19. Modern R

This handbook has used basic R throughout in order to focus on the algorithms and to make the code as accessible as possible to a wide audience. However, as you continue in R after working through this book, you should learn to use recent extensions to the language that fall under the general title of the **tidyverse**. The tidyverse is a set of R packages designed for improved data manipulation, exploration, and visualization. The packages work together well because they have common data representations and a common API design. All of them can be installed at once with command: `install.packages("tidyverse")`.

The packages include:

- dplyr - provides a grammar of data manipulation
- tibble - creates and manipulates a modern data frame
- tidyr - helps you create tidy data
- readr - is a fast reader of rectangular data
- purr - enhances R's functional programmming (FP) toolkit
- ggplot2 - implements a grammar of graphics for data visualization
- stringr - makes working with strings easier
- forcats - work with factors

These extensions to the language are exciting because they have updated R to be more competitive with newer languages. Further, the tidyverse is fast, and better able to handle larger data.

In the next two sections we provide a brief introduction to dplyr and ggplot2 because you will commonly see these used in R code on the web and it is important to understand the code well enough to read it. A terrific online resource for learning to code in the tidyverse is Garrett Grolemund and Hadley Wickham's book *R for Data Science*, most of which is available on the web site: `http://r4ds.had.co.nz/index.html`

19.1 Package dplyr

According to the package author, Hadley Wickham, the d in dplyr is for data and the rest is to evoke pliers, that is, a tool. So dplyr is used to manipulate data efficiently. The dplyr approach is designed to help manage bigger data but we will demonstrate its basic features on the PimaIndiansDiabetes2 data set in mlbench.

19.1.1 Tibble

A tibble is a data frame with enhanced features. After converting the PimaIndiansDiabetes2 data frame to tibble df, when we type df at the console we get a page at a time, as you can see in the online notebook in github. We also removed PimaIndiansDiabetes2 from the workspace. If this had been a huge data set, removing it would free up a lot of memory.

Code 19.1.1 — dplyr. Create a tibble.

```
library(dplyr)
library(mlbench)
data("PimaIndiansDiabetes2")
df <- tbl_df(PimaIndiansDiabetes2)
rm(PimaIndiansDiabetes2)
df
```

```
## # A tibble: 768 x 9
##    pregnant glucose pressure triceps insulin  mass pedigree   age diabetes
##  *    <dbl>   <dbl>    <dbl>   <dbl>   <dbl> <dbl>    <dbl> <dbl> <fct>
##  1     6.00   148       72.0    35.0    NA    33.6    0.627  50.0 pos
##  2     1.00    85.0     66.0    29.0    NA    26.6    0.351  31.0 neg
##  3     8.00   183       64.0    NA      NA    23.3    0.672  32.0 pos
##  4     1.00    89.0     66.0    23.0    94.0  28.1    0.167  21.0 neg
##  5     0      137       40.0    35.0   168    43.1    2.29   33.0 pos
##  6     5.00   116       74.0    NA      NA    25.6    0.201  30.0 neg
##  7     3.00    78.0     50.0    32.0    88.0  31.0    0.248  26.0 pos
##  8    10.0    115       NA      NA      NA    35.3    0.134  29.0 neg
##  9     2.00   197       70.0    45.0   543    30.5    0.158  53.0 pos
## 10     8.00   125       96.0    NA      NA    NA      0.232  54.0 pos
## # ... with 758 more rows
```

