An Introduction to R Graphics

Part I—Base Graphics

Dan Hall, Director of the SCC

Department of Statistics
Franklin College of Arts and Sciences
Statistical Consulting Center
UNIVERSITY OF GEORGIA

Table of Contents

Introduction

Having powerful and flexible systems for graphics is one of R's biggest strengths.

- Base Graphics. Contained in the `graphics` package distributed in base R.

- Grid graphics. `grid` package is distributed in base R.
  - Contains low-level graphics functions.
  - Useful as a platform for developing and implementing higher-level graphics functions and systems.

- Lattice Graphics. `lattice` package also distributed in base R.
  - Mimics and extends trellis graphics from S and S-PLUS.
  - Characteristic feature is plots with multiple panels.
  - Built on grid.

- ggplot2 Graphics. `ggplot2` package available on CRAN.
  - Based on Leland Wilkinson's ideas articulated in his book, *The Grammar of Graphics*.
  - Sophisticated and powerful system. Not too hard to learn.
  - Built on grid.

Introduction

Having powerful and flexible systems for graphics is one of R's biggest strengths.

- Base Graphics. Contained in the `graphics` package distributed in base R.
- Grid graphics. `grid` package is distributed in base R.
  - Contains low-level graphics functions.
  - Useful as a platform for developing and implementing higher-level graphics functions and systems.
- Lattice Graphics. `lattice` package also distributed in base R.
  - Mimics and extends trellis graphics from S and S-PLUS.
  - Characteristic feature is plots with multiple panels.
  - Built on grid.
- ggplot2 Graphics. `ggplot2` package available on CRAN.
  - Based on Leland Wilkinson's ideas articulated in his book, *The Grammar of Graphics*.
  - Sophisticated and powerful system. Not too hard to learn.
  - Built on grid.

Introduction

Having powerful and flexible systems for graphics is one of R's biggest strengths.

- Base Graphics. Contained in the `graphics` package distributed in base R.
- Grid graphics. `grid` package is distributed in base R.
  - Contains low-level graphics functions.
  - Useful as a platform for developing and implementing higher-level graphics functions and systems.
- Lattice Graphics. `lattice` package also distributed in base R.
  - Mimics and extends trellis graphics from S and S-PLUS.
  - Characteristic feature is plots with multiple panels.
  - Built on grid.
- ggplot2 Graphics. `ggplot2` package available on CRAN.
  - Based on Leland Wilkinson's ideas articulated in his book, *The Grammar of Graphics*.
  - Sophisticated and powerful system. Not too hard to learn.
  - Built on grid.

Introduction

Having powerful and flexible systems for graphics is one of R's biggest strengths.

- Base Graphics. Contained in the `graphics` package distributed in base R.

- Grid graphics. `grid` package is distributed in base R.
  - Contains low-level graphics functions.
  - Useful as a platform for developing and implementing higher-level graphics functions and systems.

- Lattice Graphics. `lattice` package also distributed in base R.
  - Mimics and extends trellis graphics from S and S-PLUS.
  - Characteristic feature is plots with multiple panels.
  - Built on grid.

- ggplot2 Graphics. `ggplot2` package available on CRAN.
  - Based on Leland Wilkinson's ideas articulated in his book, *The Grammar of Graphics*.
  - Sophisticated and powerful system. Not too hard to learn.
  - Built on grid.

Introduction

Having powerful and flexible systems for graphics is one of R's biggest strengths.

- Base Graphics. Contained in the `graphics` package distributed in base R.

- Grid graphics. `grid` package is distributed in base R.
    - Contains low-level graphics functions.
    - Useful as a platform for developing and implementing higher-level graphics functions and systems.

- Lattice Graphics. `lattice` package also distributed in base R.
    - Mimics and extends trellis graphics from S and S-PLUS.
    - Characteristic feature is plots with multiple panels.
    - Built on grid.

- ggplot2 Graphics. `ggplot2` package available on CRAN.
    - Based on Leland Wilkinson's ideas articulated in his book, *The Grammar of Graphics*.
    - Sophisticated and powerful system. Not too hard to learn.
    - Built on grid.

