한양대학교
HANYANG UNIVERSITY

Programming statistical graphics

Seoncheol Park

3.1.1 Bar charts and dot charts

- The most basic type of graph is one that summarizes a single set of numbers.
- **Bar charts** and **dot charts** do this by displaying a bar or dot whose length or position corresponds to the number.

Example 3.1

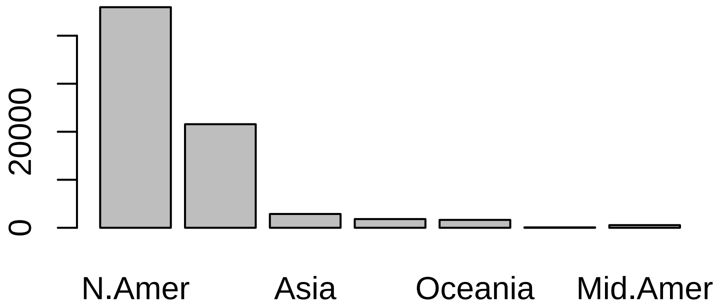
- The `WorldPhones` matrix holds counts of the numbers of telephones in the major regions of the world for a number of years. The first row of the matrix corresponds to the year 1951.
- In order to display these data graphically, we first extract that row.

```
WorldPhones51 <- WorldPhones[1, ]
WorldPhones51
```

```
  N.Amer   Europe     Asia   S.Amer  Oceania   Africa Mid.Amer
   45939    21574     2876     1815     1646       89      555
```

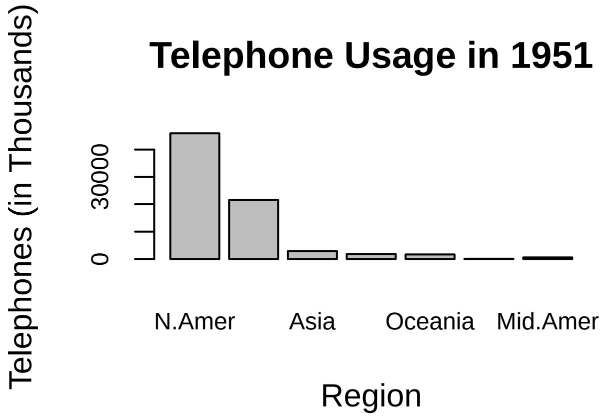
- The default code to produce a bar chart of these data using the `barplot()` function is

```
barplot(WorldPhones51)
```



- We'd like to display a title at the top, to include informative axis labels, and to reduce the size of the text associated with the axes.

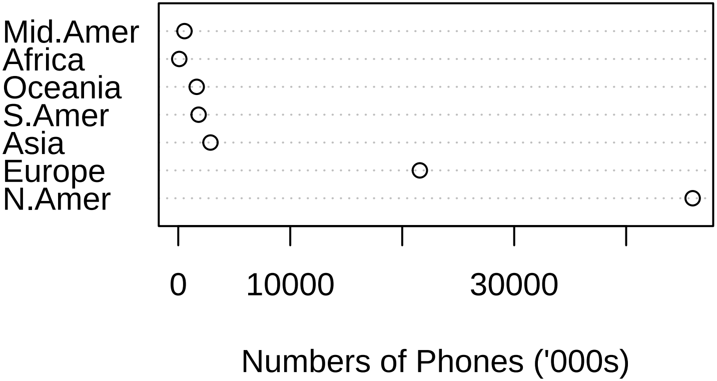
```
barplot(WorldPhones51, main = "Telephone Usage in 1951", cex.names = 0.75,
        cex.axis = 0.75, ylab = "Telephones (in Thousands)", xlab="Region")
```



- The `cex.names = 0.75` argument reduced the size of the region names to 0.75 of their former size, and the `cex.axis = 0.75` argument reduced the labels on the vertical axis by the same amount.
  - The `main` argument sets the main title for the plot, and
  - the `ylab` and `xlab` arguments are used to include axis labels.

- An alternative way to plot the same kind of data is in a dot chart

```
dotchart(WorldPhones51, xlab = "Numbers of Phones ('000s)")
```



Example 3.2

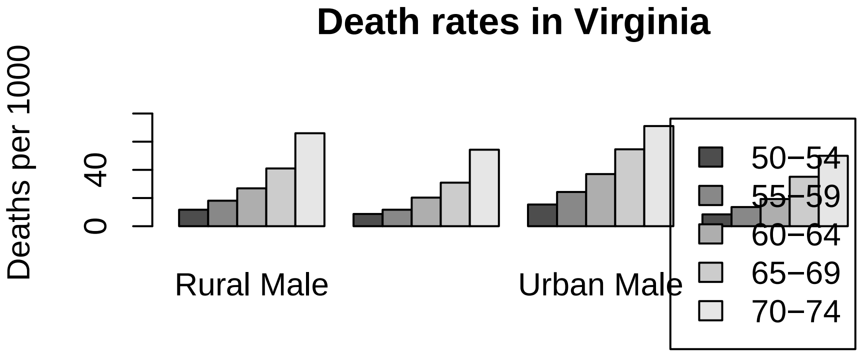
- The `VADeaths` data set in R contains death rates (number of deaths per 1000 population per year) in various subpopulations within the state of Virginia in 1940.

```
VADeaths
```

```
      Rural Male Rural Female Urban Male Urban Female
50-54       11.7          8.7       15.4          8.4
55-59       18.1         11.7       24.3         13.6
60-64       26.9         20.3       37.0         19.3
65-69       41.0         30.9       54.6         35.1
70-74       66.0         54.3       71.1         50.0
```

- This data set may be displayed as a sequence of bar charts, one for each subgroup.

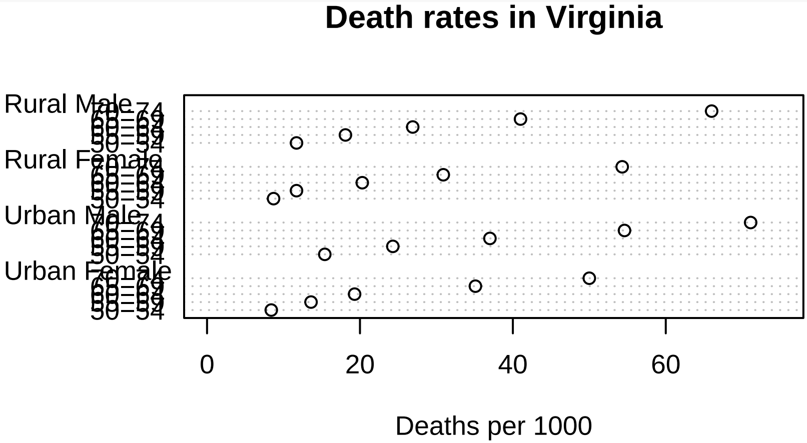
```
barplot(VADeaths, beside = TRUE, legend = TRUE, ylim = c(0, 90),
        ylab = "Deaths per 1000",
        main = "Death rates in Virginia")
```



- The bars correspond to each number in the matrix.
  - The `beside = TRUE` argument causes the values in each column to be plotted side-by-side;
  - `legend = TRUE` causes the legend in the top right to be added.
  - The `ylim = c(0, 90)` argument modifies the vertical scale of the graph to make room for the legend.
  - `main = "Death rates in Virginia"` sets the main title for the plot.

Example 3.3

```
dotchart(VADeaths, xlim = c(0, 75), xlab = "Deaths per 1000",
         main = "Death rates in Virginia", cex = 0.8)
```

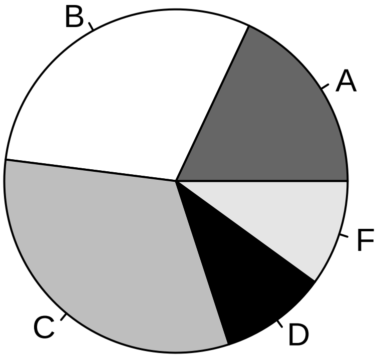


- Since it is natural to want to compare the total rates in the different groups, we set the `xlim=c(0,75)`.
- We have also set `cex=0.8`. This shrinks the plotting character to 80% of its default size, but more importantly, shrinks the axis tick labels to 80% of their default size.

3.1.2 Pie charts

- **Pie charts** display a vector of numbers by breaking up a circular disk into pieces whose angle (and hence area) is proportional to each number.
- For example, the letter grades assigned to a class might arise in the proportions, A: 18%, B: 30%, C: 32%, D: 10%, and F: 10%.

