

Functions & Data Visualization

Fundamental Techniques in Data Science



**Utrecht
University**

Kyle M. Lang

Department of Methodology & Statistics
Utrecht University

Outline

Functions

Data Visualization

- Base R Graphics

- GGPlot

- Saving Graphics



FUNCTIONS



R Functions

Functions are the foundation of R programming.

- Other than data objects, almost everything else that you interact with when using R is a function.
- Any R command written as a word followed by parentheses, `()`, is a function.
 - `mean()`
 - `library()`
 - `mutate()`
- Infix operators are aliased functions.
 - `<-`
 - `+`, `-`, `*`
 - `>`, `<`, `==`



User-Defined Functions

We can define our own functions using the `function()` function.

```
square <- function(x) {  
  out <- x^2  
  out  
}
```

After defining a function, we call it in the usual way.

```
square(5)
```

```
[1] 25
```

One-line functions don't need braces.

```
square <- function(x) x^2
```

```
square(5)
```

```
[1] 25
```

User-Defined Functions

Function arguments are not strictly typed.

```
square(1:5)
```

```
[1] 1 4 9 16 25
```

```
square(pi)
```

```
[1] 9.869604
```

```
square(TRUE)
```

```
[1] 1
```

But there are limits.

```
square("bob") # But one can only try so hard
```

```
Error in x^2: non-numeric argument to binary operator
```

User-Defined Functions

Functions can take multiple arguments.

```
mod <- function(x, y) x %% y
mod(10, 3)

[1] 1
```

Sometimes it's useful to specify a list of arguments.

```
getLsBeta <- function(datList) {
  X <- datList$X
  y <- datList$y

  solve(crossprod(X)) %*% t(X) %*% y
}
```

User-Defined Functions

```
X      <- matrix(runif(500), ncol = 5)
datList <- list(y = X %*% rep(0.5, 5), X = X)

getLsBeta(datList = datList)
```

```
      [,1]
[1,] 0.5
[2,] 0.5
[3,] 0.5
[4,] 0.5
[5,] 0.5
```


User-Defined Functions

Functions are first-class objects in R.

- We can treat functions like any other R object.

R views an unevaluated function as an object with type "closure".

```
class(getLsBeta)
[1] "function"

typeof(getLsBeta)
[1] "closure"
```

An evaluated functions is equivalent to the objects it returns.

```
class(getLsBeta(datList))
[1] "matrix" "array"

typeof(getLsBeta(datList))
[1] "double"
```

User-Defined Functions

We can use functions as arguments to other operations and functions.

```
fun1 <- function(x, y) x + y

## What will this command return?
fun1(1, fun1(1, 1))

[1] 3
```

Why would we care?

```
s2 <- var(runif(100))
x <- rnorm(100, 0, sqrt(s2))
```

User-Defined Functions

```
X[1:8, ]
```

	[,1]	[,2]	[,3]	[,4]	[,5]
[1,]	0.52431382	0.67136447	0.28228726	0.7148383	0.54204681
[2,]	0.01926742	0.11693762	0.09148502	0.6929171	0.88371944
[3,]	0.05100735	0.18432074	0.43547799	0.6097462	0.09026598
[4,]	0.60566972	0.12944127	0.21000143	0.2441917	0.68141473
[5,]	0.48737303	0.94030405	0.23988619	0.4915910	0.36353771
[6,]	0.19941958	0.96670678	0.11455820	0.1243947	0.24253273
[7,]	0.95507804	0.38705829	0.49733535	0.2968470	0.81001800
[8,]	0.11093197	0.07731757	0.84923006	0.8653987	0.61914193

```
c(1, 3, 6:9, 12)
```

```
[1] 1 3 6 7 8 9 12
```

DATA VISUALIZATION



Setup

```
dataDir <- "../.../data/"

## Load some data:
diabetes <- readRDS(paste0(dataDir, "diabetes.rds"))
titanic <- readRDS(paste0(dataDir, "titanic.rds"))
bfi      <- readRDS(paste0(dataDir, "bfi.rds"))

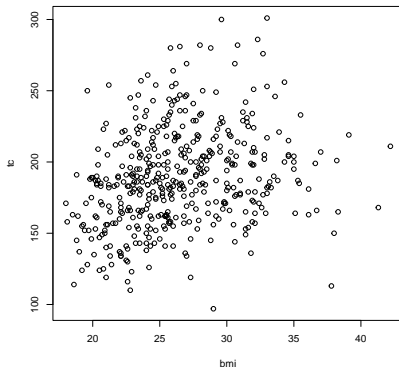
## Convert survival indicator to a numeric dummy code:
titanic <- titanic %>% mutate(survived = as.numeric(survived) - 1)
```



Base R Graphics: Scatterplots

We can create a basic scatterplot using the `plot()` function.

```
diabetes %>% plot(y = tc, x = bmi)
```