Introduction

Having powerful and flexible systems for graphics is one of R's biggest strengths.

- Base Graphics. Contained in the `graphics` package distributed in base R.

- Grid graphics. `grid` package is distributed in base R.
  - Contains low-level graphics functions.
  - Useful as a platform for developing and implementing higher-level graphics functions and systems.

- Lattice Graphics. `lattice` package also distributed in base R.
  - Mimics and extends trellis graphics from S and S-PLUS.
  - Characteristic feature is plots with multiple panels.
  - Built on grid.

- ggplot2 Graphics. `ggplot2` package available on CRAN.
  - Based on Leland Wilkinson's ideas articulated in his book, *The Grammar of Graphics*.
  - Sophisticated and powerful system. Not too hard to learn.
  - Built on grid.

Introduction

Having powerful and flexible systems for graphics is one of R's biggest strengths.

- Base Graphics. Contained in the `graphics` package distributed in base R.
- Grid graphics. `grid` package is distributed in base R.
  - Contains low-level graphics functions.
  - Useful as a platform for developing and implementing higher-level graphics functions and systems.
- Lattice Graphics. `lattice` package also distributed in base R.
  - Mimics and extends trellis graphics from S and S-PLUS.
  - Characteristic feature is plots with multiple panels.
  - Built on grid.
- ggplot2 Graphics. `ggplot2` package available on CRAN.
  - Based on Leland Wilkinson's ideas articulated in his book, *The Grammar of Graphics*.
  - Sophisticated and powerful system. Not too hard to learn.
  - Built on grid.

Introduction

Having powerful and flexible systems for graphics is one of R's biggest strengths.

- Base Graphics. Contained in the `graphics` package distributed in base R.

- Grid graphics. `grid` package is distributed in base R.
  - Contains low-level graphics functions.
  - Useful as a platform for developing and implementing higher-level graphics functions and systems.

- Lattice Graphics. `lattice` package also distributed in base R.
  - Mimics and extends trellis graphics from S and S-PLUS.
  - Characteristic feature is plots with multiple panels.
  - Built on grid.

- ggplot2 Graphics. `ggplot2` package available on CRAN.
  - Based on Leland Wilkinson's ideas articulated in his book, *The Grammar of Graphics*.
  - Sophisticated and powerful system. Not too hard to learn.
  - Built on grid.

Introduction

Having powerful and flexible systems for graphics is one of R's biggest strengths.

- Base Graphics. Contained in the `graphics` package distributed in base R.

- Grid graphics. `grid` package is distributed in base R.
  - Contains low-level graphics functions.
  - Useful as a platform for developing and implementing higher-level graphics functions and systems.

- Lattice Graphics. `lattice` package also distributed in base R.
  - Mimics and extends trellis graphics from S and S-PLUS.
  - Characteristic feature is plots with multiple panels.
  - Built on grid.

- ggplot2 Graphics. `ggplot2` package available on CRAN.
  - Based on Leland Wilkinson's ideas articulated in his book, *The Grammar of Graphics*.
  - Sophisticated and powerful system. Not too hard to learn.
  - Built on grid.

Introduction

Having powerful and flexible systems for graphics is one of R's biggest strengths.

- Base Graphics. Contained in the `graphics` package distributed in base R.

- Grid graphics. `grid` package is distributed in base R.
  - Contains low-level graphics functions.
  - Useful as a platform for developing and implementing higher-level graphics functions and systems.

- Lattice Graphics. `lattice` package also distributed in base R.
  - Mimics and extends trellis graphics from S and S-PLUS.
  - Characteristic feature is plots with multiple panels.
  - Built on grid.

- ggplot2 Graphics. `ggplot2` package available on CRAN.
  - Based on Leland Wilkinson's ideas articulated in his book, *The Grammar of Graphics.*
  - Sophisticated and powerful system. Not too hard to learn.
  - Built on grid.

Introduction

Having powerful and flexible systems for graphics is one of R's biggest strengths.

