Exploratory Data Analysis

James Balamuta

Department of Informatics, Statistics
University of Illinois at Urbana-Champaign

July 17, 2017

On the Agenda

1. Administrative Issues
   - HW4 & HW5 Assigned Tonight
   - Graded Exams available for pick up tomorrow during Office Hour
   - Group Project Proposal due date moved to **Friday July 21st, 2017**
2. Exploratory Data Analysis
   - Quantitative
   - Visual

Group Project Update

- Help me, help you by letting me know about your group's project status
- Please answer:
  1. How is the project progressing?
  2. What has been accomplished thus far?
  3. What have you learned?
  4. What issues have arisen?
- *Avoid* showing me code in the report.
- Score for the progress report is based on how much work has been completed since the project proposal was initially submitted.

On the Agenda

Starting an Analysis

> *"When one begins an analysis, the facts of the analysis will stick to oneself."*
> — *James Balamuta*

What does it mean that the "facts" are **sticking** to them?

Exploratory Data Analysis

Exploratory Data Analysis (EDA) is a philosophy for the beginning of an analysis that describes a variety of techniques that are primarily visual but sometimes quantitative in nature as pioneered by John Tukey in his 1977 book Exploratory Data Analysis

The goals are to:

- understand the structure of the data;
- detect mistakes in importing data or within the dataset;
- find outliers and anomalies; and
- test underlying assumptions;

Variable Types in Statistics

Unlike in *Base R*, Statistics views data stored in variables in two forms:

- Quantitative
  - A *number* that describes an outcome
    - **Discrete**: Integers e.g. 1 brother, 2 Starbucks Drinks
    - **Continuous**: Real number e.g. **86.25** on a test, $\pi = 3.141593\ldots$
- Categorical
  - A *character* that describes a trait
    - e.g. "Male" or "Female", "Student" or "Instructor", "Ninjas" or "Pirates"

Types of EDA

There are two types of EDA:

- Quantitative
- Visual

Both with *ups* and *downs*.

Before tackling EDA...

Let's briefly review data ingestion. Verifying the correct data set is loaded can save a lot of heartbreak at a later point. Especially if the naming conventions between data sets differ greatly.

Sample data

To investigate this, we're going to simulate some data that might be commonly associated with an experiment

```
# Make some data
n = 20

# Set seed for reproducibility
set.seed(1133)
d = data.frame(id = paste0("s", sample(n, n)),
               sex = sample(c("male", "female"),
                            n, replace = T),
               food = sample(c("cake", "pie"),
                             n, replace = T),
               trt_a = runif(n),
               trt_b = rnorm(n)
               )
```

Verify the Data

The first step to this process is to verify the `data`.

To do so, use:

- `head()` and `tail()`
  - to make sure the data has been imported correctly.
- `nrow()` and `ncol()` OR `dim()`
  - to understand the amount of observations and variables.
- `class`
  - to verify import data type of each variable.
- `is.na`
  - to obtain whether missing values exist.

Verify the Data - Head

```
head(d)          # Defaults to showing the first 6
```

```
##    id    sex food      trt_a      trt_b
## 1 s19 female  pie 0.07404679  1.8732555
## 2  s6 female cake 0.57559848 -0.4506255
## 3 s16   male  pie 0.15270363 -1.0601917
## 4 s18 female  pie 0.97374584 -0.4109764
## 5  s5 female cake 0.95795011 -0.6313457
## 6  s1   male  pie 0.65279380 -0.6462618
```

```
head(d, n = 2) # Shows the first 2
```

```
##    id    sex food      trt_a      trt_b
## 1 s19 female  pie 0.07404679  1.8732555
## 2  s6 female cake 0.57559848 -0.4506255
```

Verify the Data - Tail

```
tail(d)          # Defaults to showing the last 6
```

```
##     id    sex food      trt_a      trt_b
## 15 s17 female  pie 0.32042847 -1.4592700
## 16 s10 female  pie 0.41355622 -0.3774795
## 17 s14 female cake 0.69593115  0.1023129
## 18 s15 female  pie 0.05747749 -0.2826929
## 19  s8   male  pie 0.31232103 -0.4166077
## 20 s20 female cake 0.90455963  0.1255623
```

```
tail(d, n = 2) # Shows the last 2
```

```
##     id    sex food     trt_a      trt_b
## 19  s8   male  pie 0.3123210 -0.4166077
## 20 s20 female cake 0.9045596  0.1255623
```

Verify the Data - Observations and Variables

```
nrow(d)   # Find the number of observations
```

```
## [1] 20
```

```
ncol(d)   # Find the number of variables
```

```
## [1] 5
```

```
dim(d)    # Both observations and variables (n x p)
```

```
## [1] 20  5
```

Verify the Data - Check Data Types

```
sapply(d, FUN = class)    # Obtain each columns data type
```

```
##         id        sex       food      trt_a      trt_b
##  "factor"  "factor"  "factor" "numeric" "numeric"
```

Verify the Data - Missing Values

```
sapply(d, FUN = function(x) {
  sum(is.na(x))
})    # Missing values per column
```

```
##     id    sex   food trt_a trt_b
##      0      0      0      0      0
```

On the Agenda

1. EDA
   - Background
   - Verify Data
2. Quantitative
   - Types
   - Numeric
   - 5 Summary Statistics
   - Median
   - Categorical
3. Visual
   - A picture. . .
   - Base R Plotting
   - lattice Plotting
   - ggplot2 Plotting
   - ggvis
   - plotly
4. Exploring ggplot2
   - Background
   - Layering
   - Histogram
   - Boxplot
   - Themes

Univariate **Quantitative** Analysis

Depending on the *data type* there are different ways of obtaining univariate **quantitative** information

- **numeric**
  - 5 Summary
- **categorical**
  - frequency
  - contigency table

Univariate **Quantitative** Analysis - Numeric

The 5 Summary Statistics are defined as follows:

- **Minimum**
  - `min()`
- **1st Quartile** or **25% Quantile**
  - `quantile(x, probs = 0.25)`
- **2nd Quartile** or **50% Quantile**
  - `median()`
- **3rd Quartile** or **75% Quantile:**
  - `quantile(x, probs = 0.75)`
- **Maximum:**
  - `max()`
- **(*Optional*) Mean:**
  - `mean()`

Univariate **Quantitative** Analysis - Numeric

```
stat5summary = function(x, na.rm = T){
  if(class(x) != "numeric")
    stop("`x` must be numeric data")

  # Calculate quantiles
  q = quantile(x, probs = c(0.25, 0.5, 0.75),
               na.rm = na.rm)

  # Return
  c("min" = min(x, na.rm = na.rm),
    "q1" = q[[1]], "median" = q[[2]], "q3" = q[[3]],
    "max" = max(x, na.rm = na.rm))
}
```

What might we want to add here?

Univariate **Quantitative** Analysis - Numeric

Let's try out our function!

```
sapply(d[, 4:5], FUN = stat5summary)
```

```
##                trt_a      trt_b
## min     0.05747749 -1.4749439
## q1      0.37380872 -0.9103675
## median 0.59020143 -0.4336166
## q3      0.75354446 -0.2113880
## max     0.97374584  1.8732555
```

Univariate **Quantitative** Analysis - Numeric

Psst... the `summary()` function does this by default on `numeric` data!

```
sapply(d[, 4:5], FUN = summary)
```

```
##                trt_a      trt_b
## Min.     0.05747749 -1.4749439
## 1st Qu.  0.37380872 -0.9103675
## Median   0.59020143 -0.4336166
## Mean     0.55022097 -0.4258596
## 3rd Qu.  0.75354446 -0.2113880
## Max.     0.97374584  1.8732555
```

Regarding the `quantile` function

There are 9 different types of ways to compute quantiles. By default, we prefer to use the 7th type. See `?quantile` for more details.

Implementing a pure median

- Sort data.
- If the numbe of elements is odd then, the median is the center value.
- Else, take an average between the two center values.

```
median_r = function(x) {
  n = length(x)
  half = (n + 1L) %/% 2L

  if (n%%2L == 1L) {
    sort(x)[half]
  } else {
    mean(sort(x)[half + 0L:1L])
  }
}
```

Univariate **Quantitative** Analysis - Categorical

Categorical data normally is associated with:

- Frequency Counts
- Table Format x vs. y
- Percentages

Univariate **Quantitative** Analysis - Categorical

```
sapply(d[, 1:3], FUN = summary)
```

```
## $id
##  s1 s10 s11 s12 s13 s14 s15 s16 s17 s18 s19  s2 s20  s3  s4
##   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
##  s5  s6  s7  s8  s9
##   1   1   1   1   1
##
## $sex
## female   male
##     12      8
##
## $food
## cake  pie
##    9   11
```

Univariate **Quantitative** Analysis - Categorical Tabulate

Overall counts between two variables

```
(o = table(d[, 2], d[, 3]))
```

```
##
##           cake pie
##   female     6   6
##   male       3   5
```

Univariate **Quantitative** Analysis - Categorical Proportions

Element / Total number of observations

```
prop.table(o)  # Requires table() object
```

```
##
##            cake  pie
##    female 0.30 0.30
##    male   0.15 0.25
```

Univariate **Quantitative** Analysis - Categorical Headache

Make sure to avoid unique comparisons. . .

```
head(table(d[,1], d[,2]))
```

```
##
##        female male
##    s1       0    1
##    s10      1    0
##    s11      0    1
##    s12      0    1
##    s13      1    0
##    s14      1    0
```

Rules of Thumb

There are a couple *rules of thumb* that are slightly helpful with EDA and statistical modeling.

1. If the number of distinct numbers is less than 20, treat them as *categorical* variables.
2. Try to floor and cap *numerical* values to avoid large extrema.
   - Floor and Cap means to set a boundary point for low and high values.
   - Never tell a robust statistician this...

Exercises

1. Determine the summary information for the `PlantGrowth` dataset.
   - What variables exist, what kind of variables are there?
2. Obtain the `msos` package from cran and look at the `spam` dataset.
   - How often was spam detected?
3. Download the `faraway` package from CRAN and explore the `pima` dataset.
   - Any pattern with missing values?

On the Agenda

1. EDA
   - Background
   - Verify Data
2. Quantitative
   - Types
   - Numeric
   - 5 Summary Statistics
   - Median
   - Categorical
3. Visual
   - A picture. . .
   - Base R Plotting
   - lattice Plotting
   - ggplot2 Plotting
   - ggvis
   - plotly
4. Exploring ggplot2
   - Background
   - Layering
   - Histogram
   - Boxplot
   - Themes

Univariate **Visual** Analysis

"The greatest value of a picture is when it forces us to notice what we never expected to see."
— John Tukey in Exploratory Data Analysis (1977)

A sample data generation

```
set.seed(2016) # Set Seed for reproducibility
n = 1e4        # Number of observations

(n*2) %>%      # Generate some data
  rnorm %>%
  matrix(ncol = 2) -> a

runif(n, 0, 2 * pi) %>%
  {0.5 * cbind(sin(.), cos(.))} -> b

o = rbind(a,b)    # Combine generate data

x = as.data.frame(o[sample(nrow(o)), ])

colnames(x) = c("x","y")
```

Numerically we have...

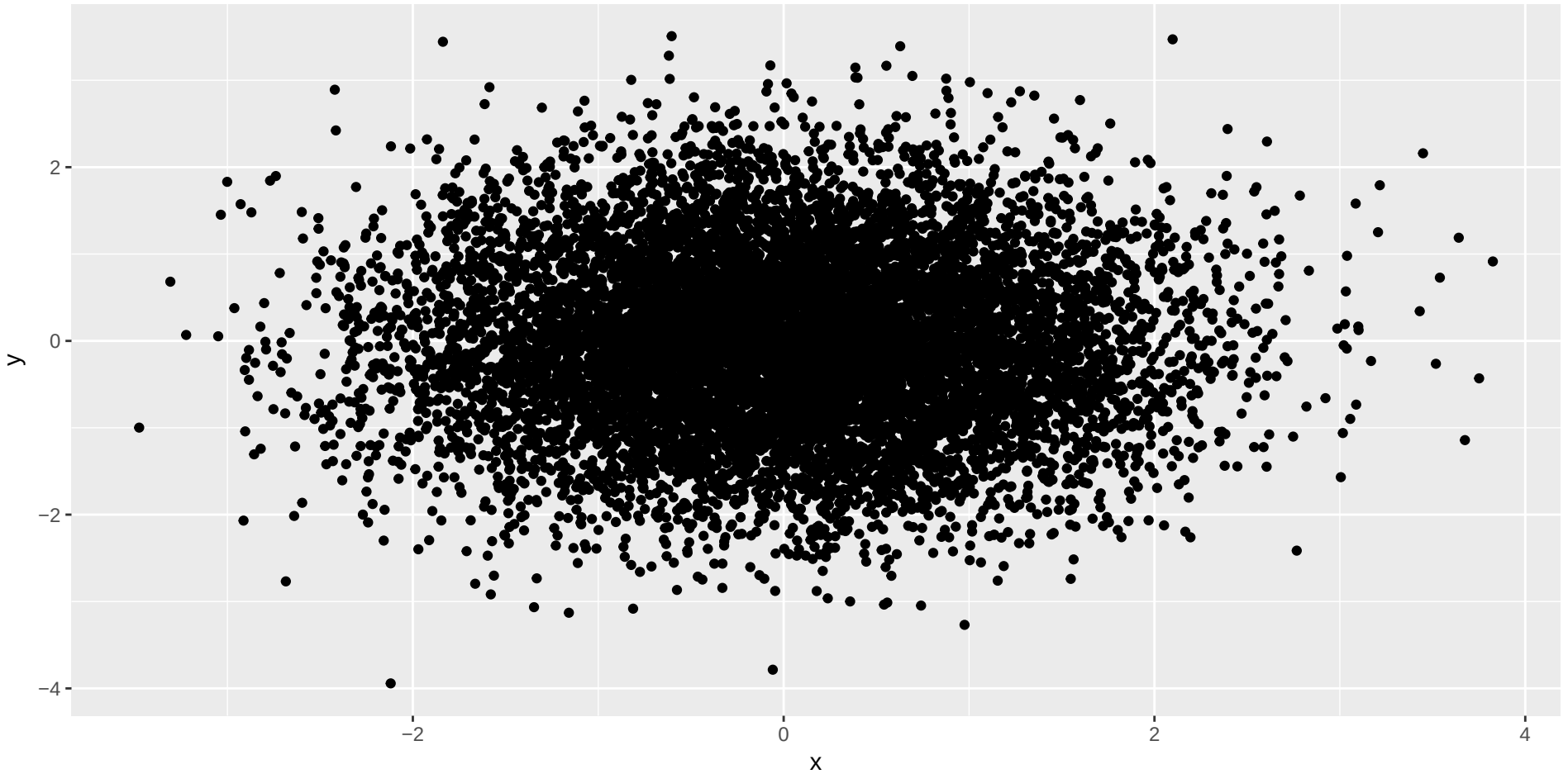
```
summary(x)     # data.frame implements summary.
```

```
##        x                       y
##  Min.   :-3.476111   Min.   :-3.94248
##  1st Qu.:-0.436574   1st Qu.:-0.43595
##  Median : 0.005087   Median : 0.00629
##  Mean   :-0.000790   Mean   : 0.00137
##  3rd Qu.: 0.430879   3rd Qu.: 0.43412
##  Max.   : 3.825197   Max.   : 3.50801
```

Insight: Data looks to be bounded between -4 and 4.