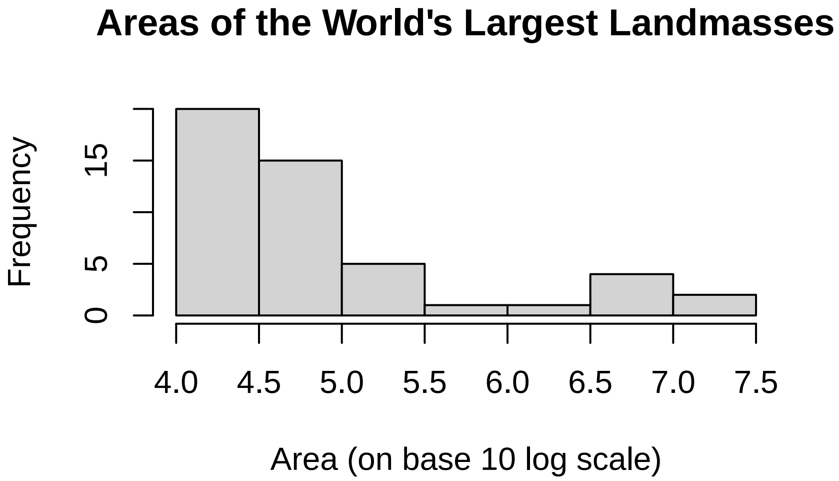
```r
groupsizes <- c(18, 30, 32, 10, 10)
labels <- c("A", "B", "C", "D", "F")
pie(groupsizes, labels, col = c("grey40", "white", "grey", "black", "grey90"))
```



3.1.3 Histograms

- A **histogram** is a special type of bar chart that is used to show the frequency distribution of a collection of numbers.
- Each bar represents the count of x values that fall in the range indicated by the base of the bar.
- Usually all bars have the same width; this is the default in R. In this case, the height of each bar is proportional to the number of observations in the corresponding interval.
- If bars have different widths, then the area of the bar should be proportional to the count; in this way the height represents the density (i.e. the frequency per unit of x).
- In base graphics, `hist(x, ...)` is the main way to plot histograms. Here x is a vector consisting of numeric observations, and optional parameters in … are used to control the details of the display.

```r
hist(log(1000*islands, 10), xlab = "Area (on base 10 log scale)",
     main = "Areas of the World's Largest Landmasses")
```



3.1.4 Boxplots

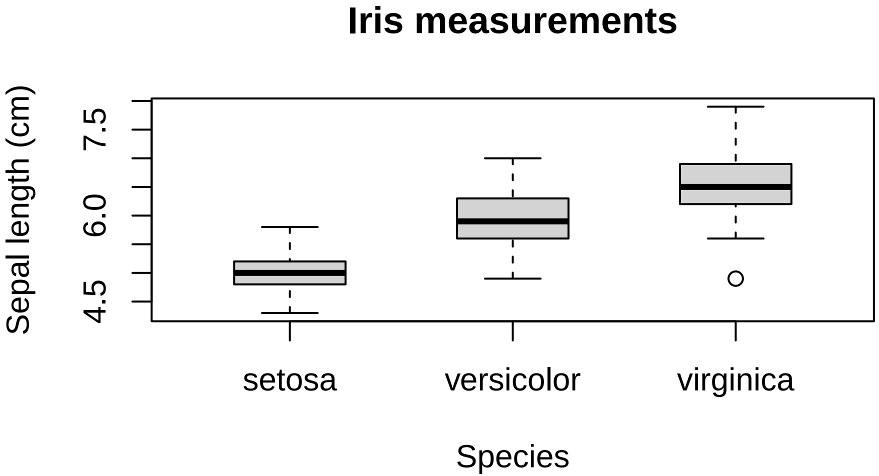
- A **boxplot** is an alternative to a histogram to give a quick visual display of the main features of a set of data.
- The box gives an indication of the location and spread of the central portion of the data, while the extent of the lines (the **whiskers**) provides an idea of the range of the bulk of the data.
- In some implementations, outliers (observations that are very different from the rest of the data) are plotted as separate points.
- The basic construction of the box part of the boxplot is as follows:

1. A horizontal line is drawn at the median.
2. Split the data into two halves, each containing the median.
3. Calculate the upper and lower quartiles as the medians of each half, and draw horizontal lines at each of these values. Then connect the lines to form a rectangular box.

- The box thus drawn defines the **interquartile range (IQR)**. This is the difference between the upper quartile and the lower quartile.
- We can use the IQR to give a measure of the amount of variability in the central

- The lower whisker is drawn from the lower end of the box to the smallest value that is no smaller than 1.5 IQR below the lower quartile.
- Similarly, the upper whisker is drawn from the middle of the upper end of the box to the largest value that is no larger than 1.5 IQR above the upper quartile.
- The rationale for these definitions is that when data are drawn from the normal distribution or other distributions with a similar shape, about 99% of the observations will fall between the whiskers.
- Boxplots are convenient for comparing distributions of data in two or more categories, with a number (say 10 or more) of numerical observations per category.

```
boxplot(Sepal.Length ~ Species, data = iris, ylab = "Sepal length (cm)",
        main = "Iris measurements", boxwex = 0.5)
```



- The syntax `Sepal.Length ~ Species` is read as **Sepal.Length depending on Species,** where both are columns of the data frame specified by `data = iris`.

3.1.5 Scatterplots

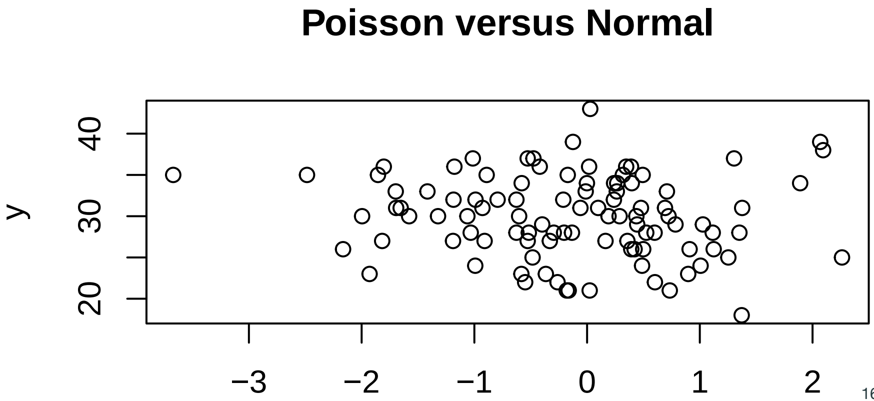
- To see the relationships between different variables, one of the most commonly used plots is the **scatterplot**, in which points $(x_i, y_i), i = 1, \ldots, n$ are drawn using dots or other symbols. These are drawn to show relationships between the x_i and y_i values.
- In R, scatterplots are drawn using the `plot()` function. Its basic usage is `plot(x, y, ...)` where `x` and `y` are numeric vectors of the same length holding the data to be plotted.
- One important optional argument is type.
  - The default is `type = "p"`, which draws a scatterplot.
  - Line plots (in which line segments join the (x_i, y_i) points in order from first to last) are drawn using `type = "l"`.
  - Many other types are available, including `type = "n"`, to draw nothing: this just sets up the frame around the plot, allowing other functions to be used to draw in it.

```
x <- rnorm(100) # assigns 100 random normal observations to x
y <- rpois(100, 30) # assigns 100 random Poisson observations
                    # to y; mean value is 30
mean(y) # the resulting value should be near 30
```

```
[1] 29.95
```

- The `main` argument sets the main title for the plot.

```
plot(x, y, main = "Poisson versus Normal")
```



- Other possibilities you should try:

```
plot(x, y, pch = 16) # changes the plot symbol to a solid dot
plot(x, y, type = 'l') # plots a broken line (a dense tangle of line
                       # segments here)
plot(sort(x), sort(y), type = 'l') # a plot of the sample "quantiles"
```

Example 3.4

- The `Orange` data frame is in the `datasets` package installed with R. It consists of 35 observations on the age (in days since December 31, 1968) and the corresponding circumference of five different orange trees, with identifiers

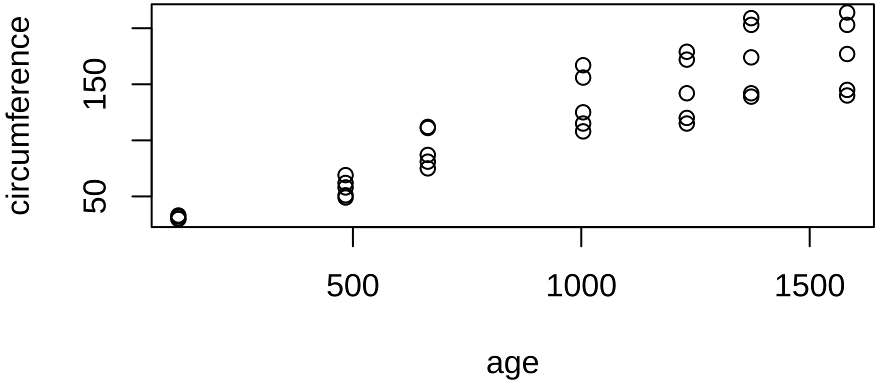
```
unique(as.character(Orange$Tree))
```

```
[1] "1" "2" "3" "4" "5"
```

(Since `Orange$Tree` is a factor, we use `as.character()` to get the displayed form, and `unique()` to select the unique values.)