- Base Graphics. Contained in the `graphics` package distributed in base R.

- Grid graphics. `grid` package is distributed in base R.
  - Contains low-level graphics functions.
  - Useful as a platform for developing and implementing higher-level graphics functions and systems.

- Lattice Graphics. `lattice` package also distributed in base R.
  - Mimics and extends trellis graphics from S and S-PLUS.
  - Characteristic feature is plots with multiple panels.
  - Built on grid.

- ggplot2 Graphics. `ggplot2` package available on CRAN.
  - Based on Leland Wilkinson's ideas articulated in his book, *The Grammar of Graphics.*
  - Sophisticated and powerful system. Not too hard to learn.
  - Built on grid.

Introduction

Having powerful and flexible systems for graphics is one of R's biggest strengths.

- Base Graphics. Contained in the `graphics` package distributed in base R.

- Grid graphics. `grid` package is distributed in base R.
    - Contains low-level graphics functions.
    - Useful as a platform for developing and implementing higher-level graphics functions and systems.

- Lattice Graphics. `lattice` package also distributed in base R.
    - Mimics and extends trellis graphics from S and S-PLUS.
    - Characteristic feature is plots with multiple panels.
    - Built on grid.

- ggplot2 Graphics. `ggplot2` package available on CRAN.
    - Based on Leland Wilkinson's ideas articulated in his book, *The Grammar of Graphics.*
    - Sophisticated and powerful system. Not too hard to learn.
    - Built on grid.

Base Graphics

- High level functions—Produce a new complete plot on the current graphics device.
  - `plot()`: generic function capable of a wide variety of plot types.
  - `boxplot()`: single and side-by-side boxplots.
  - `hist()`: histograms.
  - `qqplot()`, `qqnorm()`, `qqline()`: quantile-quantile plots.
  - `dotchart()`, `stripchart()`: dot plots.
  - `image()`, `contour()`, `persp()`: 3d plots.
  - `pairs()`: scatter plot matrices.
- Low level functions—Add features to an existing plot.
  - `lines()`, `points()`, `symbols()`, `segments()`, `arrows()`: add various features. Most have syntax similar to `plot()`.
  - `abline()`, `curve()`: add lines or curves from output of a model or at reference locations.
  - `title()`, `legend()`: add a title or legend.
  - `axis()`: adds an axis with fine control of its appearance.
  - `text()`, `mtext()`: add text within the plotting region or in the margins.

Base Graphics

- High level functions—Produce a new complete plot on the current graphics device.
  - `plot()`: generic function capable of a wide variety of plot types.
  - `boxplot()`: single and side-by-side boxplots.
  - `hist()`: histograms.
  - `qqplot()`, `qqnorm()`, `qqline()`: quantile-quantile plots.
  - `dotchart()`, `stripchart()`: dot plots.
  - `image()`, `contour()`, `persp()`: 3d plots.
  - `pairs()`: scatter plot matrices.
- Low level functions—Add features to an existing plot.
  - `lines()`, `points()`, `symbols()`, `segments()`, `arrows()`: add various features. Most have syntax similar to `plot()`.
  - `abline()`, `curve()`: add lines or curves from output of a model or at reference locations.
  - `title()`, `legend()`: add a title or legend.
  - `axis()`: adds an axis with fine control of its appearance.
  - `text()`, `mtext()`: add text within the plotting region or in the margins.

Base Graphics

- High level functions—Produce a new complete plot on the current graphics device.
  - `plot()`: generic function capable of a wide variety of plot types.
  - `boxplot()`: single and side-by-side boxplots.
  - `hist()`: histograms.
  - `qqplot()`, `qqnorm()`, `qqline()`: quantile-quantile plots.
  - `dotchart()`, `stripchart()`: dot plots.
  - `image()`, `contour()`, `persp()`: 3d plots.
  - `pairs()`: scatter plot matrices.