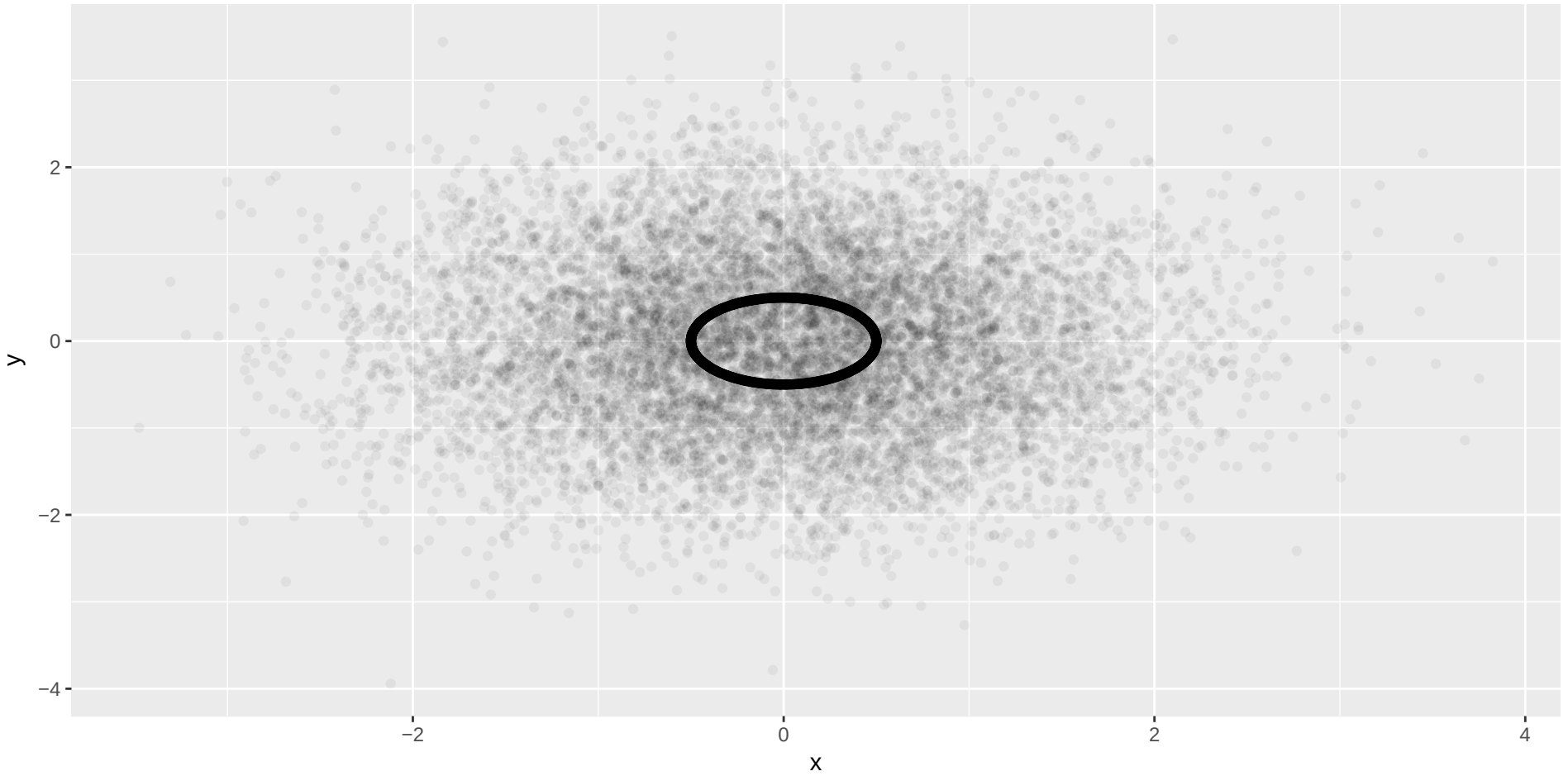
Graphically we have . . .

```
ggplot(x) + geom_point(aes(x,y))
```



Redux of Graphically we have . . .

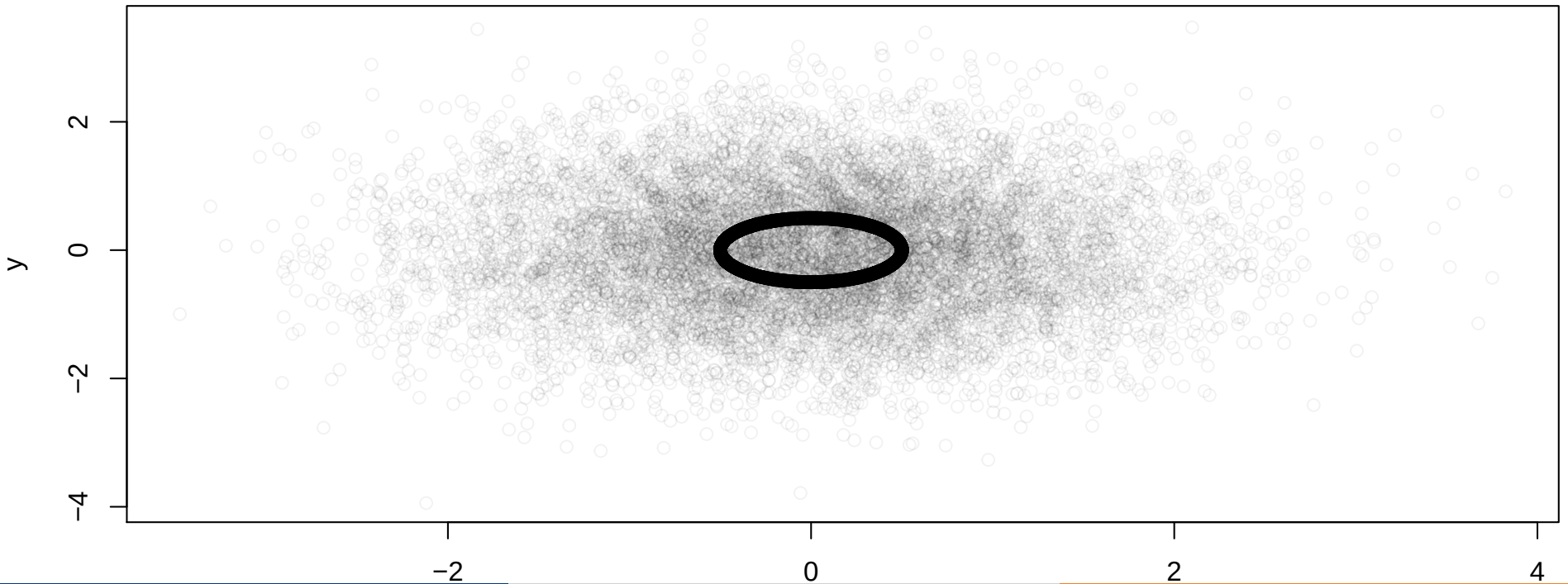
```
ggplot(x) + geom_point(aes(x,y), alpha = 0.05)
```



A note...

- Notice in the previous slides, there was no call to `plot()`.
- Instead, `ggplot()` was used to create the graphic through the use of layering via the + symbol.
- To do the same with base *R*, we would of used:

```
plot(x, col = rgb(0, 0, 0, 0.05)) # Transparent color
```



Graphing in *R*

- Before now, we never really focused on plotting.
- Instead, we aimed to understand the computing logic behind calculations in *R*.
- Now, to support visual *EDA*, we really need to start focusing on such features.

R and the Three Graphing Systems

- Dilemma: There are ***3*** graphing systems to chose from in *R*.
- Similar to the **Goldilocks and the Three Bears** problem.

Figure 1

- Selecting the graphical system is important. . .

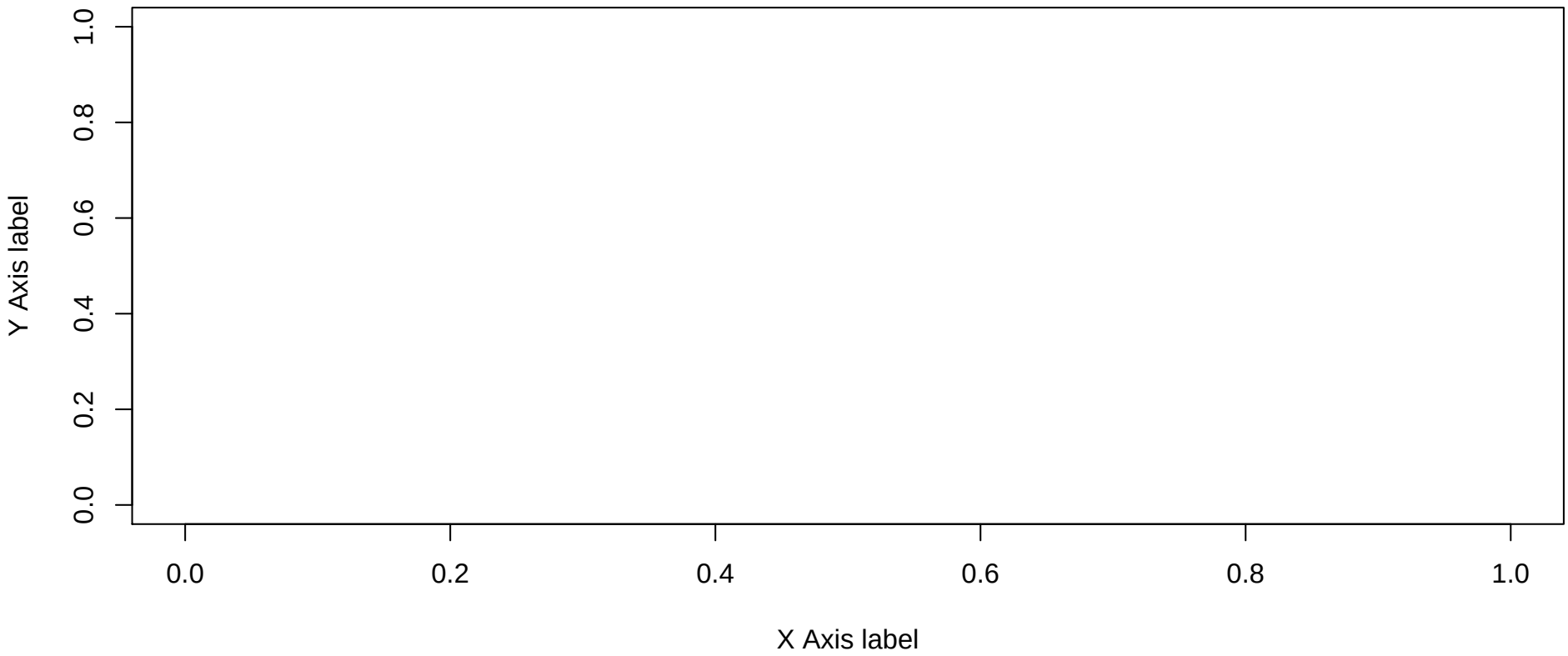
Different Plotting Systems in *R* - Overview

- R's `Base` plotting system.
  - `plot()`, `hist()`, `barplot()`
- `lattice` formulaic graphing in *R*.
  - `xyplot()`, `dotplot()`, `histogram()`, `*plot()`
- `ggplot2` rapid layered graphing approach
  - graphs start with `ggplot()` and add layers via + typically denoted by `geom_point()` , `geom_*()`

R's Base Plotting System - Example

- View it as an artists blank canvas

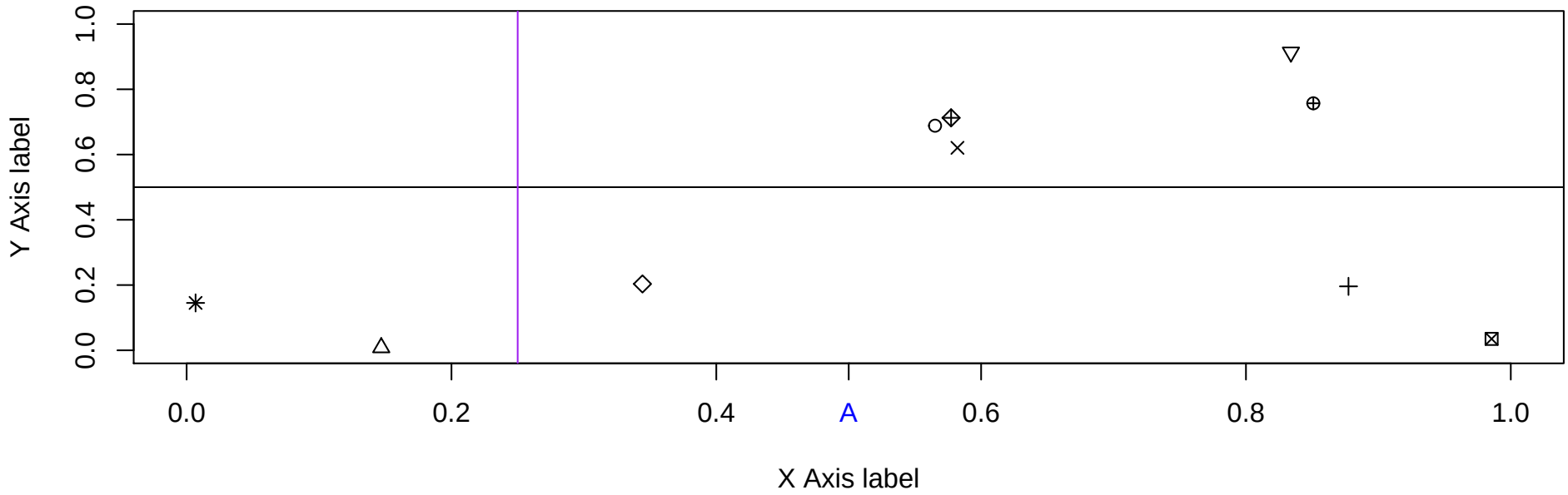
```
plot(NULL, xlim=c(0,1), ylim=c(0,1),
     ylab="Y Axis label", xlab="X Axis label")
```



R's Base Plotting System - Example

- Each subsequent function calls adds lines, points, axis, et cetera.

```
x = runif(10); y = runif(10)
abline(h = .5)                    # Horizontal Line
abline(v = .25, col="purple")     # Vertical Line w/ color
points(x, y, pch = 1:10)          # Points w/ shapes
axis(1, .5, LETTERS[1], col.axis = "blue")
```



R's Base Plotting System - Verdict

- **Con:** No ability to change plot settings (e.g. `?par` settings) or draw content added once started.
- **Pros:** Easier custom graphs and higher quality graphs (e.g. AVLR using the `tikzDevice` package)
- **Verdict: Academics only**

lattice - A formulaic approach to graphs.