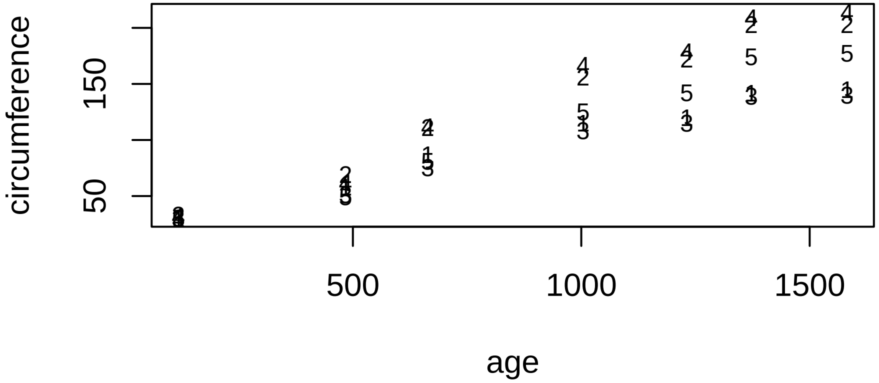
- We have used the graphics formula and the `data` argument as in the earlier boxplot example.
- The plot function finds `circumference` and age in the `Orange` data frame, and plots the ordered pairs of (`age`, `circumference`) observations.

```
plot(circumference ~ age, data = Orange)
```



- This figure hides important information: the observations are not all from the same tree, and they are not all from different trees; they are from five different trees, but we cannot tell which observations are from which tree.
- The `pch` parameter controls the plotting character.
  - The default setting `pch = 1` yields the open circular dot.
  - We can also ask for different characters to be plotted; for example, `pch = "A"` causes R to plot the character `A`.

```
plot(circumference ~ age, data = Orange, pch = as.character(Tree), cex = 0.75)
```

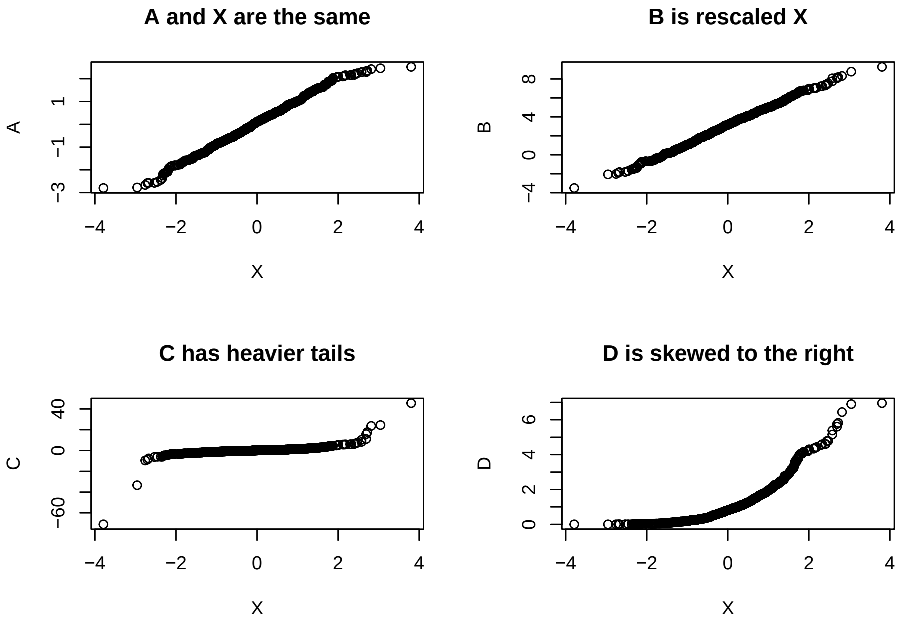


- The `cex` parameter controls the size of the plotting character, and the `pch` parameter has been assigned the levels of the Tree column; because `Tree` is a factor, care must be taken in order that the level values are used, and not the factor codes, hence the use of `as.character()`.

3.1.7 QQ plots

- **Quantile-quantile plots** (otherwise known as **QQ plots**) are a type of scatterplot used to compare the distributions of two groups or to compare a sample with a reference distribution.
- In the case where there are two groups of equal size, the QQ plot is obtained by
  - first sorting the observations in each group: $X[1] \leq \ldots \leq X[n]$ and $Y[1] \leq \ldots \leq Y[n]$.
  - Next, draw a scatterplot of $(X[i], Y[i])$, for $i = 1, \ldots, n$.
- When the groups are of different sizes, some scheme must be used to artificially match them. R reduces the size of the larger group to the size of the smaller one by keeping the minimum and maximum values, and choosing equally spaced quantiles between.
- When plotting a single sample against a reference distribution, theoretical quantiles are used for one coordinate.
- To avoid biases, quantiles are chosen corresponding to probabilities $(i - 1/2)/n$: these are centered evenly between zero and one.
- When the distributions of X and Y match, the points in the QQ plot will lie near the line $y = x$.

Example 3.6


A and X are the same
B is rescaled X
C has heavier tails
D is skewed to the right
A
B
C
D
X

- The `mfrow` parameter of the `par()` function is giving a 2×2 layout.
    - The first plot is based on identical normal distributions,
    - the second plot is based on normal distributions having different means and standard deviations,
    - the third plot is based on a standard normal and a t distribution on 2 degrees of freedom, and
    - the fourth plot is based on a standard normal compared with an exponential distribution.
- Since we used simulated random numbers here, you'll likely see slightly different results if you run the same code. More information about the functions `rnorm()`, `rt()` and `rexp()` is given in Chapter 5.

3.3 Low level graphics functions

- Functions like `barplot()`, `dotchart()`, and `plot()` do their work by using low level graphics functions to draw lines and points, to establish where they will be placed on a page, and so on.

3.3.1 Adding to plots

- Several functions exist to add components to existing graphs:

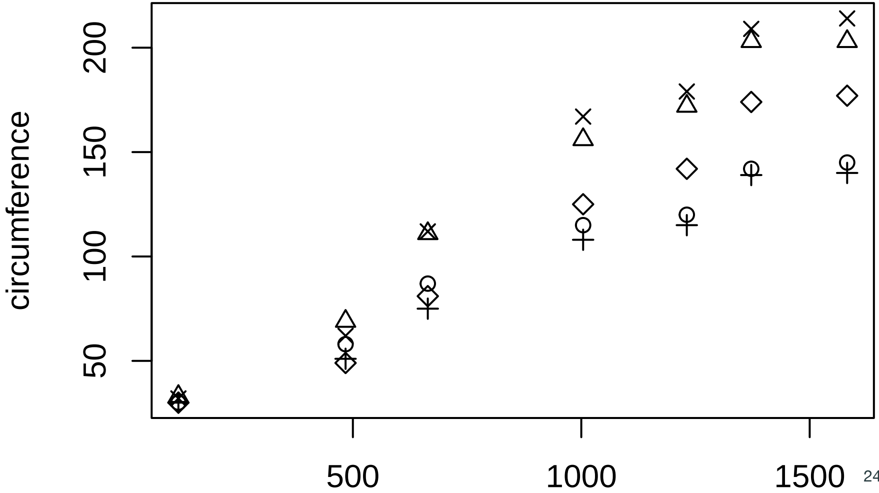
```
points(x,y,...) #adds points
lines(x,y,...)  #adds line segments
text(x,y,labels,...)  #adds text into the graph
abline(a,b,...) #adds the line $y=a+bx$
abline(h=y,...) #adds a horizontal line
abline(v=x,...) #adds a vertical line
polygon(x,y,...)  #adds a closed an possibly filled polygon
segments(x0,y0,x1,y1,...) #draws line segments
arrows(x0,y0,x1,y1,...) #draws arrows
symbols(x,y,...)  #draws circles, squares, thermometers, etc.
legend(x,y,legend,...)  #draws a legend
```

- The optional arguments to these functions specify the color, size, and other characteristics of the items being added.

Example 3.7

- Consider the `Orange` data frame again. In addition to using different plotting characters for the different trees, we will pass lines of best fit (i.e. **least-squares regression lines**) through the points corresponding to each tree.