- Low level functions—Add features to an existing plot.
  - `lines()`, `points()`, `symbols()`, `segments()`, `arrows()`: add various features. Most have syntax similar to `plot()`.
  - `abline()`, `curve()`: add lines or curves from output of a model or at reference locations.
  - `title()`, `legend()`: add a title or legend.
  - `axis()`: adds an axis with fine control of its appearance.
  - `text()`, `mtext()`: add text within the plotting region or in the margins.

Base Graphics

- High level functions—Produce a new complete plot on the current graphics device.
  - `plot()`: generic function capable of a wide variety of plot types.
  - `boxplot()`: single and side-by-side boxplots.
  - `hist()`: histograms.
  - `qqplot()`, `qqnorm()`, `qqline()`: quantile-quantile plots.
  - `dotchart()`, `stripchart()`: dot plots.
  - `image()`, `contour()`, `persp()`: 3d plots.
  - `pairs()`: scatter plot matrices.
- Low level functions—Add features to an existing plot.
  - `lines()`, `points()`, `symbols()`, `segments()`, `arrows()`: add various features. Most have syntax similar to `plot()`.
  - `abline()`, `curve()`: add lines or curves from output of a model or at reference locations.
  - `title()`, `legend()`: add a title or legend.
  - `axis()`: adds an axis with fine control of its appearance.
  - `text()`, `mtext()`: add text within the plotting region or in the margins.

Base Graphics

- High level functions—Produce a new complete plot on the current graphics device.
  - `plot()`: generic function capable of a wide variety of plot types.
  - `boxplot()`: single and side-by-side boxplots.
  - `hist()`: histograms.
  - `qqplot()`, `qqnorm()`, `qqline()`: quantile-quantile plots.
  - `dotchart()`, `stripchart()`: dot plots.
  - `image()`, `contour()`, `persp()`: 3d plots.
  - `pairs()`: scatter plot matrices.

- Low level functions—Add features to an existing plot.
  - `lines()`, `points()`, `symbols()`, `segments()`, `arrows()`: add various features. Most have syntax similar to `plot()`.
  - `abline()`, `curve()`: add lines or curves from output of a model or at reference locations.
  - `title()`, `legend()`: add a title or legend.
  - `axis()`: adds an axis with fine control of its appearance.
  - `text()`, `mtext()`: add text within the plotting region or in the margins.

Base Graphics

- High level functions—Produce a new complete plot on the current graphics device.
  - `plot()`: generic function capable of a wide variety of plot types.
  - `boxplot()`: single and side-by-side boxplots.
  - `hist()`: histograms.
  - `qqplot()`, `qqnorm()`, `qqline()`: quantile-quantile plots.
  - `dotchart()`, `stripchart()`: dot plots.
  - `image()`, `contour()`, `persp()`: 3d plots.
  - `pairs()`: scatter plot matrices.

- Low level functions—Add features to an existing plot.
  - `lines()`, `points()`, `symbols()`, `segments()`, `arrows()`: add various features. Most have syntax similar to `plot()`.
  - `abline()`, `curve()`: add lines or curves from output of a model or at reference locations.
  - `title()`, `legend()`: add a title or legend.
  - `axis()`: adds an axis with fine control of its appearance.
  - `text()`, `mtext()`: add text within the plotting region or in the margins.

Base Graphics

- High level functions—Produce a new complete plot on the current graphics device.
  - `plot()`: generic function capable of a wide variety of plot types.
  - `boxplot()`: single and side-by-side boxplots.
  - `hist()`: histograms.
  - `qqplot()`, `qqnorm()`, `qqline()`: quantile-quantile plots.
  - `dotchart()`, `stripchart()`: dot plots.
  - `image()`, `contour()`, `persp()`: 3d plots.
  - `pairs()`: scatter plot matrices.
- Low level functions—Add features to an existing plot.
  - `lines()`, `points()`, `symbols()`, `segments()`, `arrows()`: add various features. Most have syntax similar to `plot()`.
  - `abline()`, `curve()`: add lines or curves from output of a model or at reference locations.
  - `title()`, `legend()`: add a title or legend.
  - `axis()`: adds an axis with fine control of its appearance.
  - `text()`, `mtext()`: add text within the plotting region or in the margins.