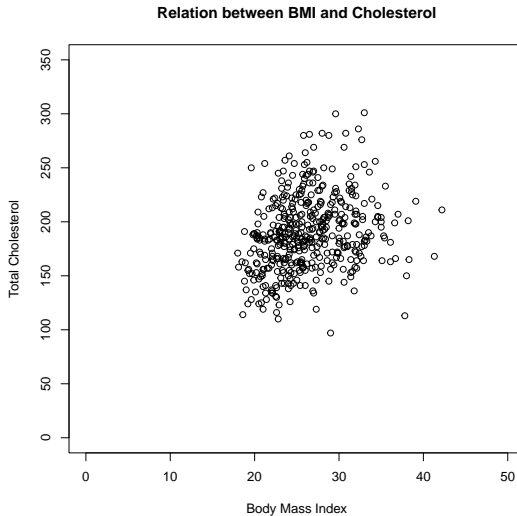


Base R Graphics: Scatterplots

```
diabetes %$% plot(y = tc,  
                 x = bmi,  
                 ylab = "Total Cholesterol",  
                 xlab = "Body Mass Index",  
                 main = "Relation between BMI and Cholesterol",  
                 ylim = c(0, 350),  
                 xlim = c(0, 50)  
                 )
```



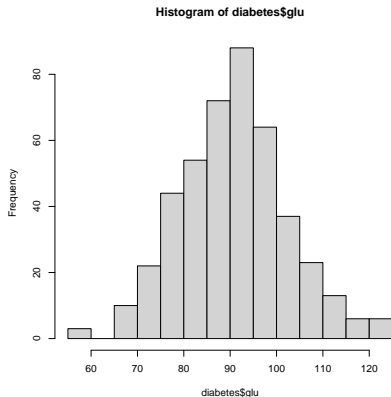
Base R Graphics: Scatterplots



Base R Graphics: Histograms

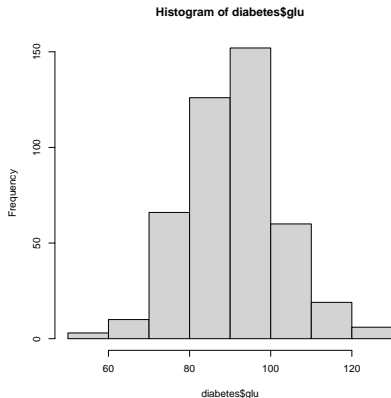
We can create a simple histogram with the `hist()` function.

```
hist(diabetes$glu)
```



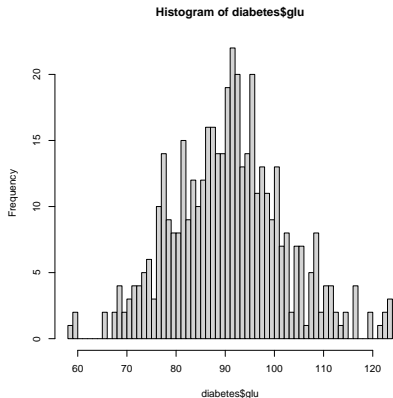
Base R Graphics: Histograms

```
hist(diabetes$glu, breaks = 5)
```



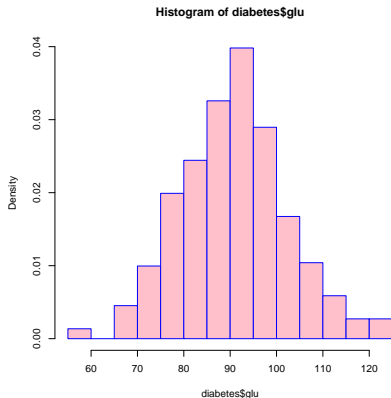
Base R Graphics: Histograms

```
hist(diabetes$glu, breaks = 50)
```



Base R Graphics: Histograms

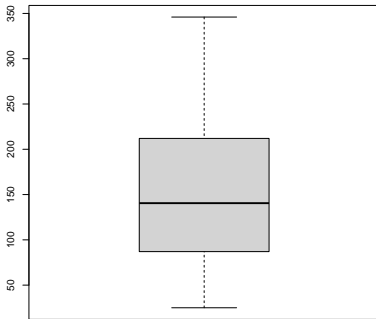
```
hist(diabetes$glu, col = "pink", border = "blue", probability = TRUE)
```



Base R Graphics: Boxplots

We can create simple boxplots via the `boxplot()` function.

```
boxplot(diabetes$progress)
```

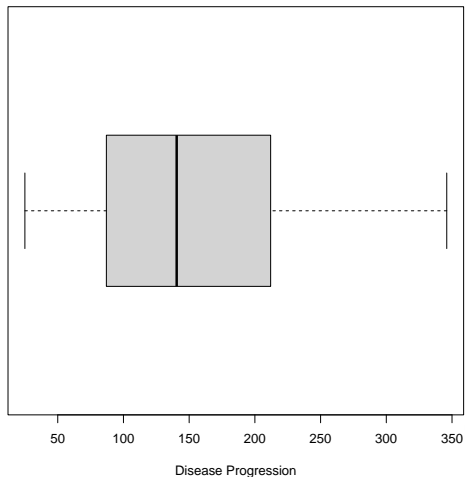


Base R Graphics: Boxplots

```
boxplot(diabetes$progress,  
        horizontal = TRUE,  
        range = 3,  
        xlab = "Disease Progression")
```



