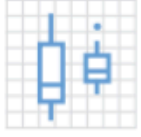
```
# Parts of a ggplot call
ggplot(DATASET, aes(MAPPING STATEMENT)) +
  geom_TYPE() +
  geom_TYPE() +
  ETC... +
  OTHER_FORMATTING() #axes, labels, legends, themes, etc.
```

- Note that ggplot uses `+`, not `%>%`

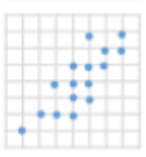
What type of aesthetic mapping depends on what type of geom



c + geom_histogram(binwidth = 5**)** x, y, alpha, color, fill, linetype, size, weight



f + geom_boxplot(), x, y, lower, middle, upper, ymax, ymin, alpha, color, fill, group, linetype, shape, size, weight



e + geom_point(), x, y, alpha, color, fill, shape, size, stroke

h + geom_bin2d(binwidth = c(0.25, 500)**)** x, y, alpha, color, fill, linetype, size, weight

What type of aesthetic mapping depends on what type of geom

- `?geom_point` and scroll down to find required aesthetics in bold

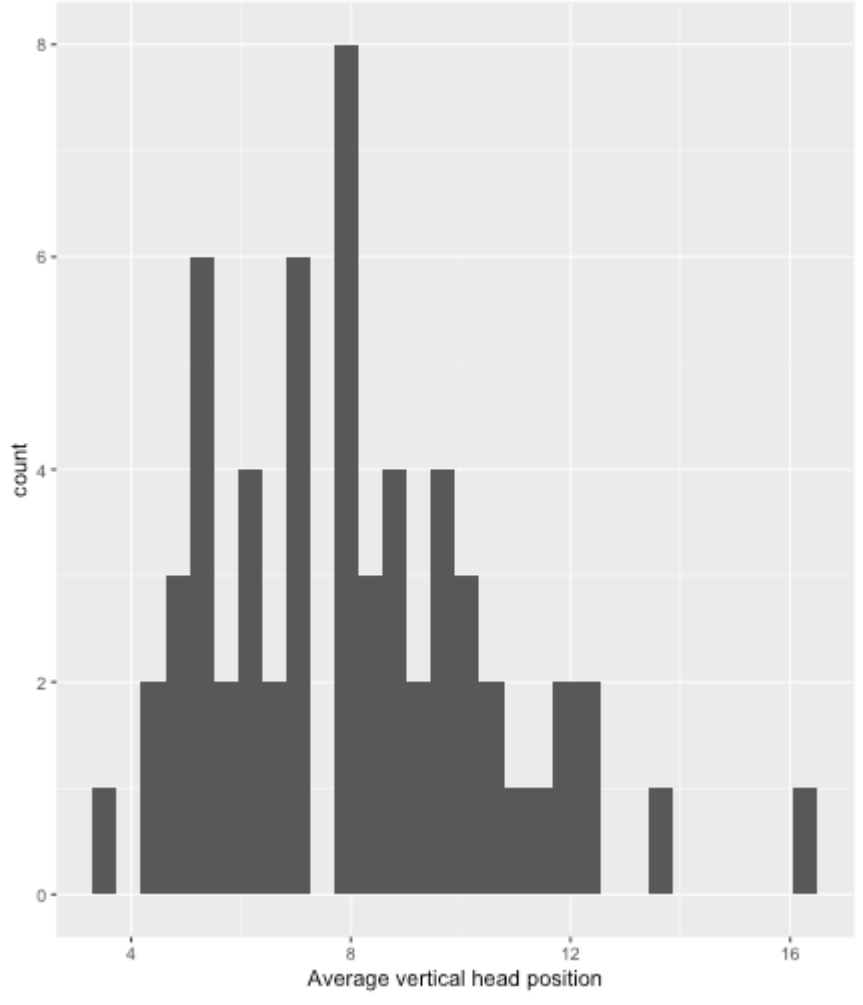
Aesthetics

`geom_point()` understands the following aesthetics (required aesthetics are in bold):

- **`x`**
- **`y`**
- `alpha`
- `colour`
- `fill`

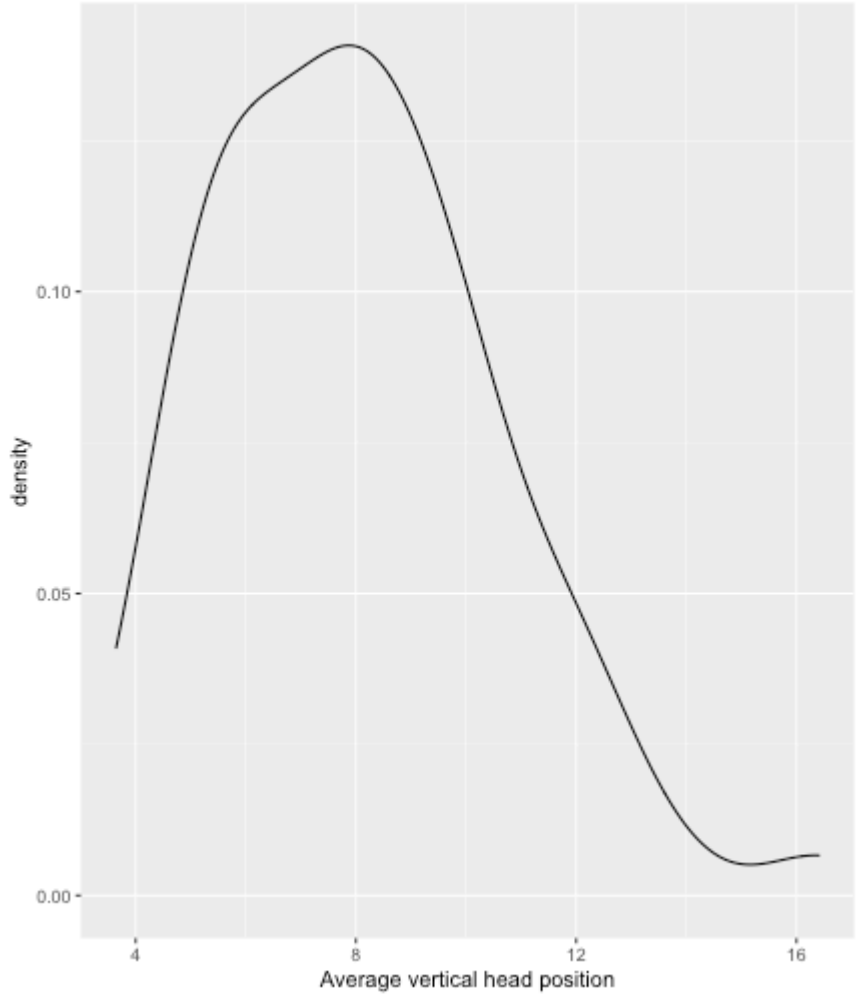
Single-aesthetic plots

```
ggplot(ds, aes(x = posy_std)) +
  geom_histogram() +
  xlab("Average vertical head position")
```



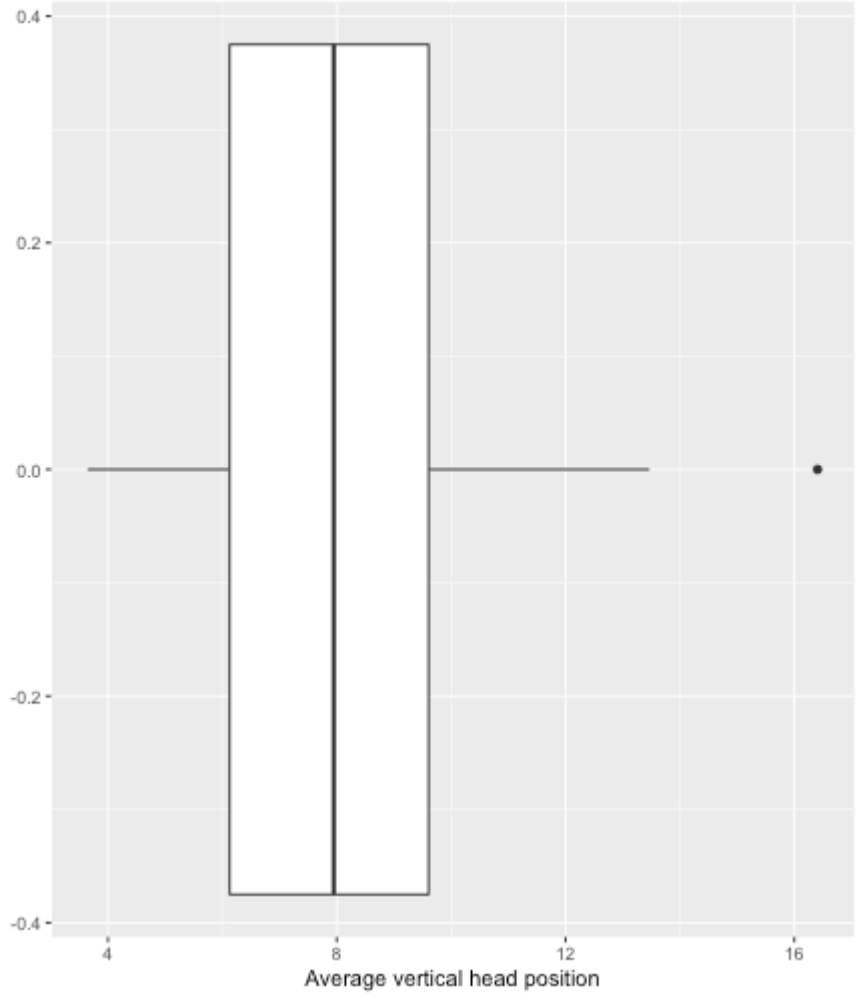
Single-aesthetic plots

```
ggplot(ds, aes(x = posy_std)) +
  geom_density() +
  xlab("Average vertical head position")
```