19.1.2 Function glimpse()

The glimpse() function is similar to str() but can handle bigger data more efficiently. The command `glimpse(df)` output the following:

```
## Observations: 768
## Variables: 9
## $ pregnant <dbl> 6, 1, 8, 1, 0, 5, 3, 10, 2, 8, 4, 10, 10, 1, 5, 7, 0,...
## $ glucose  <dbl> 148, 85, 183, 89, 137, 116, 78, 115, 197, 125, 110, 1...
## $ pressure <dbl> 72, 66, 64, 66, 40, 74, 50, NA, 70, 96, 92, 74, 80, 6...
## $ triceps  <dbl> 35, 29, NA, 23, 35, NA, 32, NA, 45, NA, NA, NA, NA, 2...
## $ insulin  <dbl> NA, NA, NA, 94, 168, NA, 88, NA, 543, NA, NA, NA, NA,...
## $ mass     <dbl> 33.6, 26.6, 23.3, 28.1, 43.1, 25.6, 31.0, 35.3, 30.5,...
## $ pedigree <dbl> 0.627, 0.351, 0.672, 0.167, 2.288, 0.201, 0.248, 0.13...
## $ age      <dbl> 50, 31, 32, 21, 33, 30, 26, 29, 53, 54, 30, 34, 57, 5...
## $ diabetes <fct> pos, neg, pos, neg, pos, neg, pos, neg, pos, pos, neg...
```

19.1.3 Select

There are 5 main functions in dplyr, select and mutate deal with columns, filter and arrange deal with rows, and summarize lets you extract summary information. Next we select two columns from the data frame and send them to the print function. Select returns a new object, but in this case we didn't save it to another variable so it does not exist in memory.

Code 19.1.2 — dplyr. Select.

```
print(select(df, diabetes, pregnant))
```

```
## # A tibble: 768 x 2
##    diabetes pregnant
##  * <fct>       <dbl>
##  1 pos          6.00
##  2 neg          1.00
##  3 pos          8.00
##  4 neg          1.00
##  5 pos          0
##  6 neg          5.00
##  7 pos          3.00
##  8 neg         10.0
##  9 pos          2.00
## 10 pos          8.00
## # ... with 758 more rows
```

19.1.4 Mutate

You can use mutate to create new columns from the existing data. In the code example below we created a new binary factor columns that is 1 if glucose is above average for the population and 0 otherwise.

Code 19.1.3 — dplyr. Mutate.

```
mutate(df, glucose_high = as.factor(
     ifelse(glucose>mean(glucose, na.rm=TRUE), 1, 0)))
```

```
## # A tibble: 768 x 10
##    pregnant glucose pressure triceps insulin  mass pedigree   age diabetes
##       <dbl>   <dbl>    <dbl>   <dbl>   <dbl> <dbl>    <dbl> <dbl> <fct>
##  1     6.00   148       72.0    35.0    NA    33.6    0.627  50.0 pos
##  2     1.00    85.0     66.0    29.0    NA    26.6    0.351  31.0 neg
##  3     8.00   183       64.0    NA      NA    23.3    0.672  32.0 pos
##  4     1.00    89.0     66.0    23.0    94.0  28.1    0.167  21.0 neg
##  5     0      137       40.0    35.0   168    43.1    2.29   33.0 pos
##  6     5.00   116       74.0    NA      NA    25.6    0.201  30.0 neg
##  7     3.00    78.0     50.0    32.0    88.0  31.0    0.248  26.0 pos
##  8    10.0    115       NA      NA      NA    35.3    0.134  29.0 neg
##  9     2.00   197       70.0    45.0   543    30.5    0.158  53.0 pos
## 10     8.00   125       96.0    NA      NA    NA      0.232  54.0 pos
## # ... with 758 more rows, and 1 more variable: glucose_high <fct>
```

19.1.5 Filter

The filter function is used to remove rows. Below we filter rows out that have NAs in either glucose or mass, then glimpse the data.