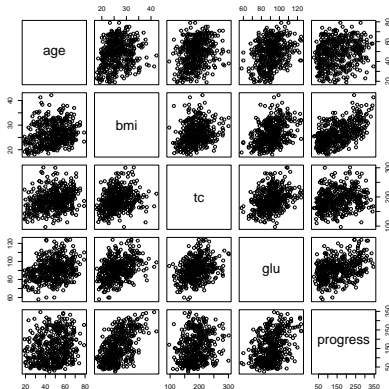
Base R Graphics: Boxplots



Base R Graphics: Fancy Things

Plotting an entire data frame produces a scatterplot matrix.

```
diabetes %>% select(age, bmi, tc, glu, progress) %>% plot()
```

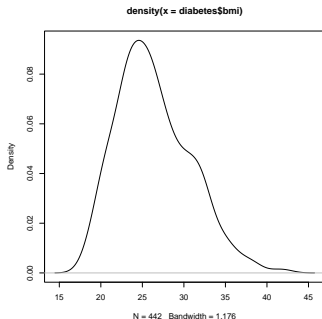


Base R Graphics: Fancy Things

The `density()` function estimates the density of a variable.

- If we plot a density object, we get a kernel density plot.

```
density(diabetes$bmi) %>% plot()
```



Base R Graphics: Fancy Things

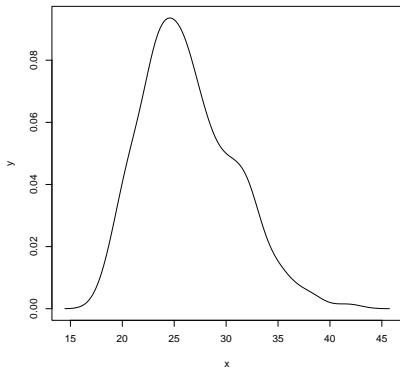
```
d <- density(diabetes$bmi)
ls(d)
```

```
[1] "bw"      "call"    "data.name" "has.na"
[5] "n"       "x"       "y"
```



Base R Graphics: Fancy Things

```
d %>% plot(y = y, x = x, type = "l")
```



Base R Graphics: Workflow

Base R graphics work by building up figures in layers.

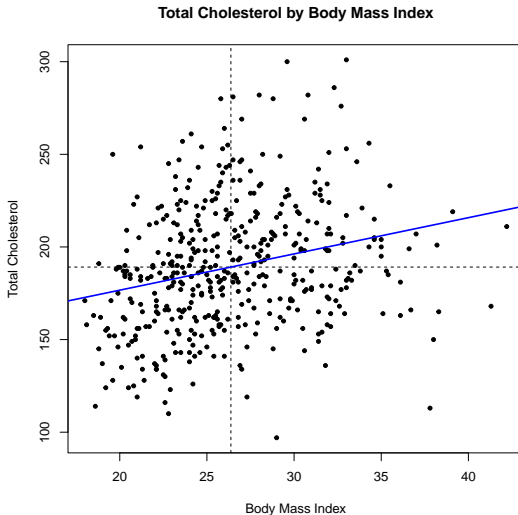
```
## Start with a simple scatterplot:
diabetes %$% plot(y = tc, x = bmi, pch = 20, xlab = "", ylab = "")

## Use the abline() function to add lines representing the means of x and y:
abline(h = mean(diabetes$tc), v = mean(diabetes$bmi), lty = 2)

## Add the best fit line from a linear regression of 'tc' onto 'bmi':
diabetes %$%
  lm(tc ~ bmi) %>%
  coef() %>%
  abline(coef = ., col = "blue", lwd = 2)

## Add titles:
title(main = "Total Cholesterol by Body Mass Index",
      ylab = "Total Cholesterol",
      xlab = "Body Mass Index")
```

Base R Graphics: Workflow

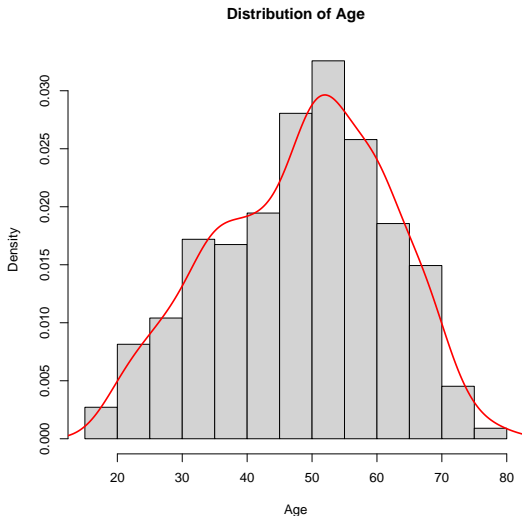


Base R Graphics: Workflow

Add a kernel density plot on top of a histogram.

```
diabetes %$%  
  hist(age,  
        probability = TRUE,  
        xlab = "Age",  
        main = "Distribution of Age")  
  
diabetes %$%  
  density(age) %>%  
  lines(col = "red", lwd = 2)
```

Base R Graphics: Workflow



GGPlot

Base R graphics are fine for quick-and-dirty visualizations (e.g., EDA, checking assumptions), but for publication quality graphics, you should probably use GGPlot.