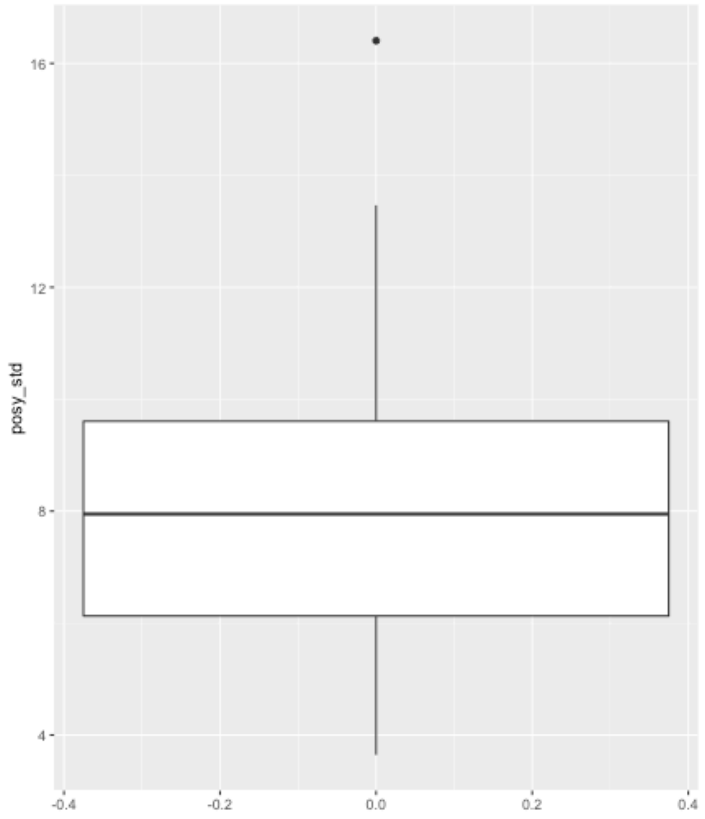
Single-aesthetic plots

```
ggplot(ds, aes(x = posy_std)) +
  geom_boxplot() +
  xlab("Average vertical head position")
```



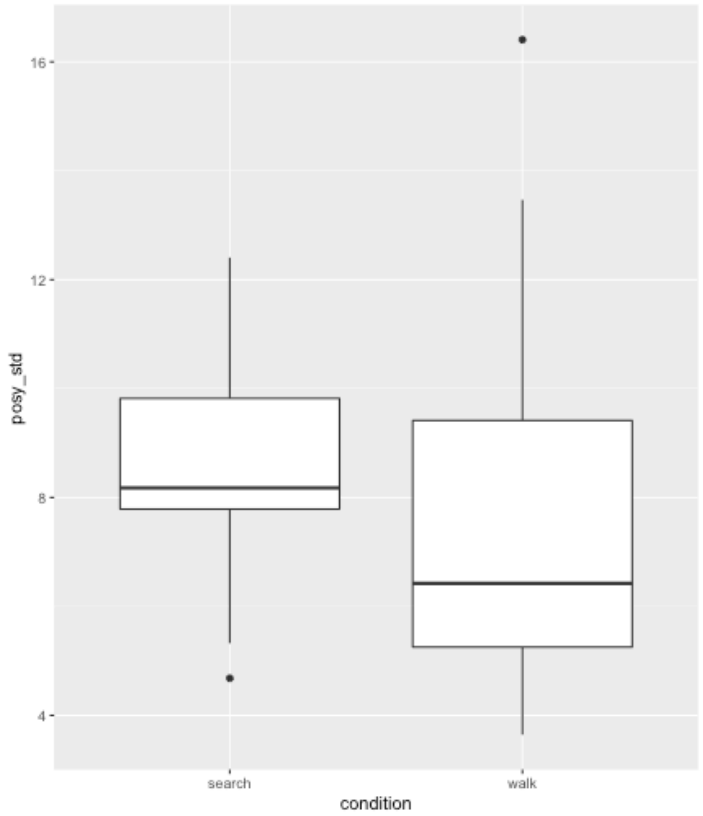
Add a category

```
ggplot(ds) +
geom_boxplot(aes(y = posy_std))
```



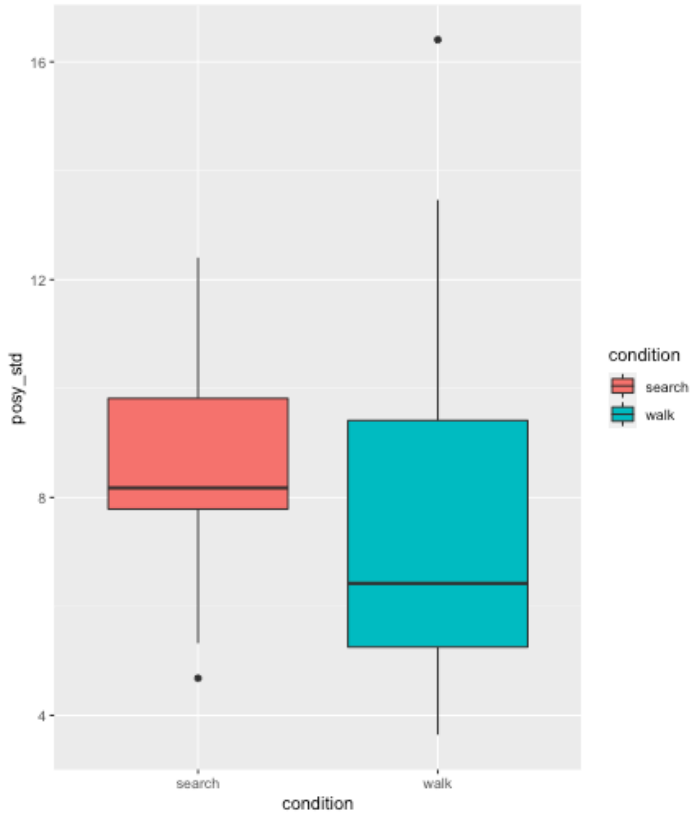
Add a category

```
ggplot(ds) +
geom_boxplot(aes(x = condition, y = posy_std))
```



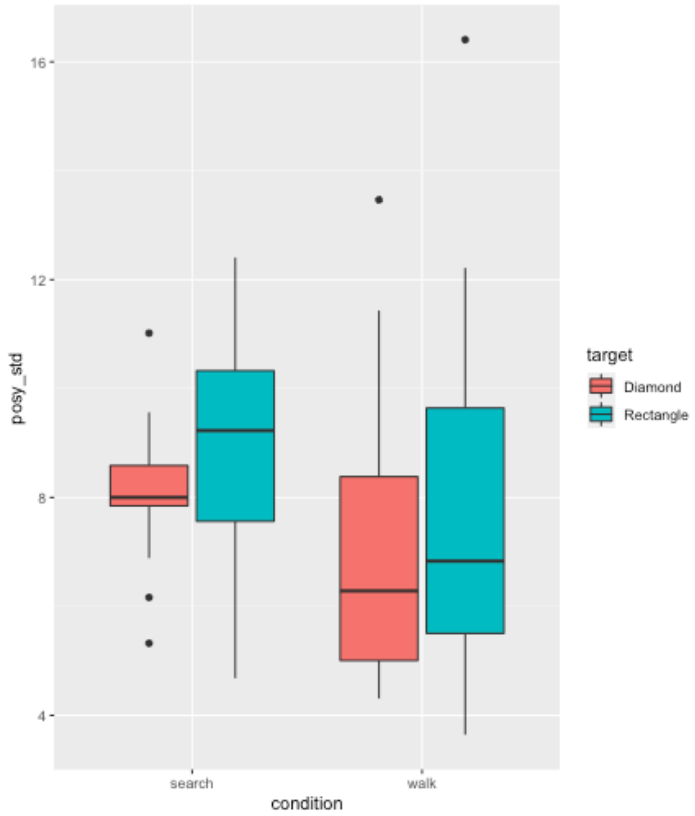
Add a category

```
ggplot(ds) +
geom_boxplot(aes(x = condition, y = posy_std, fill = condition))
```