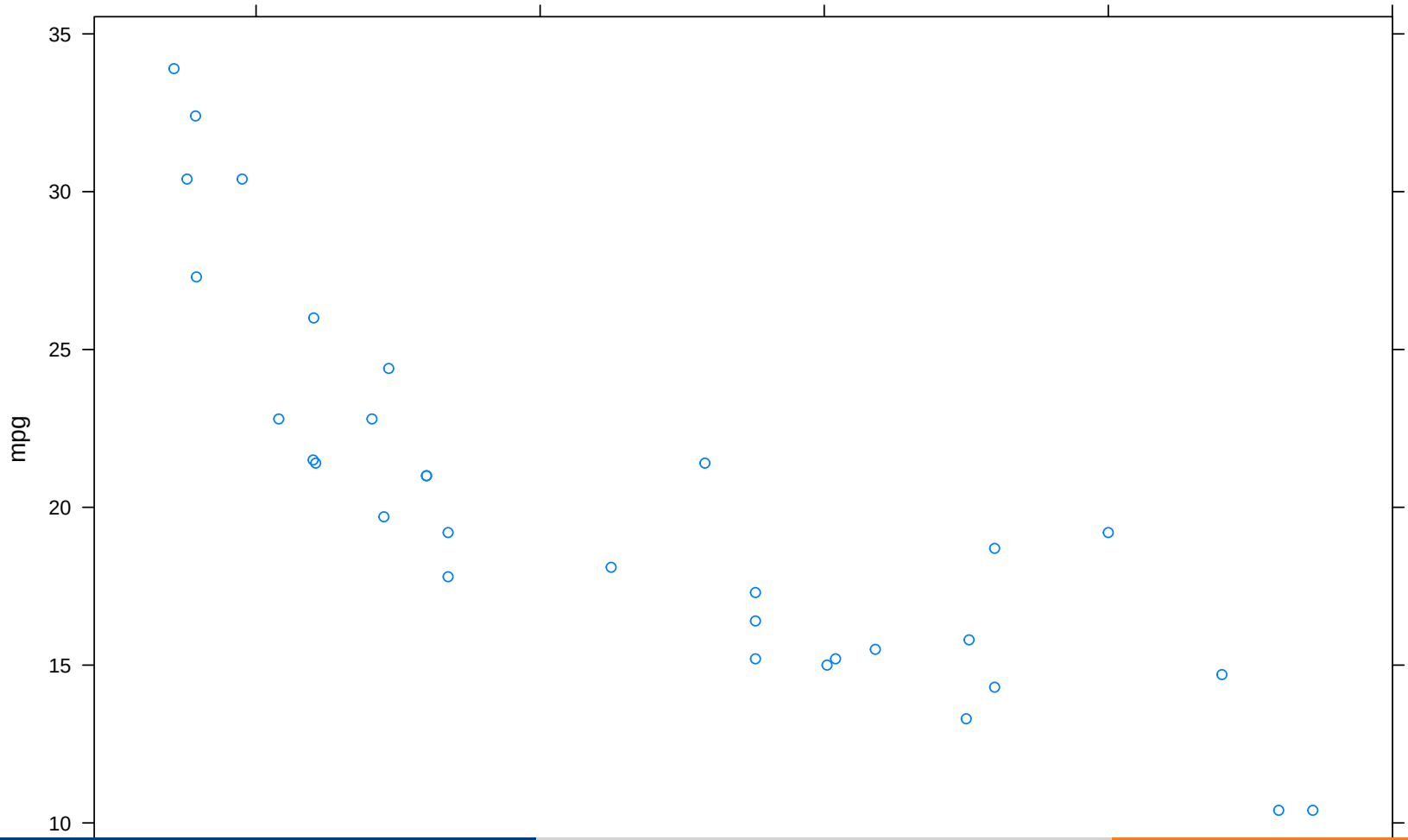
- The lattice package written by Deepayan Sarkar provides the ability to make graphs in *one* call vs. Base R's multiple calls.
- The call form is normally:

```
type_of_plot(formula, data=list())
```

- Uses the formula object associated with lm to specify: *response* (y~), *explanatory* (~x), *conditional relationships* (y~x|A).
- Great for viewing conditional relationships and multivariate data.

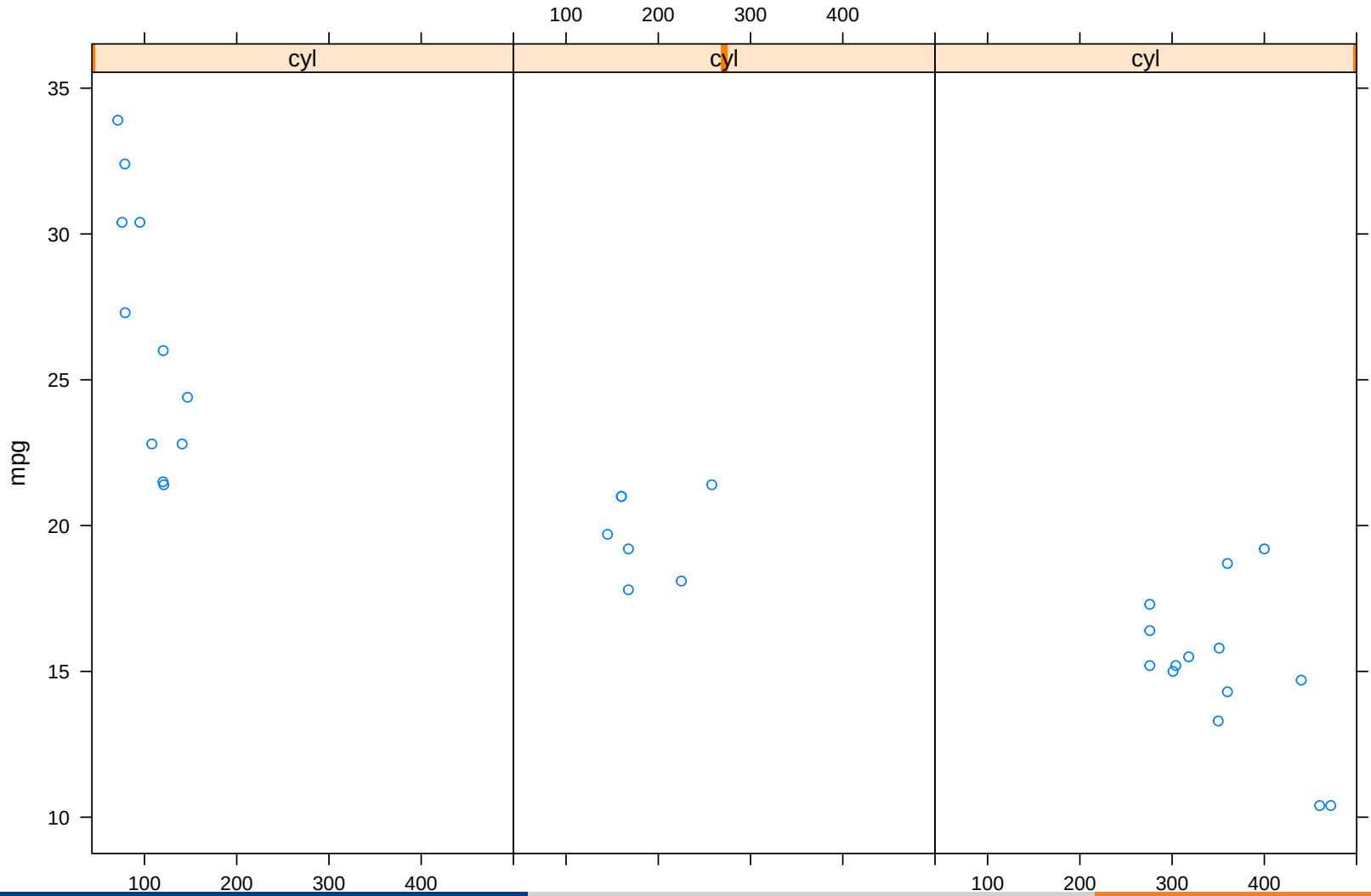
lattice - Example

```
library("lattice")
xyplot(mpg ~ disp, data = mtcars)
```



lattice - Example with Condition

```
xyplot(mpg ~ disp | cyl, data = mtcars) # Note the |
```



lattice - Verdict

- **Cons:** Everything in 1 function call is *messy* and awkward.
- **Pros:** Handle all margin settings of multiple graphs and conditioning.
- **Verdict: Casual *R* users.**

Credit: The Global Warmers

ggplot2 - Grammar of Graphics

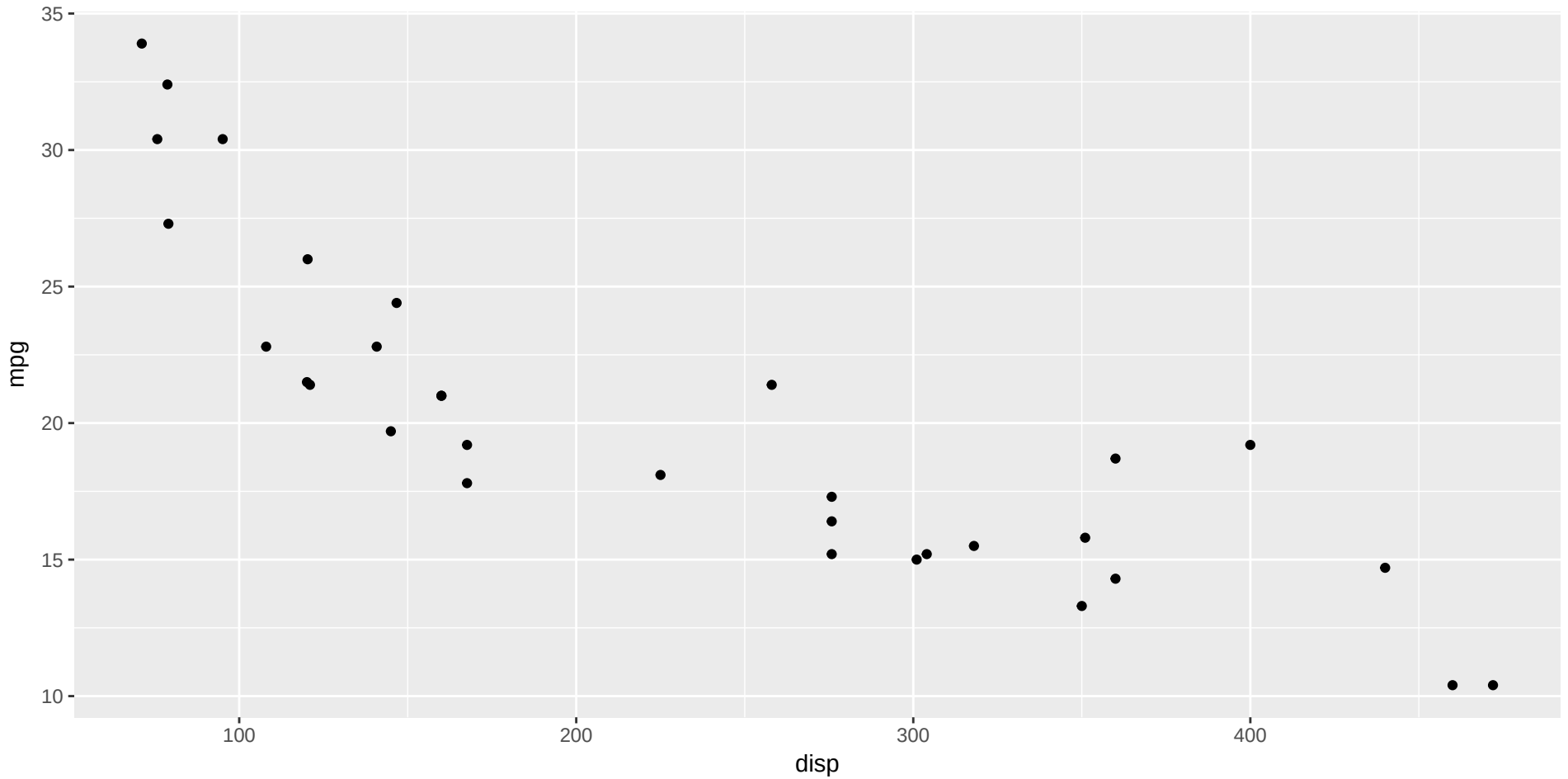
- ggplot2 is the implementation of the pivotal 1999 Book Grammar of Graphics by Leland Wilkinson.
  - Each Graph shares a common structure.
  - The difference between graphs is different component layers and rules.

ggplot2 - Grammar of Graphics

- Historical Information
  - ggplot1 written by Hadley Wickham as part of his PhD thesis.
  - ggplot2 released for ease of use alongside A Layered Grammar of Graphics
  - UseR 2016 Keynote: ggplot1 is better than ggplot2 API wise due to the piping operator
    - Time: 36:38 to 38:32

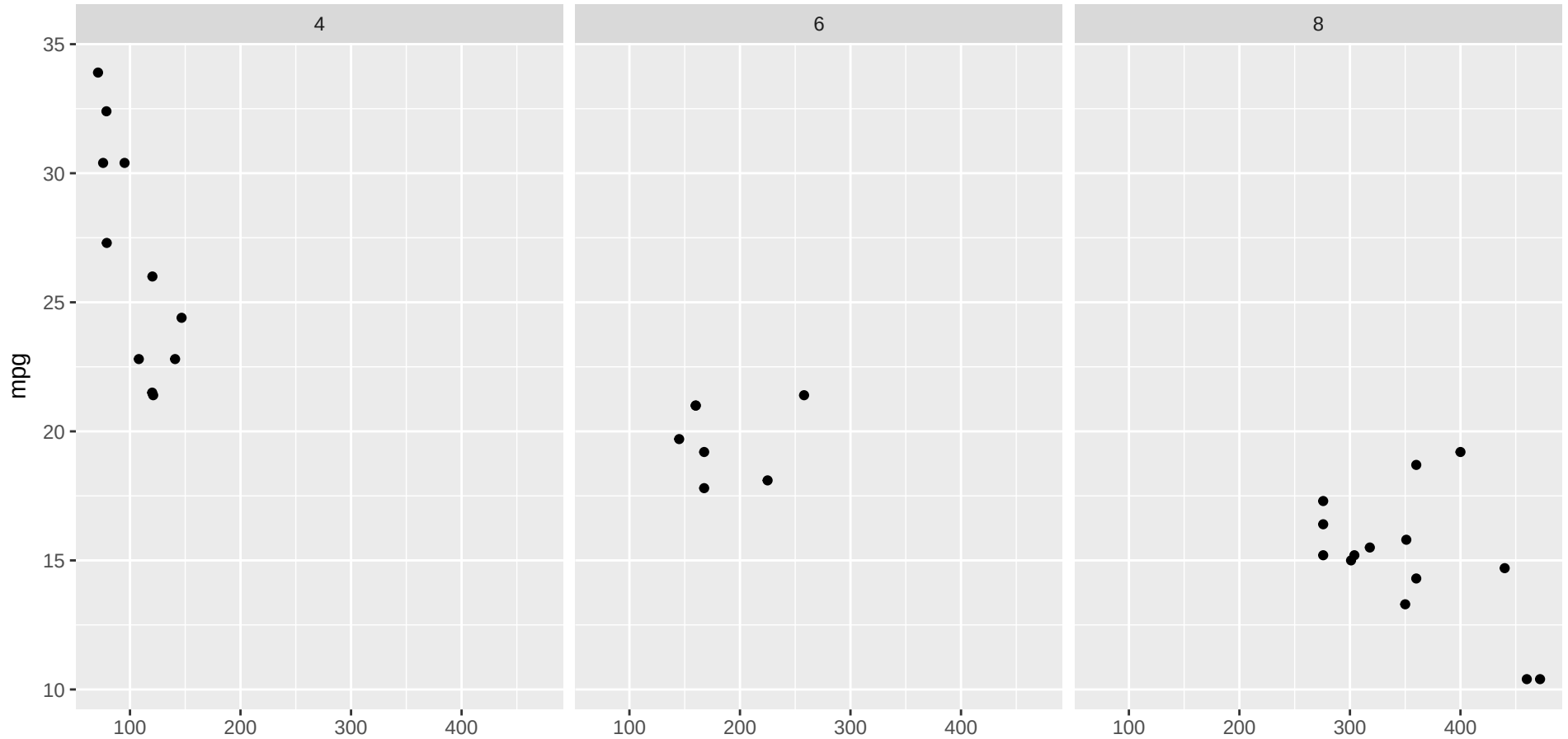
ggplot2 - Scatterplot

```
ggplot(mtcars) +                    # Supply data.frame
  geom_point(aes(disp, mpg))        # Add points to plot
```