- GGPlot uses the "grammar of graphics" and "tidy data" to build up a figure from modular components.

Describes all the non-data ink
Plotting space for the data
Statistical models & summaries
Rows and columns of sub-plots
Shapes used to represent the data
Scales onto which data is mapped
The actual variables to be plotted

Theme
Coordinates
Statistics
Facets
Geometries
Aesthetics
Data



GGPlot: Basic Setup

We start by calling the `ggplot()` function.

- We must define a data source.
- We must also give some aesthetic via the `aes()` function.

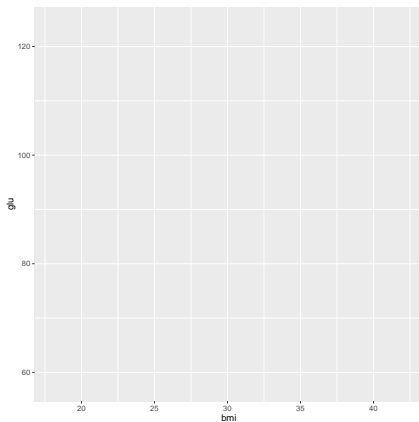
```
library(ggplot2)
p1 <- ggplot(data = diabetes, mapping = aes(x = bmi, y = glu))
```



GGPlot: Basic Setup

At this point, our plot is pretty boring.

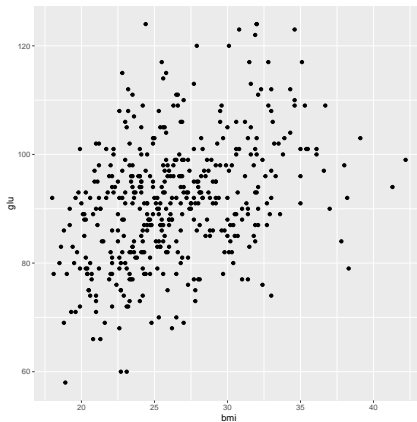
p1



GGPlot: Geometries

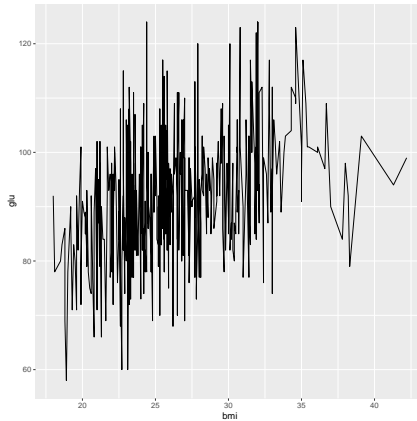
We need to define some geometry via a `geom_XXX()` function.

```
p1 + geom_point()
```



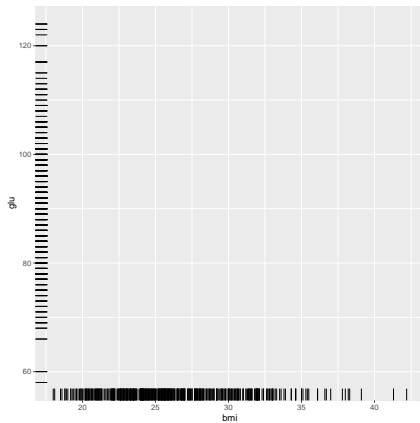
GGPlot: Geometries

```
p1 + geom_line()
```



GGPlot: Geometries

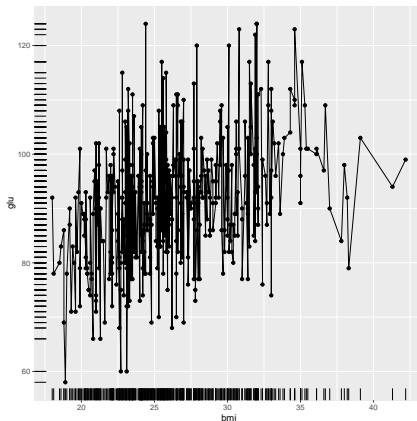
```
p1 + geom_rug()
```



GGPlot: Geometries

We can also combine different geometries into a single figure

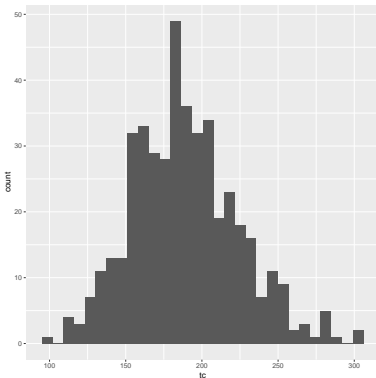
```
p1 + geom_point() + geom_line() + geom_rug()
```