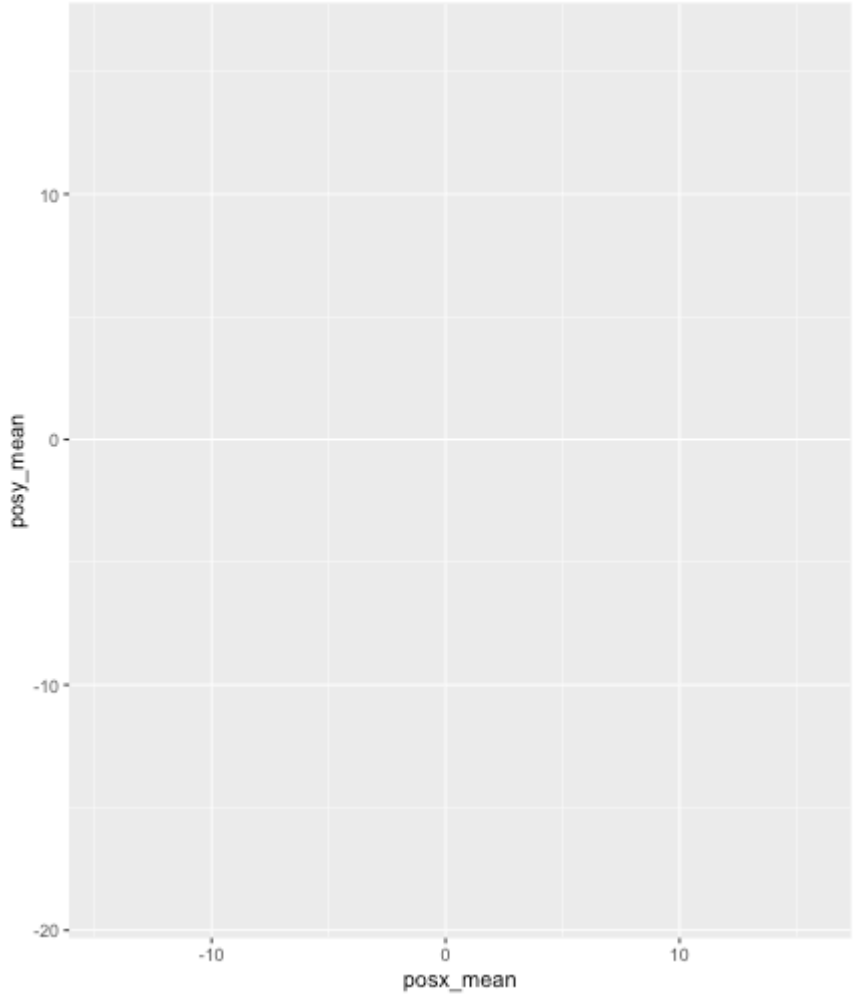
Add a category

```
ggplot(ds) +
geom_boxplot(aes(x = condition, y = posy_std, fill = target))
```



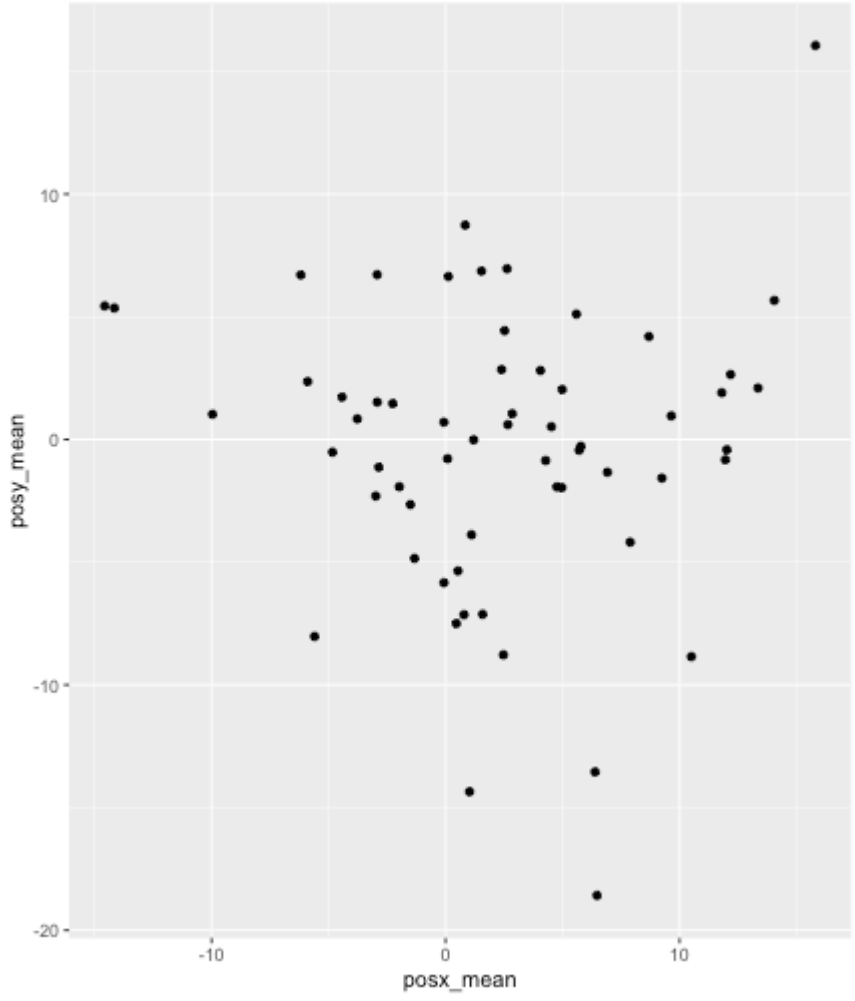
Simple scatterplot

```
ggplot(ds, aes(x = posx_mean, y = posy_mean))
```



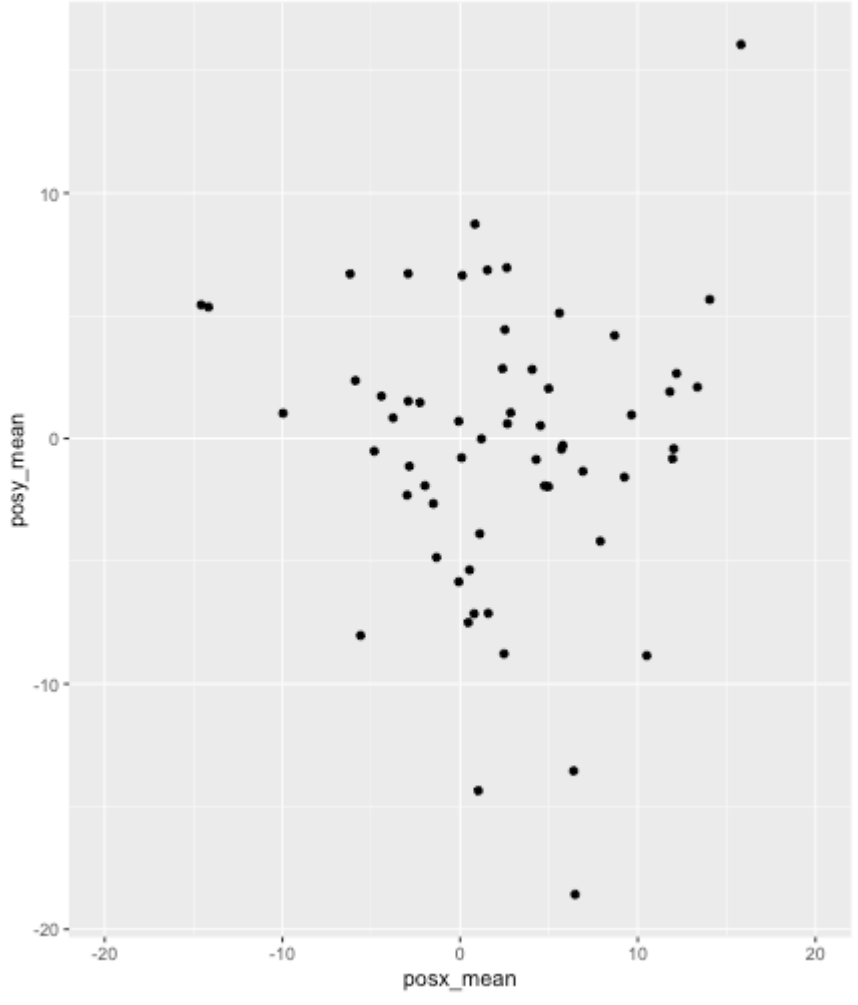
Simple scatterplot

```
ggplot(ds, aes(x = posx_mean, y = posy_mean)) +
  geom_point()
```