Code 19.1.4 — dplyr. Filter.

```
df <- filter(df, !is.na(glucose), !is.na(mass))
glimpse(df)
```

```
Observations: 752
Variables: 9
$ pregnant <dbl> 6, 1, 8, 1, 0, 5, 3, 10, 2, 4, 10, 10, 1, 5, 7, 0, 7, 1, 1, 3, 8, 7, 9, 11, 10, 7, 1, 13, 5, 5, 3, 3, ...
$ glucose  <dbl> 148, 85, 183, 89, 137, 116, 78, 115, 197, 110, 168, 139, 189, 166, 100, 118, 107, 103, 115, 126, 99, 1...
$ pressure <dbl> 72, 66, 64, 66, 40, 74, 50, NA, 70, 92, 74, 80, 60, 72, NA, 84, 74, 30, 70, 88, 84, 90, 80, 94, 70, 76...
$ triceps  <dbl> 35, 29, NA, 23, 35, NA, 32, NA, 45, NA, NA, NA, 23, 19, NA, 47, NA, 38, 30, 41, NA, NA, 35, 33, 26, NA...
$ insulin  <dbl> NA, NA, NA, 94, 168, NA, 88, NA, 543, NA, NA, NA, 846, 175, NA, 230, NA, 83, 96, 235, NA, NA, NA, 146,...
$ mass     <dbl> 33.6, 26.6, 23.3, 28.1, 43.1, 25.6, 31.0, 35.3, 30.5, 37.6, 38.0, 27.1, 30.1, 25.8, 30.0, 45.8, 29.6, ...
$ pedigree <dbl> 0.627, 0.351, 0.672, 0.167, 2.288, 0.201, 0.248, 0.134, 0.158, 0.191, 0.537, 1.441, 0.398, 0.587, 0.48...
$ age      <dbl> 50, 31, 32, 21, 33, 30, 26, 29, 53, 30, 34, 57, 59, 51, 32, 31, 31, 33, 32, 27, 50, 41, 29, 51, 41, 43...
$ diabetes <fct> pos, neg, pos, neg, pos, neg, pos, neg, pos, neg, pos, neg, pos, pos, pos, pos, pos, neg, pos, neg, ne...
```

19.1.6 Arrange

Arrange is used to control the order of rows. Here we arrange by mass in descending order.

Code 19.1.5 — dplyr. Arrange.

```
arrange(df, desc(mass)) # descending order
```

```
## # A tibble: 752 x 9
##    pregnant glucose pressure triceps insulin  mass pedigree   age diabetes
##       <dbl>   <dbl>    <dbl>   <dbl>   <dbl> <dbl>    <dbl> <dbl> <fct>
##  1     0      129      110      46.0    130    67.1    0.319  26.0 pos
##  2     0      180       78.0    63.0     14.0  59.4    2.42   25.0 pos
##  3     3.00   123      100      35.0    240    57.3    0.880  22.0 neg
##  4     1.00    88.0     30.0    42.0     99.0  55.0    0.496  26.0 pos
##  5     0      162       76.0    56.0    100    53.2    0.759  25.0 pos
##  6     5.00   115       98.0    NA       NA    52.9    0.209  28.0 pos
##  7    11.0    135       NA      NA       NA    52.3    0.578  40.0 pos
##  8     0      165       90.0    33.0    680    52.3    0.427  23.0 neg
##  9     7.00   152       88.0    44.0     NA    50.0    0.337  36.0 pos
## 10     1.00   122       90.0    51.0    220    49.7    0.325  31.0 pos
```

19.1.7 Summarize

The summarize function is a powerful way to get summary statistics. Below we have tow examples. Note that you can spell it the American or British way: summarize or summarise.

19.1.8 Pipes

Pipes were introduced in the `magrittr` package and are automatically loaded with dplyr. The pipe symbol works like the unix pipe. It makes code easier to read as you can combine several commeands in a neat group of lines instead of nesting f unctions in the typical R fashion.

```
summarize(df, min=min(mass), max=max(mass), sd(mass))
```

```
## # A tibble: 1 x 3
##     min   max `sd(mass)`
##   <dbl> <dbl>      <dbl>
## 1  18.2  67.1       6.93
```

```
summarize(df, n_diabetic = sum(diabetes=="pos"), n_not_diabetic = sum(diabetes=="neg"))
```

```
## # A tibble: 1 x 2
##   n_diabetic n_not_diabetic
##        <int>          <int>
## 1        264            488
```

Code 19.1.6 — dplyr. Pipes.

```
df %>%
  group_by(diabetes) %>%
  summarize(n_diabetic = n())
```

```
## # A tibble: 2 x 2
##   diabetes n_diabetic
##   <fct>         <int>
## 1 neg             488
## 2 pos             264
```

19.1.9 Going Further with dplyr

- The book *R for Data Science* by Garrett Grolemund and Hadley Wickham is available here: `http://r4ds.had.co.nz/`. It provides a thorough introduction to R and the tidyverse.