ggplot2 - Scatterplot Conditioned

```
ggplot(mtcars) +                    # Supply data.frame
  geom_point(aes(disp, mpg)) +      # Add points to plot
  facet_wrap(~cyl)                  # Write conditioning
```



ggplot2 vs. Base R

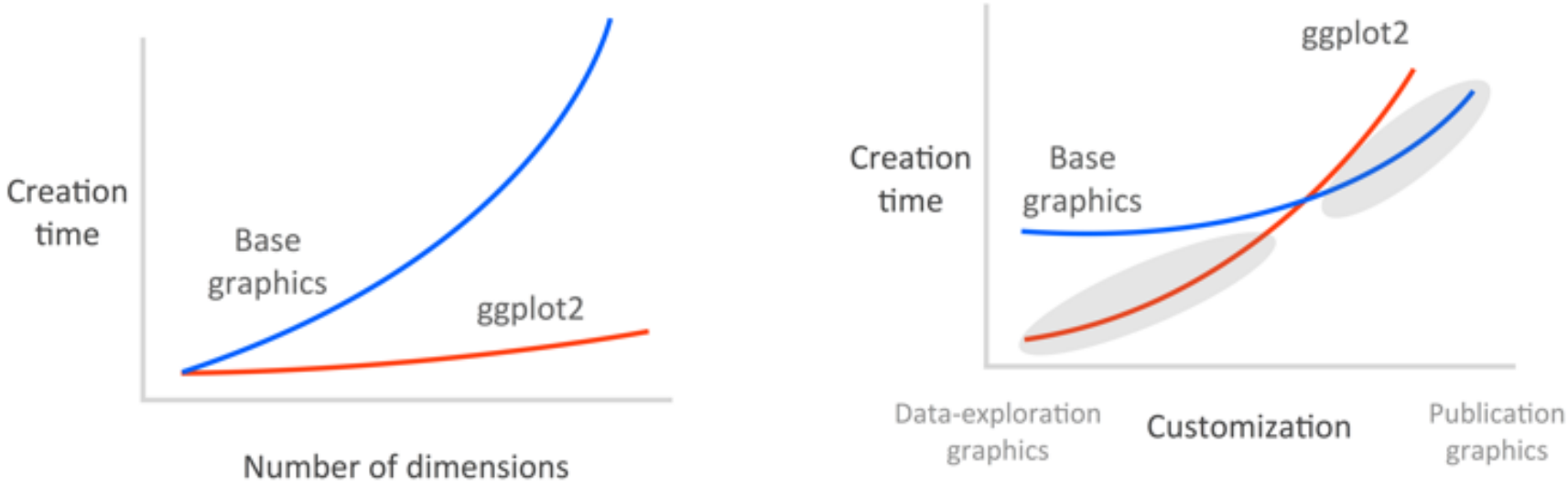


Figure 2: Credit: Sean C. Anderson

ggplot2 - Verdict

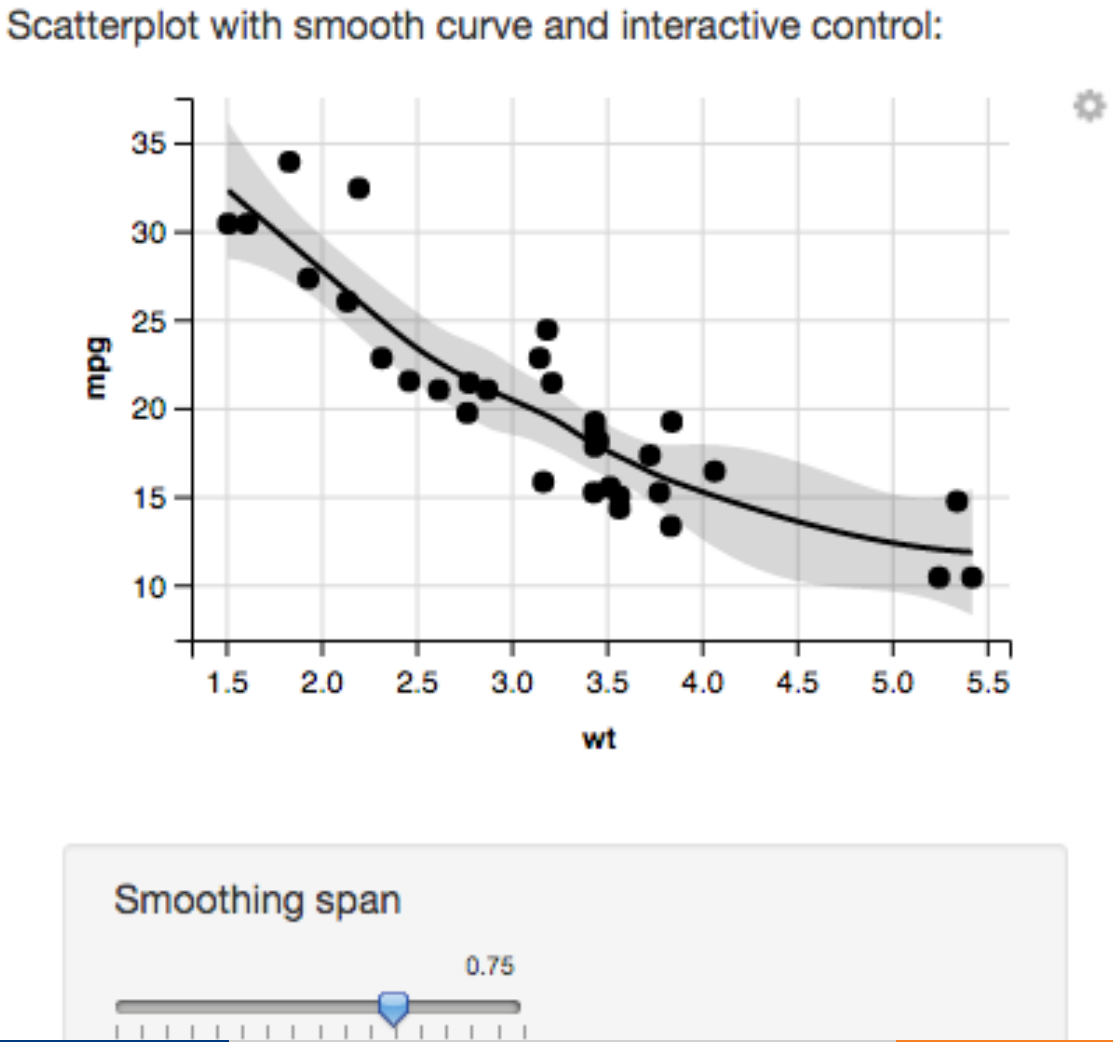
- **Cons:** Data must be in a `data.frame`, global scoping of variables, data copies, and simple things might be *complex*.
- **Pros:** Rapidly iterate visualizations, grammatical structure, and extendable graphing system.
- **Verdict: Data Scientists, Researchers, and Causal *R* users.**

I lied...

- As is the case with technology, there always a new graphing system around the corner.
- Coming Soon a 4th Graphical System for *R* using the parts of `ggplot2`....

ggvis - Coming Soon (TM)

- Introducing ggvis, the successor to ggplot2...

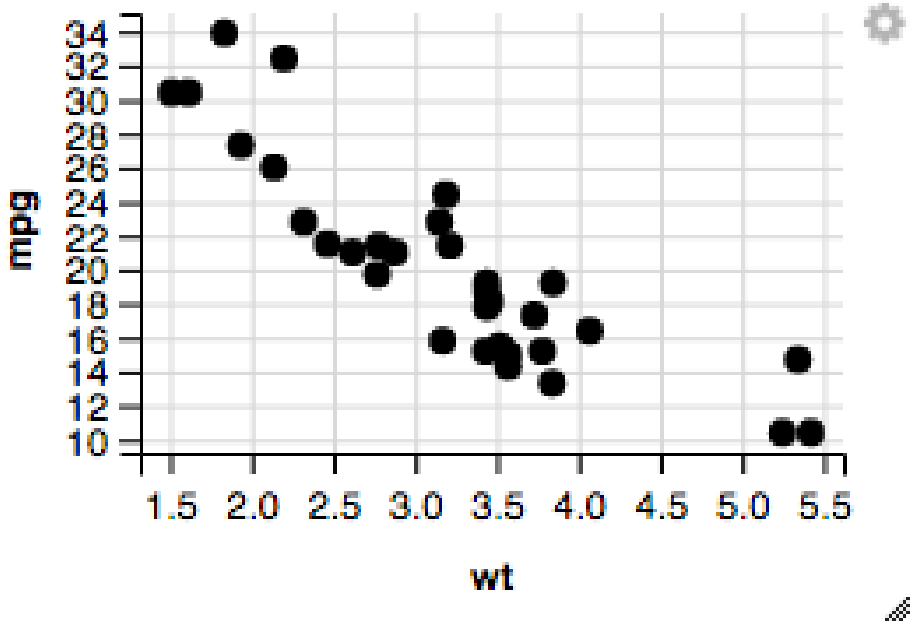


ggvis - Coming Soon (TM)

- `ggvis` will usher in a new way of interactive graphics (think `identify()`). Keep an eye on this project.
- Alas, as the API is currently in flux, we will not dedicate any time to it. However, note that `ggvis` replaces `ggplot2`'s concatenation with the %>% operator.

ggvis - Example

```
# install.packages("ggvis")
library("ggvis")
mtcars %>% ggvis(~wt, ~mpg) %>% layer_points()
```



Note: This code can only be run in *HTML* rich environments. No LaTeX environments need apply.

plotly and ggplotly

Bringing interactivity to present day *R* graphs

On the Agenda

1. EDA
   - Background
   - Verify Data
2. Quantitative
   - Types
   - Numeric
   - 5 Summary Statistics
   - Median
   - Categorical
3. Visual
   - A picture. . .
   - Base R Plotting
   - lattice Plotting
   - ggplot2 Plotting
   - ggvis
   - plotly
4. Exploring ggplot2
   - Background
   - Layering
   - Histogram
   - Boxplot
   - Themes

Visual EDA

"Use a picture. It's worth a thousand words"
— Tess Flanders in Speakers Give Sound Advice

Data Wrangling `birth` from `msos` package.

For the next section, I'll aim to use the `birth` data from `msos`. Note, the data is in **wide** form in a `matrix`. The below script sets up the data for graphing by converting it to **long** form and class `data.frame`.

```
# Extract hospital birth dates
data(births, package="msos")

library("tidyr")
df_births = as.data.frame(births)
df_births$time = seq_len(nrow(df_births))
long_births = gather(df_births, hospital, value, -time)
```

Looking into long_births

Let's peek at what the data in long_births looks like.

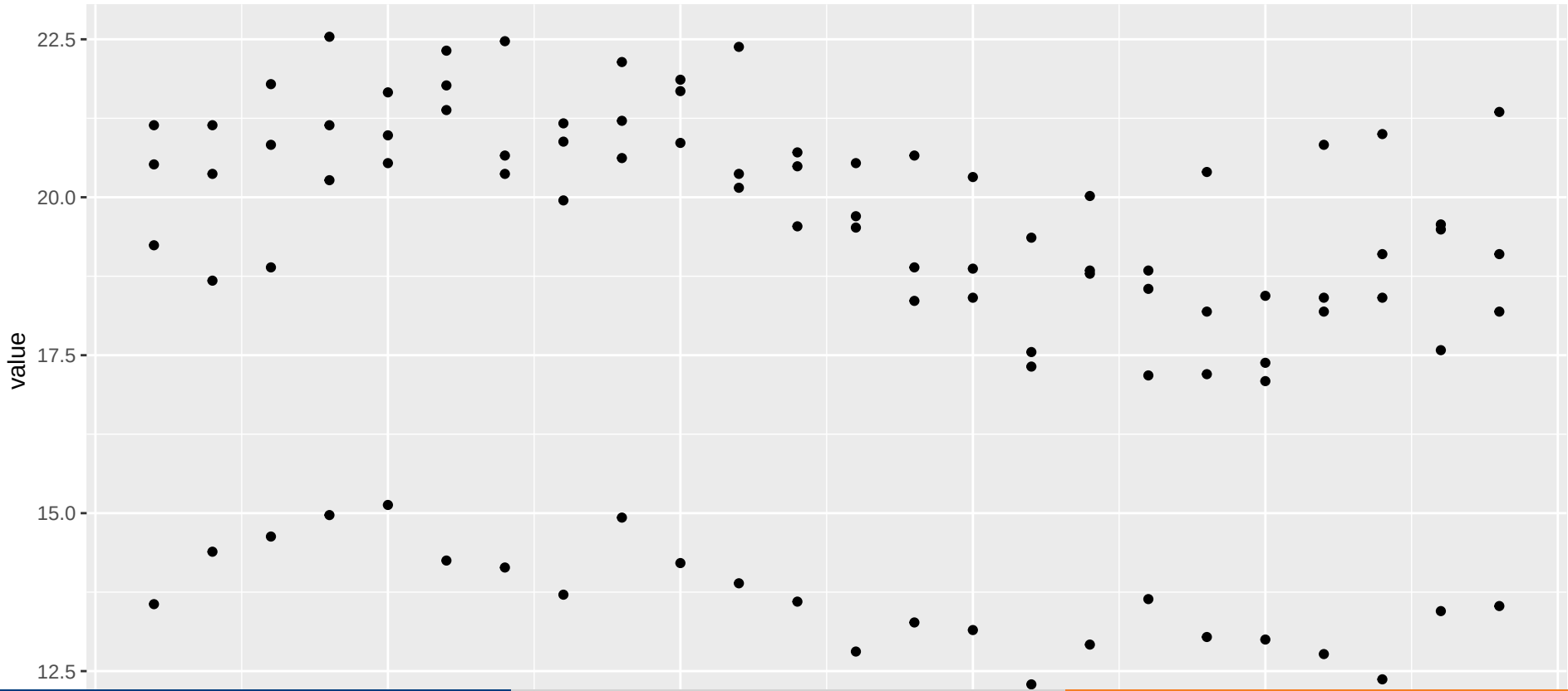
```
head(long_births)
```

```
##   time  hospital value
## 1    1 Hospital1 13.56
## 2    2 Hospital1 14.39
## 3    3 Hospital1 14.63
## 4    4 Hospital1 14.97
## 5    5 Hospital1 15.13
## 6    6 Hospital1 14.25
```

Note: long_births is of class data.frame!!