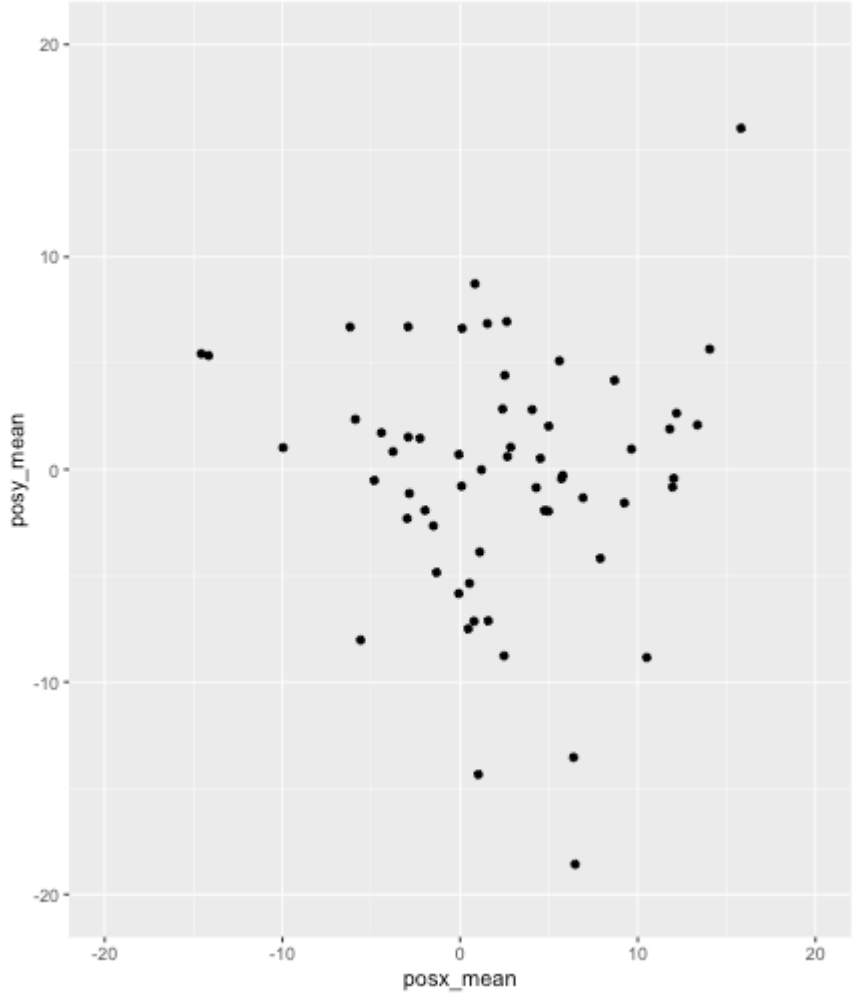
Simple scatterplot

```
ggplot(ds, aes(x = posx_mean, y = posy_mean)) +
  geom_point() +
  xlim(-20, 20)
```



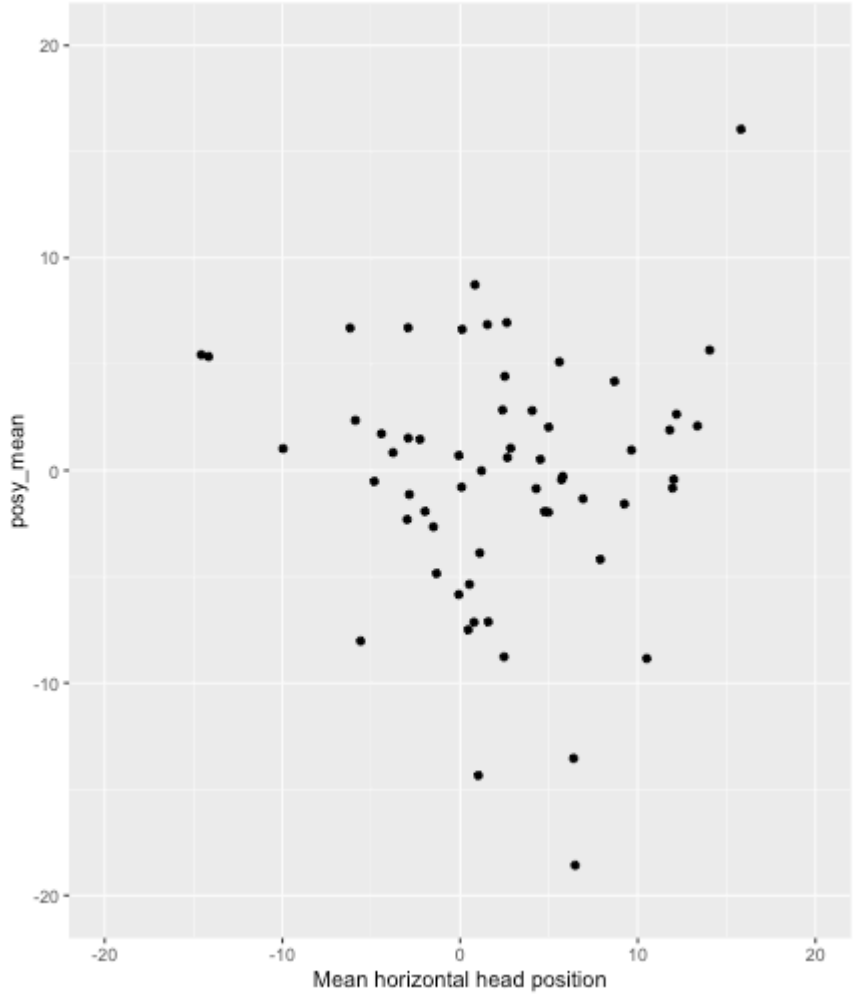
Simple scatterplot

```
ggplot(ds, aes(x = posx_mean, y = posy_mean)) +
  geom_point() +
  xlim(-20, 20) +
  ylim(-20, 20)
```



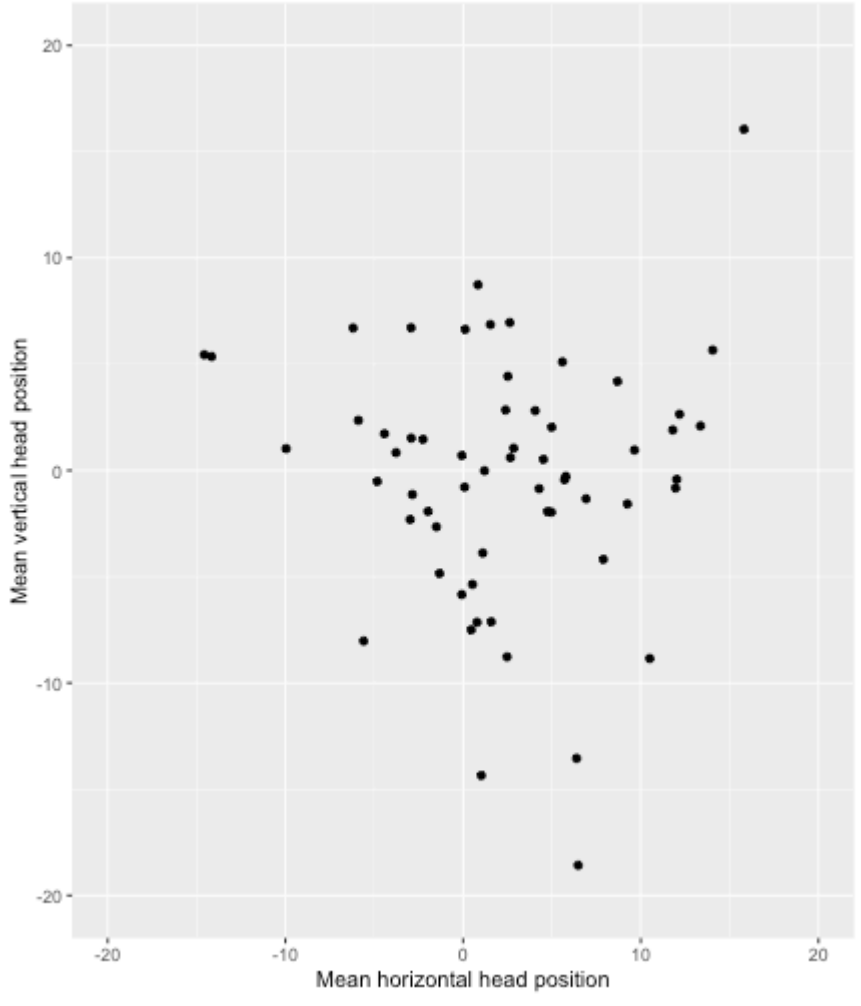
Simple scatterplot

```
ggplot(ds, aes(x = posx_mean, y = posy_mean)) +
  geom_point() +
  xlim(-20, 20) +
  ylim(-20, 20) +
  xlab("Mean horizontal head position")
```



