ISOM 3390: Business Programming in R

Instructor: Bingjie Qian (PhD candidate)

Summer 2020, HKUST

Topic 7: Base Plotting

7.1 Base Plotting in R

Topics 7 and 8 are about **data visualization** in data science.

Data visualization plays an important role in exploring data and gaining insights from the data, which can guide the choice of statistical models in the later stages. It is also important in communicating with non-data experts.

In this course, we will learn two core plotting systems in R, i.e., **the base plotting system** (Topic 7) and **the ggplot2 plotting system** (Topic 8). These two plotting systems can pretty much satisfy all your data visualization needs.

The base plotting system comes with the `graphics` package, which is automatically loaded in a standard installation of R.

The base plotting system provides some basic but powerful plotting functions:

- `plot()`: generic plotting function
- `points()`: add points to an existing plot
- `lines()`, `abline()`: add lines to an existing plot
- `text()`, `legend()`: add text to an existing plot
- `rect()`, `polygon()`: add shapes to an existing plot
- `hist()`: create histograms
- `density()`: estimate density, which can be plotted
- `curve()`: draw a curve, or add to an existing plot
- `barplot()`, `boxplot()`: create bar plots and box plots
- ...

If we don't bother to create more aesthetic graphics, the base plotting system is very handy for us to do quick plotting for data exploration.

7.2 The Generic Plotting Function: `plot()`

The `plot()` function is *generic*, which means it can cope with data in different formats and produce different types of plots accordingly.

The type of plot depends on the class of arguments.

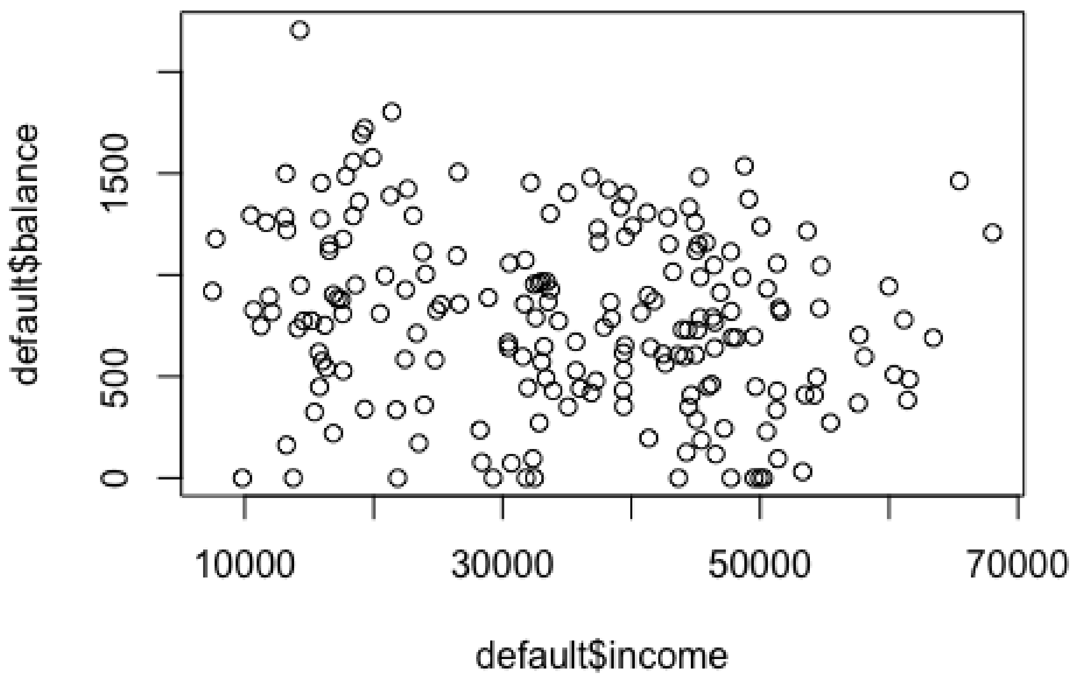
In the following examples, we will use the data set `Default.csv`. Read the first 200 rows into the R session:

```
default <- read.csv("Default.csv", nrow = 200)
default$default <- as.factor(default$default)
default$student <- as.factor(default$student)
head(default)
```

```
##   default student   balance    income
## 1      No      No  729.5265 44361.625
## 2      No     Yes  817.1804 12106.135
## 3      No      No 1073.5492 31767.139
## 4      No      No  529.2506 35704.494
## 5      No      No  785.6559 38463.496
## 6      No     Yes  919.5885  7491.559
```

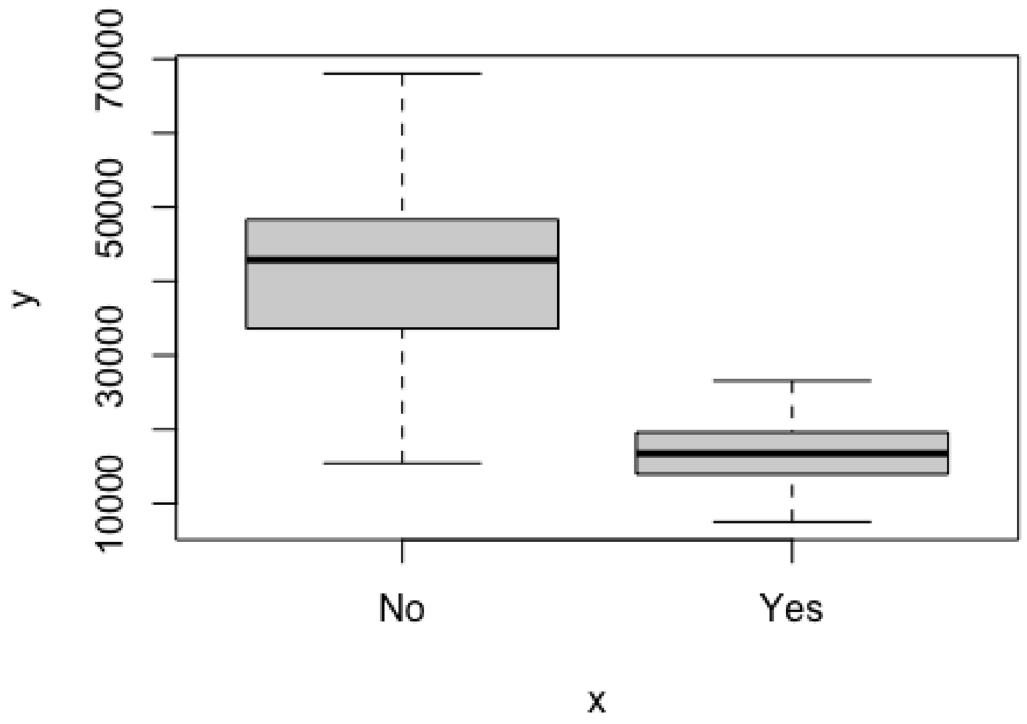
- `plot(x, y)`: a *scatter plot* if x and y are two *numeric* vectors.

```
plot(default$income, default$balance)
```



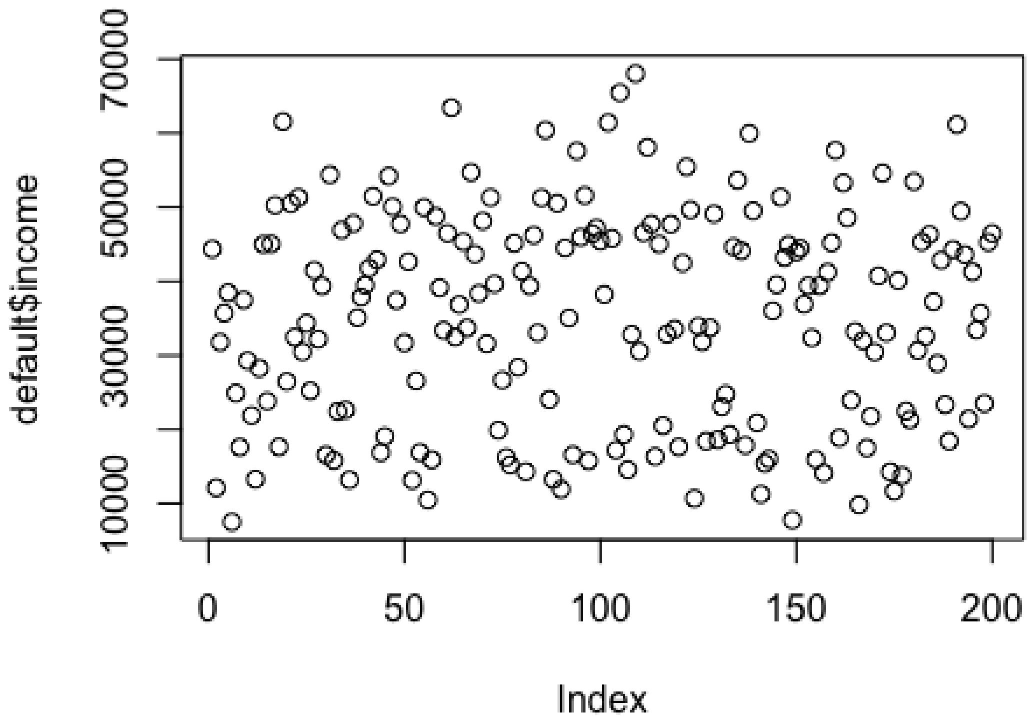
- `plot(f, y)`: a *boxplot* if `f` is a *factor* and `y` is a *numeric* vector.

```r
plot(default$student, default$income)
```

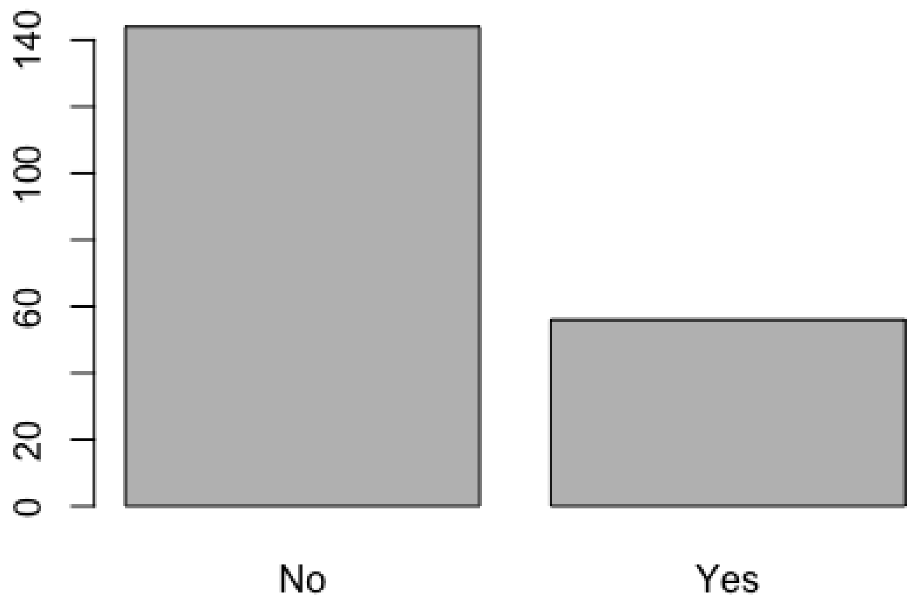


- `plot(x)` (single argument):
  - A plot showing the value of x at every index, if x is *numeric*.
  - A bar plot of counts for every level of x, if x is a *factor*.

```
plot(default$income)
```

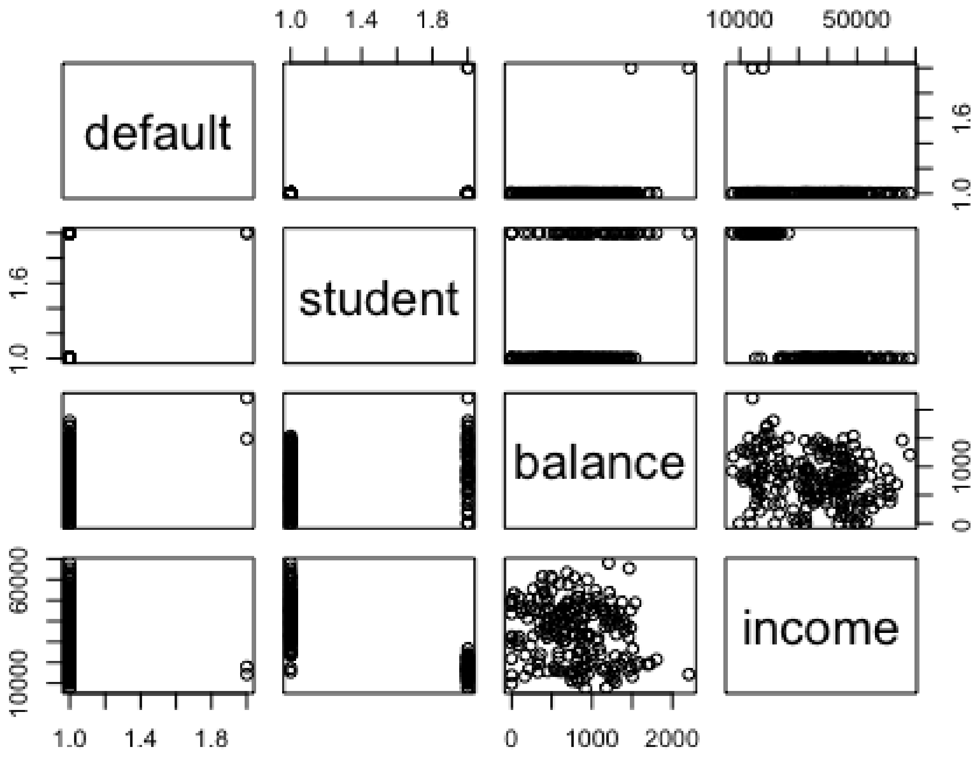


```r
plot(default$student)
```



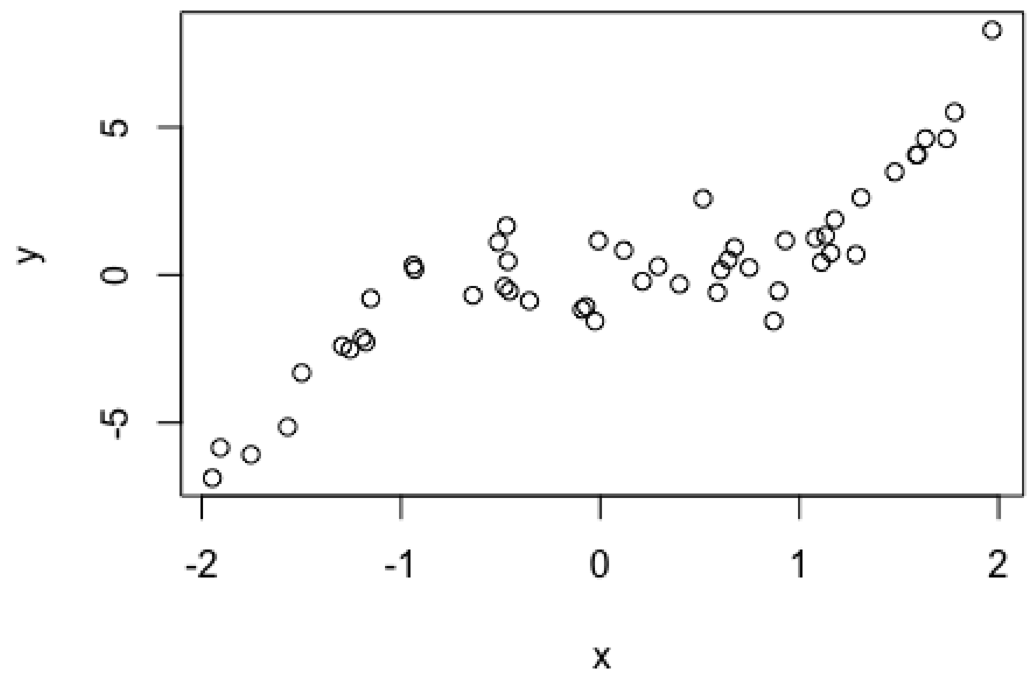
- `plot(data.frame)`: all variables plotted against each other.

```
plot(default)
```