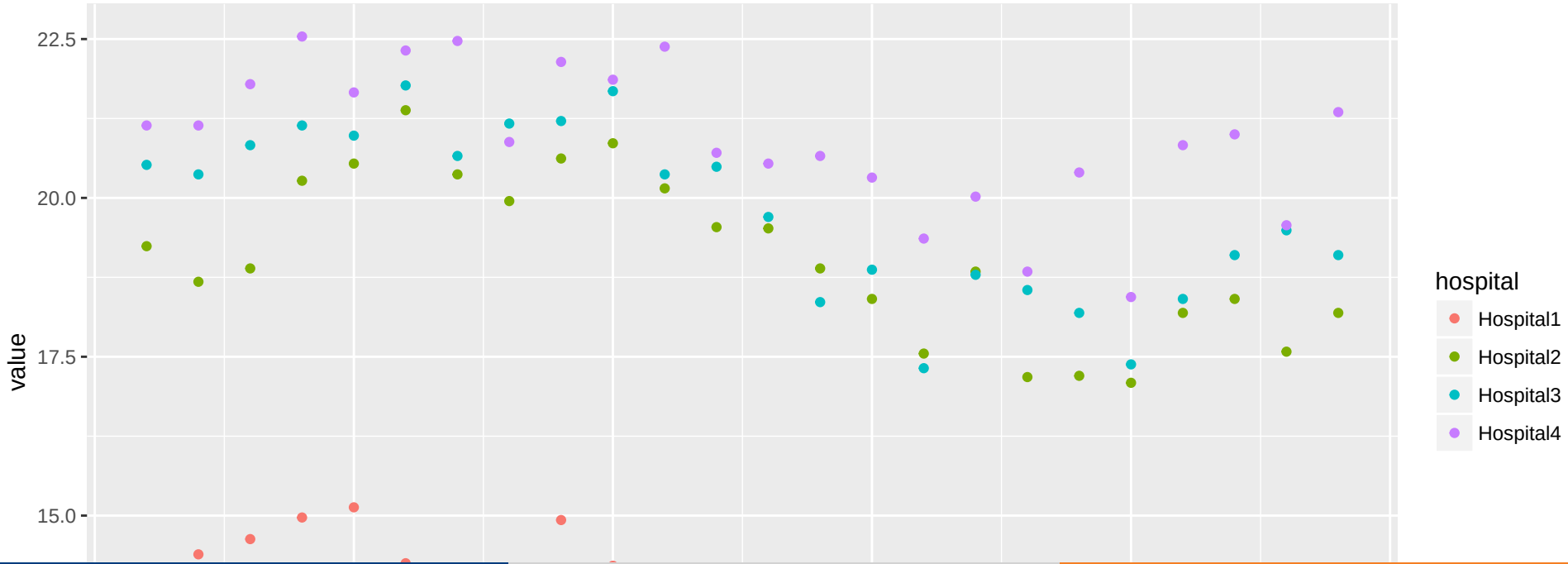
Your first ggplot!

```
ggplot(long_births) +          # Initialize ggplot w/ data
  geom_point(                  # Add a point layer
    aes(x = time, y = value)# Add an aesthetic mapping
              )
```



Your second ggplot!

```
ggplot(long_births) +     # Initialize ggplot w/ data
  geom_point(             # Add a point layer
    aes(x = time,         # Add an aesthetic mapping
        y = value,
        color = hospital)# Added color
          )
```



Key Terms and Ideas with ggplot2

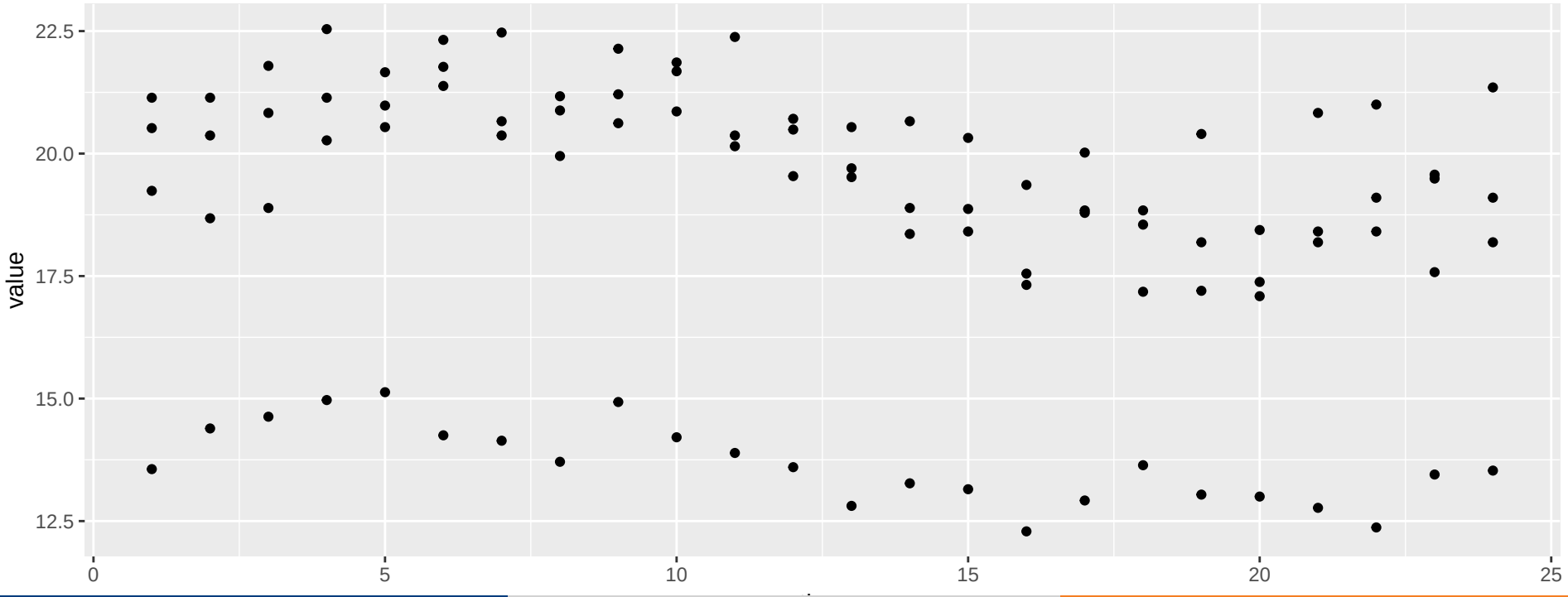
- **ggplot**: Initialization function of the graph
- **geom_**: Geometric (shape) objects
- **aes**: Provides the aesthetic options the `geom` should take.
  - Examples: color, fill, transparency (alpha), linetype, and point shape.
- **scales**: Specifies the type of `axis` to use (very problematic)
  - Examples: Continuous, Discrete, log , $\sqrt{}$, and so on.
- **facet**: Panel layout
  - Examples: Grid ($x \times y$) or Wrapped
- **stats**: statistical methods (e.g. regression, splines, binning)

Reusing ggplot2 base objects

Each ggplot2 object can be saved individually and added to in the future

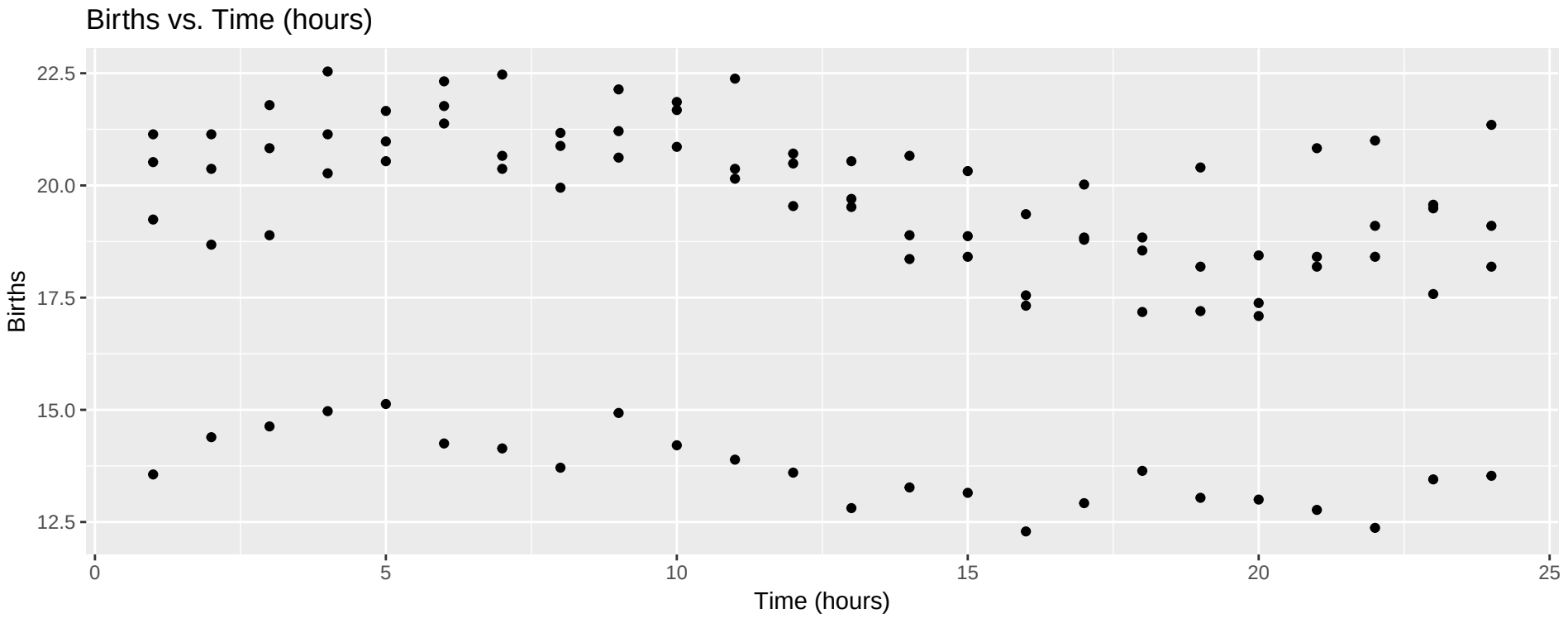
```
g = ggplot(long_births) +
  geom_point(aes(x = time, y = value))

g
```



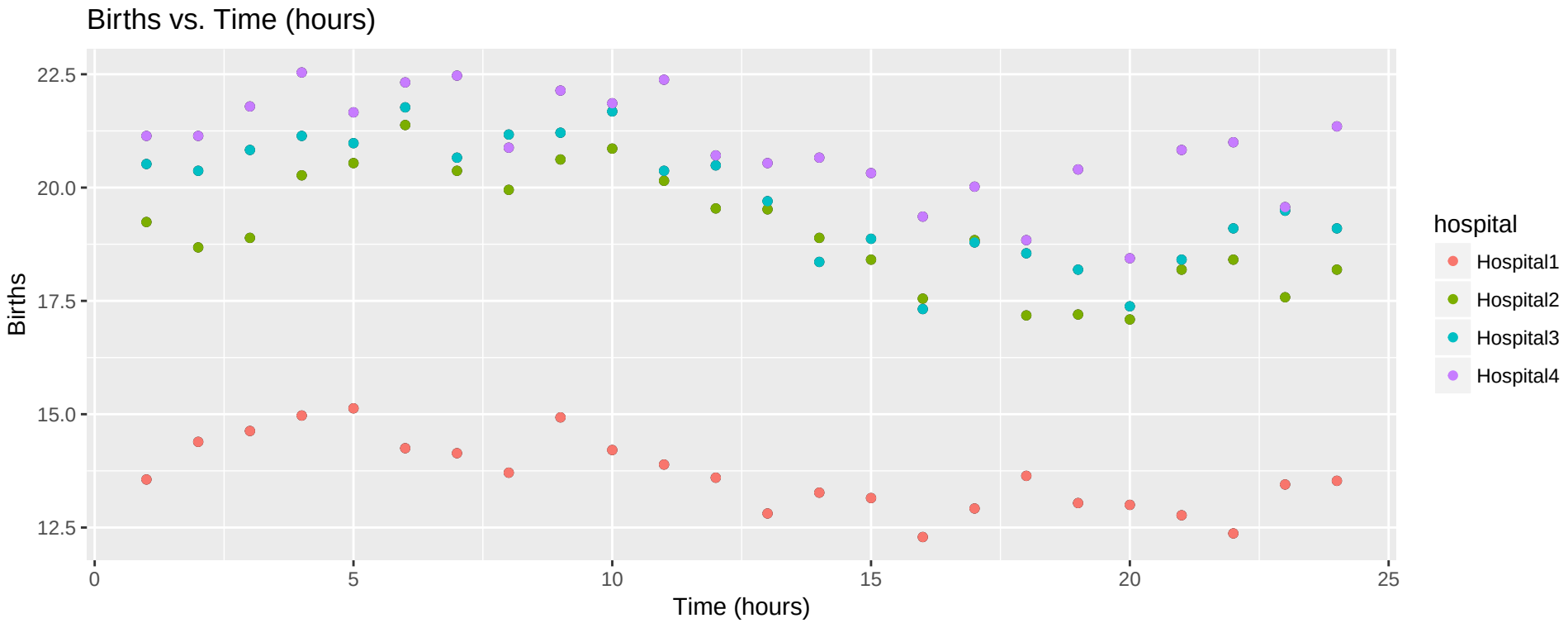
Adding Label Information to ggplot2

```
(g = g + xlab("Time (hours)") + ylab("Births") +
  ggtitle("Births vs. Time (hours)"))
```



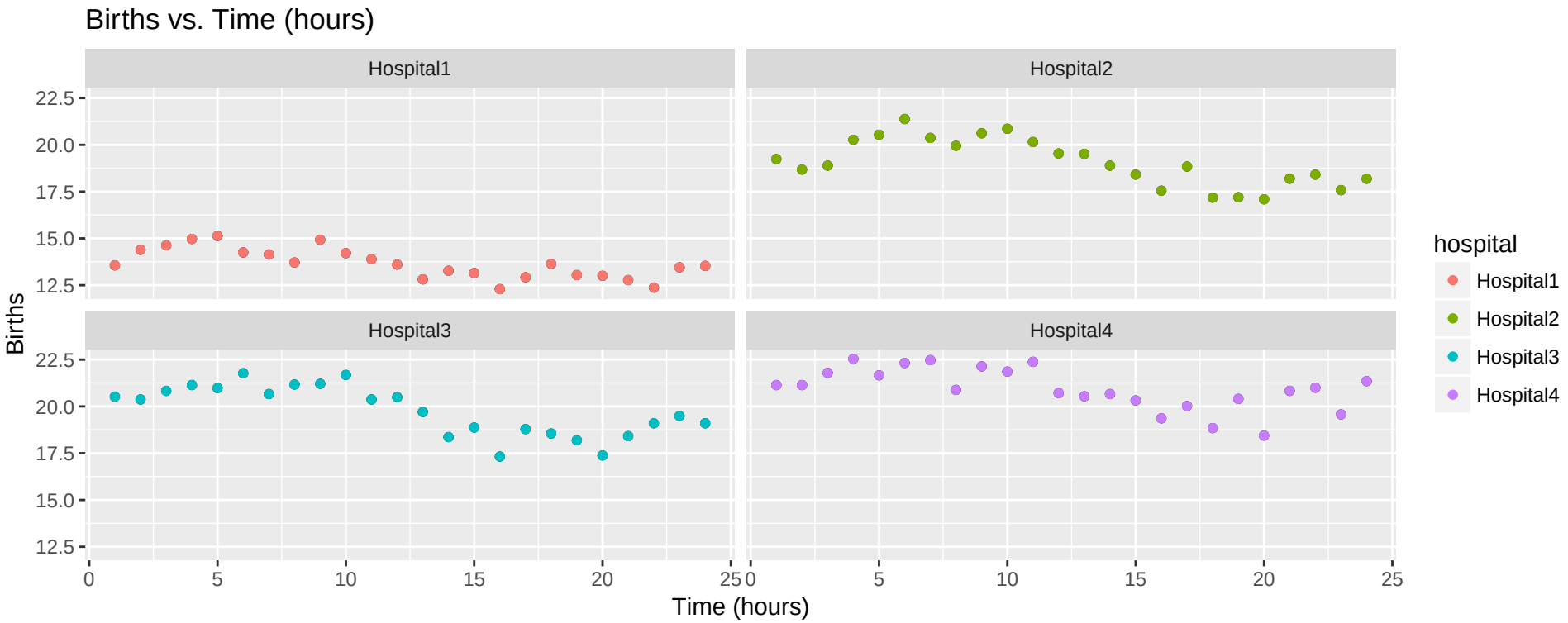
Changing aes for geom_point

```
(g = g + geom_point(aes(x = time, y = value, color = hospital)
```