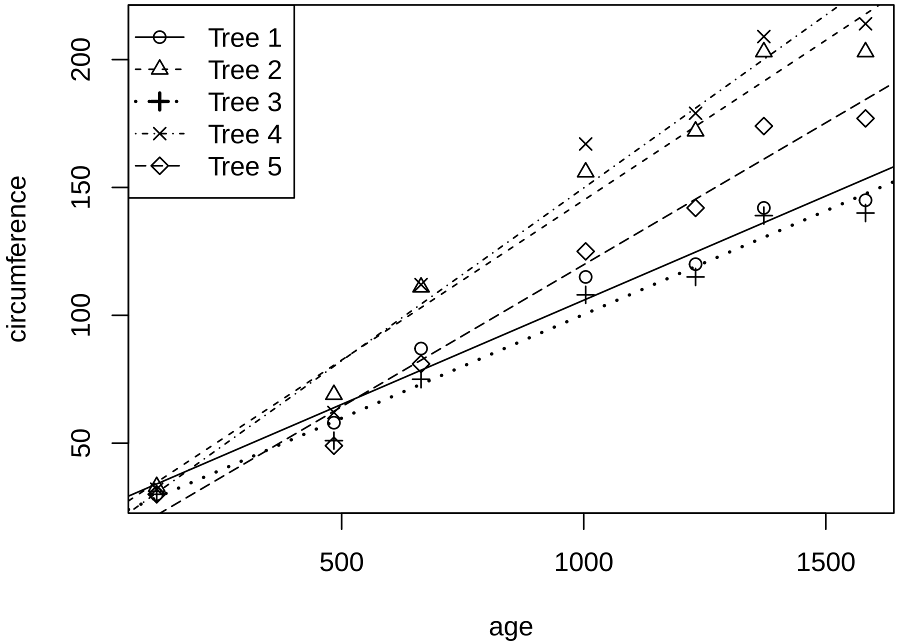
```r
plot(circumference ~ age, pch = as.numeric(as.character(Tree)), data = Orange)
```



- The best-fit lines for the five trees can be obtained using the `lm()` function which relates circumference to age for each tree. A legend has been added to identify which data points come from the different trees.

```
plot(circumference ~ age, pch = as.numeric(as.character(Tree)), data = Orange)
abline(lm(circumference ~ age, data = Orange, subset = Tree == "1"), lty = 1)
abline(lm(circumference ~ age, data = Orange, subset = Tree == "2"), lty = 2)
abline(lm(circumference ~ age, data = Orange, subset = Tree == "3"), lty = 3, lwd = 2)
abline(lm(circumference ~ age, data = Orange, subset = Tree == "4"), lty = 4)
abline(lm(circumference ~ age, data = Orange, subset = Tree == "5"), lty = 5)
legend("topleft", legend = paste("Tree", 1:5), lty = 1:5, pch = 1:5, lwd = c(1, 1, 2,
```

- In these plots `lty` gives the line type, and `lwd` gives the line width.


Tree 1
Tree 2
Tree 3
Tree 4
Tree 5
circumference
50
100
150
200
500
1000
1500
age

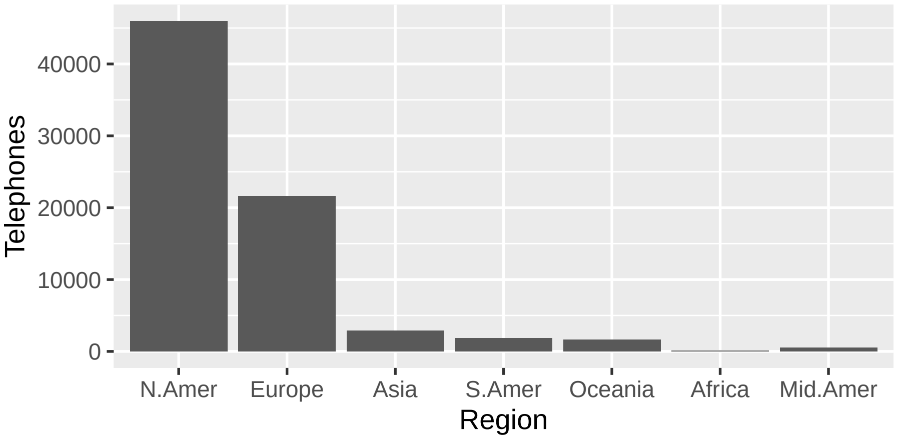
`ggplot2`

- The ideas behind `ggplot2` were first described in a 1999 book called *The Grammar of Graphics* by Leland Wilkinson, and a second expanded edition was published in 2005. They were expanded again and popularized when Hadley Wickham published `ggplot2` in 2007.
- Most `ggplot2` plot expressions start with a call to the `ggplot()` function. Its first argument is data, and that's where we specify the data component of the plot, which is always a data frame or tibble.
- The second component of every plot is called the **aesthetic mapping** of the plot, or **aesthetics** for short. This doesn't refer to the appreciation of beauty; it refers to the ways that quantities in our data are expressed in the plot. We use the `aes()` function to specify the aesthetics.
- Because Region is a factor, `geom_col()` displays one bar per level.
- Because we had `aes(x = Region,y = Telephones)` the bars are vertical. We could get horizontal bars by using `aes(y = Region, x = Telephones)`. Try it!

Example 3.8

- To plot the world phone data that we saw at the start of this chapter

```
library(ggplot2)
region <- names(WorldPhones51)
phones51 <- data.frame(Region = factor(region, levels = region), Telephones = WorldPhon
ggplot(data = phones51, aes(x = Region, y = Telephones)) + geom_col()
```



- The new feature is in the `ggplot` invocation where `aes` says that we want the Region names on the x-axis in their original order, and the telephone counts on the y-axis.
- We want to display the data using bars, hence the use of the `geom_col` function. The statistic to display is the identity, i.e. just the value itself.

Structure of `ggplot2`

- The general idea in `ggplot2` is that plots are described by a sum of objects produced by function calls. As with any addition in R, we use `+`, but you should think of the whole expression as a way to describe the plot as a combination of different components.

`ggplot` name	Description
`data`	your data frame
`aes()`	"aesthetics": things like position, color, size, and shape that control how your data is encoded in the display
`geom_*()`	a geometric object or layer of the graph that represents the data

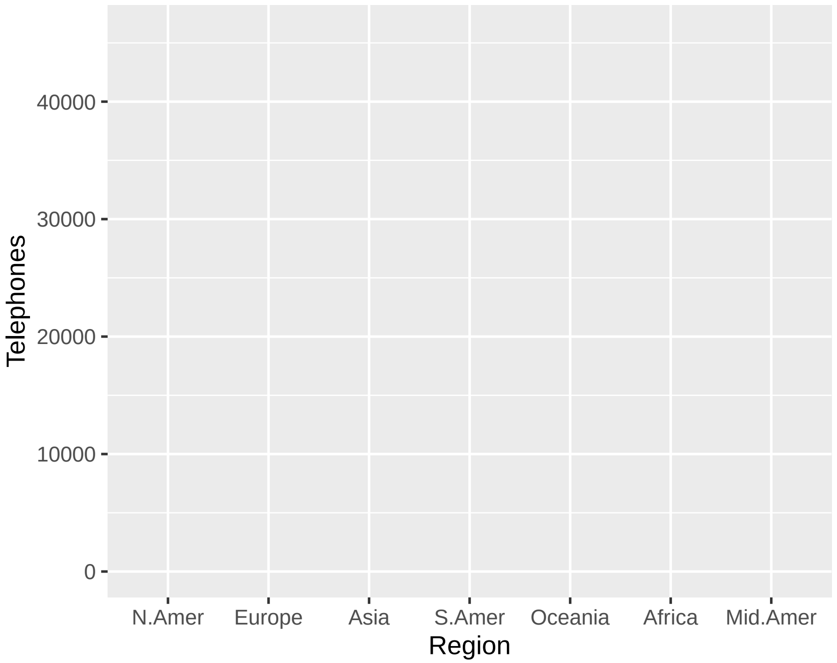
3.4.1 Details of the ggplot2 grammar

- As we've seen, `ggplot2` plots are usually created as a sum of function calls. Each of those function calls produces a special object, which the `ggplot2` code knows how to combine, provided you follow certain rules.
- First, you need to start with a `ggplot` object. This can be produced by a call to `ggplot()` or to some other function that calls it, and it can be saved in a variable and used later in a different plot.
- The `ggplot` object sets certain defaults which can be used by the layers of the plot. Normally the first argument specifies a data frame or tibble, and that data can be used in all layers of the plot. It's also common to specify aesthetics as the second argument, and again, these will be used everywhere unless overridden by other settings. Thus a first component to a `ggplot2` plot could be produced as

```
g1 <- ggplot(phones51, aes(Region, Telephones))
```

- Because we assigned the result to `g1`, it is not printed, and no graph is displayed. To display it, we can print that object:

```
g1
```