GGPlot: Geometries

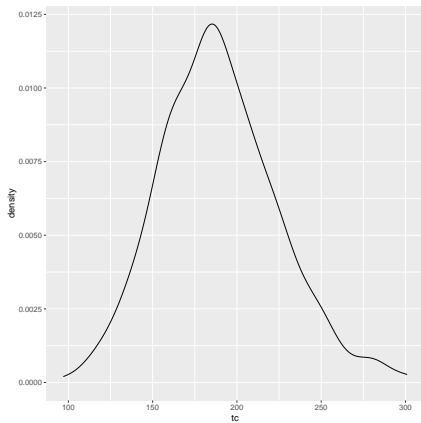
We can use different flavors of geometry for different types of data.

```
p2 <- ggplot(diabetes, aes(tc))  
p2 + geom_histogram()
```



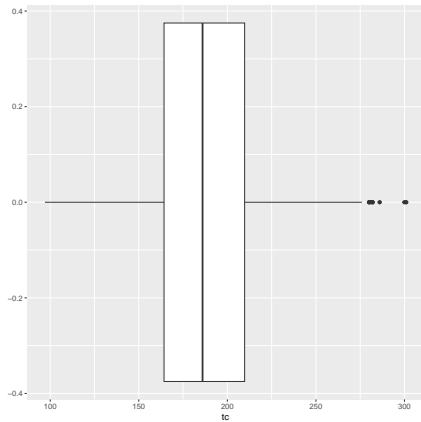
GGPlot: Geometries

```
p2 + geom_density()
```



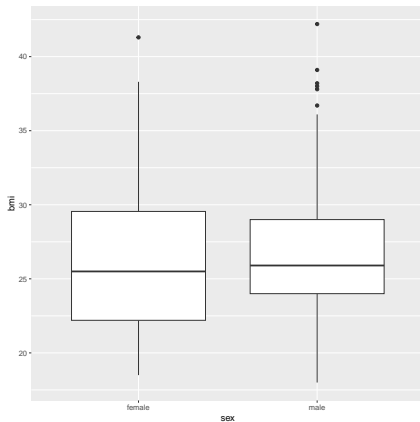
GGPlot: Geometries

```
p2 + geom_boxplot()
```



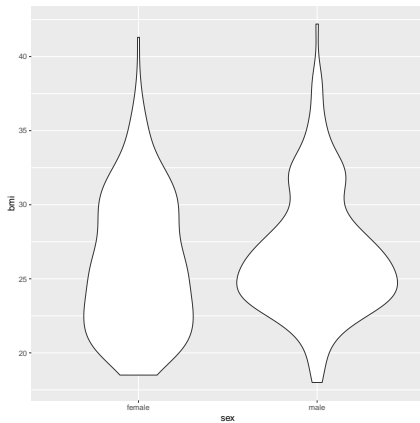
GGPlot: Geometries

```
p3 <- ggplot(diabetes, aes(sex, bmi))  
p3 + geom_boxplot()
```



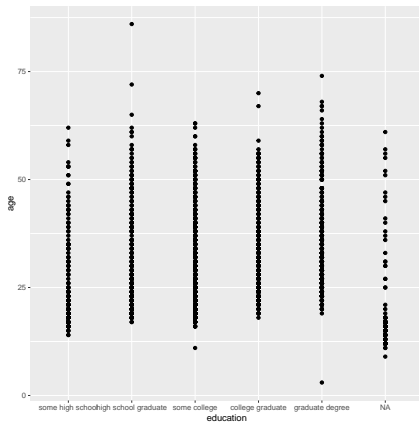
GGPlot: Geometries

```
p3 + geom_violin()
```



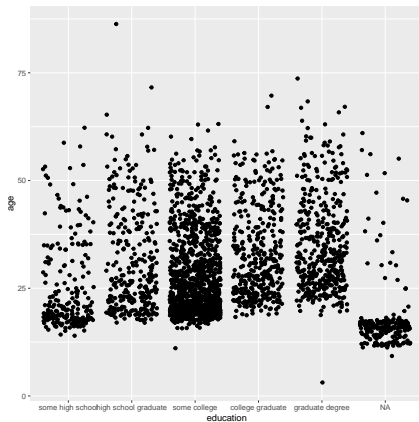
GGPlot: Geometries

```
p4 <- ggplot(bfi, aes(education, age))  
p4 + geom_point()
```



GGPlot: Geometries

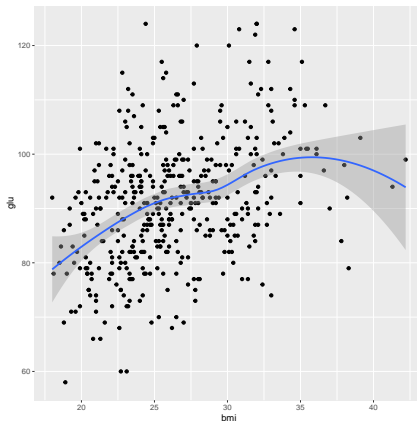
```
p4 + geom_jitter()
```



GGPlot: Statistics

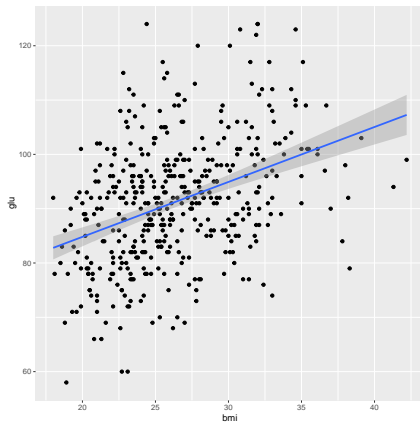
We can also add statistical summaries of the data.

```
p1 + geom_point() + geom_smooth()
```