Customize the Plot

```r
set.seed(0)  # set seed to ensure reproducibility
x <- sort(runif(50, min = -2, max = 2))
y <- x^3 + rnorm(50)
plot(x, y)  # a scatter plot
```

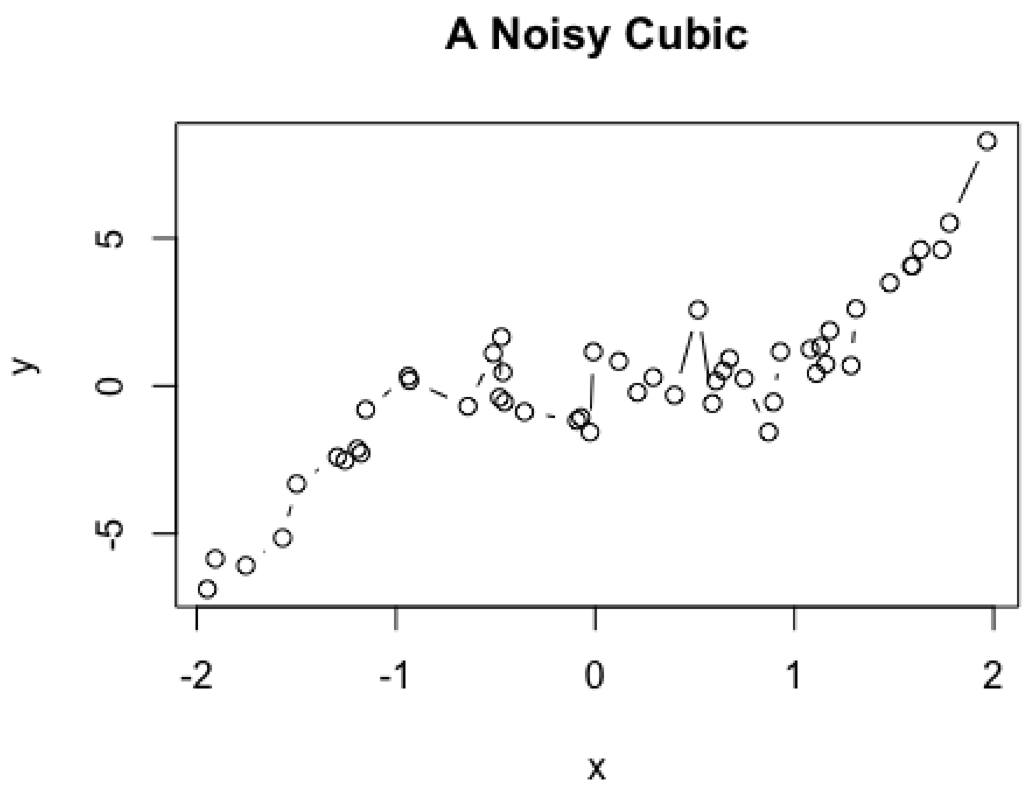


We can customize the plot by setting the arguments of `plot()`.

Customize **the style of the plot**:

- `type`: `"p"` for points, `"l"` for lines, `"b"` for both points and lines, `"c"` for empty points joined by lines, `"o"` for overplotted points and lines, `"s"` and `"S"` for stair steps and `"h"` for histogram-like vertical lines. Finally, `"n"` does not produce any points or lines.
- `pch`: plotting "character", i.e., symbol to use. This can either be a single character or an integer code for one of a set of graphics symbols. E.g., `pch = 1`, `pch = "*"`.
- `lty`: line type. Line types can either be specified as an integer (0=blank, 1=solid (default), 2=dashed, 3=dotted, 4=dotdash, 5=longdash, 6=twodash) or as one of the character strings "blank", "solid", "dashed", "dotted", "dotdash", "longdash", or "twodash", where "blank" uses "invisible lines" (i.e., does not draw them).
- `lwd`: line width, a positive number, defaulting to 1.
- `col`: a specification for the default plotting color.

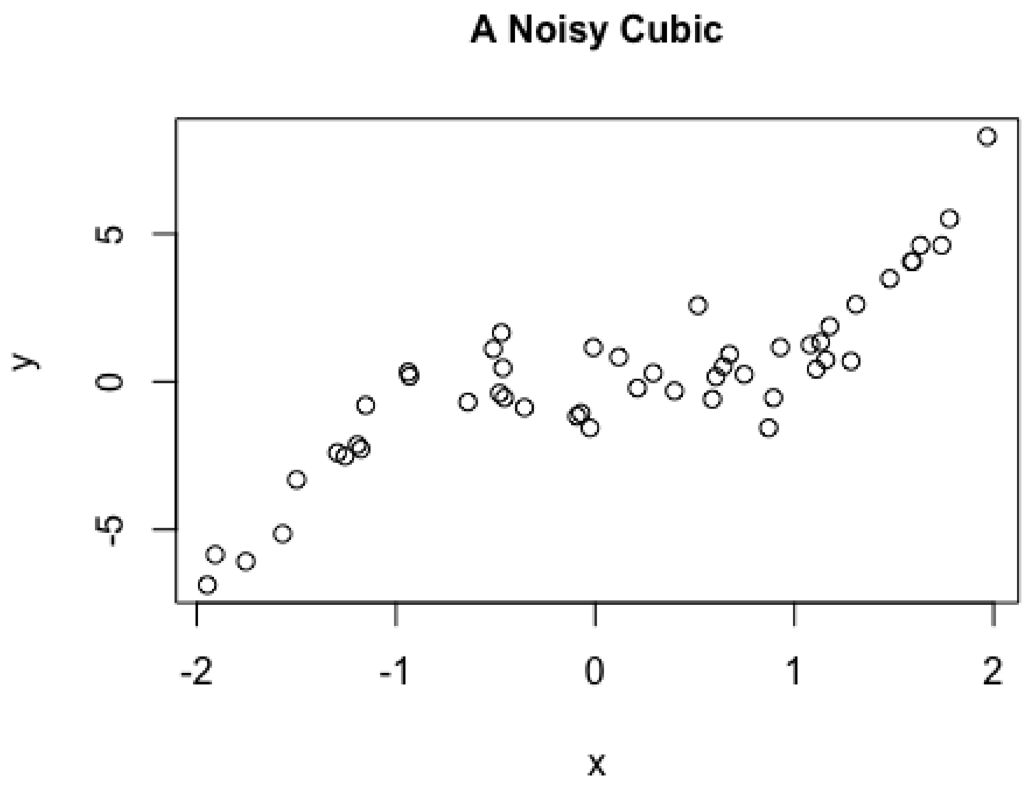
```r
# plot(x, y)
plot(x, y, type = "b", pch = 1, lty = "solid", lwd = 1, col = "black", main =
"A Noisy Cubic")
```



Customize **the size of the text**:

- `cex.main` for the size of the main title
- `cex.lab` for the x and y labels
- `cex.axis` for the axis annotation

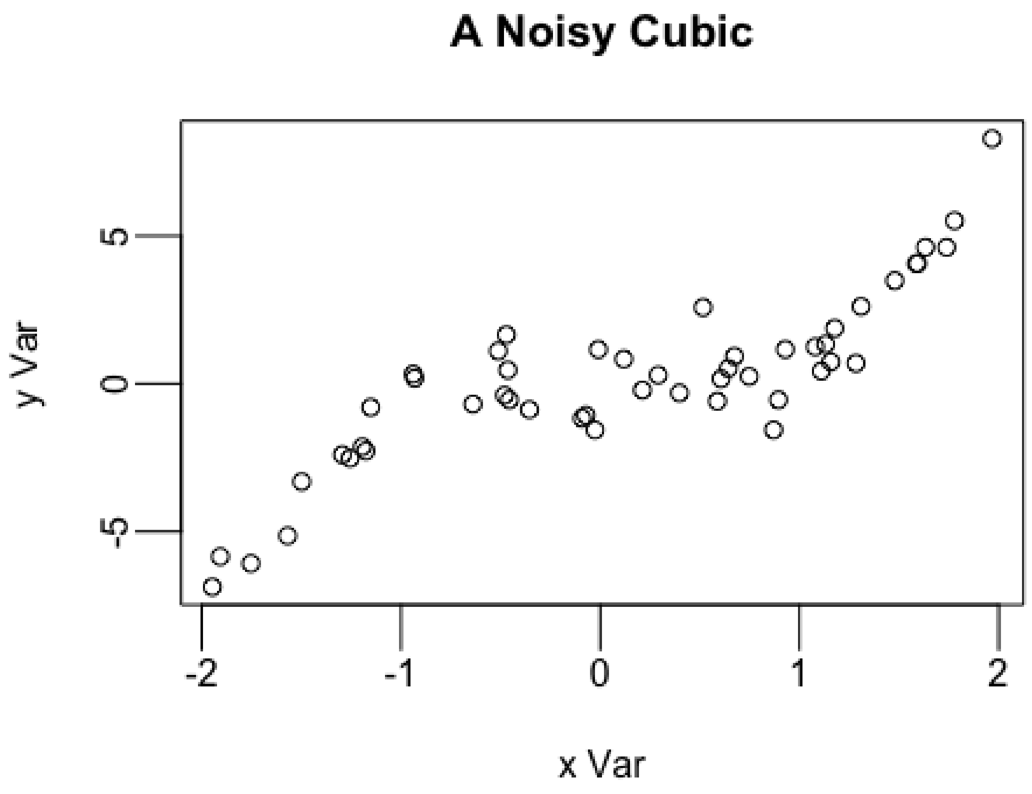
```
plot(x, y, cex.main = 1, cex.lab = 1, cex.axis = 1, main = "A Noisy Cubic")
```



Customize **the appearance of the axes**:

- `xlab` for x axis label
- `ylab` for y axis label
- `mgp` for the margin line for the axis title, axis labels and axis line; defaults to `c(3, 1, 0)`
- `tcl` for the length of tick marks as a fraction of the height of a line of text; defaults to `-0.5`

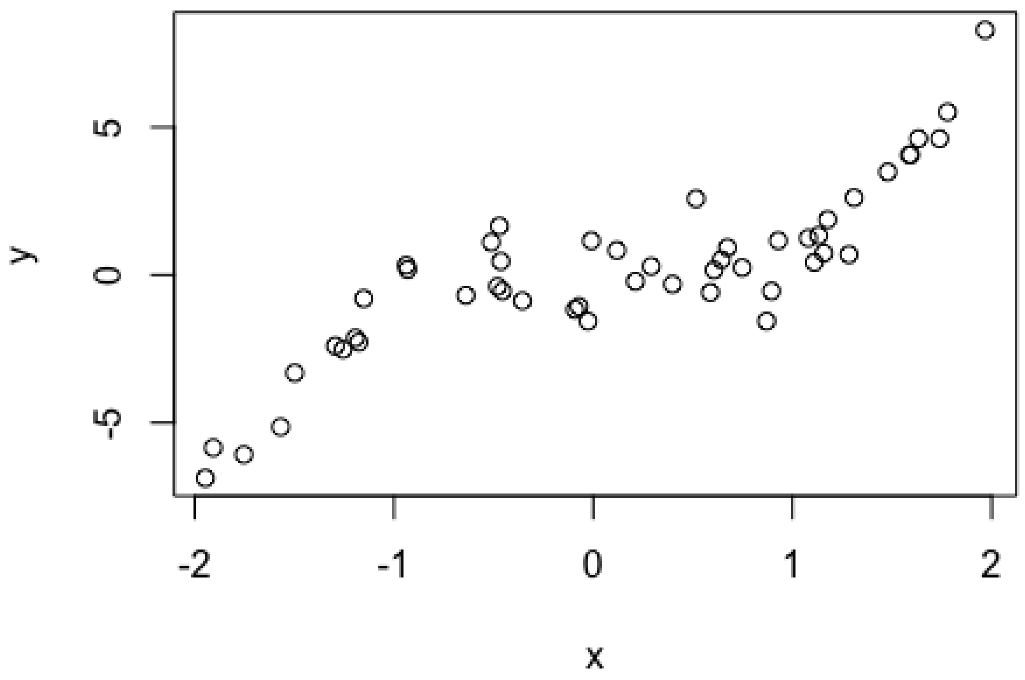
```
plot(x, y, main = "A Noisy Cubic", xlab = "x Var", ylab = "y Var", mgp = c(3,
1,
    0), tcl = -1)
```



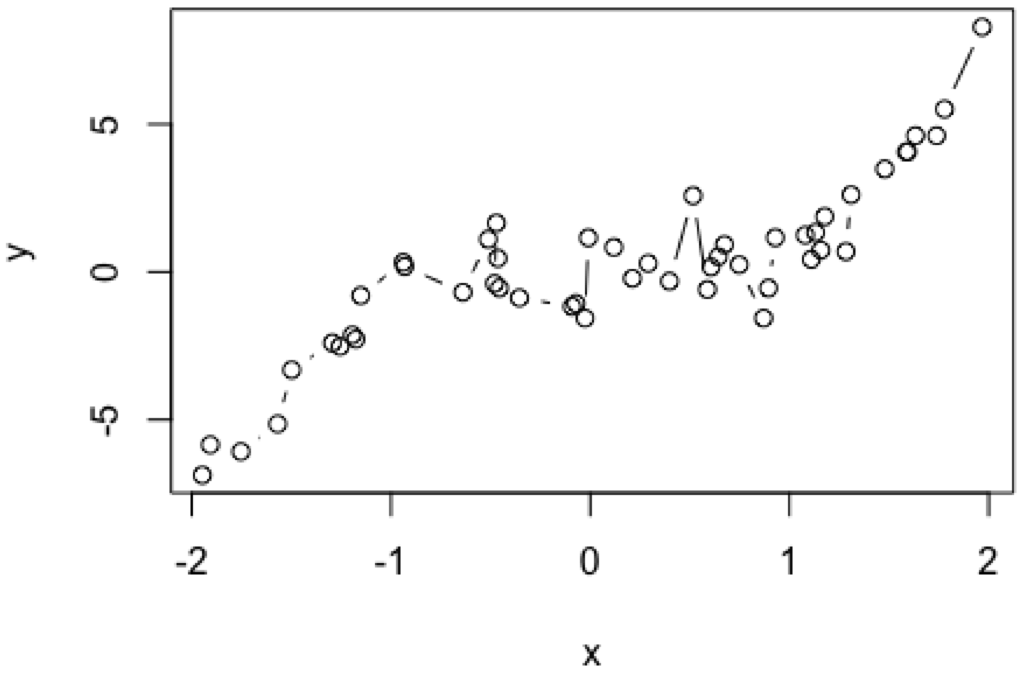
[Task 1: Plotting Basics]

(a) Read the following code:

```
set.seed(0)
# x <- runif(50, min = -2, max = 2) # a key difference y <- x^3 + rnorm(50)
plot(x, y, type = "p")
```



```r
plot(x, y, type = "b")
```



Explain why does the `plot()` with `type = "p"` looks normal, but the `plot()` with `type = "b"` looks abnormal, having crossing lines?