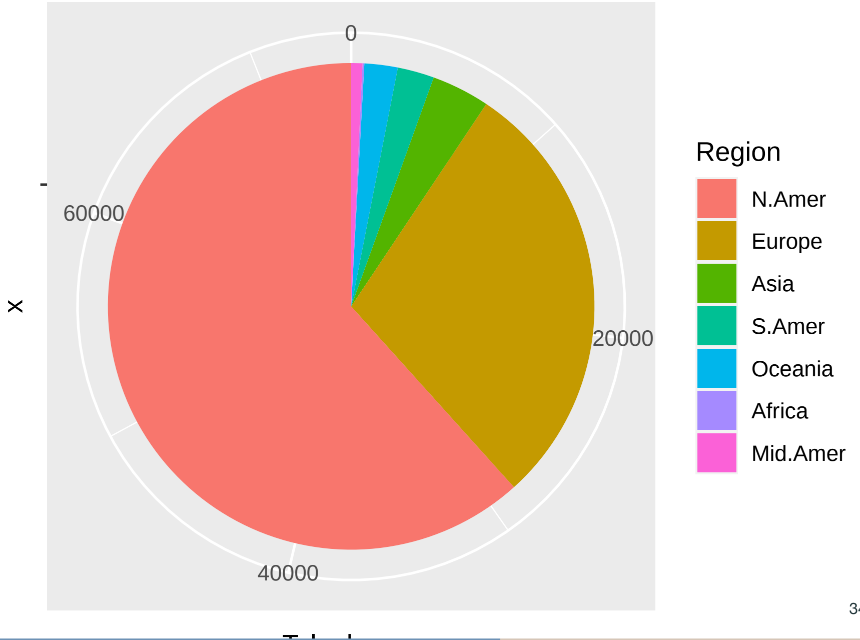
Telephones
40000
30000
20000
10000
0
N.Amer
Europe
Asia
S.Amer
Oceania
Africa
Mid.Amer
Region

- After our call to `ggplot()`, we will add other objects to change the display:

`ggplot2` name	Description
`scale_*()`	scales: how the data maps to the plot
`trans`	transformations to the data within the scale
`coord_*()`	coordinate systems
`theme_*()`	themes: overall choice of color and other parts of the appearance
`facet_*()`	facets: subsettting the plot.

- Because `Region` is a factor, it is automatically displayed using a discrete scale, and because `Telephones` is a number, it is displayed on a continuous scale.
- These automatic choices could be changed by adding in a call to a different `scale_*()` function. For example, using `scale_y_binned(n.breaks = 4)` would round the y-axis values into five bins before plotting.
- Transformations are changes to values before plotting. For example, `scale_y_continuous(trans = "log10")` will take the base 10 logarithm of the y-axis values before plotting.
- Finally, the coordinate system determines how the x and y values are displayed on the plot. For example, to display a pie chart in `ggplot2`, you display a bar plot in polar coordinates:
- The theme of a plot controls the general appearance. All of our plots have used the default `theme_gray()`, but others are available.

```
ggplot(phones51, aes(x = "", y = Telephones, fill = Region)) +
  coord_polar(theta = "y") + geom_col()
```



3.4.2 Layers in ggplot2

- There are many ways to display data, and `ggplot2` puts **ways to display data** into the `geom_*()` layer functions. Some common `ggplot2` layers are below:

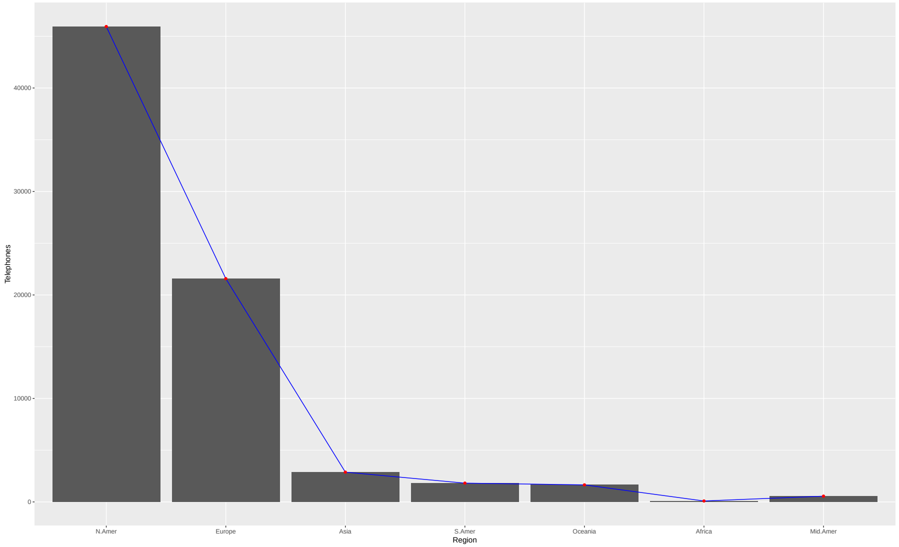
Function	Description
`geom_col()`	bar plots
`geom_bar()`	bar plots of counts
`geom_histogram()`	histograms
`geom_points()`	scatterplots
`geom_jitter()`	scatterplots with jittering
`geom_line()`	line plots
`geom_contour()`	contour plots
`geom_contour_filled()`	filled contour plots
`geom_density_2d()`	contour plots of density estimates
`geom_boxplot()`	boxplots
`geom_violin()`	violin plots

- Each kind of layer works with a different set of aesthetics. We have already seen `x` and `y` aesthetics; others that are commonly supported are
  - `alpha`: transparency
  - `color` (or `colour`)
  - `group`: group
- Where appropriate, some others are supported:
  - `linetype`
  - `size`
  - `fill`: the fill color
  - `weight`: the statistical weight to give to each observation • shape: the shape of point to plot
  - `stroke`: the thickness of parts of points being plotted

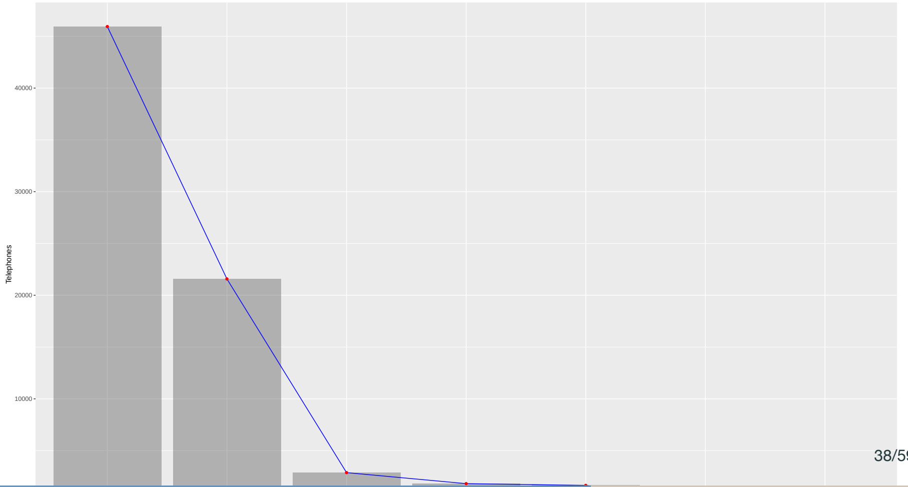
Example 3.9

- If we had used `geom_point()` or `geom_line()` instead, we would see points or lines.