Adding a `facet_wrap` to distinguish variables

```
(g = g + facet_wrap(~hospital))
```



Graphing with ggplot2

- ggplot2 makes available various geometric objects via geom_.
- These objects determine how the data is rendered on the plot.
- Some of the geoms_*() typically used:

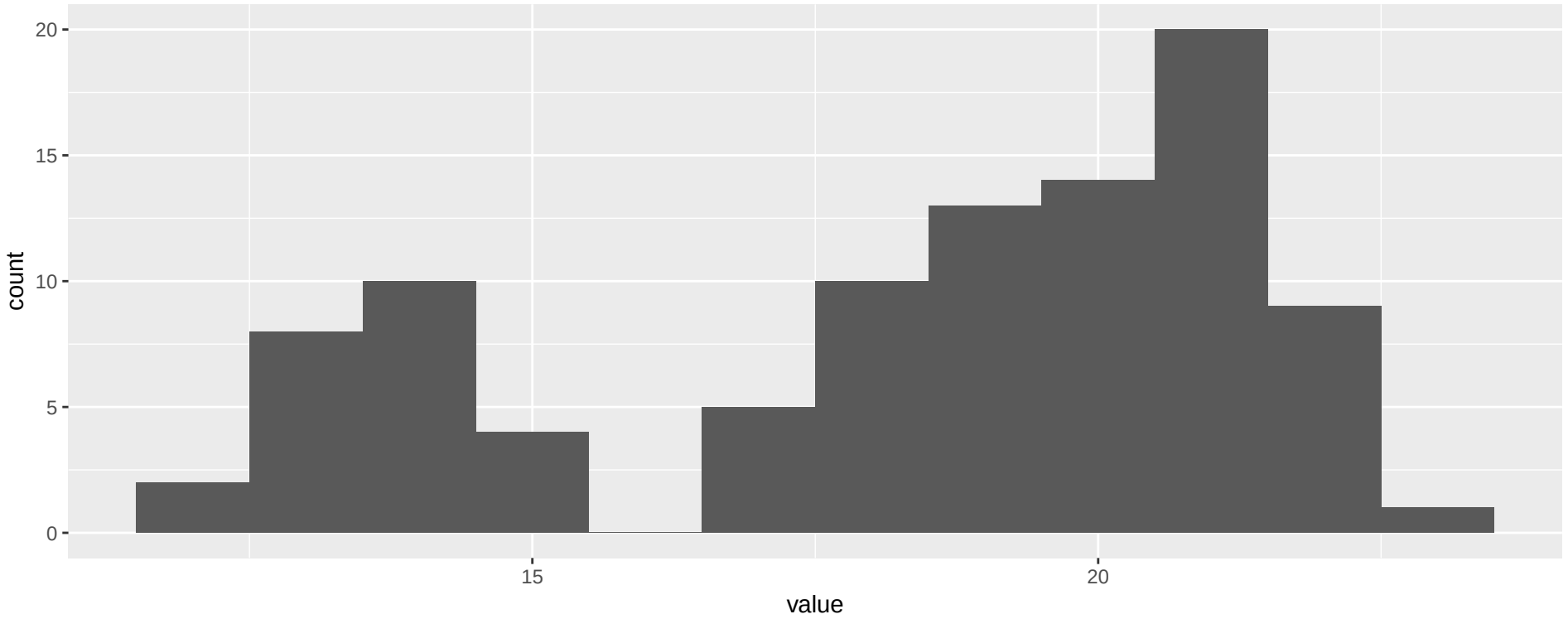
geoms_*()	Description
geom_point()	Adds data points to plot
geom_line()	Adds connected lines to the plot
geom_histogram()	Makes a histogram
geom_bar()	Creates a bar chart
geom_text()	Adds text annotations
geom_violin()	Makes a violin plot

- Many more geoms_*() exists and can be found at docs.ggplot2.org with graphing examples!

Making a histogram

Histograms typically provide count or frequency values on the y-axis

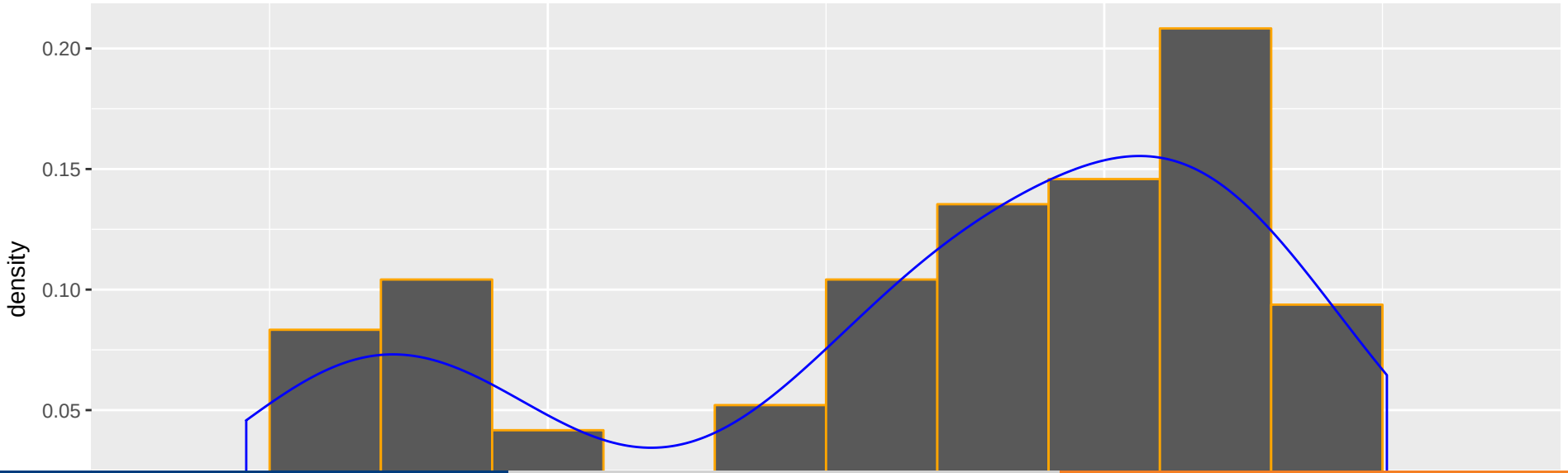
```
ggplot(long_births) +
  geom_histogram(aes(value), binwidth = 1)
```



Making a histogram with a density plot

Density plots alongside a histogram require `density` (bounded between 0 and 1) to be on the y-axis. If `count` is on the y-axis, then results are not valid.

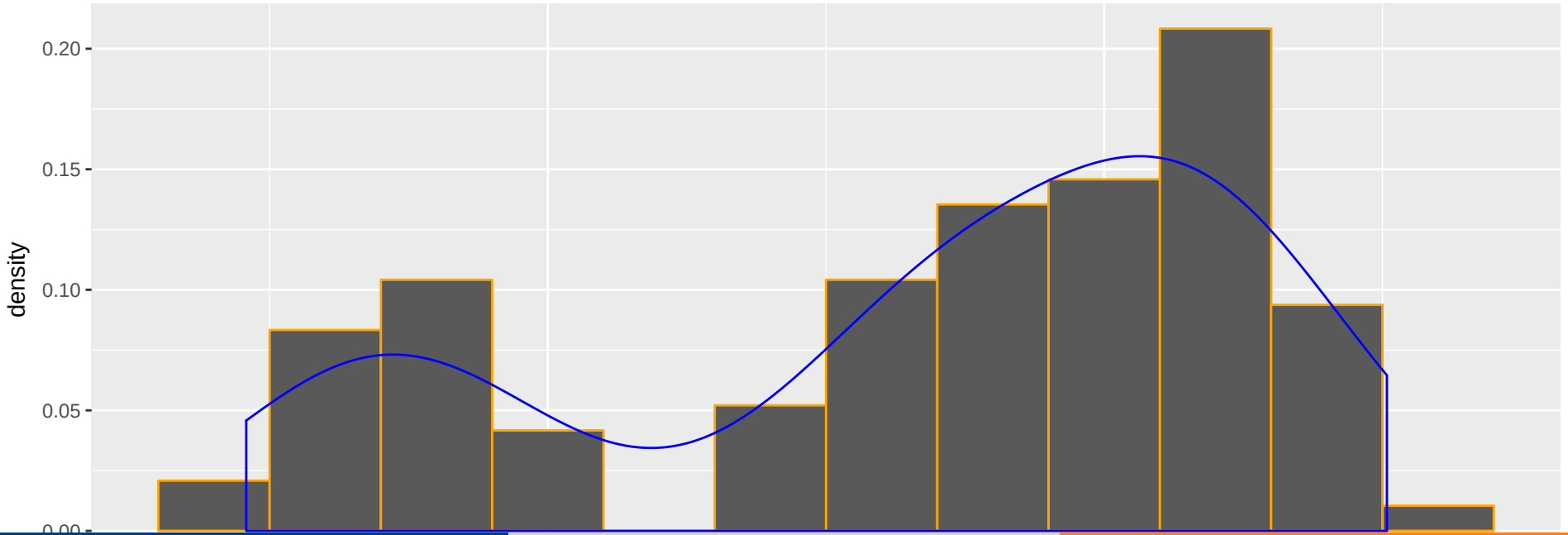
```
ggplot(long_births) +
  geom_histogram(aes(x = value, y = ..density..),
                 binwidth = 1, color = "orange") +
  geom_density(aes(value), color = "blue")
```



Storing aes in construction

Specify `aes()` does not necessarily have to be done in the `geom_*()` call. Some users prefer to specify the relationship in the `ggplot()` creation.

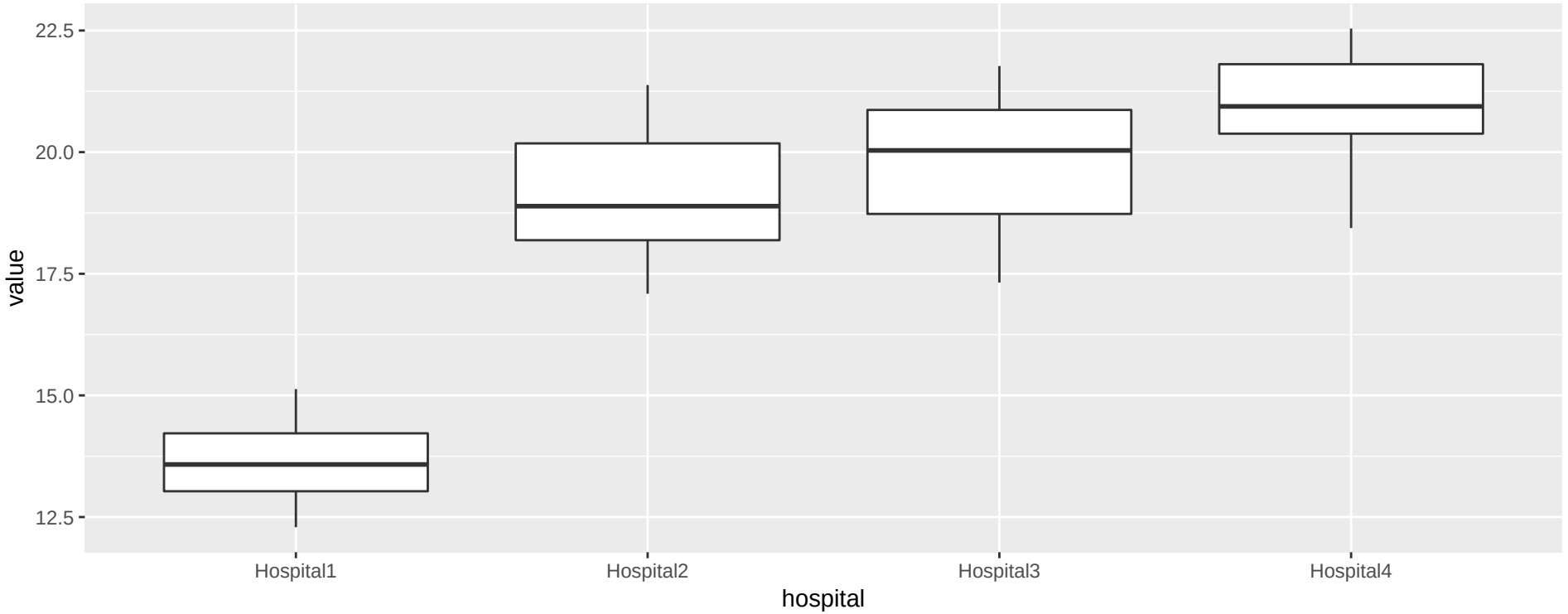
```
ggplot(long_births, aes(x = value)) +
  geom_histogram(aes(y = ..density..), # Notice no `x=`
                 binwidth = 1, color = "orange") +
  geom_density(color = "blue")          # Notice no `x=`
```



Boxplot

Boxplot are a helpful way to visualize Q1, Q2, Q3, and outlier information. They are may be referred to as a *box and whiskers* plot.