(b) Modify the code above so that the lines on the second plot do not cross.

(c) The `cex` argument can be used to shrink or expand *the size of the points* that are drawn. Its default value is 1 (no shrinking or expansion). Values between 0 and 1 will shrink points, and values larger than 1 will expand points. Plot `y` versus `x`, first with `cex` equal to 0.5 and then 2 (so, two separate plots). Give titles "Shrunken points", and "Expanded points", to the plots, respectively.

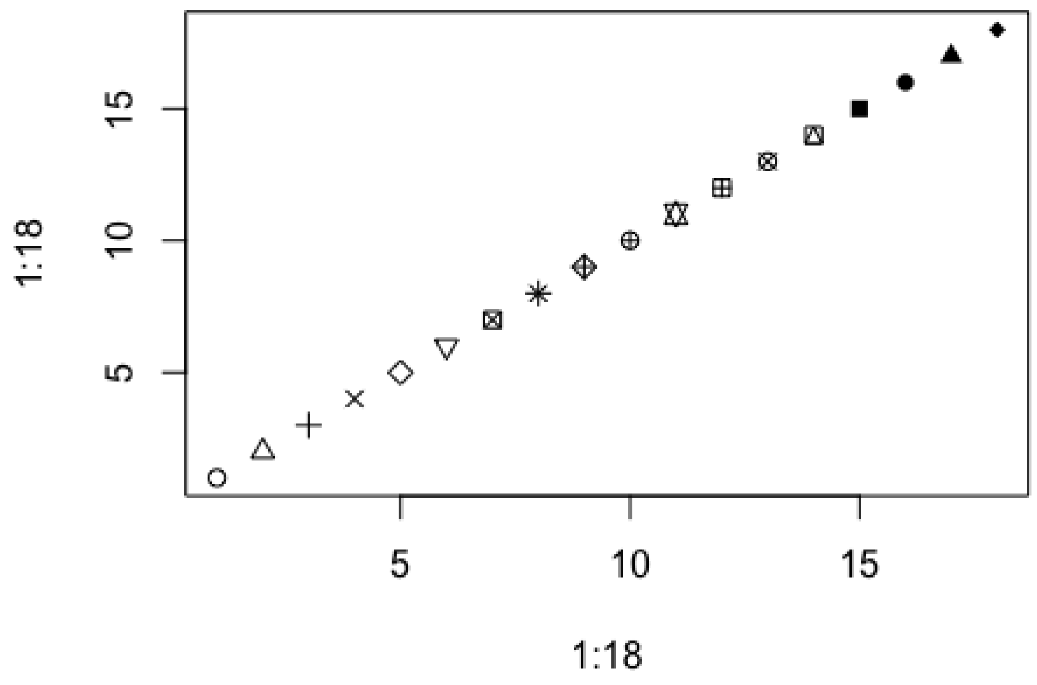
(d) The `xlim` and `ylim` arugments can be used to change *the limits on the x-axis and y-axis*, repsectively. Each argument takes a vector of length 2, as in `xlim = c(-1, 0)`, to set the x limit to be from -1 to 0. Plot `y` versus `x`, with the x limit set to be from -1 to 1, and the y limit set to be from -5 to 5. Assign x and y labels "Trimmed x" and "Trimmed y", respectively.

(e) Again plot `y` versus `x`, *only showing points whose x values are between -1 and 1*. But this time, define `x.trimmed` to be the subset of `x` between -1 and 1, and define `y.trimmed` to be the corresponding subset of `y`. Then plot `y.trimmed` versus `x.trimmed` without setting `xlim` and `ylim`: now you should see that the y limit is (automatically) set as "tight" as possible.

Tips: use logical indexing to define `x.trimmed`, `y.trimmed`.

(f) The `pch` argument controls *the point type* in the display. In the lecture examples, we set it to a single number. But it can also be a vector of numbers (similar to `col`), with one entry per point in the plot.

```
plot(1:18, 1:18, pch = 1:18)
```



The above plot displays the first 18 marker types. If `pch` is a vector whose length is shorter than the total number of points to be plotted, then its entries are recycled, as appropriate. Plot `y` versus `x`, and repeat the following pattern for the displayed points: a black empty circle, a blue filled circle, a black empty circle, a red filled circle.

Tips: use strings "blue", "red", and "black" to specify colors.

[End of Task 1]

7.3 Setting Graphical Parameters: `par()`

The function `par()` can be used to set or query **graphical parameters**.

Parameters can be set by specifying them as arguments to `par` in `tag = value` form, or by passing them as a list of tagged values.

Plotting Regions

In R's base graphics, the graph area is split into three parts:

- **Plot region**: area within the axes
- **Figure margin**: including axes, axes labels, tick mark labels etc.
- **Outer margin**

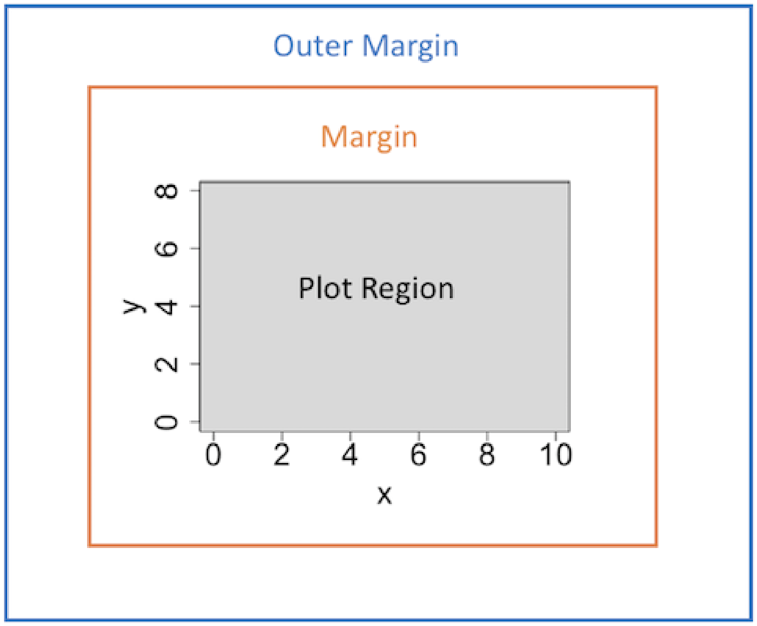
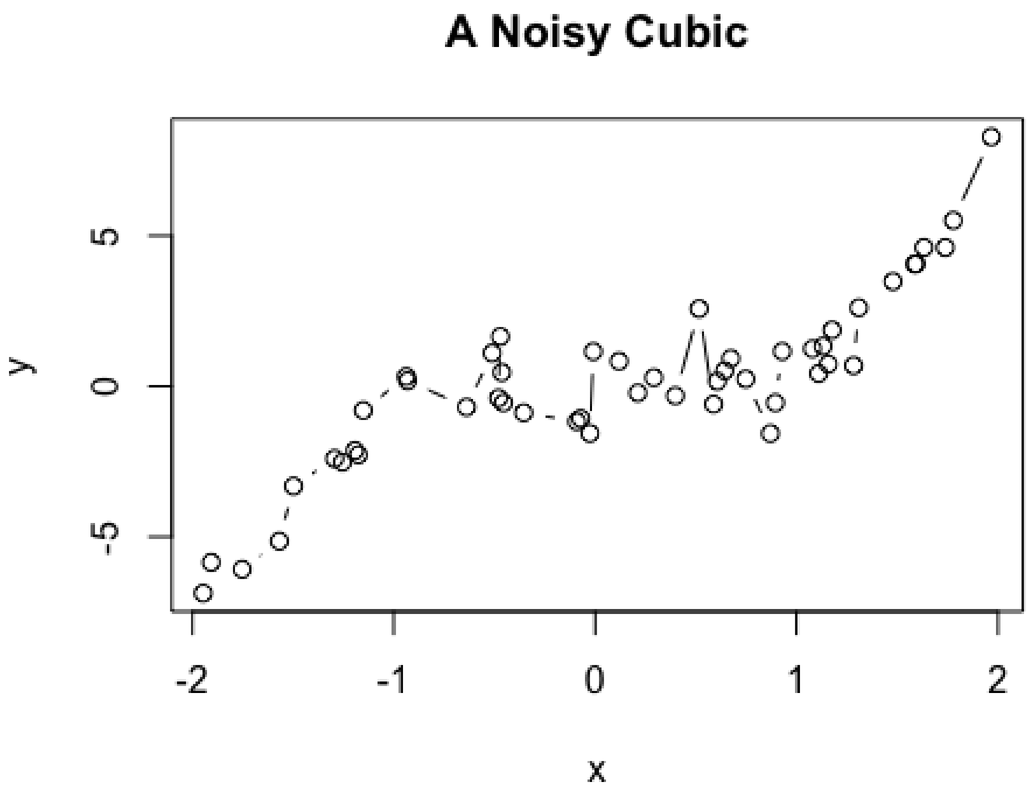


Figure 1. Plotting Regions

Figure Margins: `mar` in `par()`

The parameter `mar` is a numerical vector of the form `c(bottom, left, top, right)` which gives the number of lines of margin to be specified on the four sides of the plot. The default is `c(5, 4, 4, 2) + 0.1`.

```
par(mar = c(5, 4, 4, 2))
plot(x, y, type = "b", pch = 1, lty = 1, lwd = 1, col = 1, main = "A Noisy
Cubic")
```



Multiple Plots: `mfrow` and `mfcol` in `par()`

The parameter `mfrow` or `mfcol` is a vector of the form `c(nr, nc)`, which sets up a **plotting grid** of arbitrary dimensions. Subsequent figures will be drawn in an `nr-by-nc` array on the device by columns (`mfcol`), or rows (`mfrow`), respectively.

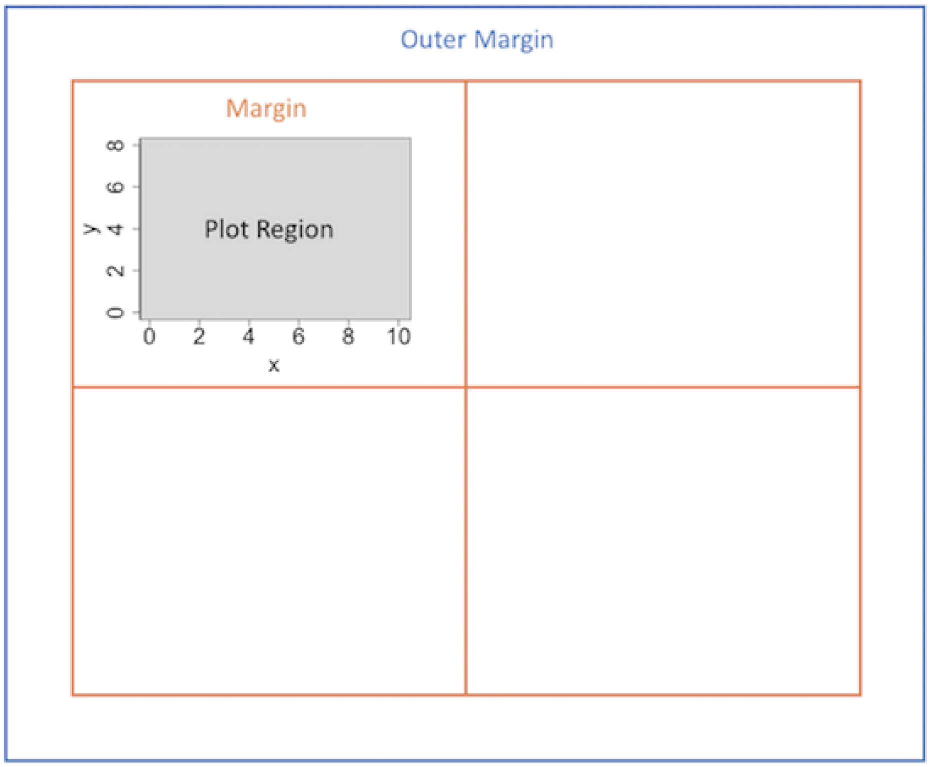
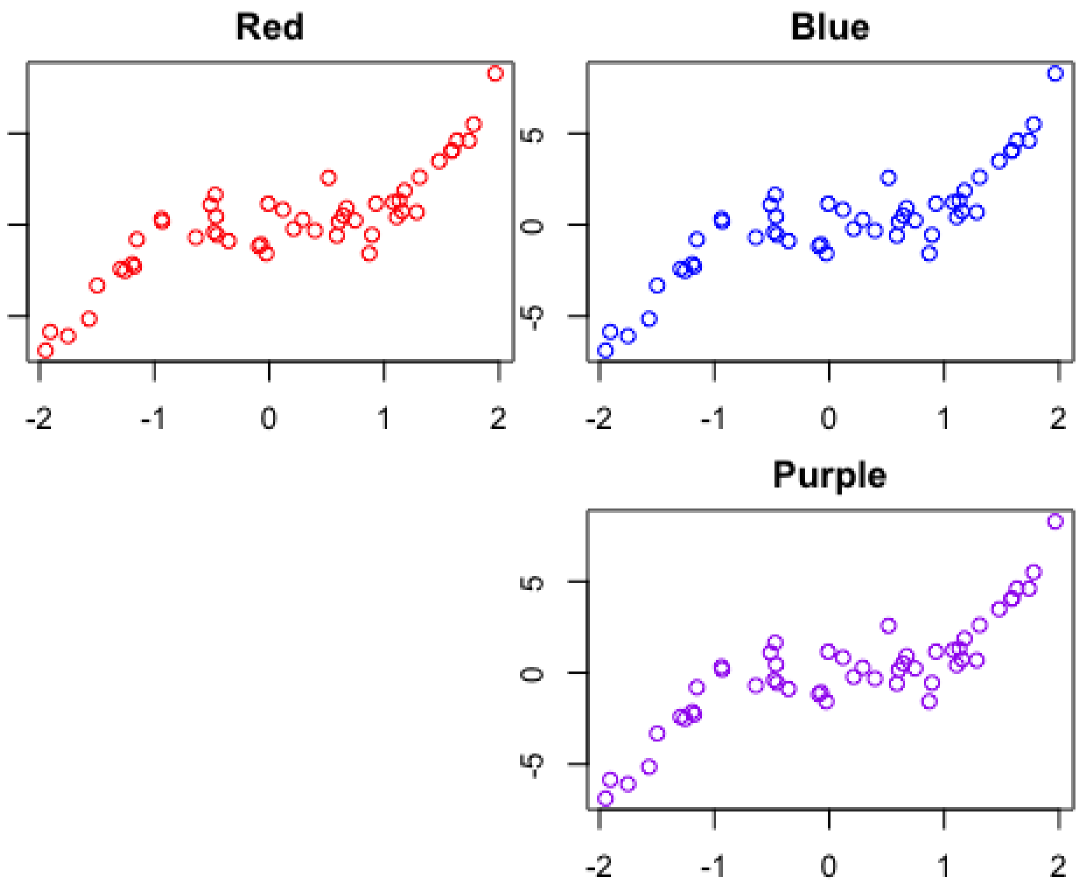