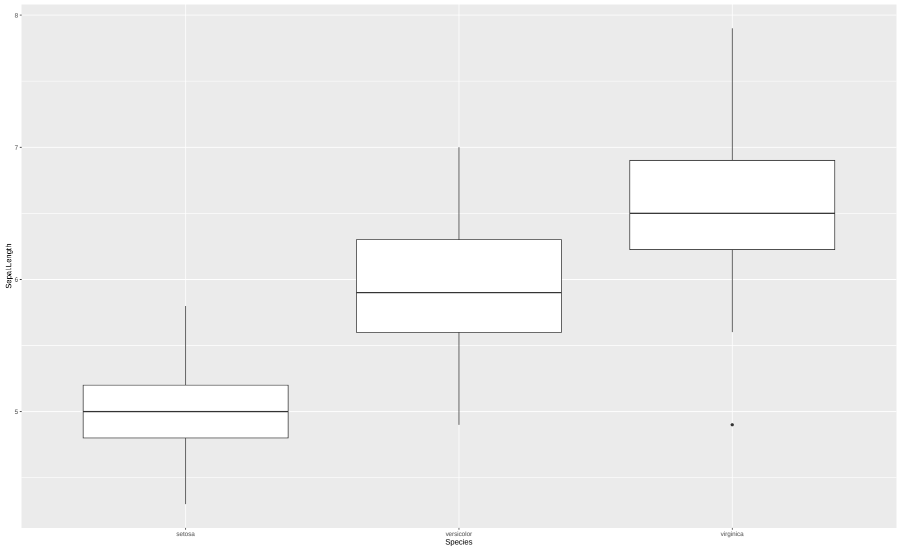
```
ggplot(phones51, aes(Region, Telephones)) +
  geom_col() +
  geom_line(col = "blue", aes(x = as.numeric(Region))) + geom_point(col = "red")
```



- The `ggplot()` call sets up the scales and coordinate system, and sets the default theme. The `geom_col()` call draws the bars in the plot as before, and `geom_point(col = "red")` draws points in red. The middle layer `geom_line(col = "blue", aes(x = as.numeric (Region)))` draws the blue line.
- To gain a clearer view of the lines and points, we might increase the transparency of the bars, by replacing the line of code

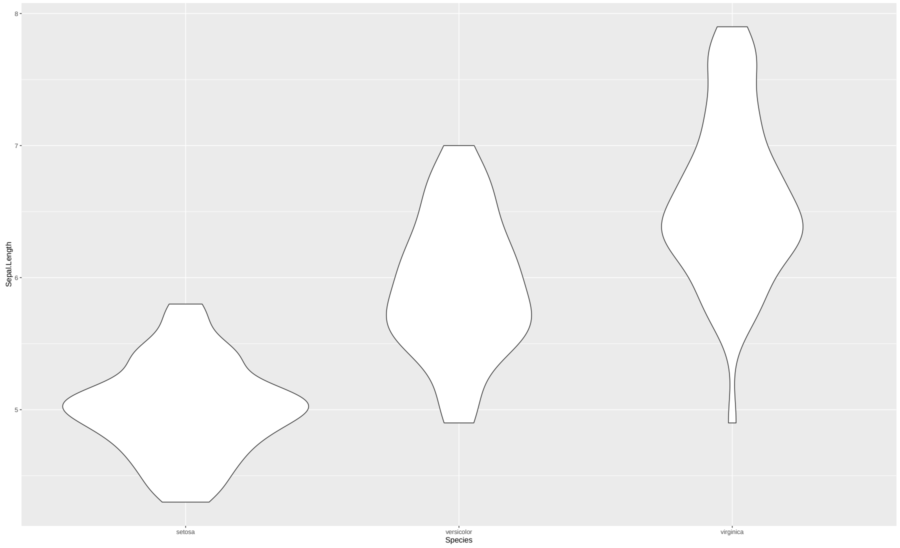


```r
ggplot(iris, aes(x = Species, y = Sepal.Length)) + geom_boxplot()
```



- Another view of the same data using violin plots is produced by the similar code:

```
ggplot(iris, aes(x = Species, y = Sepal.Length)) + geom_violin()
```



3.4.3 Setting colors

- There are several different ways to identify colors in R. They can be spec- ified by name; the function `colors()` lists hundreds of names recognized by R:

```
str(colors())
```

```
 chr [1:657] "white" "aliceblue" "antiquewhite" "antiquewhite1" ...
```

- They can also be constructed using hexadecimal (base 16) codes for the levels of red, green, and blue. For example, red would be specified as `"#FF0000"`, where `FF`, the base 16 representation of 255, is the maximum level of red, and both green and blue have zero contribution.
- R also maintains a palette of a small number of colors that can be referenced by number.

```
palette.pals()
```

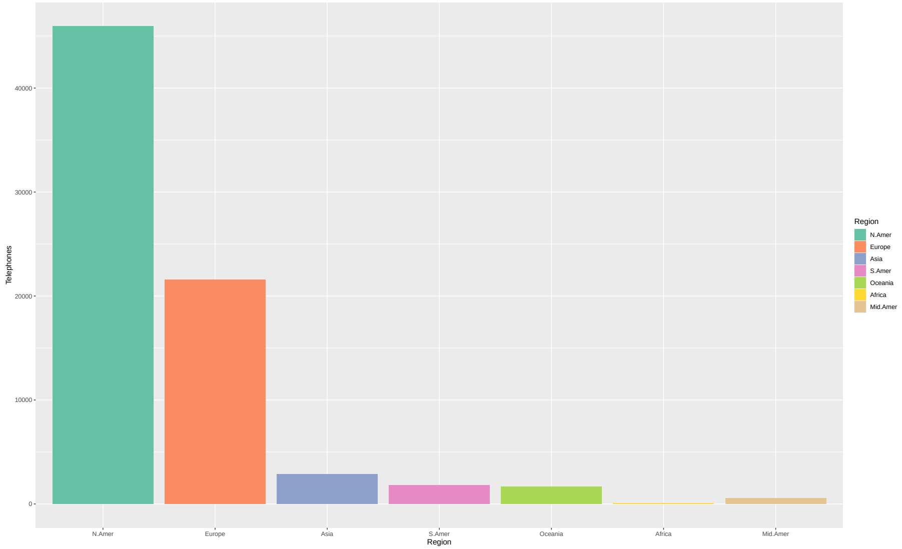
```
 [1] "R3"              "R4"              "ggplot2"         "Okabe-Ito"      
 [5] "Accent"          "Dark 2"          "Paired"          "Pastel 1"       
 [9] "Pastel 2"        "Set 1"           "Set 2"           "Set 3"          
[13] "Tableau 10"      "Classic Tableau" "Polychrome 36"   "Alphabet"       
```

```
palette("R3")
```

- With this choice of colors, we see

```
palette()
```

```r
ggplot(phones51, aes(Region, Telephones, fill = Region)) +
  geom_col() + scale_fill_brewer(palette = "Set2")
```



3.4.4 Customizing the look of a graph

- There are several functions to change the labeling on the graph.
  - The `ggtitle()` function sets a title at the top,
  - `xlab()` and `ylab()` set titles on the axes.
  - The `theme()` and `theme_*()` functions can be used to change many details of the overall look of a graph.
  - The `scale_*()` functions can be used to customize the mapping for each aesthetic.
- The `annotate()` function is interesting. It works like a layer function, but with fixed vectors of aesthetics, not values taken from the data set for the plot. For example, to add some text to a plot at location `x = 1, y = 2`, one could use `annotate("text", x = 1, y = 2, text = "Label")`. The first argument (`"text"` in this case) is part of the name of the *geom* to use.

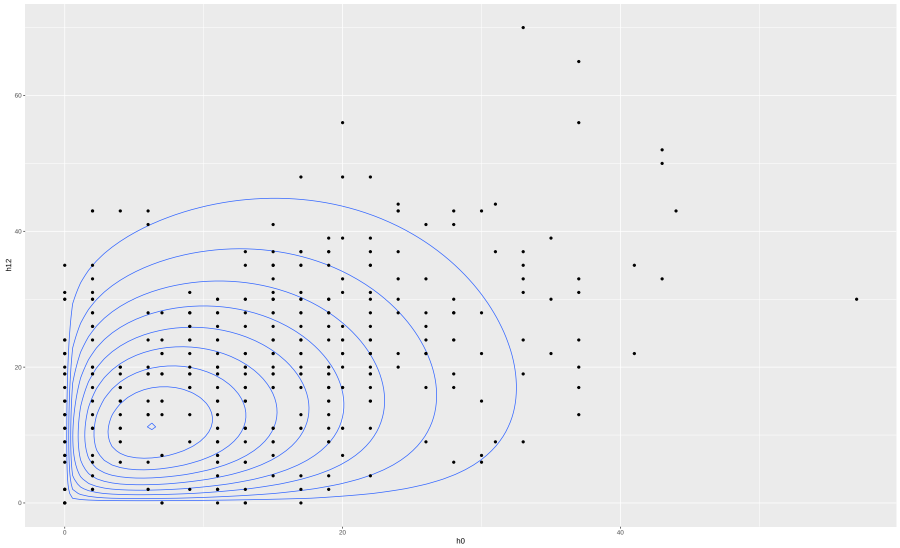
Example 3.11

- The data set windWin80 from the MPV package contains pairs of observations of wind speed at the Winnipeg International Airport.
- The columns give the wind speed in km per hour at midnight (h0) and noon (h12).
- A proposed model for the joint probability density function of these two measurements is

$$f_{X_1,X_2}(x_1,x_2) = \frac{\alpha_1\alpha_2 x_1^{\alpha_1-1} x_2^{\alpha_2-1} e^{-x_1^{\alpha_1}/\beta_2 - x_2^{\alpha_2}/(\beta_0+\beta_1 x_1)}}{\beta_2(\beta_0+\beta_1 x_1)}, \quad \text{for } x_1 \geq 0, x_2$$