GGPlot: Statistics

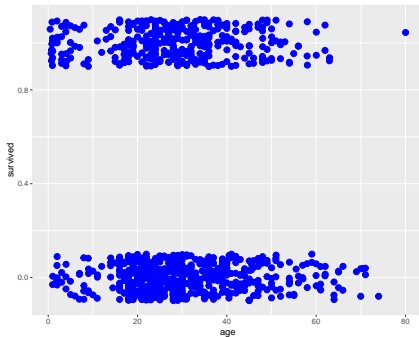
```
p1 + geom_point() + geom_smooth(method = "lm")
```



GGPlot: Styling

Changing style options outside of the `aes()` function applies the styling to the entire plot.

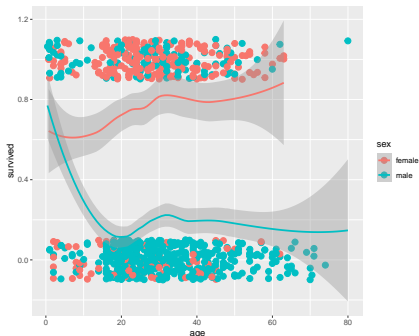
```
p5 <- ggplot(titanic, aes(age, survived))  
p5 + geom_jitter(color = "blue", size = 3, height = 0.1)
```



GGPlot: Styling

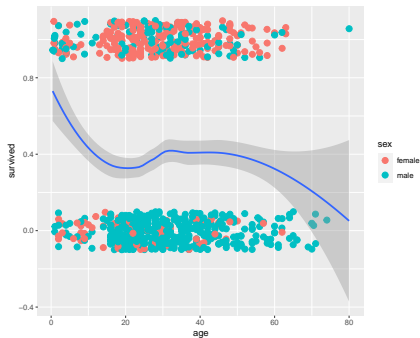
We can also apply styles as a function of variables by defining the style within the `aes()` function.

```
p6.1 <- ggplot(titanic, aes(age, survived, color = sex))  
p6.1 + geom_jitter(size = 3, height = 0.1) + geom_smooth()
```



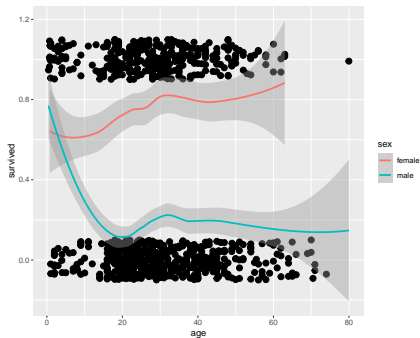
GGPlot: Styling

```
p6.2 <- ggplot(titanic, aes(age, survived))  
p6.2 + geom_jitter(aes(color = sex), size = 3, height = 0.1) +  
  geom_smooth()
```



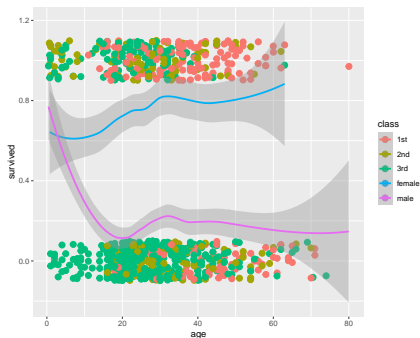
GGPlot: Styling

```
p6.2 + geom_jitter(size = 3, height = 0.1) +  
  geom_smooth(aes(color = sex))
```



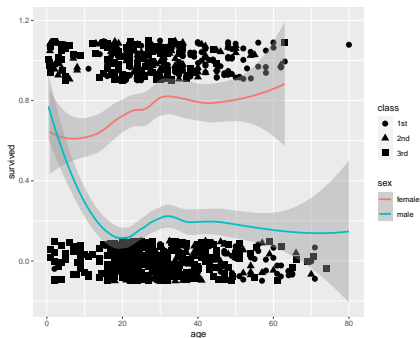
GGPlot: Styling

```
p6.2 + geom_jitter(aes(color = class), size = 3, height = 0.1) +  
  geom_smooth(aes(color = sex))
```



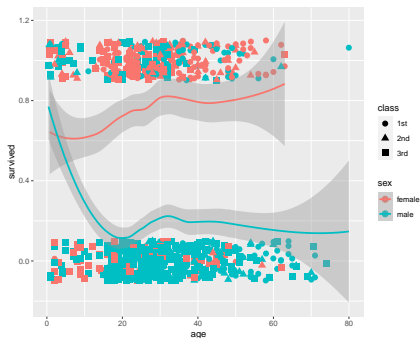
GGPlot: Styling

```
p6.2 + geom_jitter(aes(shape = class), size = 3, height = 0.1) +  
  geom_smooth(aes(color = sex))
```