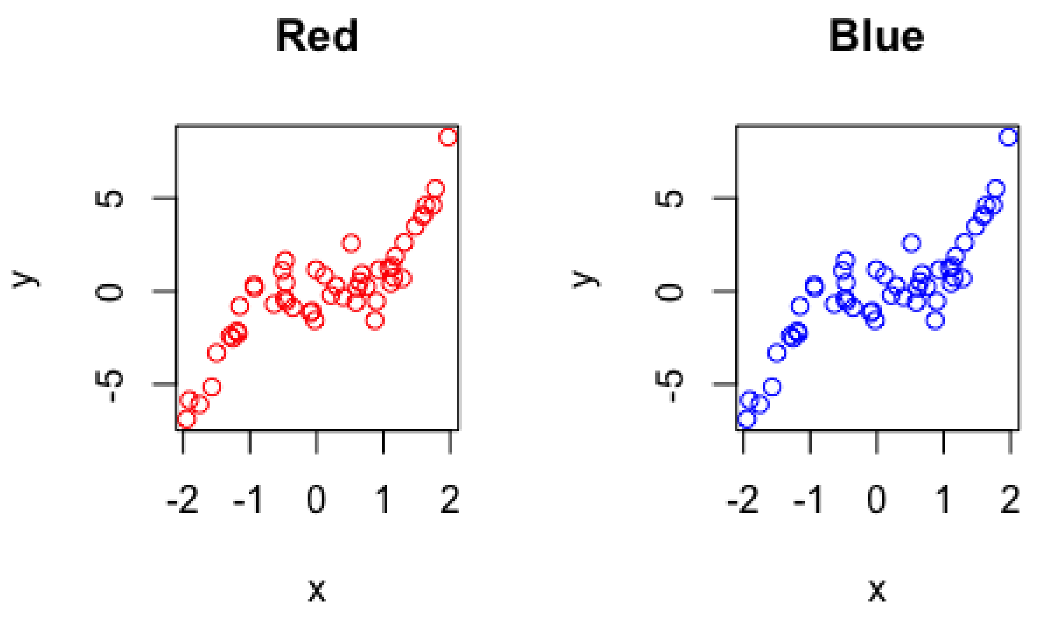
Figure 2. Multi-Panel Grid

```
par(mfrow = c(2, 2), mar = c(2, 1, 2, 1))  # Grid cells are filled by rows
plot(x, y, main = "Red", col = "red")
plot(x, y, main = "Blue", col = "blue")
plot(1, type = "n", axes = F, xlab = "", ylab = "")  # this plot does not
produce anything
plot(x, y, main = "Purple", col = "purple")
```



Outer Margins: `oma` in `par()`

```
par(mfrow = c(1, 2), oma = c(2, 0, 2, 0))
plot(x, y, main = "Red", col = "red")
plot(x, y, main = "Blue", col = "blue")
```



Precise Appearance Control

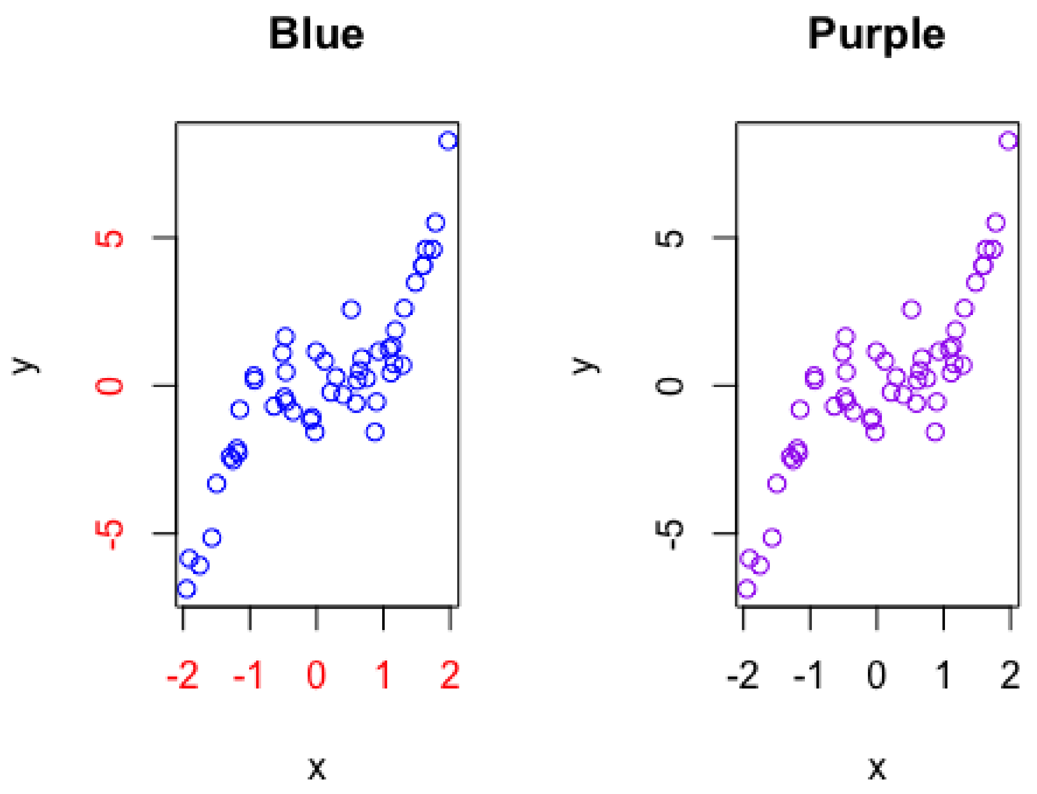
The base graphics system has around 70 graphics parameters, controlling things, such as line style, colors, figure arrangement and text justification among many others, in all 3 regions.

```
names(par())

##  [1] "xlog"      "ylog"      "adj"       "ann"       "ask"       "bg"       
##  [7] "bty"       "cex"       "cex.axis"  "cex.lab"   "cex.main"  "cex.sub"  
## [13] "cin"       "col"       "col.axis"  "col.lab"   "col.main"  "col.sub"  
## [19] "cra"       "crt"       "csi"       "cxy"       "din"       "err"      
## [25] "family"    "fg"        "fig"       "fin"       "font"      
"font.axis"
## [31] "font.lab"  "font.main" "font.sub"  "lab"       "las"       "lend"     
## [37] "lheight"   "ljoin"     "lmitre"    "lty"       "lwd"       "mai"      
## [43] "mar"       "mex"       "mfcol"     "mfg"       "mfrow"     "mgp"      
## [49] "mkh"       "new"       "oma"       "omd"       "omi"       "page"     
## [55] "pch"       "pin"       "plt"       "ps"        "pty"       "smo"      
## [61] "srt"       "tck"       "tcl"       "usr"       "xaxp"      "xaxs"     
## [67] "xaxt"      "xpd"       "yaxp"      "yaxs"      "yaxt"      "ylbias"
```

- Most of these parameters, except a few read-only ones, can be set *globally* by the `par()` function to affect all plots in an R session.

- A majority of them can be *overridden* by arguments to specific plotting functions (e.g., `plot()`, `lines()`, `abline()`, `axis()`, `title()`, `text()`, etc.).

```
par(mfrow = c(1, 2), col.axis = "red")
plot(x, y, main = "Blue", col = "blue")
plot(x, y, main = "Purple", col = "purple", col.axis = "black")
```



7.4 A Painters Model

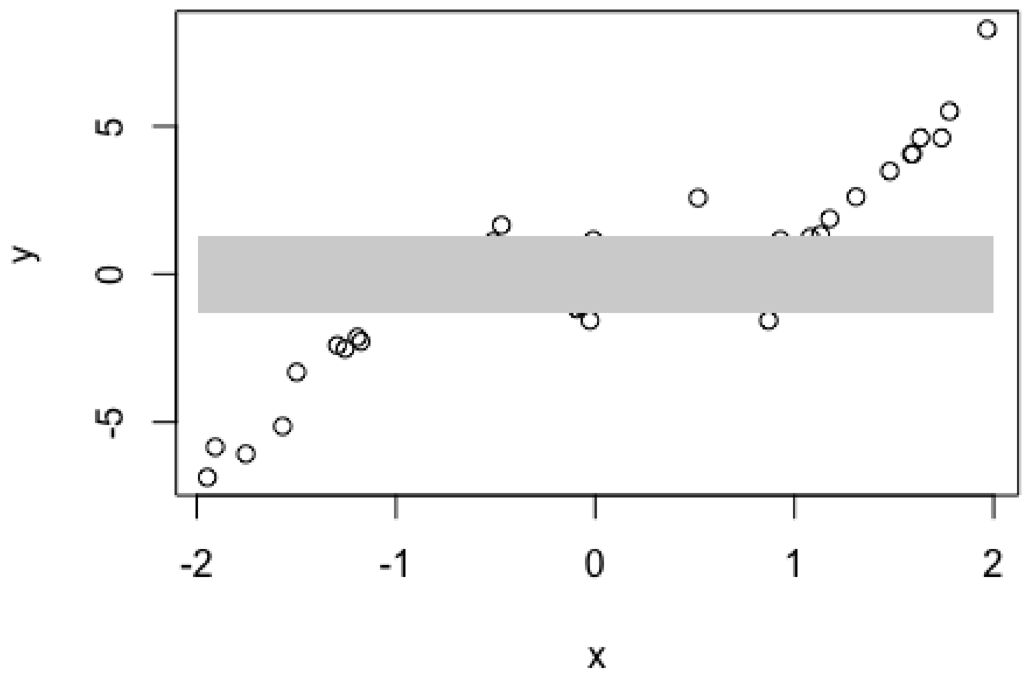
The base graphics system follows a **painters model**, starting with a "blank canvas" and adding items like we would add ink/paint to the canvas, with later items drawn on top of any previous items.

- **High-level functions**, such as `plot()`, `barplot()` and `hist()`, create complete plots, charts, etc. If there is an existing plot, it will wipe the "canvas" clean and then paint a new plot.

- **Low-level functions**, such as `points()`, `lines()`, and `rect()`, add graphics elements (including points, lines, arrows, rectangles, etc.) to existing plots and obscure what are below them.

Adding Items

```
plot(x, y)  # create a new plot
# points(x, x^3, pch = 20, col = 'red') # add points lines(x, x^3, lwd = 2) #
add
# a line abline(a = 0, b = 3) # add a straight line abline(h = 0, lty = 2,
col =
# 'red') # add a straight (horizontal) line text(0, -2.5, labels = 'This is a
# plot') # add text
rect(-2, qnorm(0.1), 2, qnorm(0.9), col = "lightgrey", border = NA)  # add a
rectangle
```



```
# polygon(c(-2, 0, 2, 2, 0, -2), c(-3, -1, -3, 3, 1, 3), col = 'pink',
density =
# 30, border = NA) # add a polygon
```

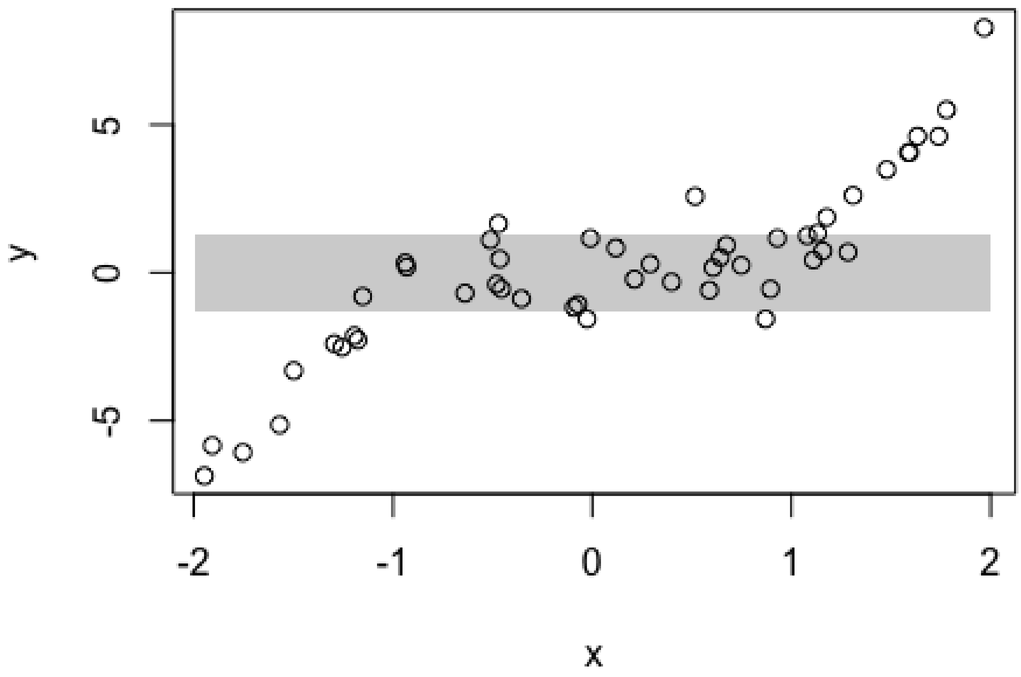
Later items will be drawn on top of previous items and obscure them. For example, the added rectangle covers the data points.