Base Graphics

- High level functions—Produce a new complete plot on the current graphics device.
  - `plot()`: generic function capable of a wide variety of plot types.
  - `boxplot()`: single and side-by-side boxplots.
  - `hist()`: histograms.
  - `qqplot()`, `qqnorm()`, `qqline()`: quantile-quantile plots.
  - `dotchart()`, `stripchart()`: dot plots.
  - `image()`, `contour()`, `persp()`: 3d plots.
  - `pairs()`: scatter plot matrices.

- Low level functions—Add features to an existing plot.
  - `lines()`, `points()`, `symbols()`, `segments()`, `arrows()`: add various features. Most have syntax similar to `plot()`.
  - `abline()`, `curve()`: add lines or curves from output of a model or at reference locations.
  - `title()`, `legend()`: add a title or legend.
  - `axis()`: adds an axis with fine control of its appearance.
  - `text()`, `mtext()`: add text within the plotting region or in the margins.

Base Graphics

- High level functions—Produce a new complete plot on the current graphics device.
  - `plot()`: generic function capable of a wide variety of plot types.
  - `boxplot()`: single and side-by-side boxplots.
  - `hist()`: histograms.
  - `qqplot()`, `qqnorm()`, `qqline()`: quantile-quantile plots.
  - `dotchart()`, `stripchart()`: dot plots.
  - `image()`, `contour()`, `persp()`: 3d plots.
  - `pairs()`: scatter plot matrices.

- Low level functions—Add features to an existing plot.
  - `lines()`, `points()`, `symbols()`, `segments()`, `arrows()`: add various features. Most have syntax similar to `plot()`.
  - `abline()`, `curve()`: add lines or curves from output of a model or at reference locations.
  - `title()`, `legend()`: add a title or legend.
  - `axis()`: adds an axis with fine control of its appearance.
  - `text()`, `mtext()`: add text within the plotting region or in the margins.

Base Graphics

- High level functions—Produce a new complete plot on the current graphics device.
  - `plot()`: generic function capable of a wide variety of plot types.
  - `boxplot()`: single and side-by-side boxplots.
  - `hist()`: histograms.
  - `qqplot()`, `qqnorm()`, `qqline()`: quantile-quantile plots.
  - `dotchart()`, `stripchart()`: dot plots.
  - `image()`, `contour()`, `persp()`: 3d plots.
  - `pairs()`: scatter plot matrices.

- Low level functions—Add features to an existing plot.
  - `lines()`, `points()`, `symbols()`, `segments()`, `arrows()`: add various features. Most have syntax similar to `plot()`.
  - `abline()`, `curve()`: add lines or curves from output of a model or at reference locations.
  - `title()`, `legend()`: add a title or legend.
  - `axis()`: adds an axis with fine control of its appearance.
  - `text()`, `mtext()`: add text within the plotting region or in the margins.

Base Graphics

- High level functions—Produce a new complete plot on the current graphics device.
  - `plot()`: generic function capable of a wide variety of plot types.
  - `boxplot()`: single and side-by-side boxplots.
  - `hist()`: histograms.
  - `qqplot()`, `qqnorm()`, `qqline()`: quantile-quantile plots.
  - `dotchart()`, `stripchart()`: dot plots.
  - `image()`, `contour()`, `persp()`: 3d plots.
  - `pairs()`: scatter plot matrices.

- Low level functions—Add features to an existing plot.
  - `lines()`, `points()`, `symbols()`, `segments()`, `arrows()`: add various features. Most have syntax similar to `plot()`.
  - `abline()`, `curve()`: add lines or curves from output of a model or at reference locations.
  - `title()`, `legend()`: add a title or legend.
  - `axis()`: adds an axis with fine control of its appearance.
  - `text()`, `mtext()`: add text within the plotting region or in the margins.