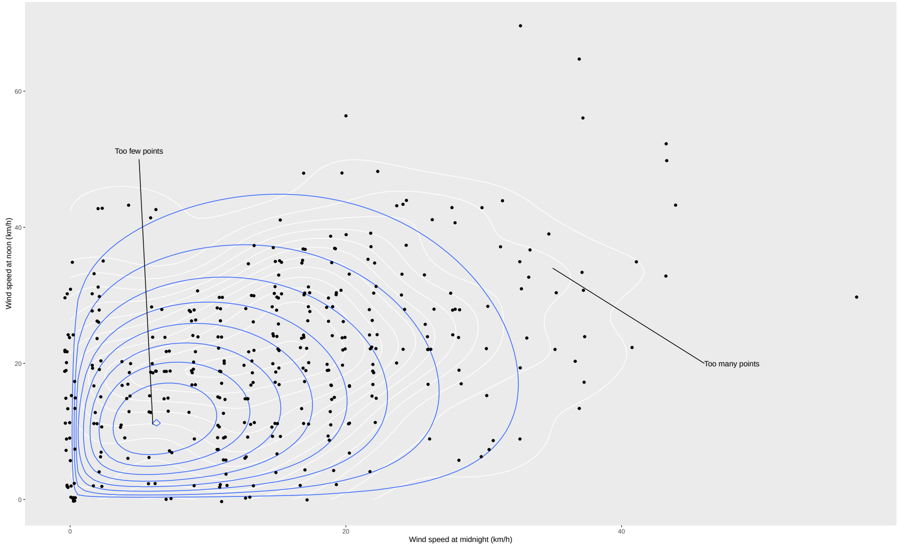
- The parameters $\alpha_1, \alpha_2, \beta_0, \beta_1$, and β_2 must be nonnegative.
- The values $\alpha_1 = 3/2, \alpha_2 = 5/3, \beta_0 = 108, \beta_1 = 5.3$, and $\beta_2 = 65$ were estimated from the data.
- We used a1, a2, b0, b1 and b2 for $\alpha_1, \alpha_2, \beta_0, \beta_1$, and β_2 respectively.

```
# Compute values from the proposed density function
a1 <- 3/2; a2 <- 5/3; b0 <- 108; b1 <- 5.3; b2 <- 65
# expand.grid(): create a grid of values in x1 and x2
grid <- with(MPV::windWin80,
             expand.grid(x1 = seq(0, max(h0), len=101), x2 = seq(0, max(h12), len=101)
grid$z <- with(grid, a1*a2*exp(-x1^a1/b2 - x2^a2/ (b0+b1*x1))*x1^(a1-1)*x2^(a2-1)/(b2*
# Plot the data on top of the density function
ggplot(MPV::windWin80, aes(x = h0, y = h12)) + geom_contour(data = grid, aes(x = x1, y
```



- From the plot, you can see these issues:
    - The points appear to fall on a regular grid of values, because the original data were rounded to the nearest integer. This might hide some points if they exactly overlap. A way to deal with overlapping points is **jittering**: move each point slightly by a random amount. (`geom_jitter()` layer function in `ggplot2`)
    - The labels `h0` and `h12` on the plot mean should be made more informative using `xlab()` and `ylab()`.
    - There appear to be too many points outside the widest contour line, and too few inside the smallest ones where the peak of the PDF occurs. We could add an estimated PDF function to the plot using `geom_density2d()`: this will estimate the PDF from the points.
    - We'll use white in `geom_density2d()`, since we are only adding it for reference: it's not the main point of the plot.
    - The grid lines are going to make the plot too busy, so we can remove them using a call to `theme()`.
    - Finally, we'll add some annotations to point out our areas of concern.

```r
ggplot(MPV::windWin80, aes(x = h0, y = h12)) +
  geom_density2d(col = "white") +
  geom_contour(data = grid, aes(x = x1, y = x2, z = z) ) +
  geom_jitter() +
  xlab("Wind speed at midnight (km/h)") +
  ylab("Wind speed at noon (km/h)") +
  theme(panel.grid = element_blank()) +
  annotate("text", x = 46, y = 20, hjust = 0, label = "Too many points") +
  annotate("segment", x = 46, y = 20, xend = 35, yend = 34) +
  annotate("text", x = 5, y = 50, vjust = -1, label = "Too few points") +
  annotate("segment", x = 5, y = 50, xend = 6, yend = 11)
```



- To remove the grid, we set it to the special value `element_blank()`.
- We drew our text annotations outside the cloud of points to make them more visible, then added line segments to point to the region we were actually talking about.

3.4.5 Faceting

- A strategy for displaying relations among three or more variables is to divide the data into subsets using the values of some of the variables, and then draw multiple plots of the values of the other variables in each of those subsets.
- In `ggplot2` this is called **faceting**, and the `facet_wrap()` and `facet_grid()` functions are used to implement it. The `facet_wrap()` function draws these plots side-by-side, wrapping the display to the next line once there are more than a few. Its first argument describes the variables to use for subsetting. This can be expressed by listing them in the `vars()` function, or giving a formula using `~`.
- The `facet_grid()` function arranges the plots in a rectangle whose rows define one set of variables for subsetting, and whose columns represent another. These can be specified using `vars()` in the `rows` and `cols` arguments, or using a formula for `rows` and skipping `cols`.
- It also has an argument `margins`, which adds a row ignoring the `cols` variables, and a column ignoring the `rows` variables.

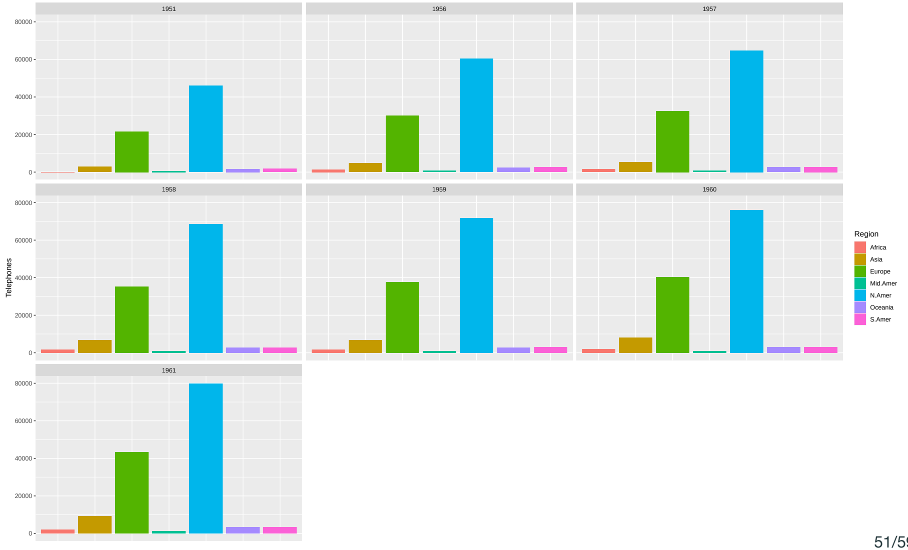
Example 3.12

- To study the trends over time in the `WorldPhones` data, we first need to convert it to a data frame.

```
phones <- data.frame(Year = as.numeric(rep(rownames(WorldPhones), 7)),
                     Region = rep(colnames(WorldPhones), each = 7),
                     Telephones = as.numeric(WorldPhones))
```

- If we are particularly interested in the changes in the distribution between regions over time, we might plot similar bar charts for each year by specifying `facet_wrap(vars(Year))` or `facet_wrap(~ Year)`.