We can change the order of items to solve the problem:

- First, draw a plot that contains only the axes and labels using `plot()` by setting `type = "n"`.
- Next, add a rectangle with `rect()`.

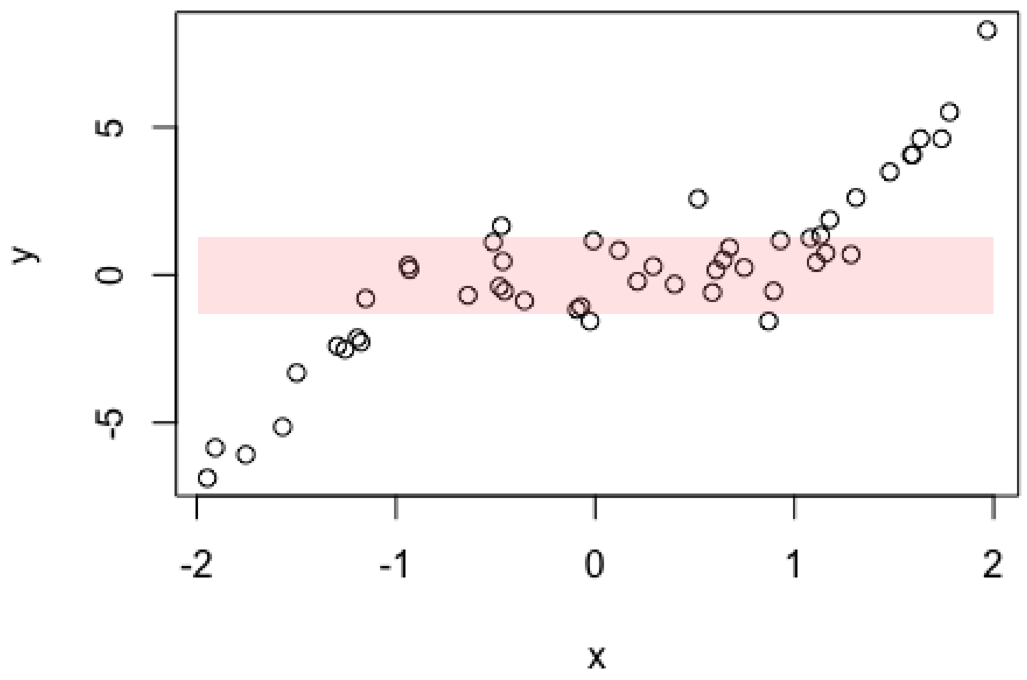
- Then, add data points with `points()`.

```
plot(x, y, type = "n")
rect(-2, qnorm(0.1), 2, qnorm(0.9), col = "lightgrey", border = NA)
points(x, y)
```



Adjust color and transparency of the rectangle with `rgb()`:

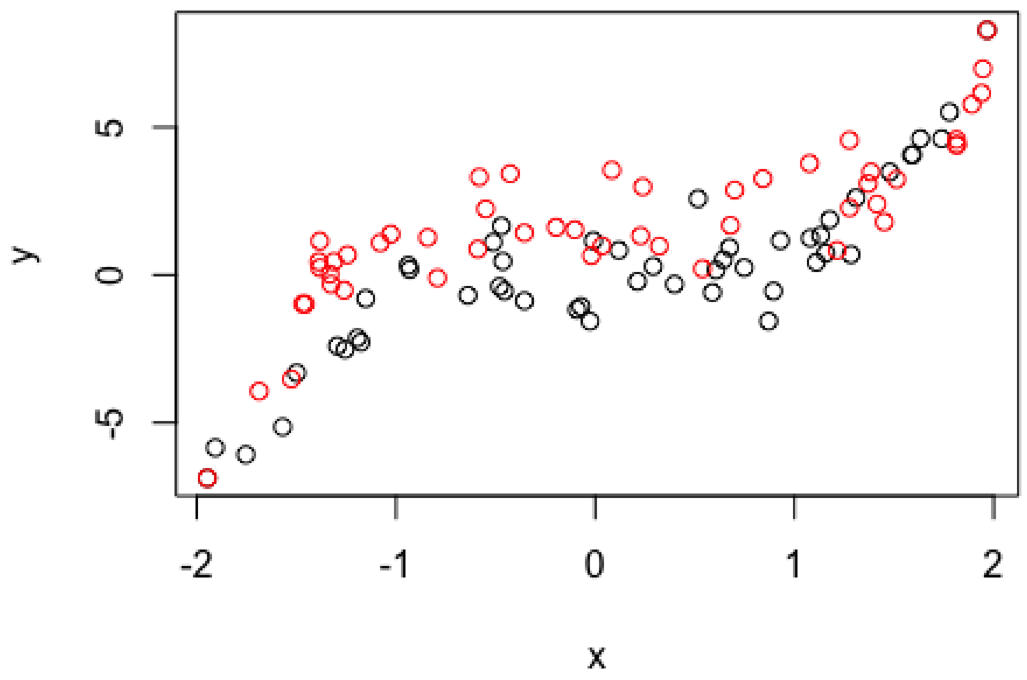
```
plot(x, y, type = "n")
points(x, y)
rect(-2, qnorm(0.1), 2, qnorm(0.9), col = rgb(1, 0.5, 0.5, 0.2), border = NA)
# red, green, blue, transparency
```



Plot Multiple Data Series

We can set `new = TRUE` with `par()` to *make the second plot without cleaning the first*.

```
plot(x, y)  # the first data series
set.seed(2)
x1 <- sort(runif(50, min = -2, max = 2))
y1 <- x^3 + rnorm(50)
par(new = TRUE)
plot(x1, y1, axes = FALSE, xlab = "", ylab = "", col = "red")  # the second
data series on the same plot
```



Adding Axes

In `plot()`, we can set `xaxt = "n"`, `yaxt = "n"`, or `axes = FALSE` to suppress the plotting of one or both axes.

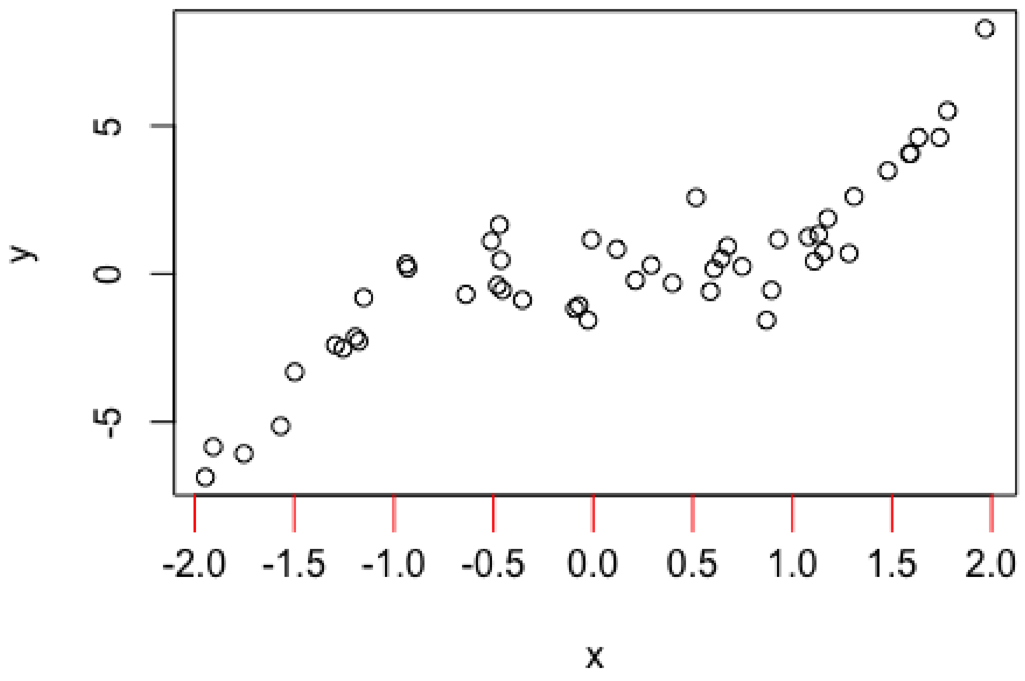
We can use the function `axis()` to add an axis to the existing plot with fine-tuning parameters.

Usage:

```
axis(side, at = NULL, labels = TRUE, tick = TRUE, line = NA,
     pos = NA, outer = FALSE, font = NA, lty = "solid",
     lwd = 1, lwd.ticks = lwd, col = NULL, col.ticks = NULL,
     hadj = NA, padj = NA, gap.axis = NA, ...)
```

- `side`: an integer specifying which side of the plot the axis is to be drawn on. 1=below, 2=left, 3=above and 4=right.
- `at`: the points at which tick-marks are to be drawn. Non-finite (infinite, `NaN` or `NA`) values are omitted. By default (when NULL) tickmark locations are computed.
- `col.ticks`: tick mark color
- `tcl`: tick mark length

```
plot(x, y, xaxt = "n")
axis(side = 1, at = seq(-2, 2, by = 0.5), col.ticks = "red", tcl = -0.8)
```

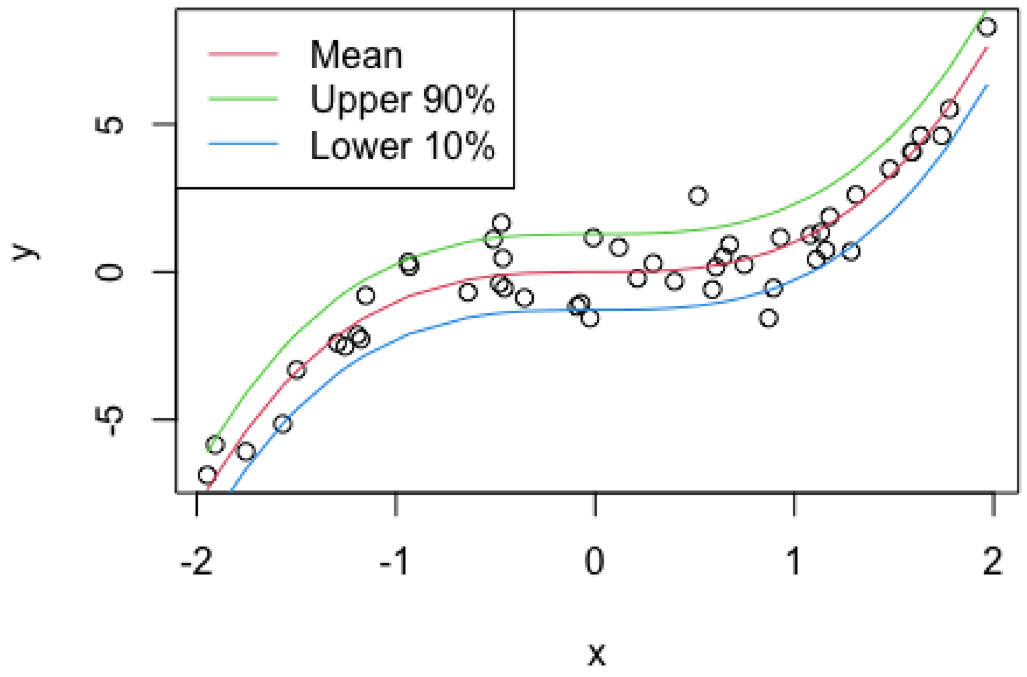


Adding a Legend

Use the function `legend()` to add a legend to an existing plot.

The legend position can be represented by x and y coordinates or one of "topleft", "topright", "bottomleft", "bottomright":

```
plot(x, y)
lines(x, x^3, col = 2)
lines(x, x^3 + qnorm(0.9), col = 3)
lines(x, x^3 + qnorm(0.1), col = 4)
legend("topleft", legend = c("Mean", "Upper 90%", "Lower 10%"), col = 2:4,
lty = 1)
```



[Task 2: Adding Items]

(a) Produce a *scatter plot* of `y` versus `x` (which contains empty black circles of `y` versus `x`), and set the title and axes labels as you see fit. Then overlay a *scatter plot* of `y2` versus `x2` on top of the plot, using the `points()` function, where `x2` and `y2` are as defined below. In the call to `points()`, set the `pch` and `col` arguments appropriately so that the overlaid points are drawn as filled blue circles.

```
x <- sort(runif(50, min = -2, max = 2))
y <- x^3 + rnorm(50)
x2 <- sort(runif(50, min = -2, max = 2))
y2 <- x2^2 + rnorm(50)
```

(b) Starting with your solution code from (a), overlay a *line plot* of `y2` versus `x2` on top of the plot (which contains empty black circles of `y` versus `x`, and filled blue circles of `y2` versus `x2`), using the `lines()` function. In the call to `lines()`, set the `col` and `lwd` arguments so that the line is drawn in red, with twice the normal thickness.

(c) Starting with your solution code from (b), add a *legend* to the bottom right corner of the the plot using `legend()`. The legend should display the text: "Cubic" and

"Quadratic", with corresponding symbols: an empty black circle and a filled blue circle, respectively.

Tips: pch and col.

(d) Produce a plot of y versus x, but with a gray *rectangle* (`rect()`) displayed underneath the points. This rectangle has a lower left corner at `(-2, qnorm(0.1))`, and an upper right corner at `(2, qnorm(0.9))`.

(e) Produce a plot of y versus x, but with a gray *tube* displayed underneath the points. Specifically, this tube should fill in the space between the two curves defined by y = x^3 - q and y = x^3 + q, where q is the 90th percentile of the standard normal distribution (i.e., equal to `qnorm(0.90)`).

Add a legend to the bottom right corner of the plot, with the text: "Data" and "Confidence band", and corresponding symbols: an empty circle and a very thick gray line, respectively.

Try to reproduce the following plot:

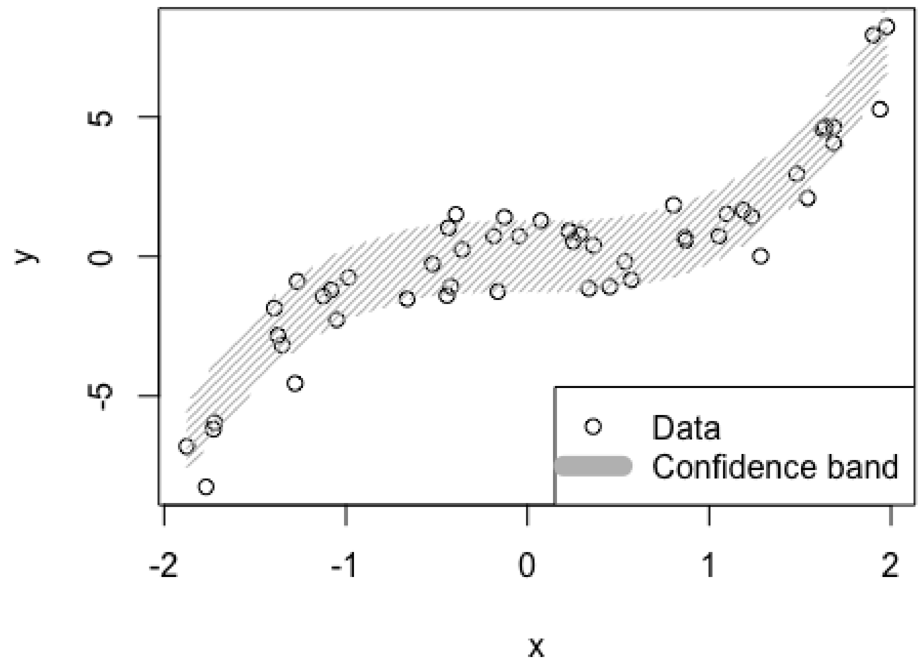


Fig 3. Task 2

Tips:

- `polygon()`: this function requires that the x coordinates of the polygon be passed in an appropriate order; you might find it useful to use `c(x, rev(x))` for the x coordinates.