19.2 Package ggplot2

Hadley Wickham developed ggplot2 in 2005, inspired by a grammar of graphics developed by Leland Wildinson in 1999. The ggplot2 functions are much more powerful than standard R graphs but also slower.

There are 7 grammatical elements in ggplot2, the first 3 of these are essential to getting something plotted.

- data = the data being plotted should be the first argument, or specify data=...
- aesthetics - the scales onto which we plot; use aes() to specify at least x= and y= if needed as well as other parameters for customization
- geometries - visual elements such as points, lines, etc.
- facets - for plotting multiples
- statistics - representations to aid understanding
- coordinates - space on which data will be plotted
- themes - you can customize your own theme to use over and over

To get started, we have a short example below showing important components of building a ggplot. First we specify the data, then the aesthetics which are how the data is represented, followed by the geometry and finally labels.

Code 19.2.1 — ggplot. Basic Scatterplot.

```
library(ggplot2)
ggplot(df, aes(x=mass, y=glucose)) +
  geom_point() +
  labs(title="Glucose and BMI", x="BMI", y="Glucose")
```

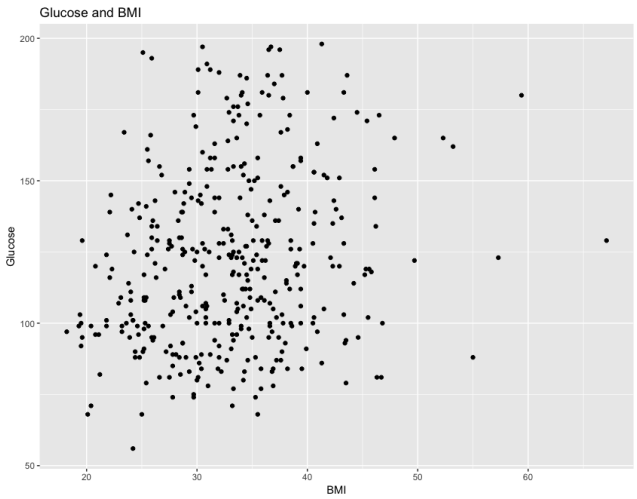

Figure 19.1: Scatterplot of BMI and Glucose

19.2.1 Improving Appearance

The above graph is a little blah. Let's add some color. Also, in the graph above the points seem to be a blog and it's hard to find a trend. We can add a smoothing line to show that as BMI increases, glucose increases.

Code 19.2.2 — ggplot. Scatterplot and Smoothing Line.

```
ggplot(df, aes(x=mass, y=glucose)) +
  geom_point(pch=20, color='blue', size=1.5) +
  geom_smooth(method='lm', color='red', linetype=2) +
  labs(title="Glucose and BMI", x="BMI", y="Glucose")
```

19.2.2 Facet Grid

In the notebook on the github, we made two new factor columns, glucose_high and insulin_high which are 1 if the value is above the mean and 0 otherwise. Then we plot with mass on the x axis and age on the y axis. We have 4 plots within the space based on the two new factor columns. Also the points are shaped according to whether the observation has diabetes or n ot and the color indicates the number of pregnancies.

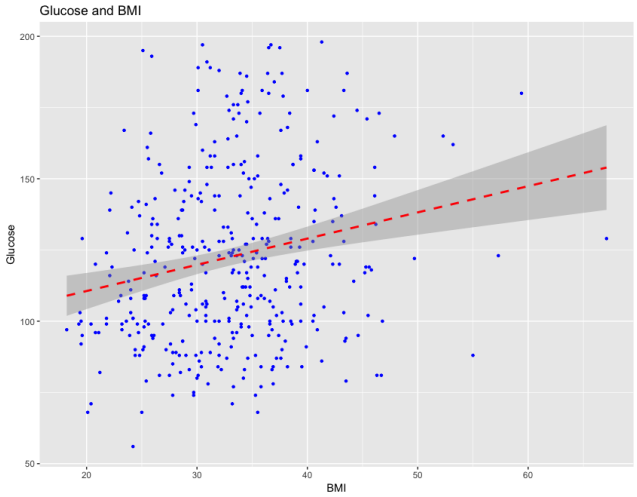

Figure 19.2: Scatterplot with Smoothing Line

Code 19.2.3 — ggplot. Facet Grid.

```
ggplot(df,
  aes(x=df$mass, y=df$age, shape=diabetes, col=pregnant)) +
  geom_point(size=2) +
  facet_grid(df$glucose_high~df$insulin_high)
```