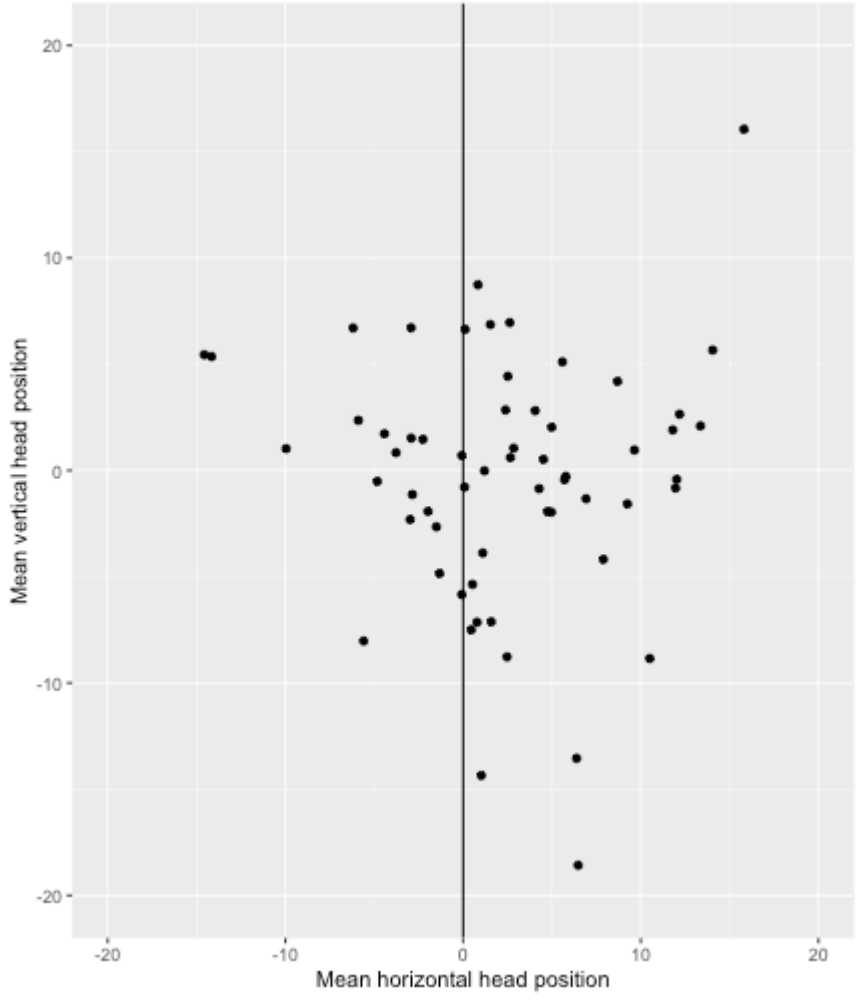
Simple scatterplot

```
ggplot(ds, aes(x = posx_mean, y = posy_mean)) +
  geom_point() +
  xlim(-20, 20) +
  ylim(-20, 20) +
  xlab("Mean horizontal head position") +
  ylab("Mean vertical head position")
```



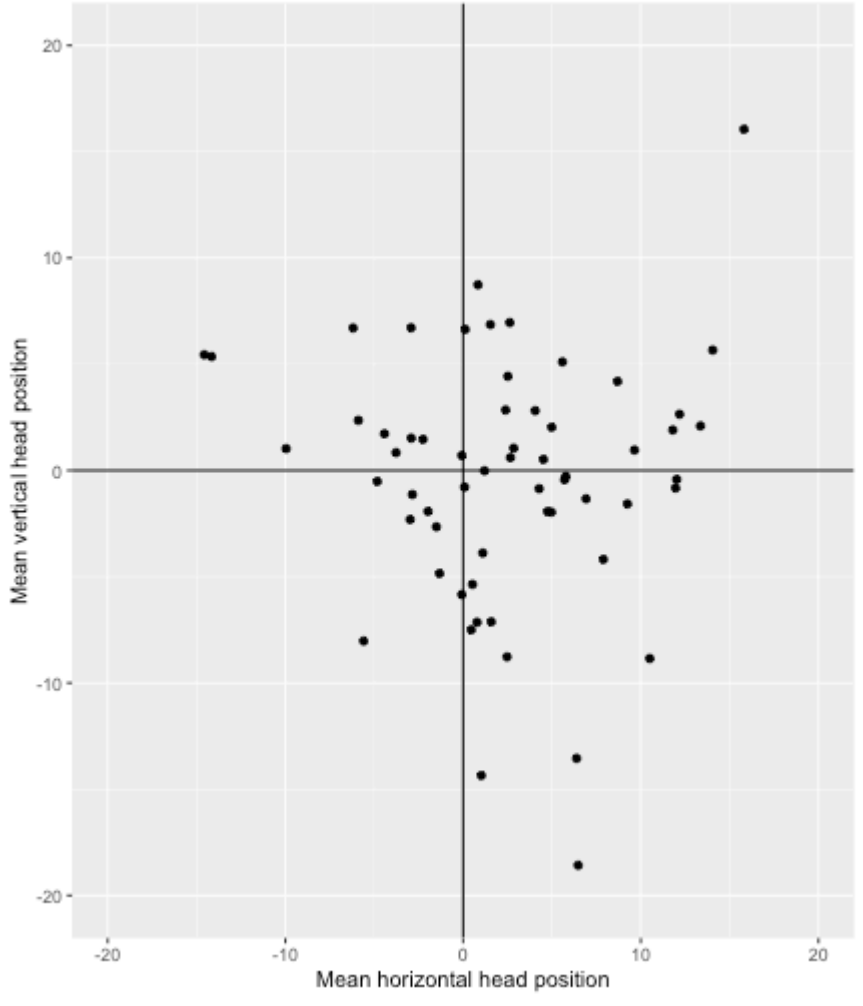
Simple scatterplot

```
ggplot(ds, aes(x = posx_mean, y = posy_mean)) +
  geom_point() +
  xlim(-20, 20) +
  ylim(-20, 20) +
  xlab("Mean horizontal head position") +
  ylab("Mean vertical head position") +
  geom_vline(xintercept = 0)
```



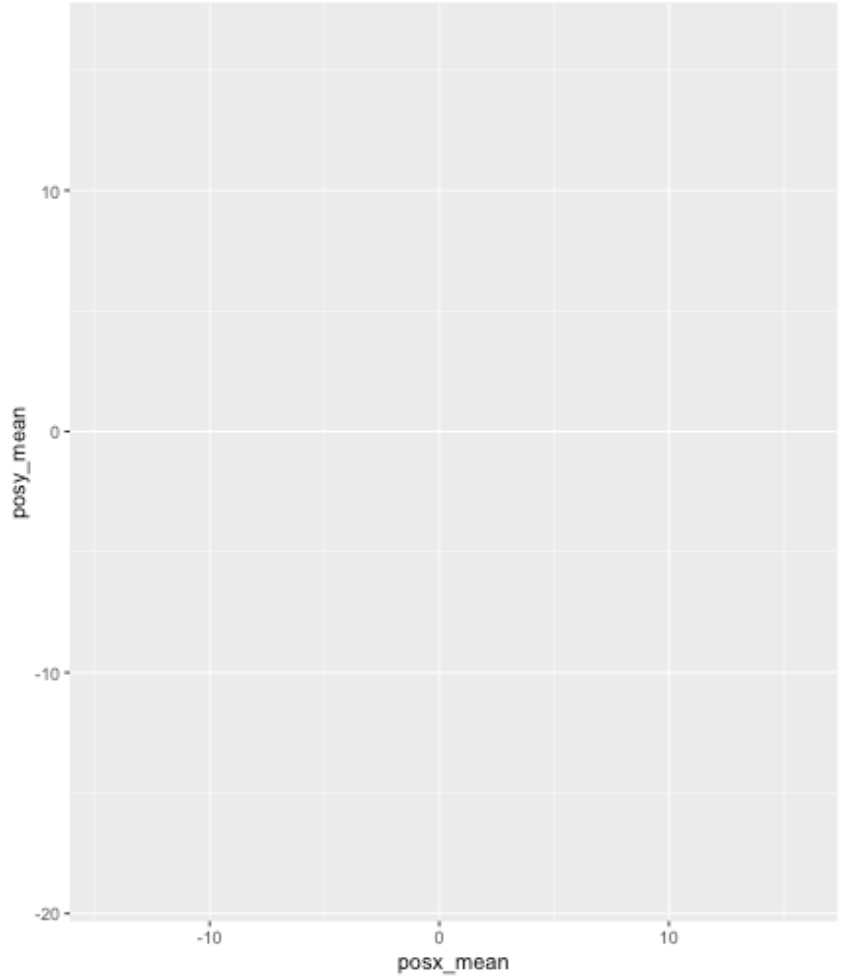
Simple scatterplot

```
ggplot(ds, aes(x = posx_mean, y = posy_mean)) +
  geom_point() +
  xlim(-20, 20) +
  ylim(-20, 20) +
  xlab("Mean horizontal head position") +
  ylab("Mean vertical head position") +
  geom_vline(xintercept = 0) +
  geom_hline(yintercept = 0)
```



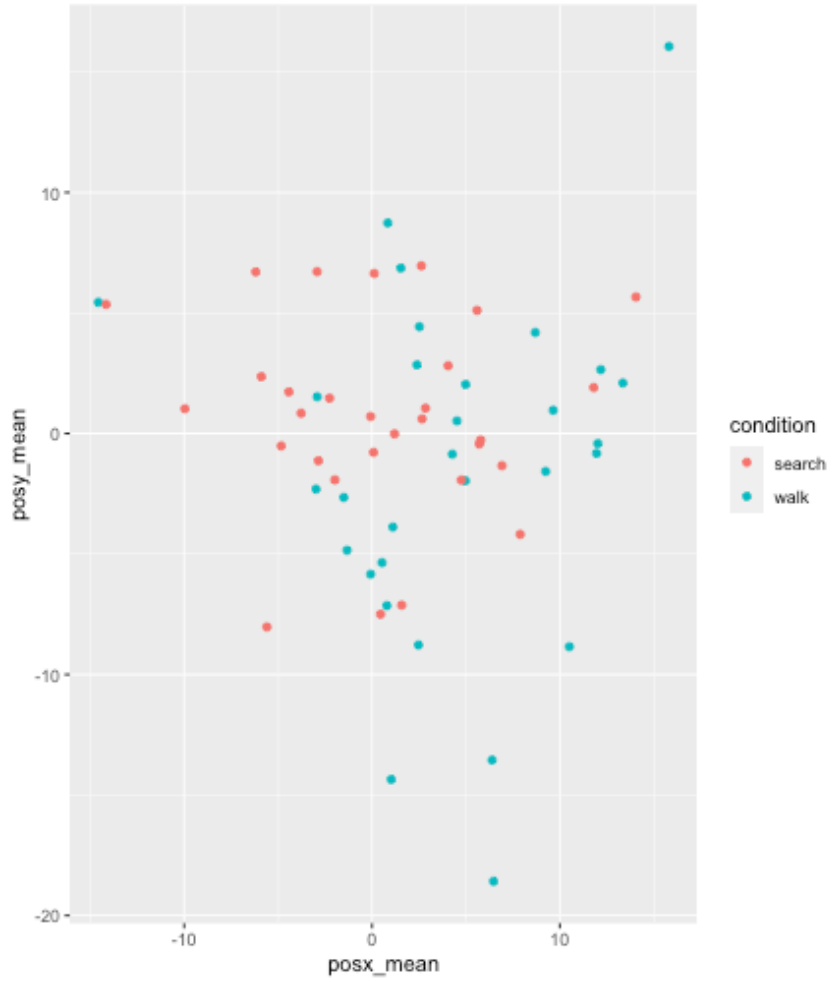
Add category to scatterplot

```
ggplot(ds, aes(x = posx_mean, y = posy_mean, color = condition))
```