GGPlot: Styling

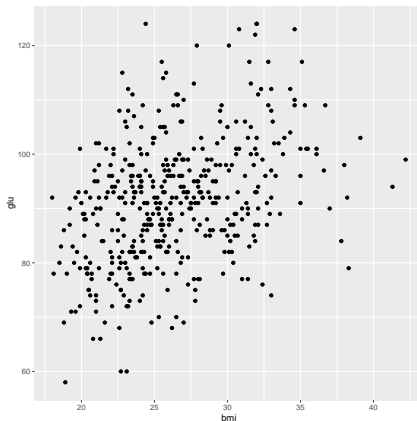
```
p6.1 + geom_jitter(aes(shape = class), size = 3, height = 0.1) +  
  geom_smooth()
```



GGPlot: Themes

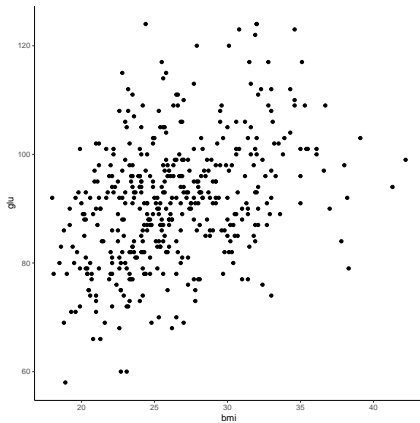
We can apply canned themes to adjust a plot's overall appearance.

```
(p1.1 <- p1 + geom_point())
```



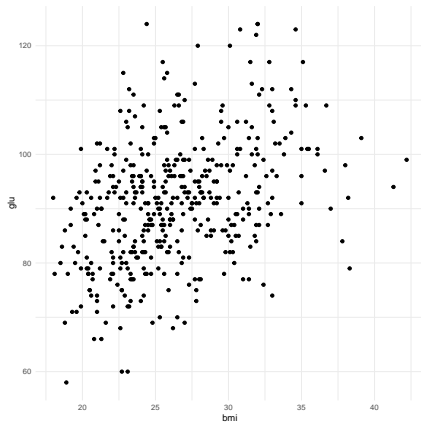
GGPlot: Themes

```
p1.1 + theme_classic()
```



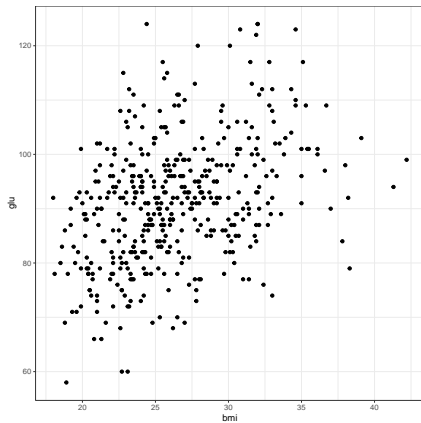
GGPlot: Themes

```
p1.1 + theme_minimal()
```



GGPlot: Themes

```
p1.1 + theme_bw()
```



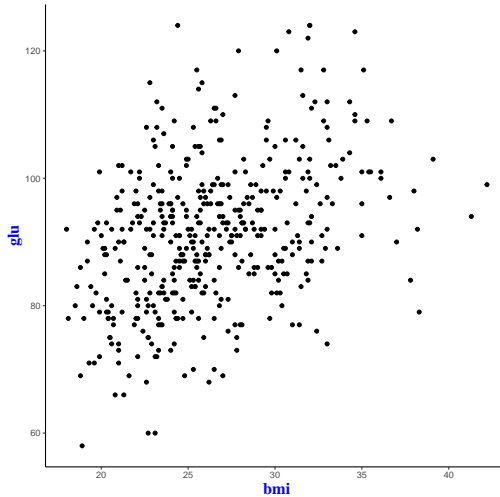
GGPlot: Themes

We can also modify individual theme elements.

```
p1.1 + theme_classic() +  
  theme(axis.title = element_text(size = 16,  
                                   family = "serif",  
                                   face = "bold",  
                                   color = "blue"),  
        aspect.ratio = 1)
```



GGPlot: Themes



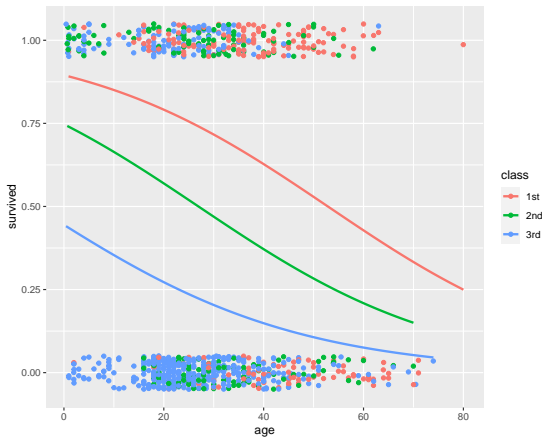
GGPlot: Facets

Faceting allow us to make arrays of conditional plots.

```
(p7 <- ggplot(titanic, aes(age, survived, color = class)) +  
  geom_jitter(height = 0.05) +  
  geom_smooth(method = "glm",  
              method.args = list(family = "binomial"),  
              se = FALSE)  
)
```