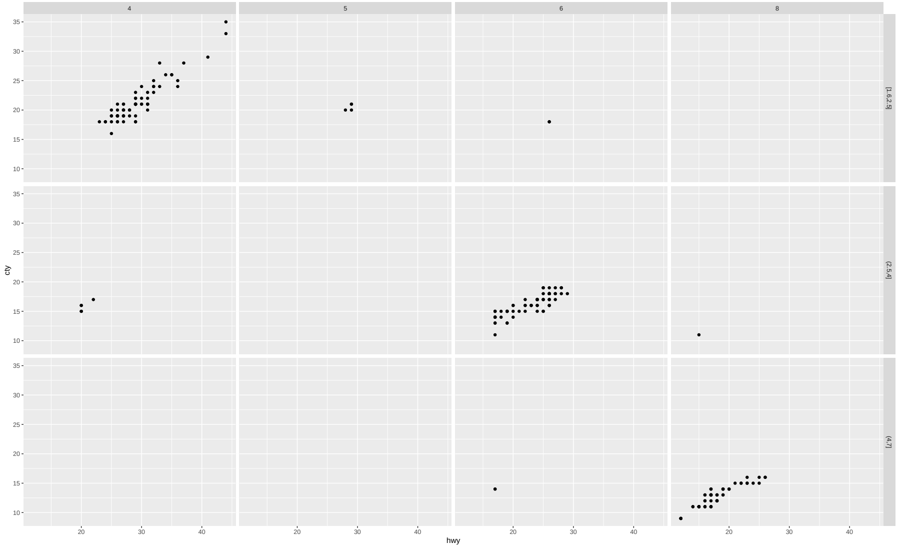
```r
ggplot(phones, aes(x = Region, y = Telephones, fill = Region)) +
  geom_col() + facet_wrap(vars(Year)) +
  theme(axis.text.x = element_blank(), axis.ticks.x = element_blank()) +
  xlab(element_blank())
```



Example 3.13

- The mpg data frame in ggplot2contains fuel economy data for 38 models of cars from 1999 to 2008.
  - The cty and hwy columns measure miles per gallon in city and highway, respectively.
  - Engine displacement in litres is given in the displ column, and cyl holds the number of cylinders.
- One would expect that fuel economy is worse in larger engines and with more cylinders, but it's not obvious how these would affect the relation between city and highway efficiency.
- To study this, we could subset the data according to values of displ and cyl and draw scatterplots of cyl versus hwy for each. However, displ is a continuous variable, and even though it is rounded to one decimal place, there are still 35 different values, so the subsets would be too small.
- A way to address this is to break displ into a smaller number of subsets based on ranges of values. The cut_*() functions in ggplot2 do this in a few different ways:
  - cut_number() gives equal numbers of cases per subset,
  - cut_interval() gives equal ranges of values per subset, and
  - cut_width() gives ranges with a specified width.

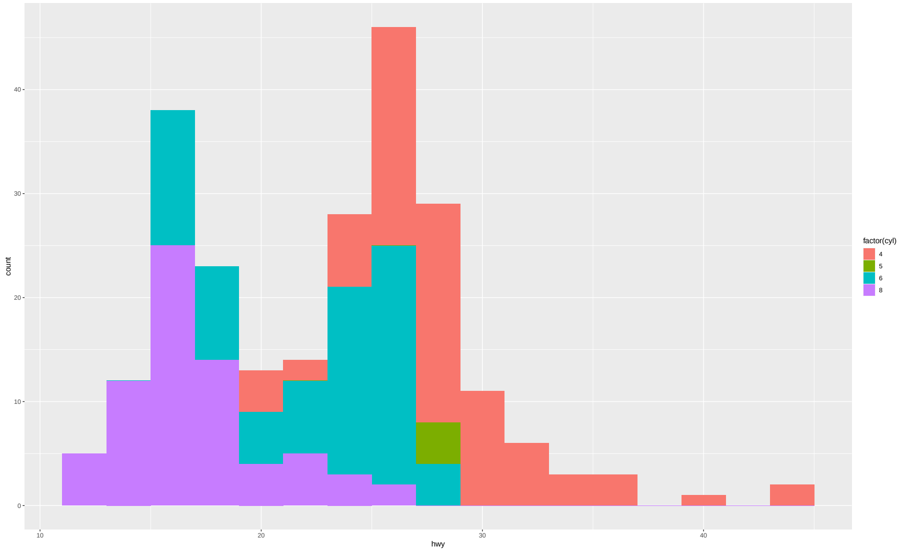
```r
ggplot(mpg, aes(hwy, cty)) +
  geom_point() +
  facet_grid(cut_number(displ, 3) ~ cyl)
```



3.4.6 Groups in ggplot2

- There are two ways that `ggplot2` uses to determine groups.
  - The usual way is to look at all of the discrete variables mentioned in the plot, and creating a group out of every unique combination of levels.
  - Groups can also be set explicitly, by specifying the `group` aesthetic.
- The `geom_histogram()` parameter `binwidth` sets the size of each bin. To draw the bars side-by- side, you could use the parameter `position = "dodge"`.
- If we had used `aes(hwy, group = cyl)` we would get the same grouping, but it would be invisible, since all groups would be drawn in the default color.

```
ggplot(mpg, aes(hwy, fill = factor(cyl))) +
  geom_histogram(binwidth = 2)
```

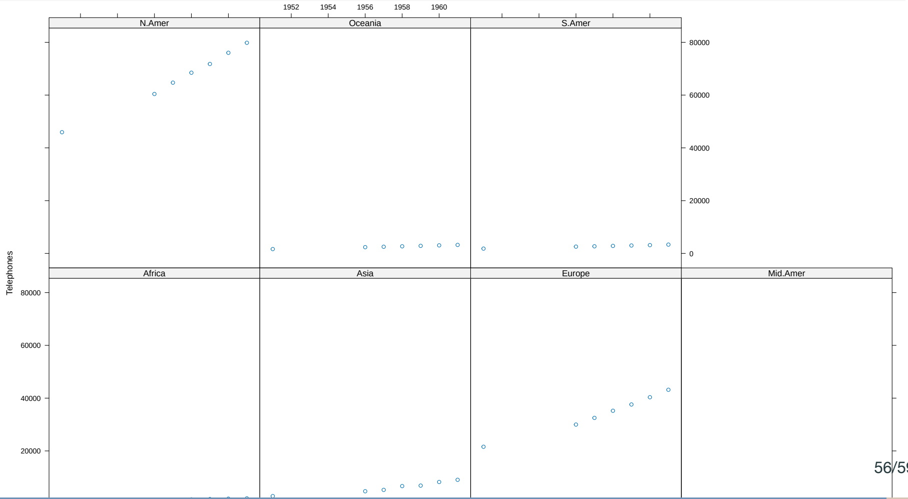


3.5 Other graphics systems

3.5.1 The lattice package

- The `lattice` package is a high level graphics system for R that predated `ggplot2`.

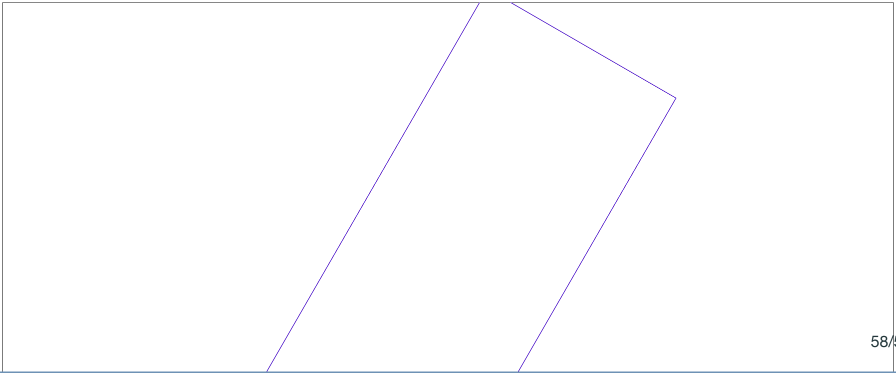
```
library(lattice)
xyplot(Telephones ~ Year | Region, data = phones)
```



3.5.2 The grid package

- Both `ggplot2` and `lattice` are implemented using the `grid` package.
- In this example, a rectangle is drawn, containing the original viewport (a 1×1 square).
- A viewport having height 0.4 and width 0.6, and rotated at an angle of 60°, is then pushed.
- Finally, the same operation is repeated, creating a viewport, rotated by a further 60° and which has a height and width which are 40% and 60% of the size of the previously drawn viewport.
- R Graphics

```
library(grid)
grid.rect() # draw black rectangle containing original viewport
vp <- viewport(h = 0.4, w = 0.6, angle = 60)
pushViewport(vp) # create viewport rotated 60 degrees (counter clockwise)
# with height 40% and width 60% of original box
grid.rect(gp = gpar(col = "red")) # draw red rectangle around new viewport pushViewpor
# create new viewport nested in previous viewport,
# rotated 60 degrees and with height 40%
# and width 60% of previous viewport.
grid.rect(gp = gpar(col = "blue")) # draw blue rectangle around viewport
```



3.5.3 Interactive graphics

- The `rgl` package is designed for three-dimensional, rotatable displays. It includes functions modeled on the base graphics functions to set up plots, and also ways to display them on screen or on a web page.
- The `plotly` package is designed for interactive graphics on web pages using the open source `plotly.js` Javascript library. Especially helpful is the function `ggplotly()` that can convert most `ggplot2` graphs into interactive form.
- The `leaflet` package provides an interface to the Leaflet library that is written in Javascript for web browsers. It is particularly well integrated with RStudio.

Example 3.16

Executing this code:

```
library(leaflet)
leaflet() %>%
  addTiles() %>%
  addMarkers(lng = 174.768, lat = -36.852, popup = "The birthplace of R")
```

in RStudio will result in the display of an interactive map with a marker in Auckland, New Zealand.

- NOTE: The `%>%` symbols will be described in Chapter 5.