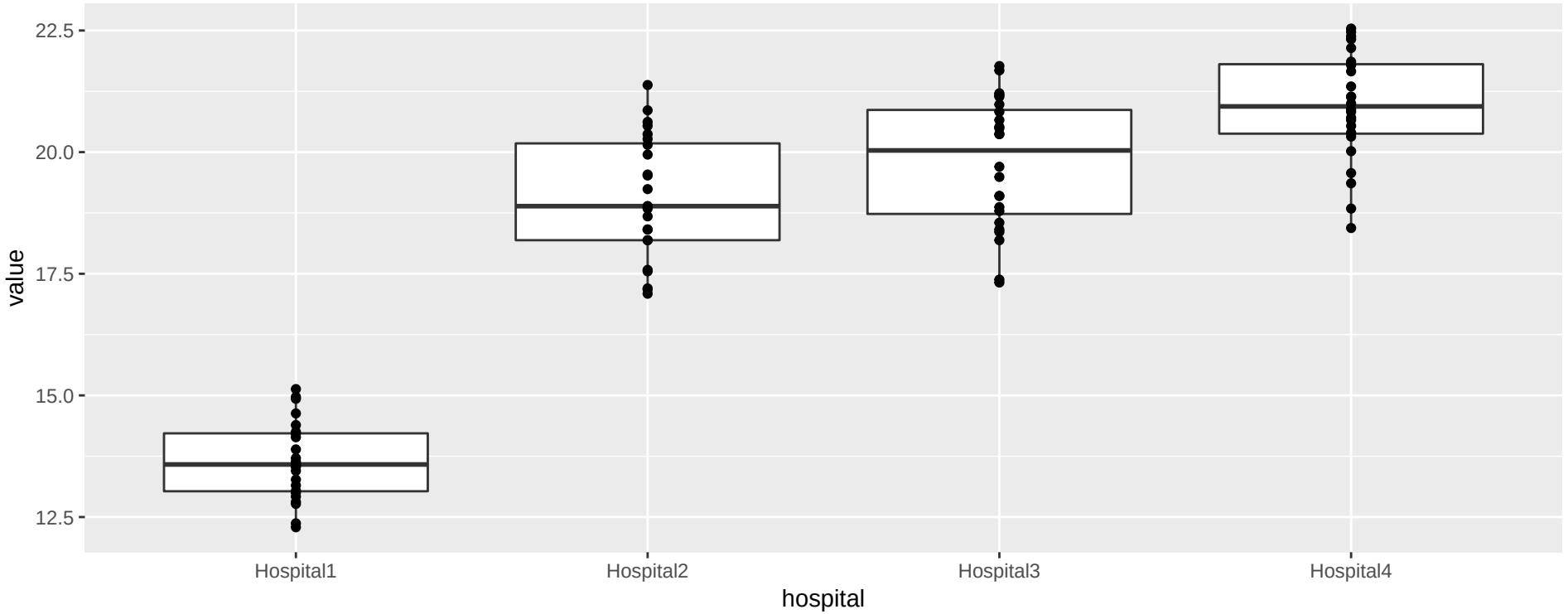
```
ggplot(long_births, aes(x = hospital, y = value)) +
  geom_boxplot()
```



Boxplot with Points

What is nice, is instead of only seeing outliers, you can also see where all the points lie just by adding `geom_point()`

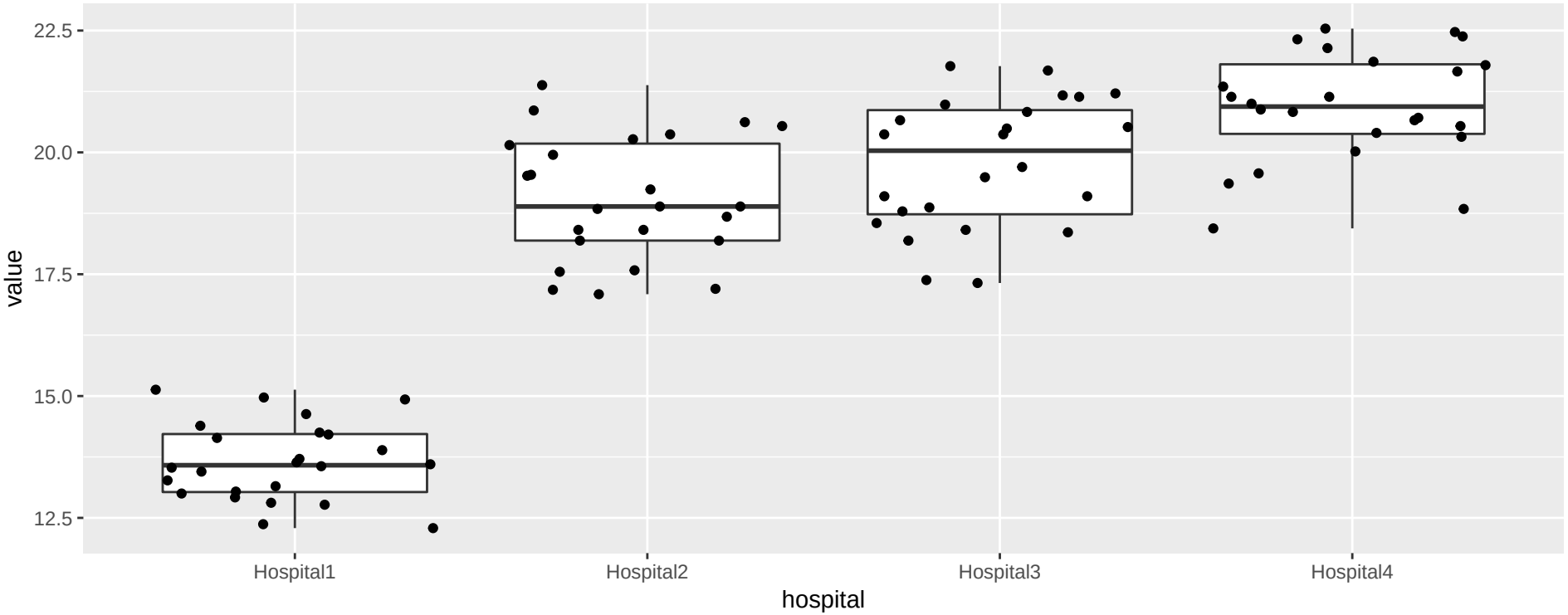
```
ggplot(long_births, aes(x = hospital, y = value)) +
  geom_boxplot() + geom_point()
```



Boxplot with Points Redux

However, adding points without *jittering* them will lead to non-informative clumping. To avoid this, use a jitter: `geom_jitter()`

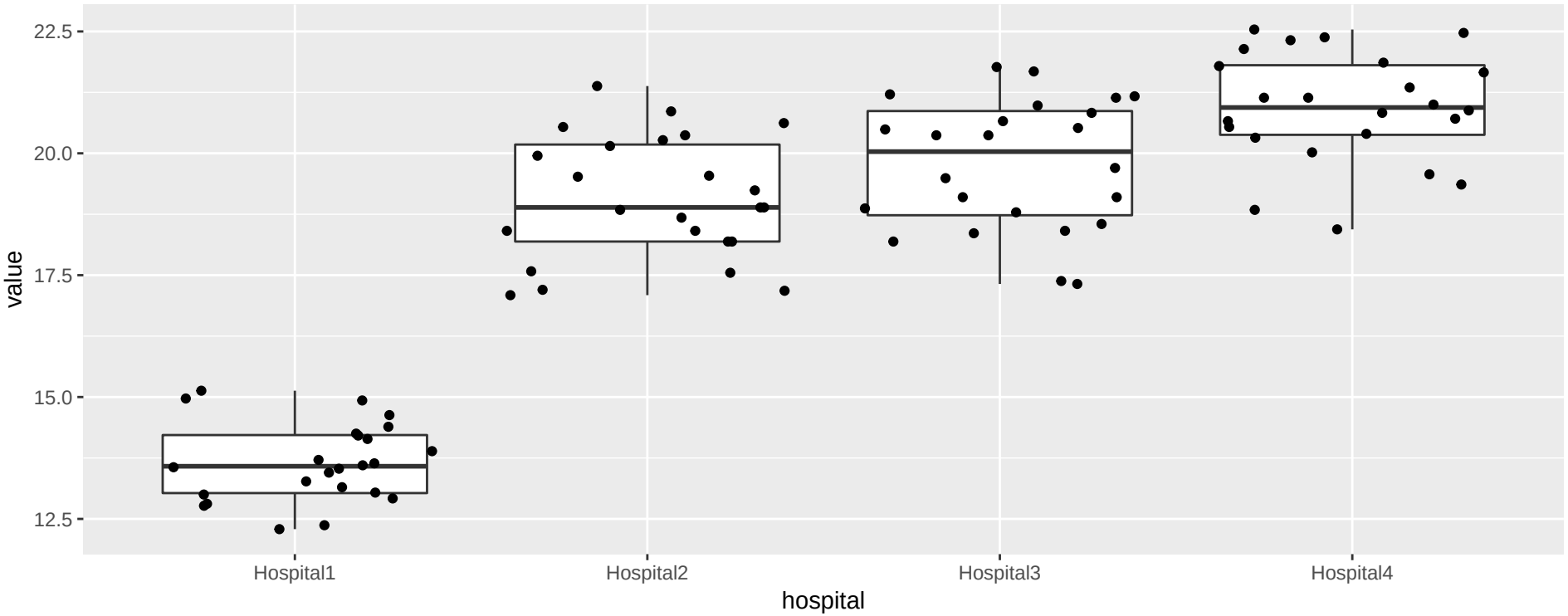
```
ggplot(long_births, aes(x = hospital, y = value)) +
  geom_boxplot() + geom_jitter(height = 0, width = 0.4)
```



Flipping My Box

Note: The jitter is applied at random (switch between slides)...

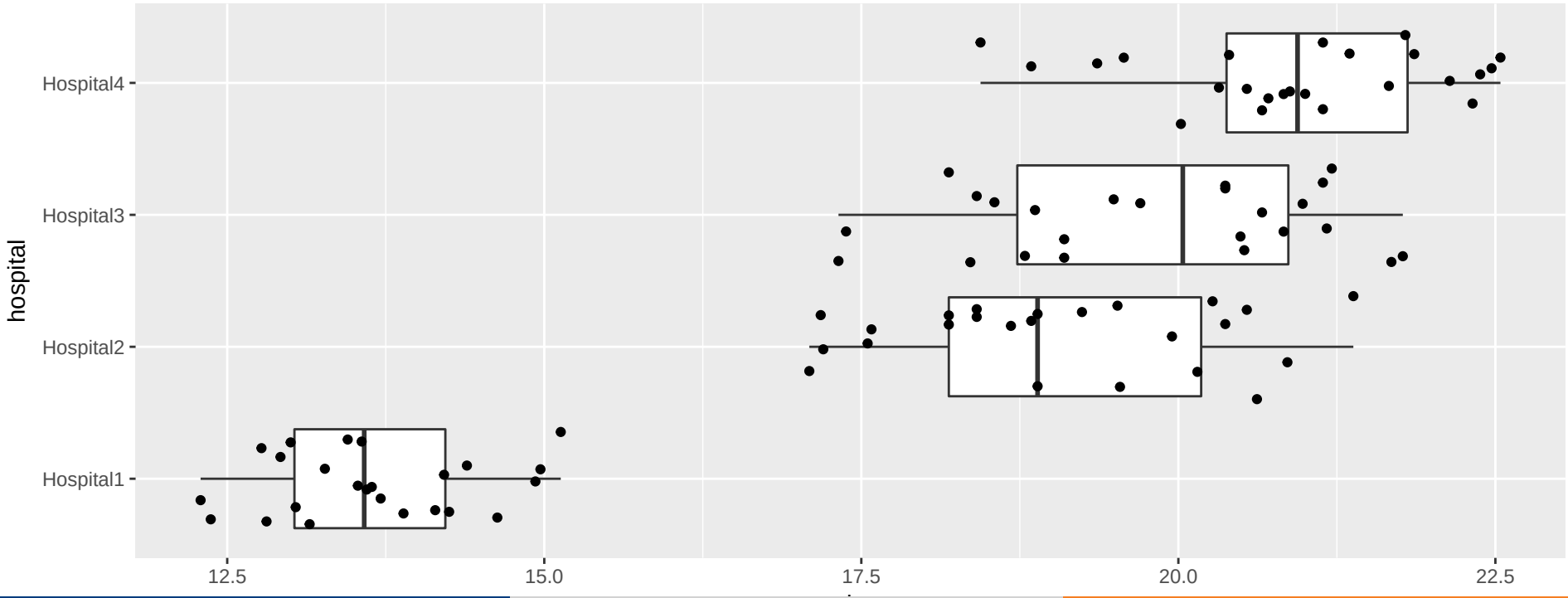
```
ggplot(long_births, aes(x = hospital, y = value)) +
  geom_boxplot() + geom_jitter(height = 0, width = 0.4)
```



Flipping My Box

The coordinate system can also change from being y-based to x-based via `coord_flip()`.

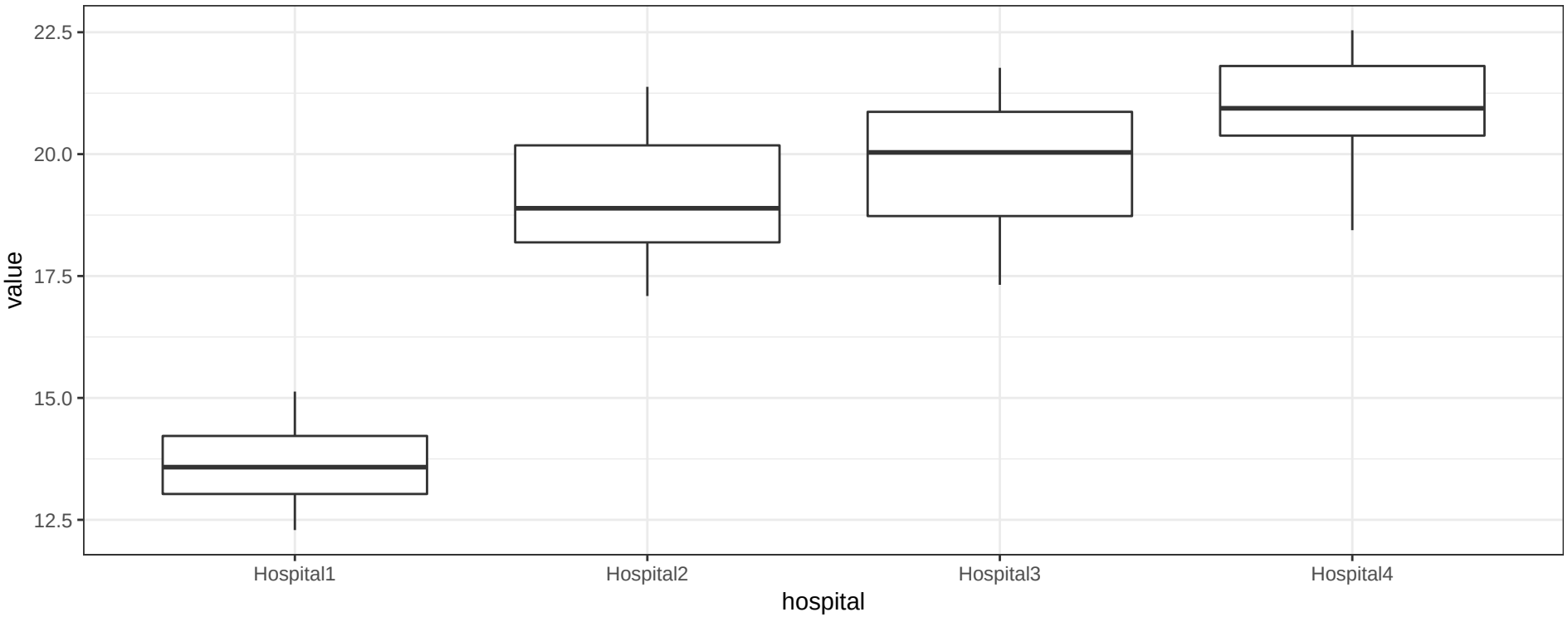
```
ggplot(long_births, aes(x = hospital, y = value)) +
  geom_boxplot() + geom_jitter(height = 0, width = 0.4) +
  coord_flip()
```