19.2.3 Histogram

We have seen many histograms throughout the book with base R but ggplot has its own take on the histogram, boxplot, and other graph types.

Code 19.2.4 — ggplot. Histogram.

```
ggplot(df, aes(x=mass)) +
  geom_histogram(fill="cornsilk4")
```

19.2.4 Boxplot and Rug

The rug on the side shows the distribution. The points overlaying the boxplot add more info. The boxes are notched at the median. The result of the following code can be seen in Figure 19.5

Code 19.2.5 — ggplot. Boxplot and Rug.

```
ggplot(df, aes(x=diabetes, y=mass)) +
  geom_boxplot(notch=TRUE) +
  geom_point(position="jitter", color="cornflowerblue", alpha=.5) +
  geom_rug(color="cornflowerblue")
```

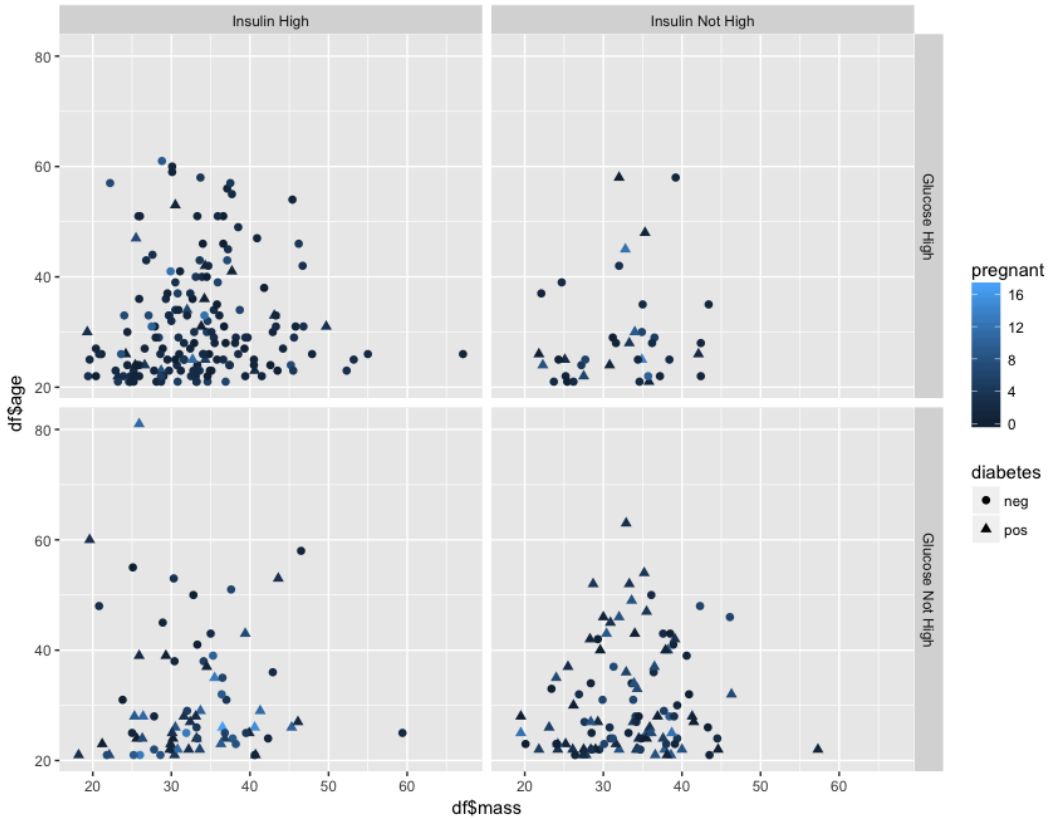

Figure 19.3: Facet Grid

19.2.5 Density Plot

This plot lets you see the density of diabetes positive and negative overlapping each other because the alpha parameter makes them semi translucent. See Figure 19.6.