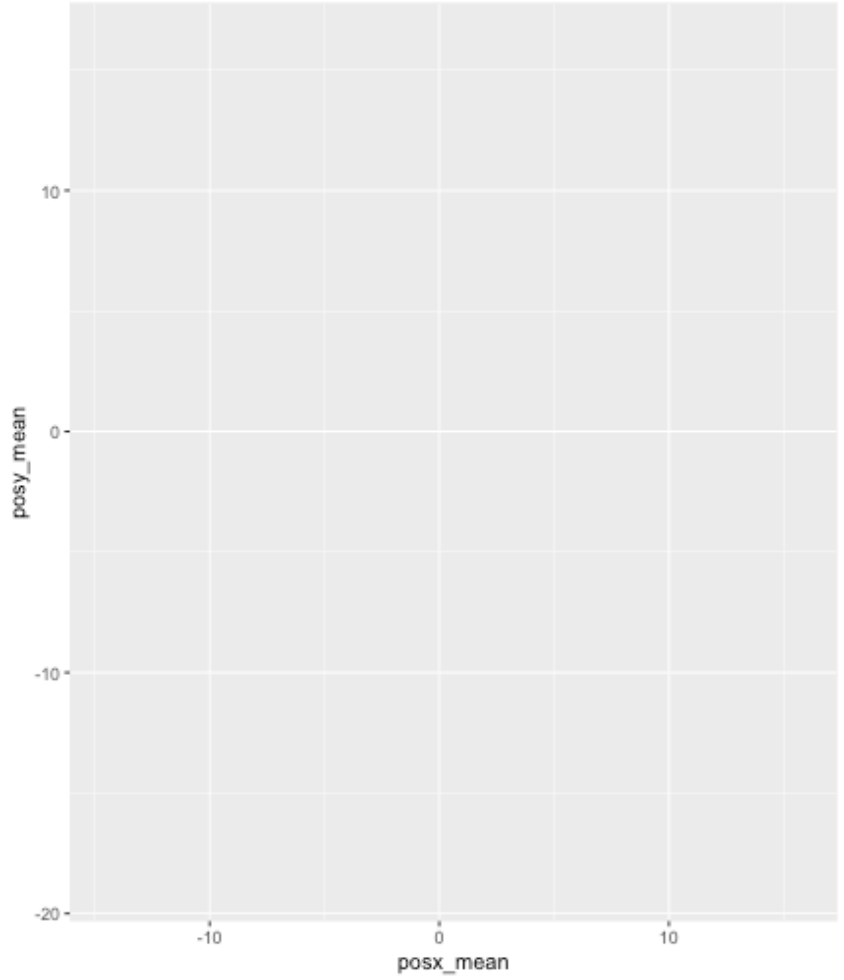
Add category to scatterplot

```
ggplot(ds, aes(x = posx_mean, y = posy_mean, color = condition)) +
  geom_point()
```



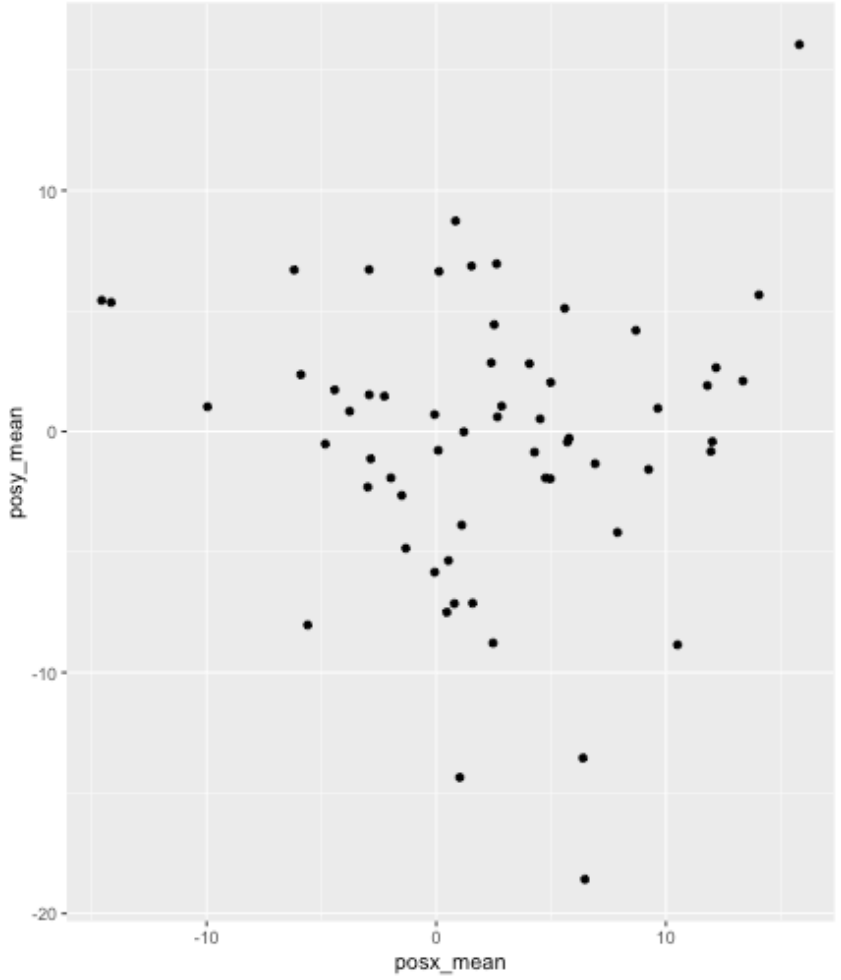
Facetting a scatterplot

```
ggplot(ds, aes(x = posx_mean, y = posy_mean))
```



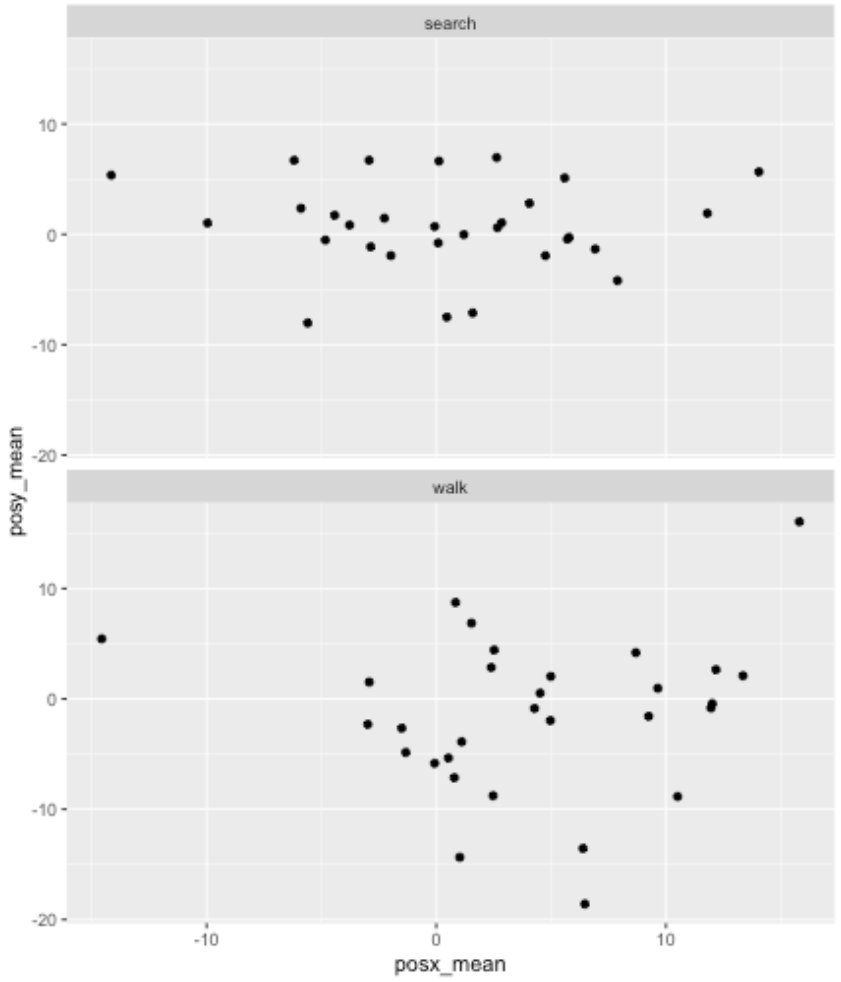
Facetting a scatterplot

```
ggplot(ds, aes(x = posx_mean, y = posy_mean)) +
  geom_point()
```