- Try `pch = c(1, NA), col = c("black", "gray"), lwd = c(NA, 10)` for the legend.

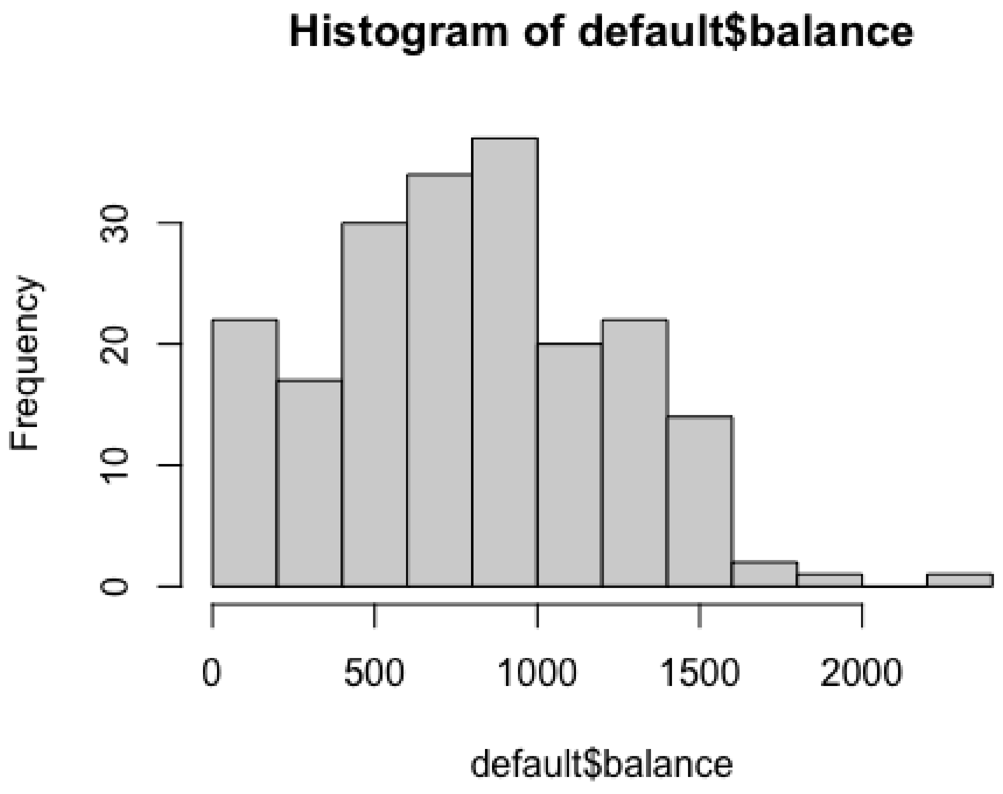
[End of Task 2]

7.5 Other High-Level Plotting Functions

7.5.1 Histogram: `hist()`

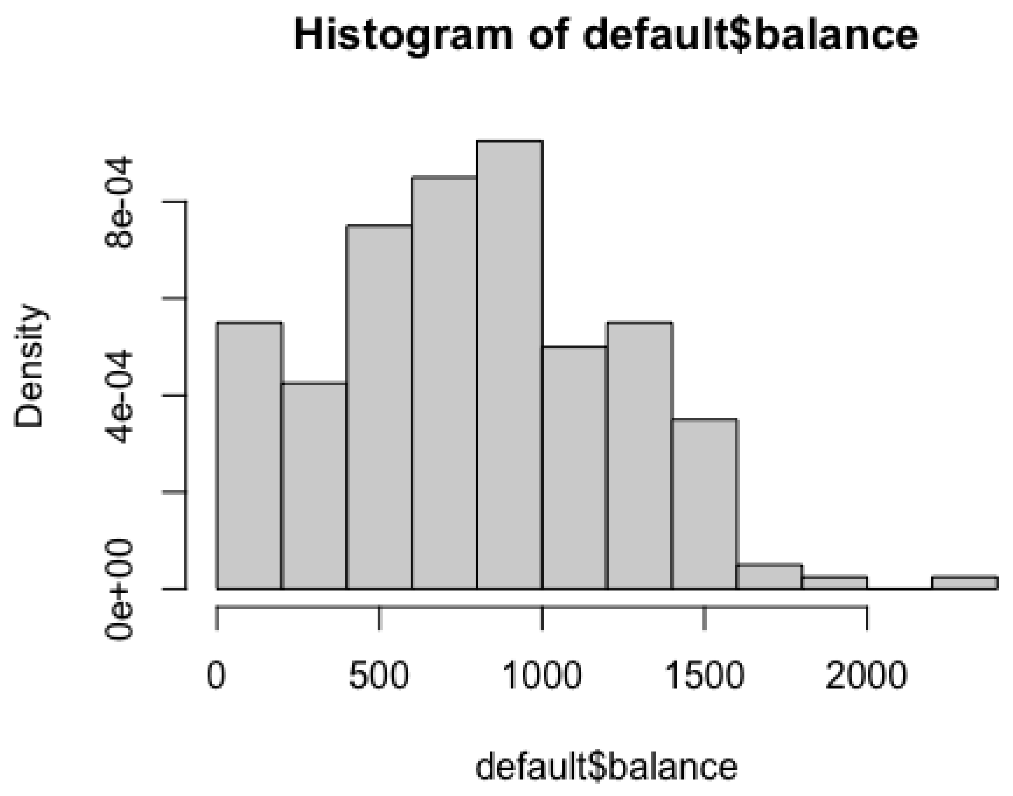
A histogram investigates the distribution of a numeric variable.

```r
hist(default$balance)
```



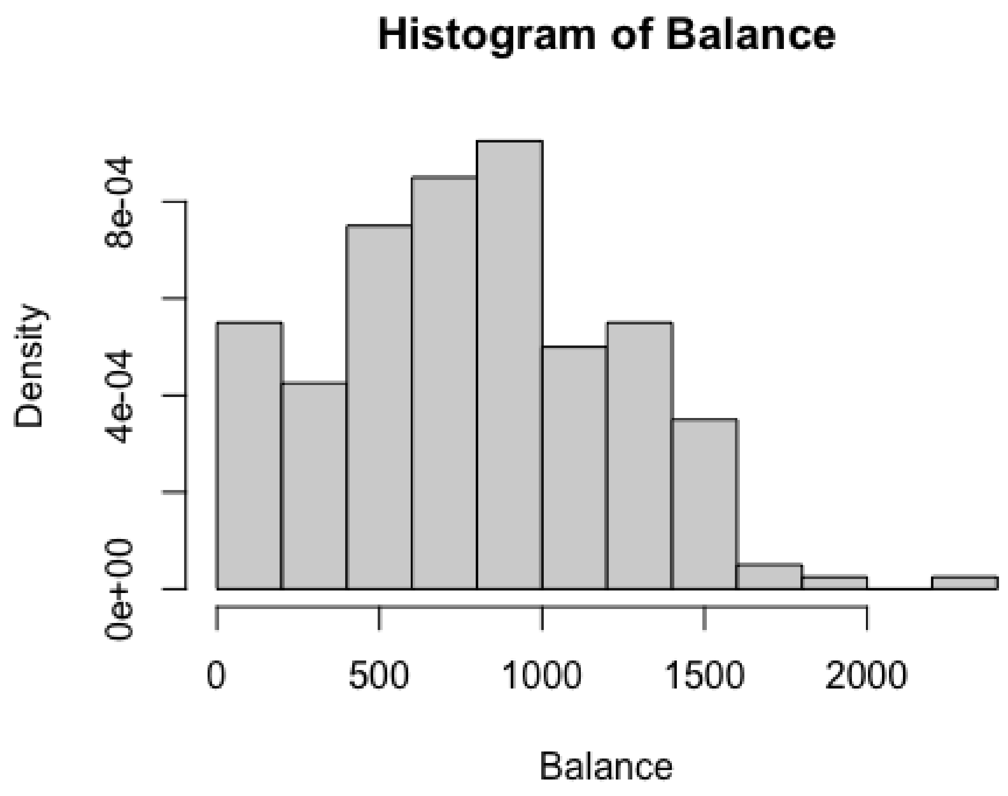
- The default scale of y axis is *frequency*. Change it to *probability density* by setting `freq = FALSE`:

```r
hist(default$balance, freq = FALSE)
```



- Change the title and the x axis label:

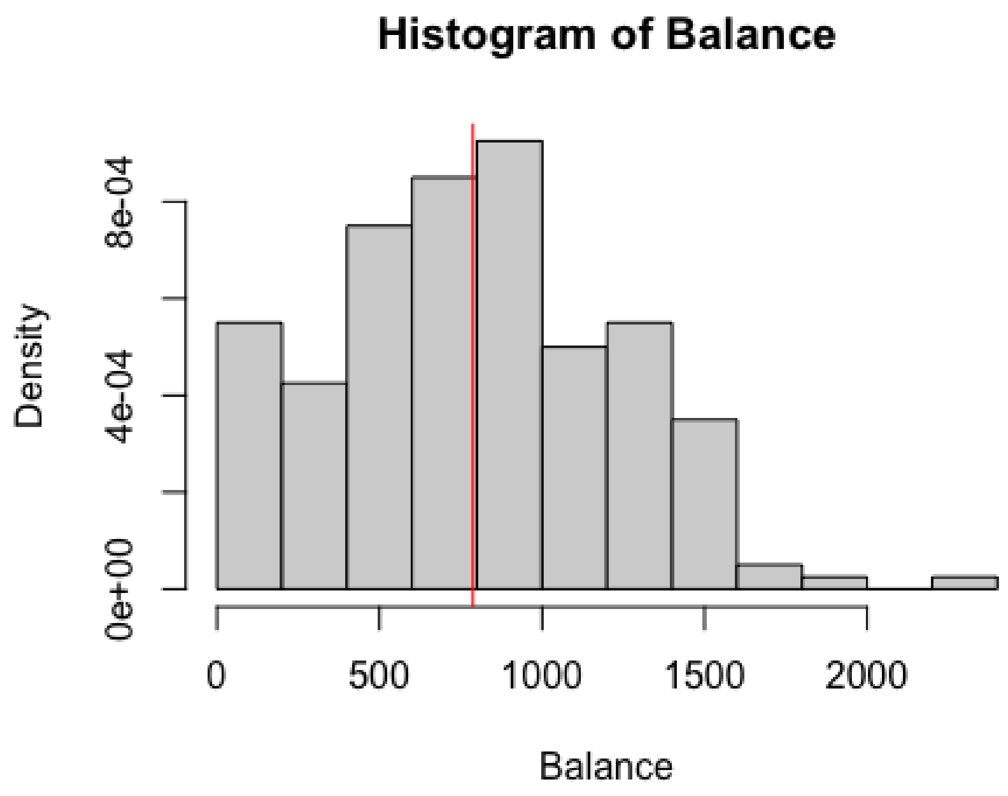
```
hist(default$balance, freq = FALSE, main = "Histogram of Balance", xlab =
"Balance")
```



Since `hist()` is a high-level plotting function, we can add items to it using low-level functions.

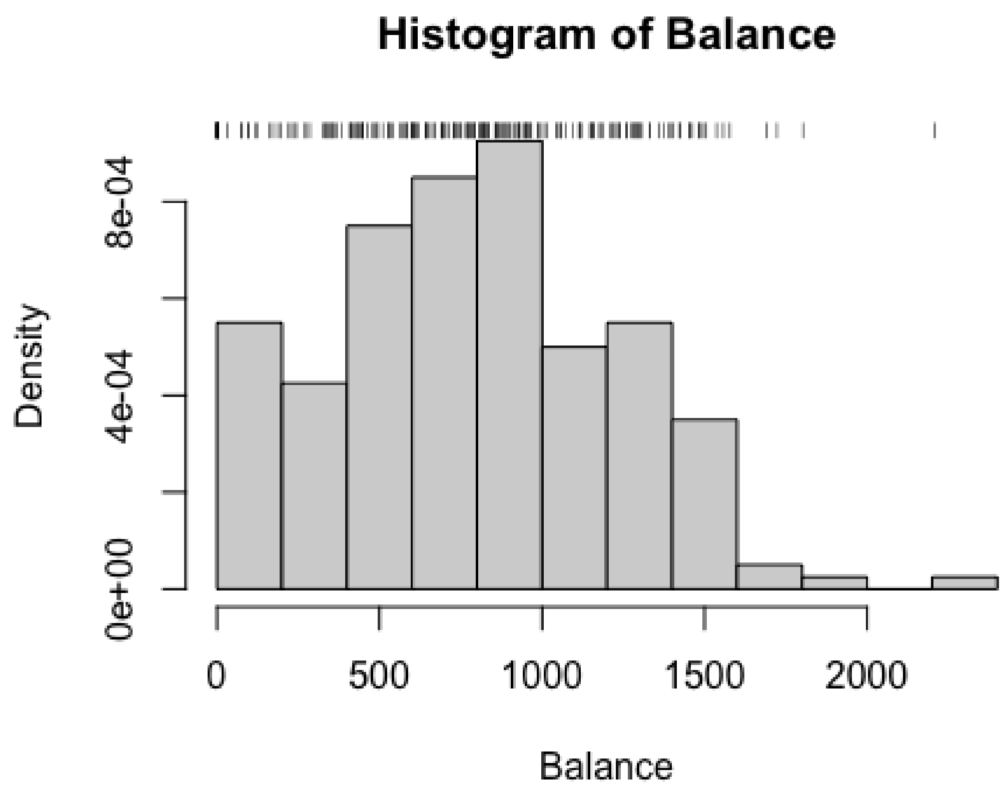
- Add a line to denote the median of the values:

```
hist(default$balance, freq = FALSE, main = "Histogram of Balance", xlab =
"Balance")
abline(v = median(default$balance), col = "red")
```



- Add a rug to provide a one-dimensional view of the distribution:

```
hist(default$balance, freq = FALSE, main = "Histogram of Balance", xlab =
"Balance")
rug(default$balance, side = 3)
```