Theming - Black & White

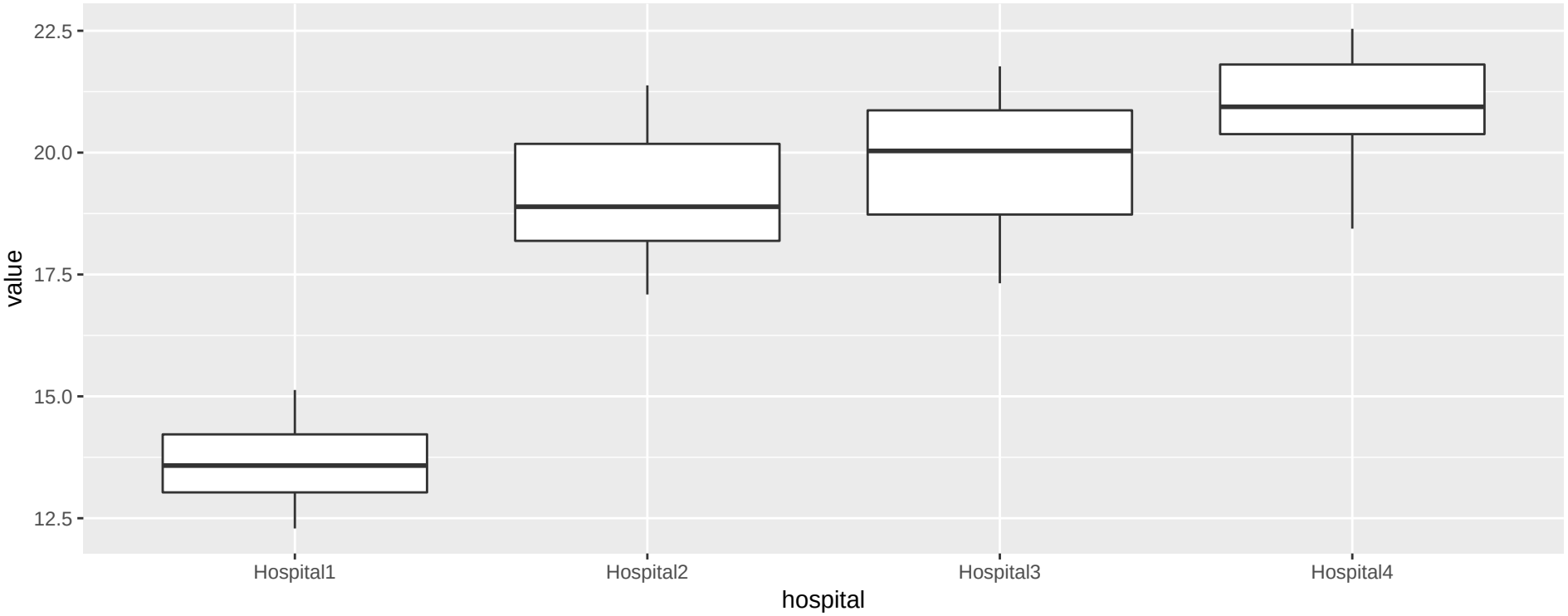
ggplot2 can easily switch to different color themes. By default, the theme_gray() is used. Some prefer the theme_bw().

```
ggplot(long_births, aes(x = hospital, y = value)) +
  geom_boxplot()  + theme_bw()
```



Theming - Original

```
ggplot(long_births, aes(x = hospital, y = value)) +
  geom_boxplot()
```



The main question:

Are you team `theme_gray()` or `theme_bw()`?

Twitter Question

Exercises

1. Load `sportsranks` from `msos` and transform it to long form. Make sure to add an indicator. Try to create boxplots of the different ratings.
2. Open `states` in the `msos` data set. Explore the different levels of school enrollment and crime. Try out other dimensions as well!
3. Last but not least, try to explore the `SAheart` data in `msos`.

Practical Comparison

For the next example, we are going to craft a Q-Q Plot in `Base R` and `ggplot2`. The Q-Q plot is normally used to check to see if the residuals of a model follow a normal distribution.

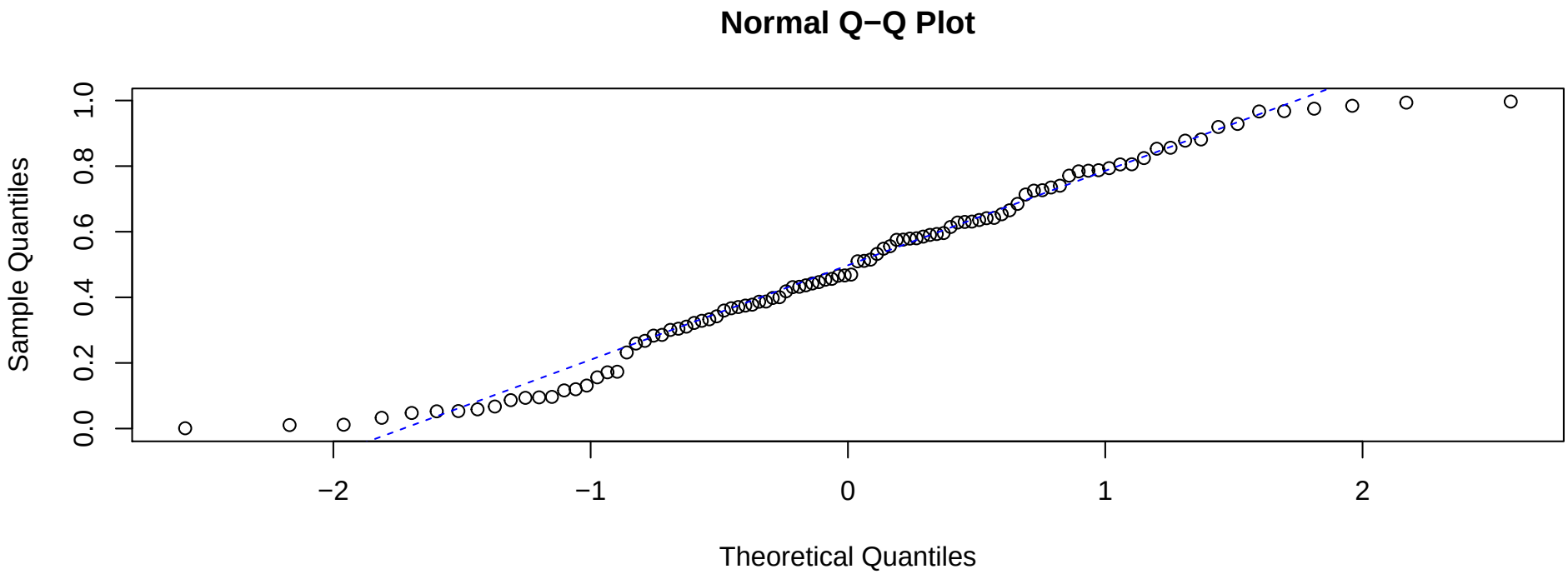
```
# Set seed for reproducibility
set.seed(111)

# Generate data
x = runif(100,0,1)
```

Traditional q-q plot in R

Included in *Base R*, without any modification is `qqnorm()`, which provides the Q-Q Plot. Though, it is missing the traditional line connecting the first and third quartiles.

```
qqnorm(x)                       # normal q-q plot
qqline(x,lty=2,col="blue") # line through the Q1 and Q3 quart
```

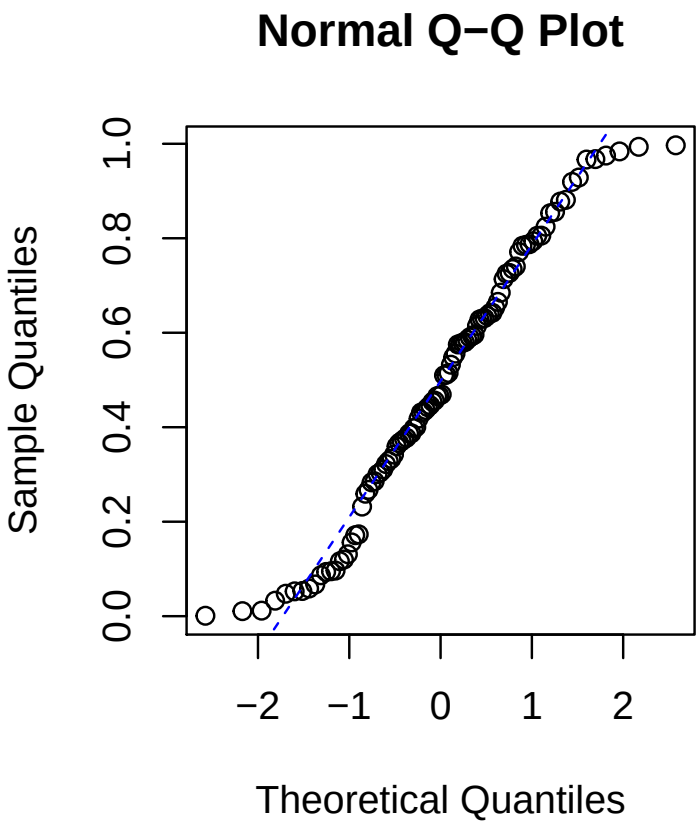


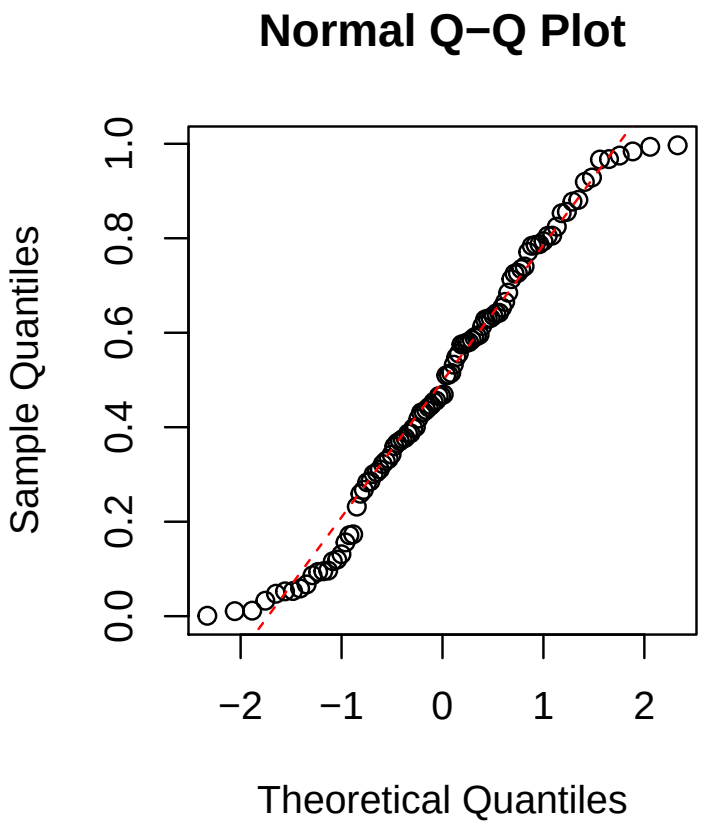
Crafting the q-q plot

```
qqn = function(w) {
  n = length(w)
  nv = qnorm((1:n)/(n+1)) # Quantiles of Normal Dist.
  plot(nv, sort(w),       # X,Y
       xlab = "Theoretical Quantiles",
       ylab = "Sample Quantiles")
  title("Normal Q-Q Plot")
  m = (quantile(w,0.75)-quantile(w,0.25))/
      (qnorm(0.75)-qnorm(0.25))
  b = quantile(w,0.25) - m*qnorm(0.25)
  abline(b, m, lty=2, col="red") # Line through Q1 & Q3
}
```

Comparing Base implementations

```
par(mfrow = c(1, 2), pty= "s") #  Two plots, square pts
qqnorm(x)                      # Base R first
qqline(x,lty=2,col="blue")
qqn(x)                    # Our Plot
```

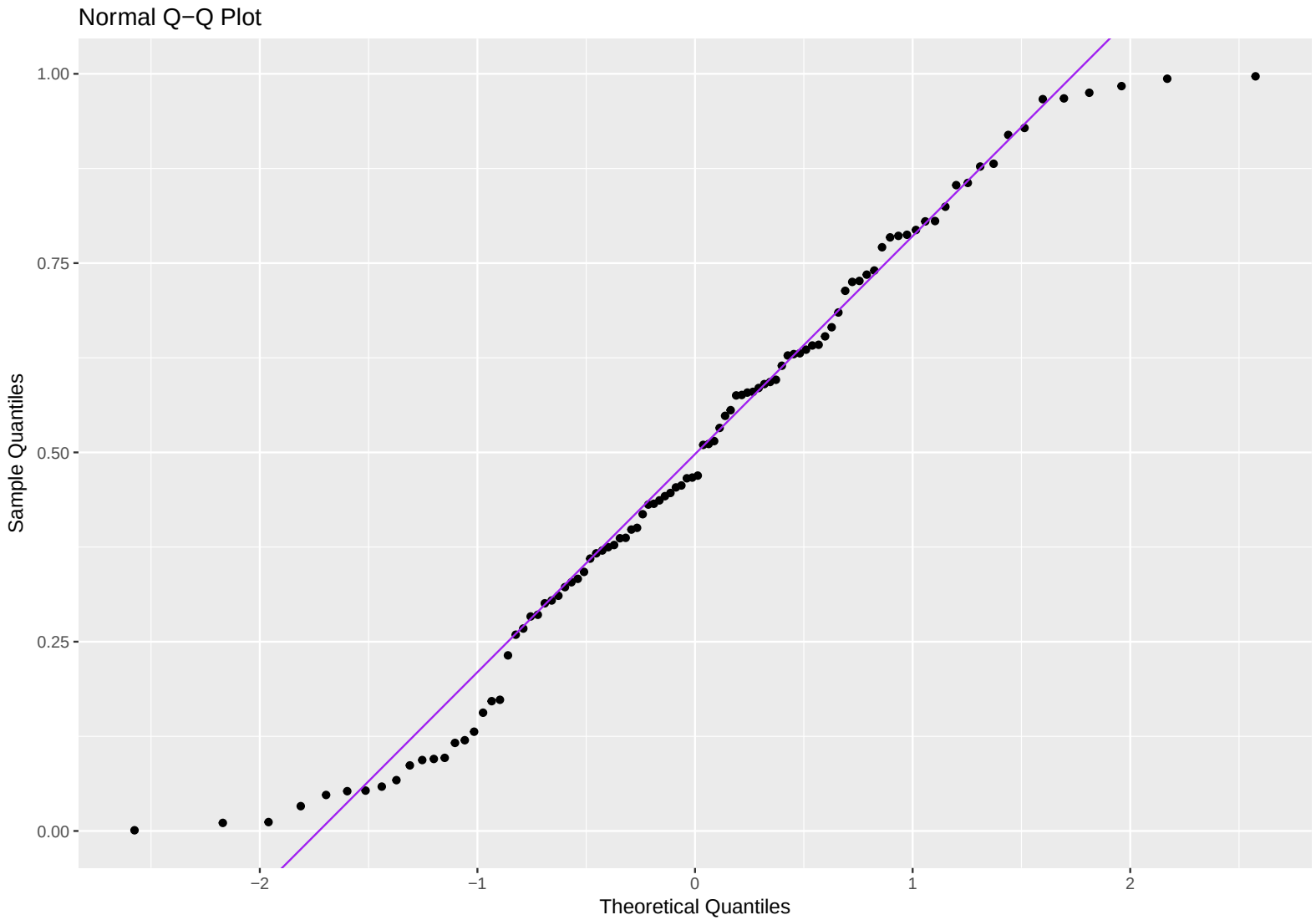




ggplot2 implementation

```
df_x = as.data.frame(x)
n = nrow(df_x)
m = (quantile(x,0.75)-quantile(x,0.25))/
      (qnorm(0.75)-qnorm(0.25))
b = quantile(x,0.25) - m*qnorm(0.25)
g = ggplot(df_x, aes(sample=x)) +
      stat_qq() +
      geom_abline(intercept = b,
                  slope = m,
                  color = "purple") +
      xlab("Theoretical Quantiles") +
      ylab("Sample Quantiles") +
      ggtitle("Normal Q-Q Plot")
```

ggplot2 implementation



ggplot2 implementation

```
g + theme_dark() # Welcome to the dark side!
```