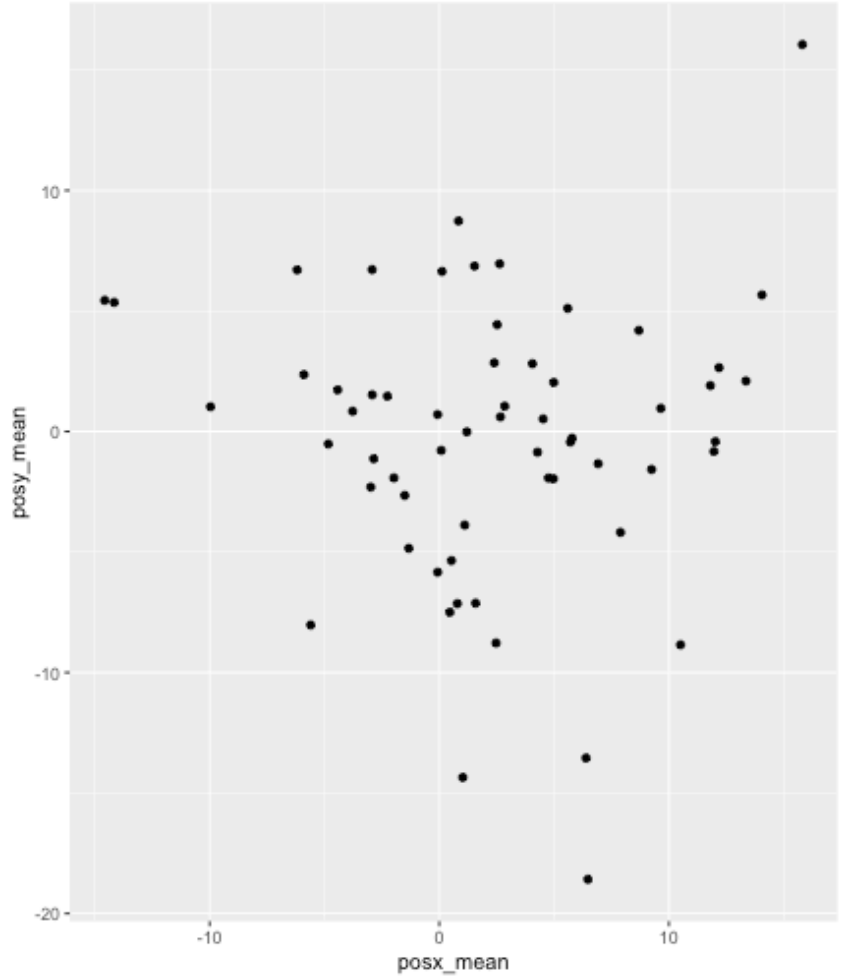
Facetting a scatterplot

```
ggplot(ds, aes(x = posx_mean, y = posy_mean)) +
  geom_point() +
  facet_wrap("condition", ncol = 1)
```



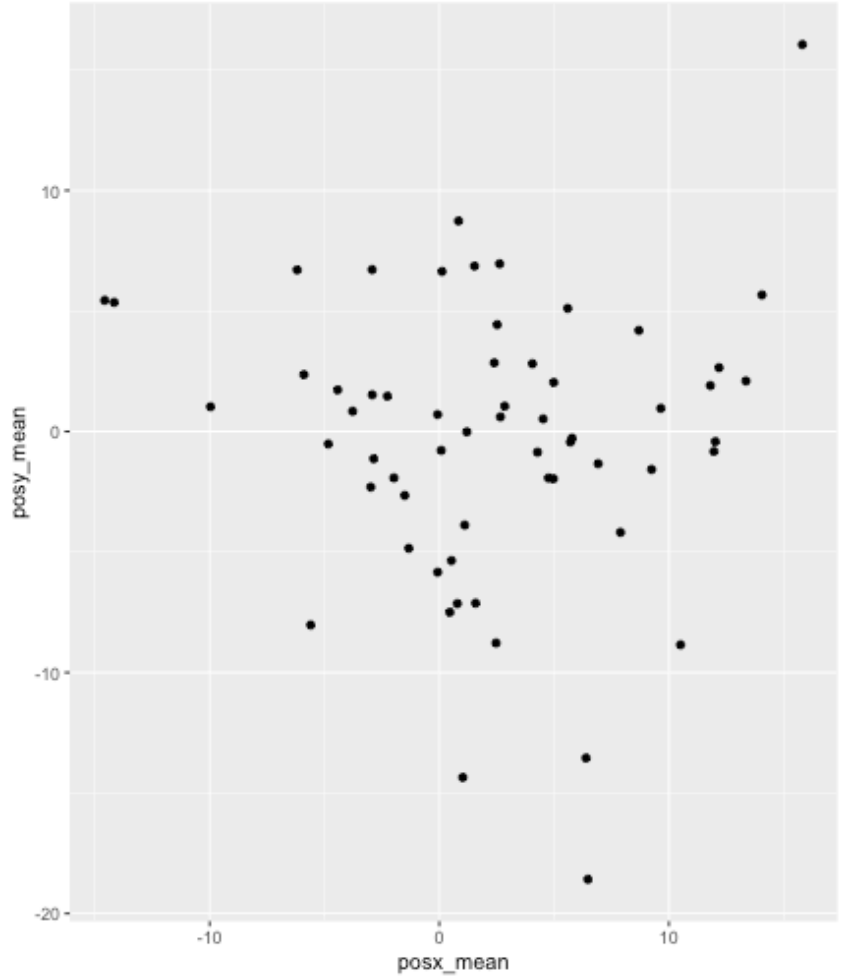
Saving plots

```
ggplot(ds, aes(x = posx_mean, y = posy_mean)) + geom_point()
```



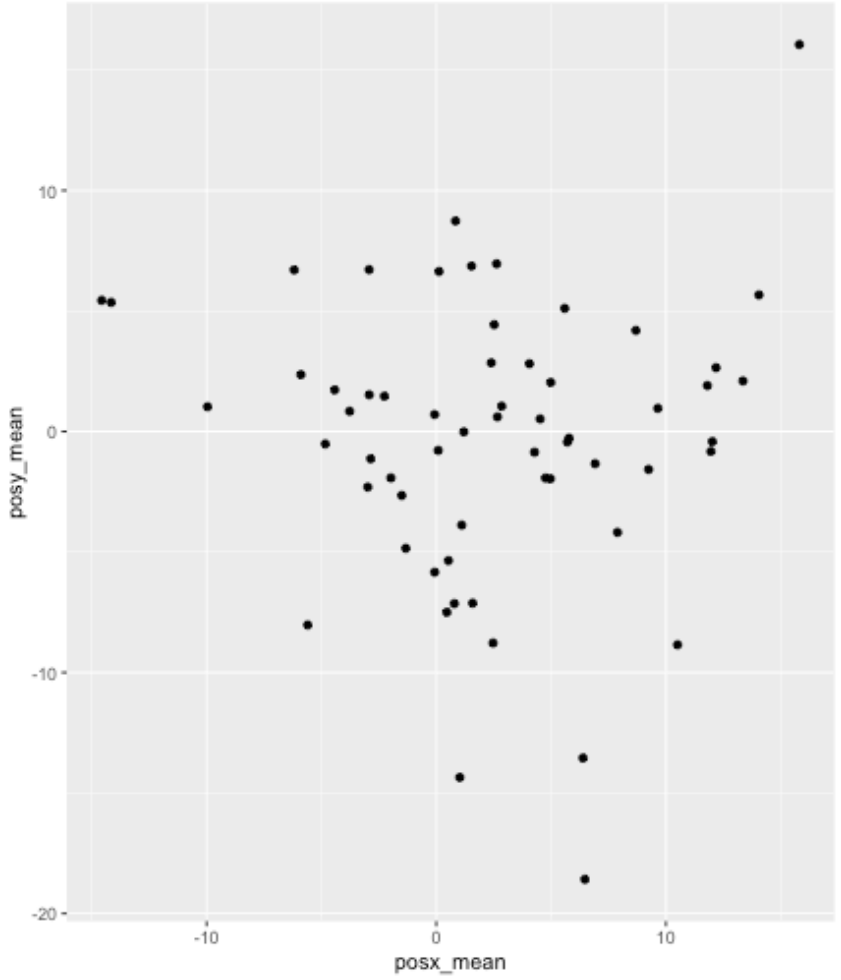
Saving plots

```
ggplot(ds, aes(x = posx_mean, y = posy_mean)) + geom_point()
#ggsave will save the last plot to file
ggsave("eda/head-position-scatter.jpg")
```