Code 19.2.6 — ggplot. Density Plot.

```
ggplot(df, aes(x=mass, fill=diabetes)) +
  geom_density(alpha=0.4)
```

19.2.6 Bubble Chart

The bubble size is indicative of the number of pregnancies. See Figure 19.7.

Code 19.2.7 — ggplot. Bubble Chart.

```
ggplot(df,
       aes(x=mass, y=glucose, size=pregnant)) +
  geom_point(shape=21, fill="cornflowerblue")
```

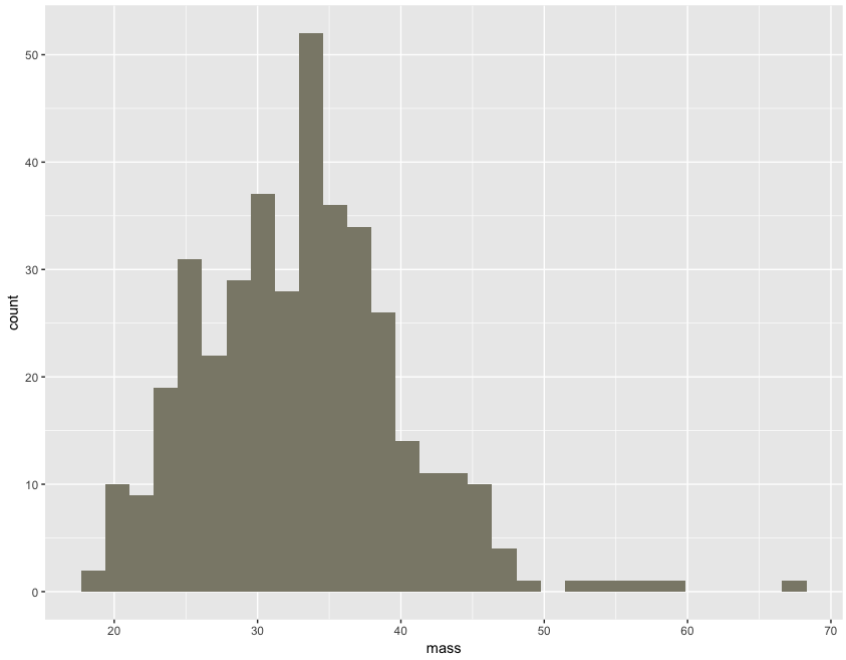

Figure 19.4: Histogram

19.2.7 Grid

In standard R we arrange plots in a grid layour with par(). In ggplot we use grid.arrange(). See Figure 19.8.

Code 19.2.8 — ggplot. Grid Arrange.

```
library(gridExtra)
p1 <- ggplot(df, aes(x=insulin_high)) + geom_bar(fill="blue")
p2 <- ggplot(df, aes(x=glucose_high)) + geom_bar(fill="blue")
grid.arrange(p1, p2, ncol=2)
```

19.2.8 Going Further with Graphs

- A cheat sheet for ggplot2 from RStudio: ggplot
- Colors in R can be referenced numerically or by name. Here is a list of names: R colors
- Quick-R reference for things like plotting symbols in base R and ggplot: Quick-R

19.3 Shiny

An overview of modern R would not be complete without a mention of Shiny, an R package that helps you make interactive web apps. Shiny apps have two components: the user interface and the server function. The shiny package has eleven built-in demos to show how it works. If you are interested in creating Shiny web apps, there is ample information to get started here: `https://shiny.rstudio.com/`