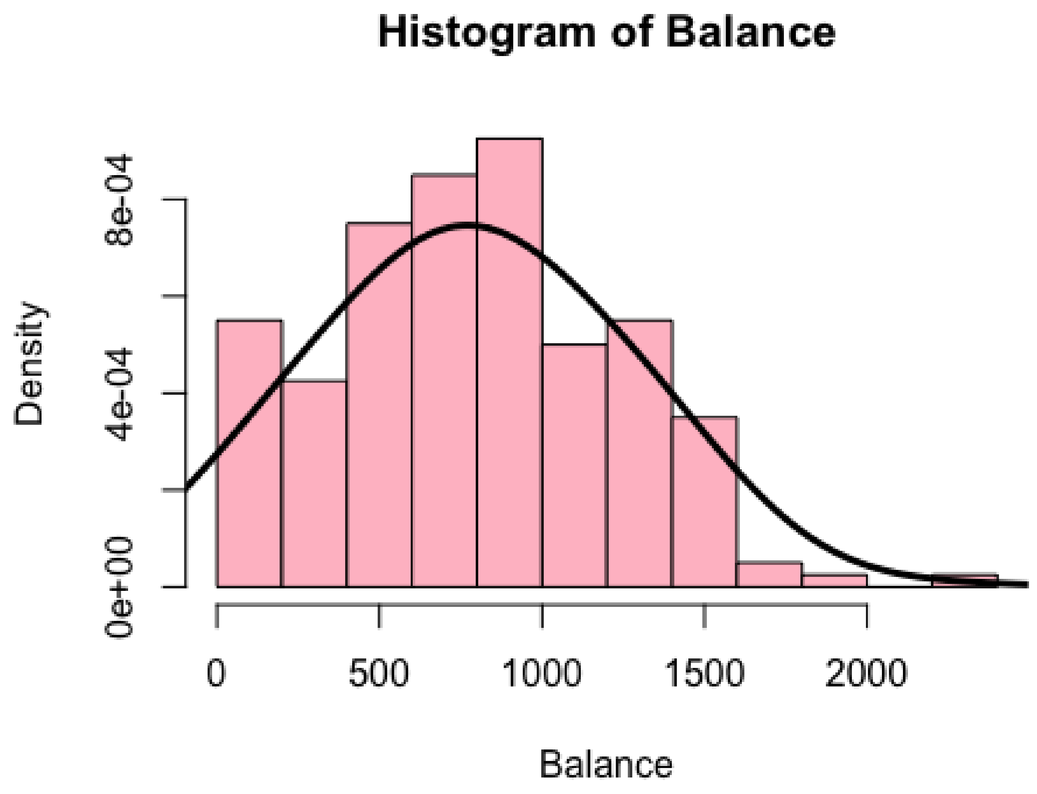


- Add a density curve that depicts the density estimate of the variable:

First, use `density()` to estimate the density.

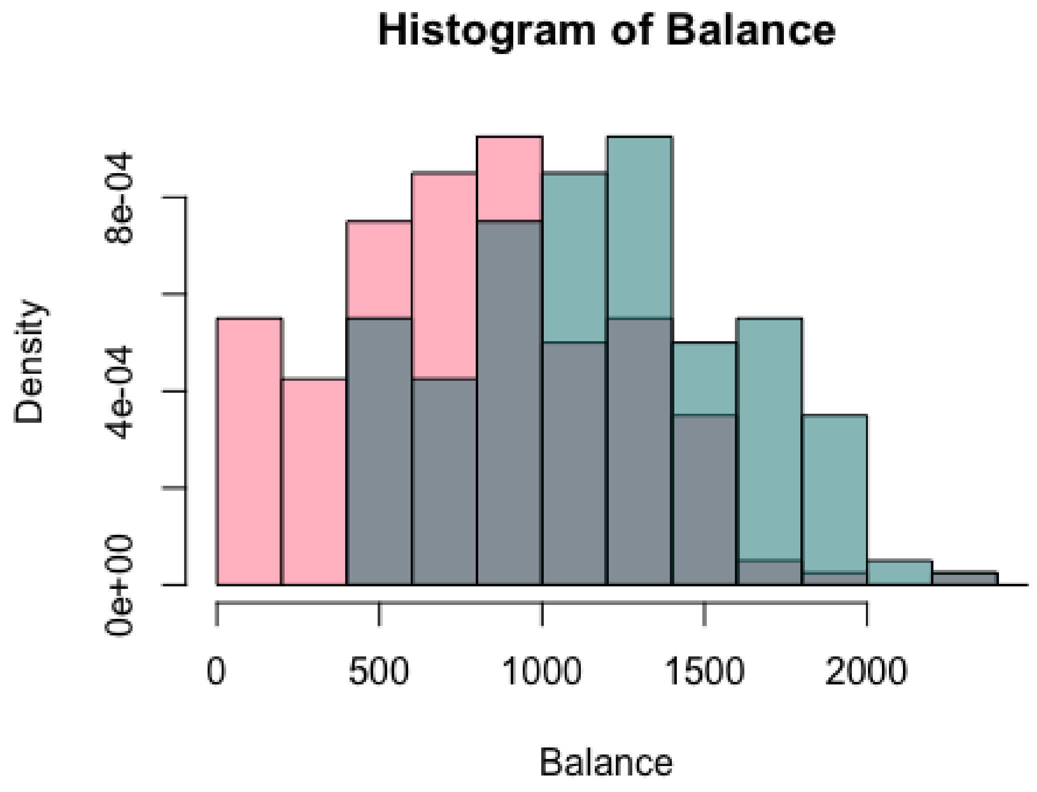
Then, use `lines()` to plot this density estimate.

```
density.est <- density(default$balance, adjust = 2)  # twice the default
smoothing bandwidth
hist(default$balance, col = "pink", freq = FALSE, xlab = "Balance", main =
"Histogram of Balance")
lines(density.est, lwd = 3)
```



We can also add a histogram to an existing plot (say, another histogram) by setting `add = TRUE` in `hist()`:

```
hist(default$balance, col = "pink", freq = FALSE, xlab = "Balance", main =
"Histogram of Balance")
hist(default$balance + 400, col = rgb(0, 0.5, 0.5, 0.5), freq = FALSE, add =
TRUE)
```



In the second histogram, we set the transparency of the color to prevent it from obscuring the first one.

[Task 3: Plotting Histograms]

Use `par(new = TRUE)` to create the same plot (two histograms) as the one created by the above code.

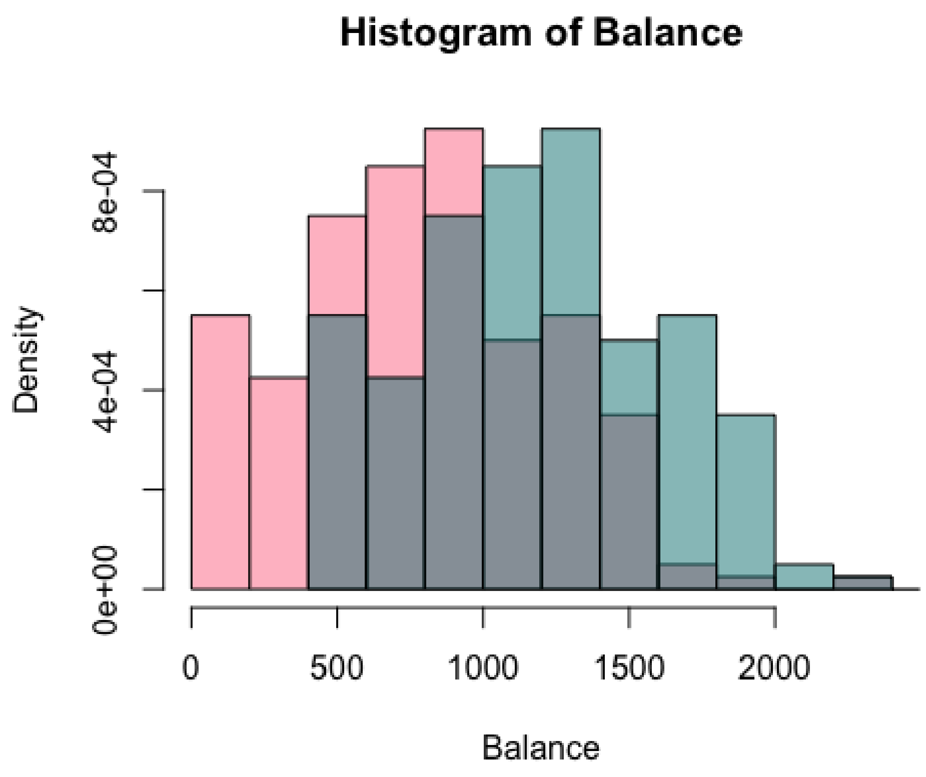


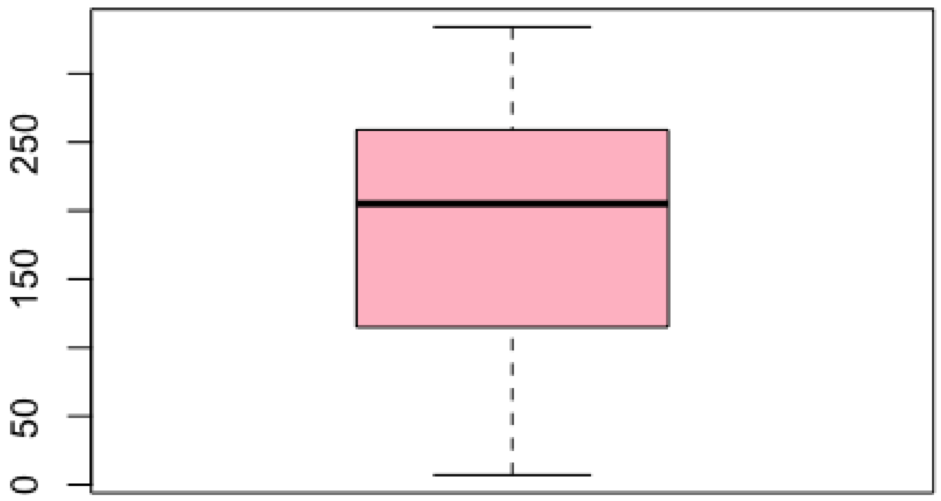
Fig 4. Task 3

[End of Task 3]

7.5.2 Boxplot: `boxplot()`

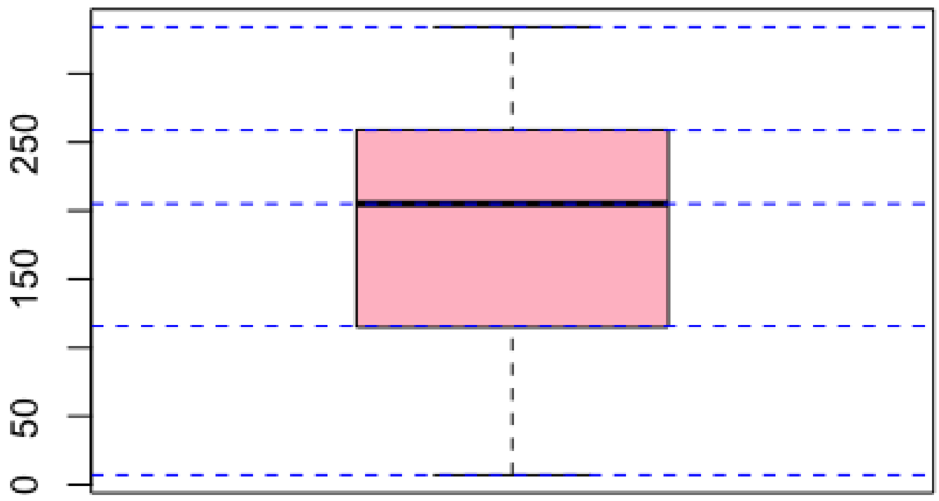
`boxplot()` produces a box-and-whisker plot to show the distribution of a variable.

```
# here we use a built-in data set: airquality
head(airquality)

##   Ozone Solar.R Wind Temp Month Day
## 1    41     190  7.4   67     5   1
## 2    36     118  8.0   72     5   2
## 3    12     149 12.6   74     5   3
## 4    18     313 11.5   62     5   4
## 5    NA      NA 14.3   56     5   5
## 6    28      NA 14.9   66     5   6

boxplot(airquality$Solar.R, col = "pink")
```



In the boxplot, the extreme of the lower whisker, the lower hinge, the median, the upper hinge and the extreme of the upper whisker correspond to the 0, 0.25, 0.5, 0.75 and 1 quantile of the variable, respectively.

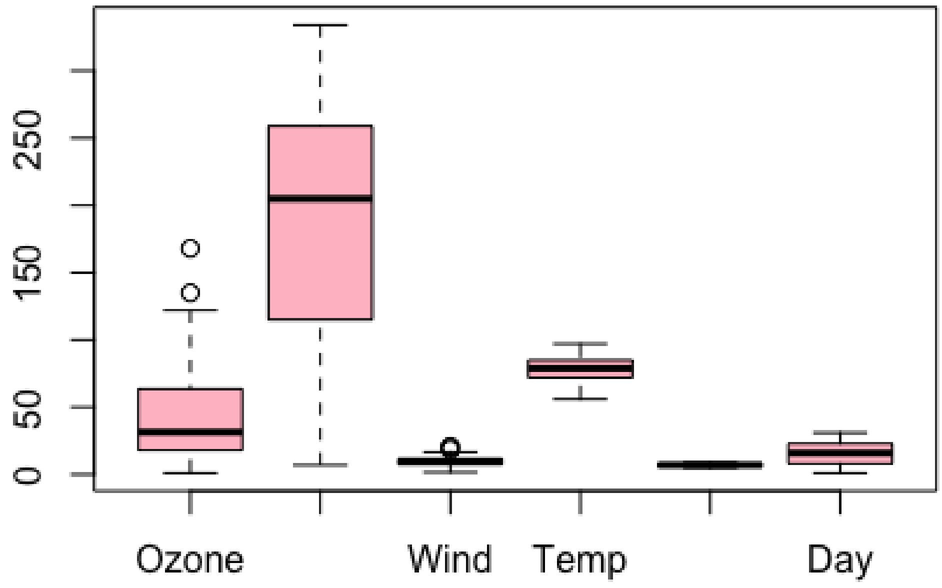
```
boxplot(airquality$Solar.R, col = "pink")
abline(h = quantile(airquality$Solar.R, c(0, 0.25, 0.5, 0.75, 1), na.rm = T),
col = "blue",
    lty = 2)
```



Boxplots for Multiple Variables

`boxplot()` can show the distribution of multiple variables in a data frame:

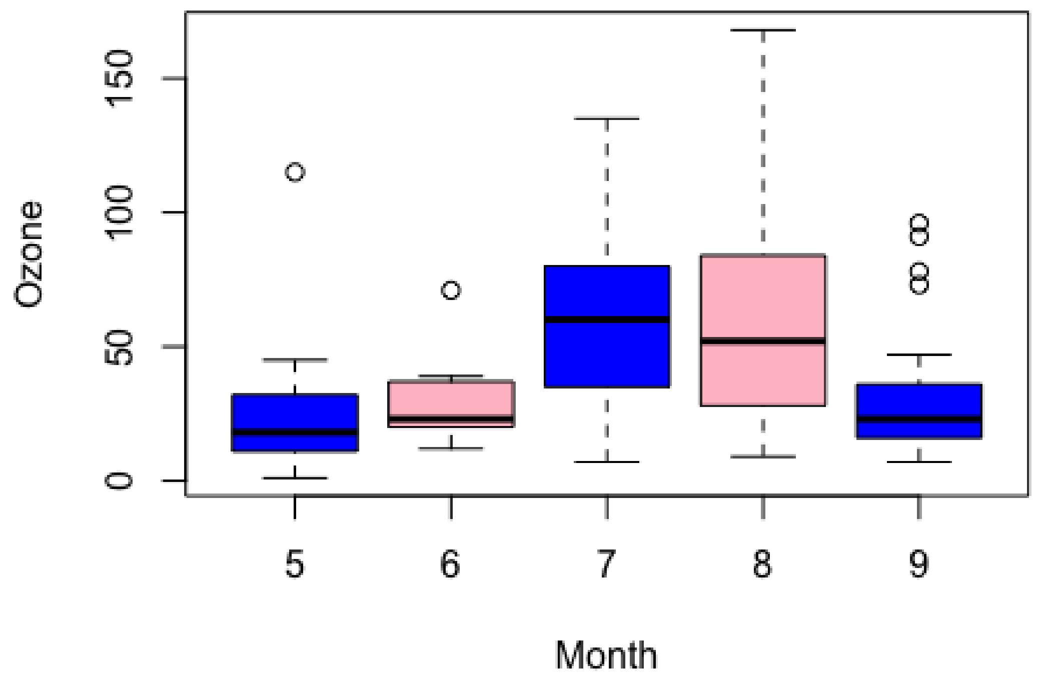
```r
boxplot(airquality, col = "pink")
```



Grouped Boxplots

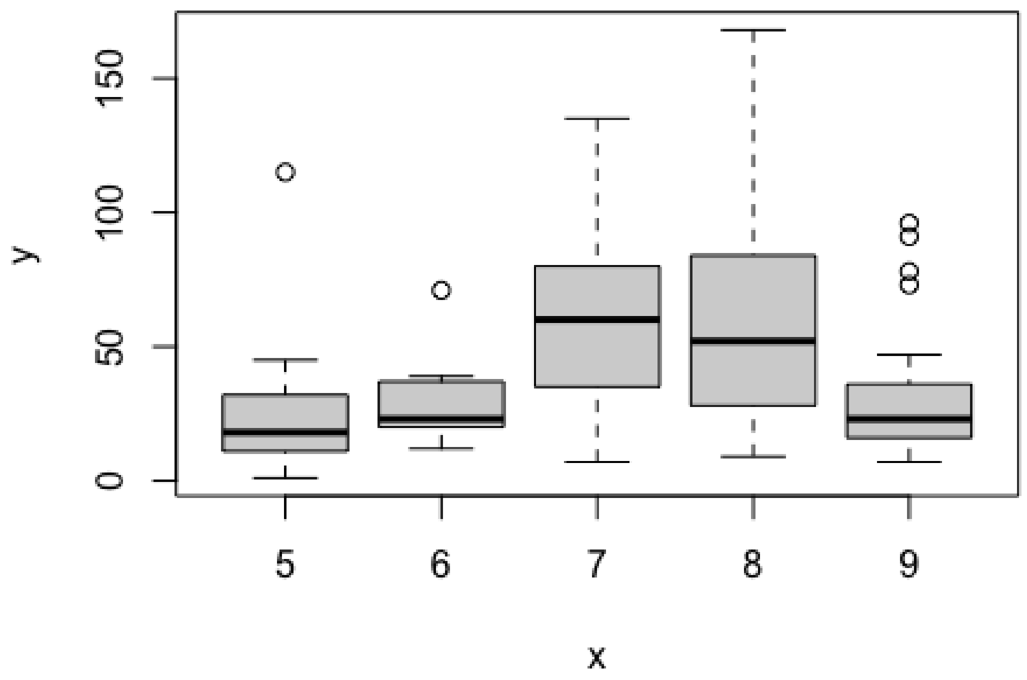
`boxplot()` can accept a formula such as `y ~ x` as an argument, where `y` is a vector of numeric values to be split into groups according to the grouping variable `x` (usually a factor).

```
boxplot(Ozone ~ Month, data = airquality, col = c("blue", "pink"))
```



Alternatively, we can use `plot()` to generate grouped boxplots. Remember that `plot(f, y)` generates a boxplot if `f` is a factor and `y` is a numeric vector (see Section 7.2).

```
plot(factor(airquality$Month), airquality$Ozone)
```

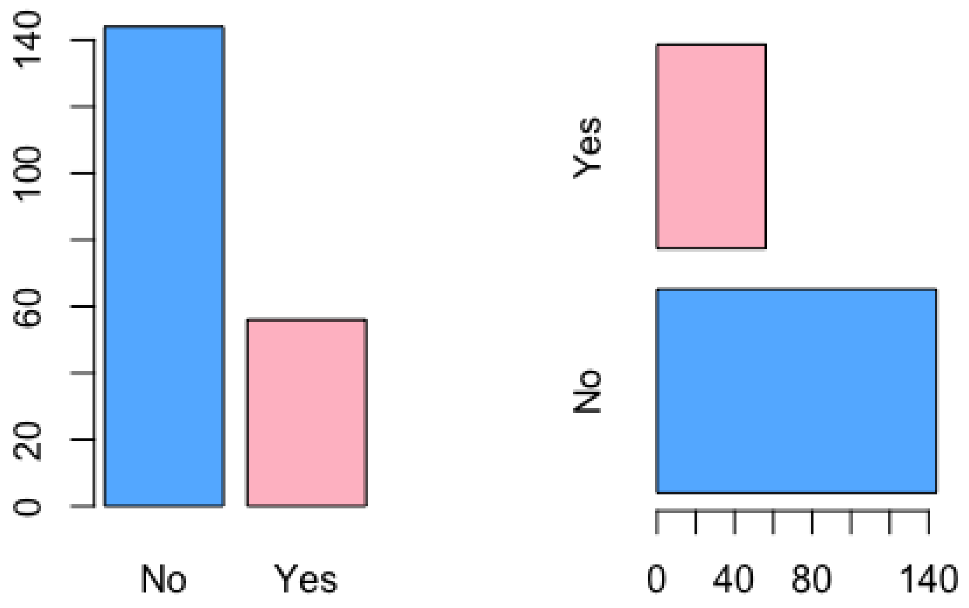


7.5.3 Barplot: `barplot()`

A barplot is a frequently used type of display that compares counts, frequencies, totals or other summary measures for a series of categories.

`barplot()` creates a barplot with vertical or horizontal bars.

```
table(default$student)

##
##  No Yes
## 144  56

par(mfrow = c(1, 2))
barplot(table(default$student), col = c("steelblue1", "pink"))
barplot(table(default$student), horiz = TRUE, col = c("steelblue1", "pink"))
```



7.6 Summary

The base plotting system uses a *painters model* to conceptualize the buildup of a plot.

We start with a blank canvas and add things to it one by one to piece together the plot. Every piece of the plot takes another line(s) of code.

[Task 4: Plotting Stock Prices]

`weeklyprice.csv` contains time series data for stock prices for three large Internet companies (Facebook, Google, and Amazon) from Oct. 30, 2012 to Oct. 30, 2017, as seen in previous in-class exercise.

```
weekly <- read_csv("weeklyprice.csv", col_names = TRUE, col_types = cols())
weekly

## # A tibble: 3 x 263
##   company `2012-10-29` `2012-11-05` `2012-11-12` `2012-11-19` `2012-11-26`
##   <chr>          <dbl>        <dbl>        <dbl>        <dbl>        <dbl>
## 1 FB              21.2         19.2         23.6           24           28
## 2 GOOG           342.         329.         321.         332.         347.
## 3 AMZN           232.         226.         225.         240.         252.
## # … with 257 more variables: `2012-12-03` <dbl>, `2012-12-10` <dbl>,
## #   `2012-12-17` <dbl>, `2012-12-24` <dbl>, `2012-12-31` <dbl>,
## #   `2013-01-07` <dbl>, `2013-01-14` <dbl>, `2013-01-21` <dbl>,
## #   `2013-01-28` <dbl>, `2013-02-04` <dbl>, `2013-02-11` <dbl>,
## #   `2013-02-18` <dbl>, `2013-02-25` <dbl>, `2013-03-04` <dbl>,
## #   `2013-03-11` <dbl>, `2013-03-18` <dbl>, `2013-03-25` <dbl>,
## #   `2013-04-01` <dbl>, `2013-04-08` <dbl>, `2013-04-15` <dbl>,
## #   `2013-04-22` <dbl>, `2013-04-29` <dbl>, `2013-05-06` <dbl>,
## #   `2013-05-13` <dbl>, `2013-05-20` <dbl>, `2013-05-27` <dbl>,
## #   `2013-06-03` <dbl>, `2013-06-10` <dbl>, `2013-06-17` <dbl>,
## #   `2013-06-24` <dbl>, `2013-07-01` <dbl>, `2013-07-08` <dbl>,
## #   `2013-07-15` <dbl>, `2013-07-22` <dbl>, `2013-07-29` <dbl>,
## #   `2013-08-05` <dbl>, `2013-08-12` <dbl>, `2013-08-19` <dbl>,
## #   `2013-08-26` <dbl>, `2013-09-02` <dbl>, `2013-09-09` <dbl>,
## #   `2013-09-16` <dbl>, `2013-09-23` <dbl>, `2013-09-30` <dbl>,
## #   `2013-10-07` <dbl>, `2013-10-14` <dbl>, `2013-10-21` <dbl>,
## #   `2013-10-28` <dbl>, `2013-11-04` <dbl>, `2013-11-11` <dbl>,
## #   `2013-11-18` <dbl>, `2013-11-25` <dbl>, `2013-12-02` <dbl>,
## #   `2013-12-09` <dbl>, `2013-12-16` <dbl>, `2013-12-23` <dbl>,
## #   `2013-12-30` <dbl>, `2014-01-06` <dbl>, `2014-01-13` <dbl>,
## #   `2014-01-20` <dbl>, `2014-01-27` <dbl>, `2014-02-03` <dbl>,
## #   `2014-02-10` <dbl>, `2014-02-17` <dbl>, `2014-02-24` <dbl>,
## #   `2014-03-03` <dbl>, `2014-03-10` <dbl>, `2014-03-17` <dbl>,
## #   `2014-03-24` <dbl>, `2014-03-31` <dbl>, `2014-04-07` <dbl>,
## #   `2014-04-14` <dbl>, `2014-04-21` <dbl>, `2014-04-28` <dbl>,
## #   `2014-05-05` <dbl>, `2014-05-12` <dbl>, `2014-05-19` <dbl>,
## #   `2014-05-26` <dbl>, `2014-06-02` <dbl>, `2014-06-09` <dbl>,
## #   `2014-06-16` <dbl>, `2014-06-23` <dbl>, `2014-06-30` <dbl>,
## #   `2014-07-07` <dbl>, `2014-07-14` <dbl>, `2014-07-21` <dbl>,
## #   `2014-07-28` <dbl>, `2014-08-04` <dbl>, `2014-08-11` <dbl>,
## #   `2014-08-18` <dbl>, `2014-08-25` <dbl>, `2014-09-01` <dbl>,
## #   `2014-09-08` <dbl>, `2014-09-15` <dbl>, `2014-09-22` <dbl>,
## #   `2014-09-29` <dbl>, `2014-10-06` <dbl>, `2014-10-13` <dbl>,
## #   `2014-10-20` <dbl>, `2014-10-27` <dbl>, …
```

Plot time series of stock prices for the three companies with seperate curves, each having a different color. Try to reproduce the following plot:

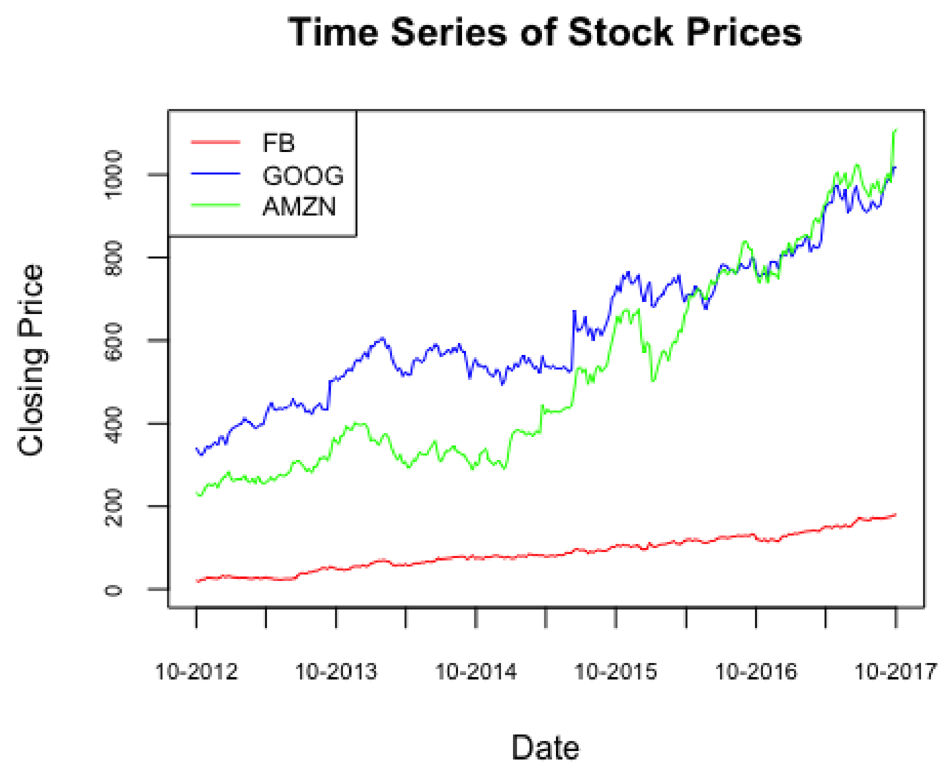


Fig 5. Task 4

Tips:

- Use `as.Date()` to convert strings to the "Date" type;
- Use the following code to customize the x axis representing calendar dates:
```
axis.Date(side = 1, at = seq(minDate, maxDate, by = "6 mon"), format =
"%m-%Y", cex.axis = 0.7)
```

[End of Task 4]