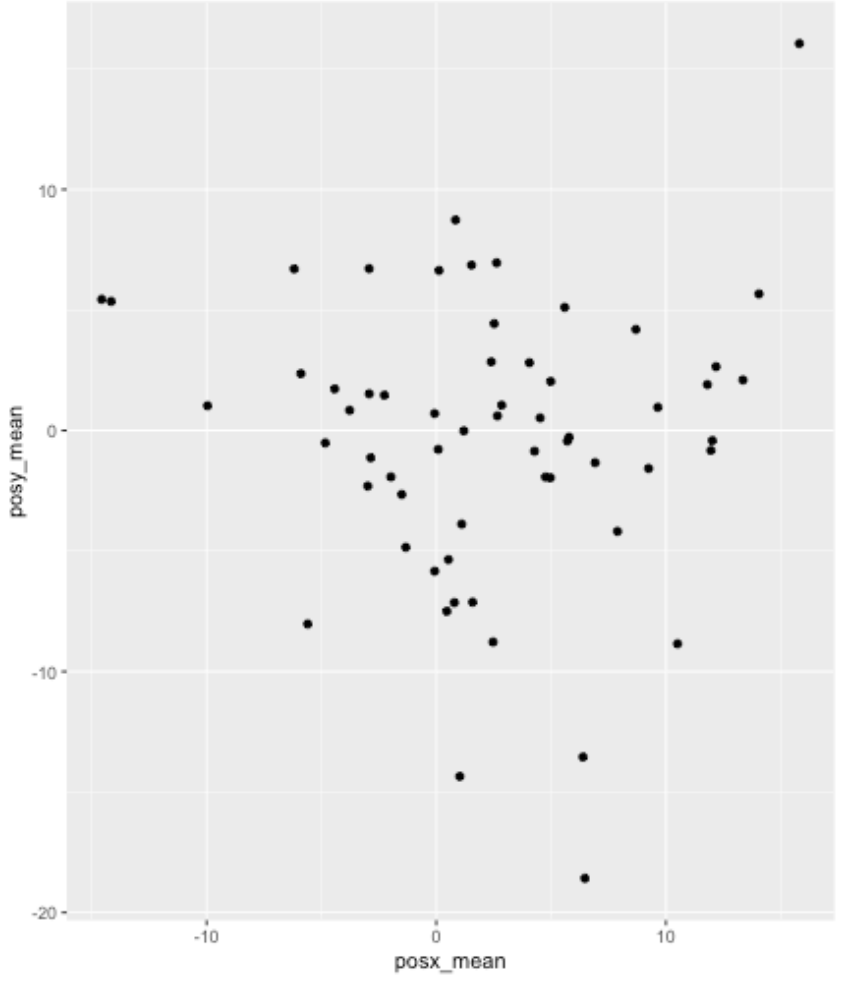
Saving plots

```
ggplot(ds, aes(x = posx_mean, y = posy_mean)) + geom_point()
#ggsave will save the last plot to file
ggsave("eda/head-position-scatter.jpg")

#You can also save plots to objects (they are lists)  ou can also s
```



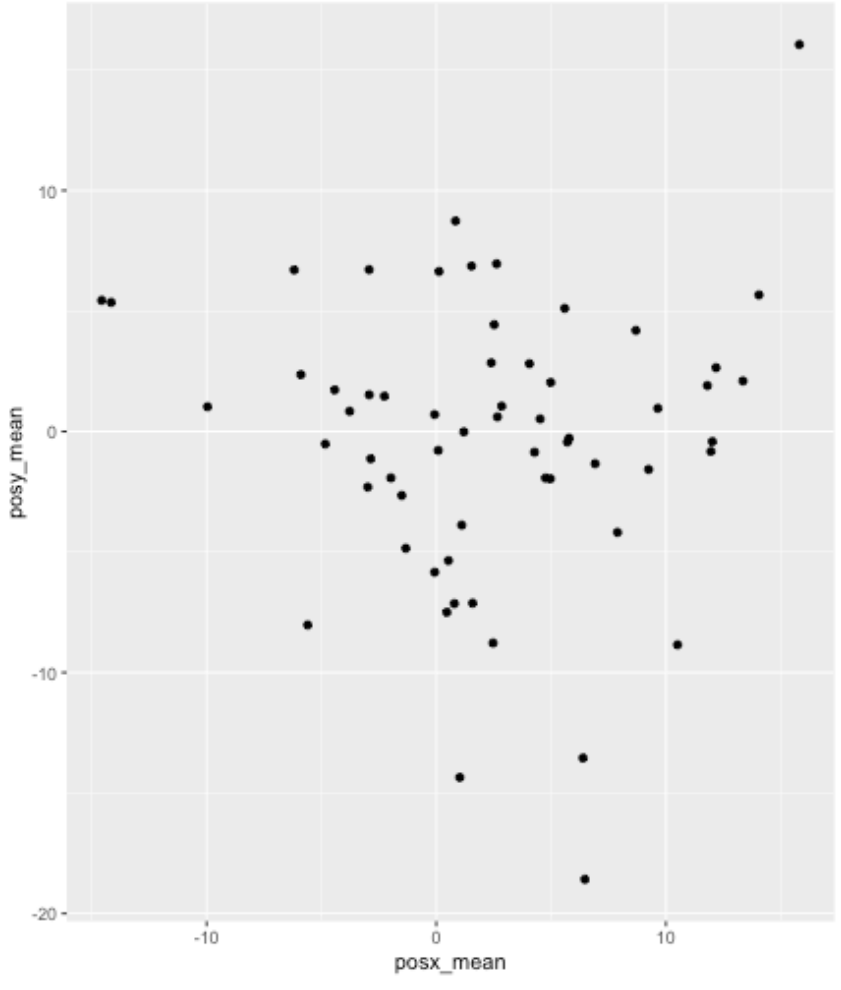
Saving plots

```
ggplot(ds, aes(x = posx_mean, y = posy_mean)) + geom_point()
#ggsave will save the last plot to file
ggsave("eda/head-position-scatter.jpg")

#You can also save plots to objects (they are lists)  ou can also s
p1 <- ggplot(ds, aes(x = posx_mean, y = posy_mean)) + geom_point()
```



Saving plots

```r
ggplot(ds, aes(x = posx_mean, y = posy_mean)) + geom_point()
#ggsave will save the last plot to file
ggsave("eda/head-position-scatter.jpg")

#You can also save plots to objects (they are lists)  ou can also s
p1 <- ggplot(ds, aes(x = posx_mean, y = posy_mean)) + geom_point()
ggsave("eda/head-position-scatter.jpg", plot = p1)
```

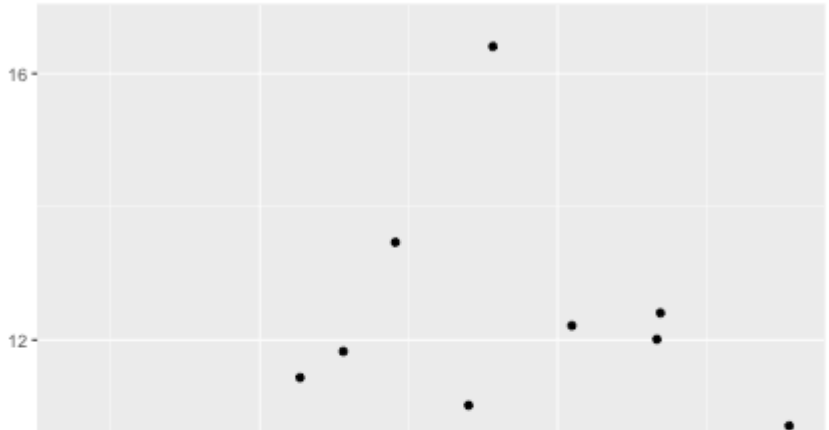


Automation + plotting

- Each plot you create will appear in the plots tab in RStudio
- Flipping through a few plots is OK, but not hundreds
- Write plotting functions that can save plots to file to better organize them

Automation + plotting

```
two_var_scatter <- function(df, var1, var2) {
  df_to_plot <- df %>%
    select(c(var1, var2)) %>%
    rename(var_1 = 1, var_2 = 2)
  p <- ggplot(df_to_plot, aes(x = var_1, y = var_2)) +
    geom_point() +
    xlab(var1) +
    ylab(var2)
  ggsave(str_glue("eda/{var1}_{var2}.jpg"), plot = p)
  return(p)
}
two_var_scatter(ds, "posx_std", "posy_std")
```



Automation + plotting

```
pred_x <- colnames(select(ds, posx_mean:eyey_speed)) %>% keep(str_detect, "x_")
pred_x
```

```
[1] "posx_mean"  "posx_std"   "posx_speed" "eyex_mean"  "eyex_std"
[6] "eyex_speed"
```

```
pred_y <- colnames(select(ds, posx_mean:eyey_speed)) %>% keep(str_detect, "y_")
pred_y
```

```
[1] "posy_mean"  "posy_std"   "posy_speed" "eyey_mean"  "eyey_std"
[6] "eyey_speed"
```

```
pred_x_by_y <- map2(pred_x, pred_y, ~ two_var_scatter(ds, .x, .y))
pred_x_by_y
```

```
[[1]]
```