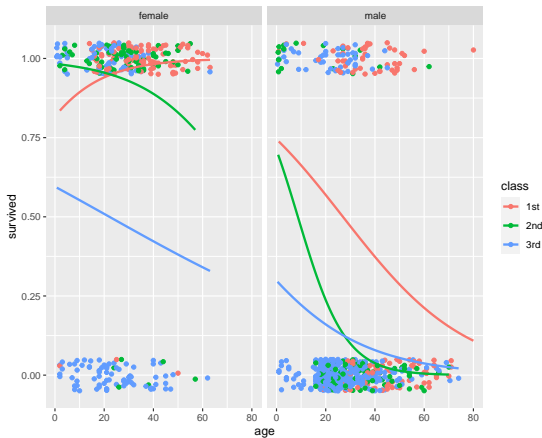


GGPlot: Facets



GGPlot: Facets

```
p7 + facet_wrap(vars(sex))
```

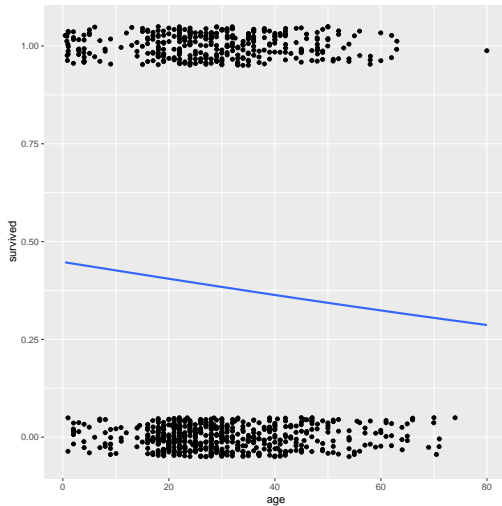


GGPlot: Facets

```
(p8 <- ggplot(titanic, aes(age, survived)) +  
  geom_jitter(height = 0.05) +  
  geom_smooth(method = "glm",  
              method.args = list(family = "binomial"),  
              se = FALSE)  
)
```

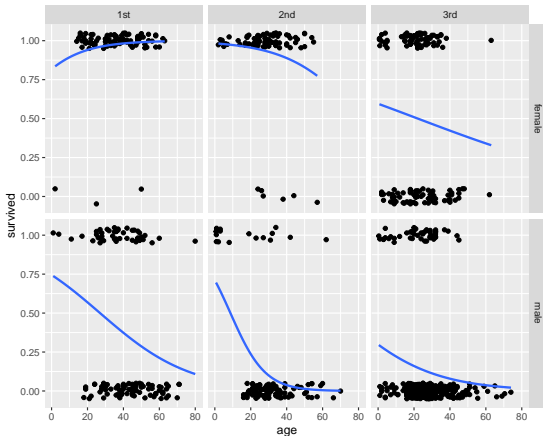


GGPlot: Facets



GGPlot: Facets

```
p8 + facet_grid(vars(sex), vars(class))
```



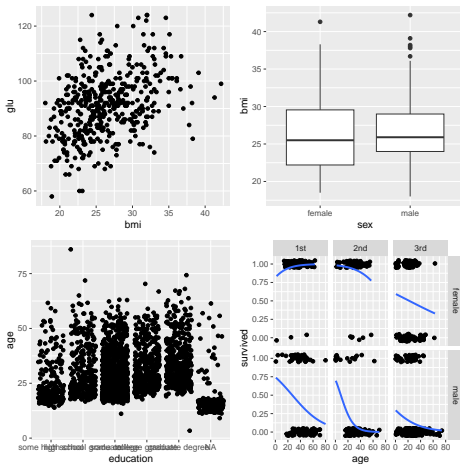
GGPlot: Joining Multiple Figures

If we want to paste several different plots into a single figure (without faceting), we can use the utilities in the **gridExtra** package.

```
library(gridExtra)

grid.arrange(p1 + geom_point(),
             p3 + geom_boxplot(),
             p4 + geom_jitter(),
             p8 + facet_grid(vars(sex), vars(class)),
             ncol = 2)
```

GGPlot: Joining Multiple Figures



Saving Graphics

To save a graphic that we've created in R, we simply redirect the graphical output to a file using an appropriate function.

```
figDir <- "figures/"

## Save as PDF
pdf(paste0(figDir, "example_plot.pdf"))

p7 + facet_wrap(vars(sex))

dev.off()

pdf
  2
```

Saving Graphics

```
## Save as JPEG
jpeg(paste0(figDir, "example_plot.jpg"))

p7 + facet_wrap(vars(sex))

dev.off()

pdf
  2

## Save as PNG
png(paste0(figDir, "example_plot.png"))

p7 + facet_wrap(vars(sex))

dev.off()

pdf
  2
```

Saving Graphics

With PDF documents, we can save multiple figures to a single file.

```
pdf(paste0(figDir, "example_plot2.pdf"))

p6.1 + geom_jitter(size = 3, height = 0.1) + geom_smooth()
p7 + facet_wrap(vars(sex))
p8 + facet_grid(vars(sex), vars(class))

dev.off()

pdf
2
```