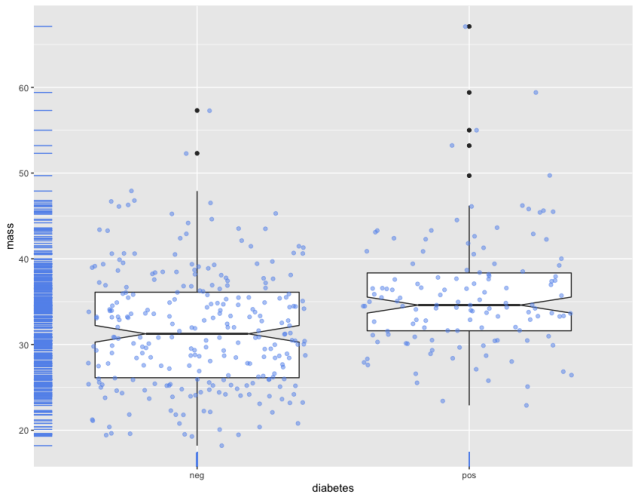

Figure 19.5: Boxplot and Rug

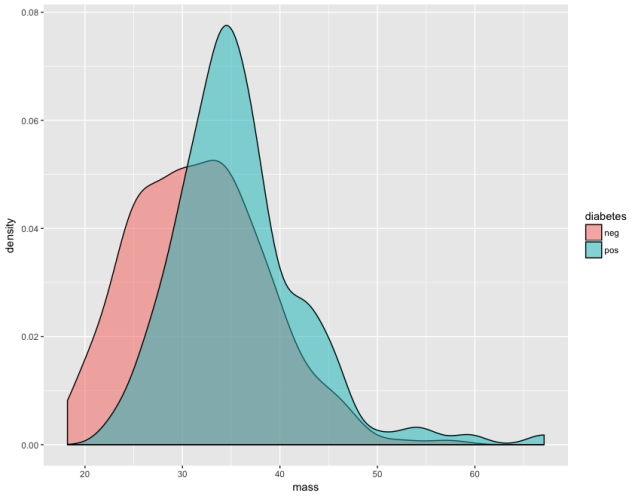

Figure 19.6: Density Plot

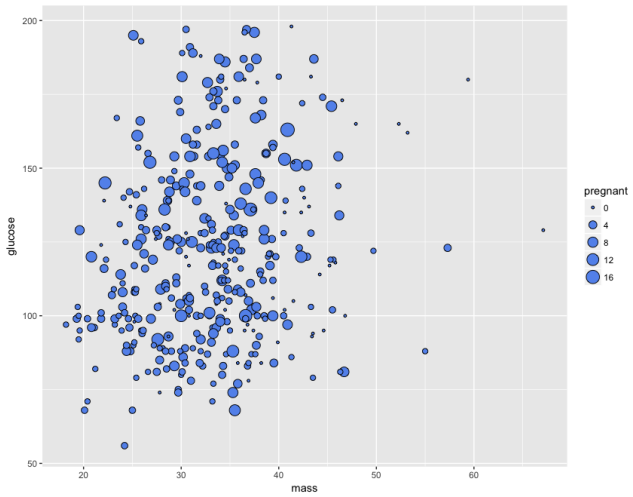

Figure 19.7: Bubble Chart

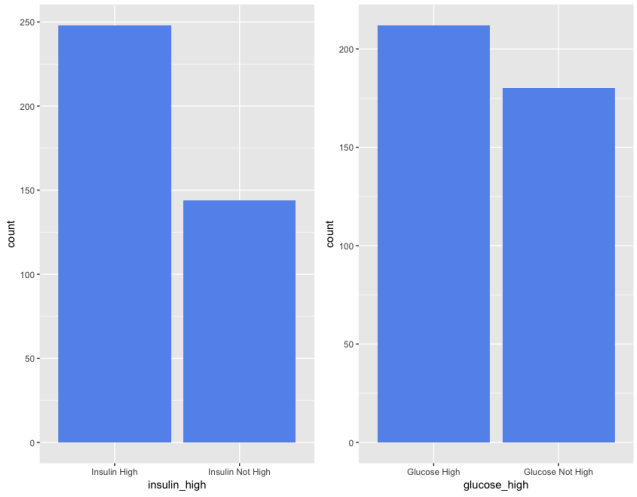

Figure 19.8: Grid Arrange