Base Graphics

- High level functions—Produce a new complete plot on the current graphics device.
  - **`plot()`**: generic function capable of a wide variety of plot types.
  - **`boxplot()`**: single and side-by-side boxplots.
  - **`hist()`**: histograms.
  - **`qqplot()`, `qqnorm()`, `qqline()`**: quantile-quantile plots.
  - **`dotchart()`, `stripchart()`**: dot plots.
  - **`image()`, `contour()`, `persp()`**: 3d plots.
  - **`pairs()`**: scatter plot matrices.

- Low level functions—Add features to an existing plot.
  - **`lines()`, `points()`, `symbols()`, `segments()`, `arrows()`**: add various features. Most have syntax similar to **`plot()`**.
  - **`abline()`, `curve()`**: add lines or curves from output of a model or at reference locations.
  - **`title()`, `legend()`**: add a title or legend.
  - `axis()`: adds an axis with fine control of its appearance.
  - `text()`, `mtext()`: add text within the plotting region or in the margins.

Base Graphics

- High level functions—Produce a new complete plot on the current graphics device.
  - `plot()`: generic function capable of a wide variety of plot types.
  - `boxplot()`: single and side-by-side boxplots.
  - `hist()`: histograms.
  - `qqplot()`, `qqnorm()`, `qqline()`: quantile-quantile plots.
  - `dotchart()`, `stripchart()`: dot plots.
  - `image()`, `contour()`, `persp()`: 3d plots.
  - `pairs()`: scatter plot matrices.

- Low level functions—Add features to an existing plot.
  - `lines()`, `points()`, `symbols()`, `segments()`, `arrows()`: add various features. Most have syntax similar to `plot()`.
  - `abline()`, `curve()`: add lines or curves from output of a model or at reference locations.
  - `title()`, `legend()`: add a title or legend.
  - `axis()`: adds an axis with fine control of its appearance.
  - `text()`, `mtext()`: add text within the plotting region or in the margins.

Base Graphics

- High level functions—Produce a new complete plot on the current graphics device.
  - `plot()`: generic function capable of a wide variety of plot types.
  - `boxplot()`: single and side-by-side boxplots.
  - `hist()`: histograms.
  - `qqplot()`, `qqnorm()`, `qqline()`: quantile-quantile plots.
  - `dotchart()`, `stripchart()`: dot plots.
  - `image()`, `contour()`, `persp()`: 3d plots.
  - `pairs()`: scatter plot matrices.

- Low level functions—Add features to an existing plot.
  - `lines()`, `points()`, `symbols()`, `segments()`, `arrows()`: add various features. Most have syntax similar to `plot()`.
  - `abline()`, `curve()`: add lines or curves from output of a model or at reference locations.
  - `title()`, `legend()`: add a title or legend.
  - `axis()`: adds an axis with fine control of its appearance.
  - `text()`, `mtext()`: add text within the plotting region or in the margins.

Graphics Devices

- Plots are sent to a graphics device, typically a window or file.
- Screen device functions:
  - Mac-OS: `quartz()`. Also allows output to files of different formats.
  - Unix/Linux: `x11()`.
  - Windows OS: `windows()` (or `x11()` or `X11()`).
    - Use `windows(record=TRUE)` to record the plots so you can page through them.
- In RStudio, built-in device is `RStudioGD`. Plots can be copied and pasted or saved to files of different formats from it.
- There are also file devices such as `pdf` and `postscript`.
- Multiple devices can be open simultaneously, but only one is the *current device*.
  - Switching devices, turning them off, etc. with functions such as `dev.off()`, `dev.cur()`, `dev.next()`, etc.
- Each device has its own graphical parameters. Setting parameters (e.g., with `par()`) affects those of the current device.

Graphics Devices

- Plots are sent to a graphics device, typically a window or file.
- Screen device functions:
  - Mac-OS: `quartz()`. Also allows output to files of different formats.
  - Unix/Linux: `x11()`.
  - Windows OS: `windows()` (or `x11()` or `X11()`).
    - Use `windows(record=TRUE)` to *record* the plots so you can page through them.
- In RStudio, built-in device is `RStudioGD`. Plots can be copied and pasted or saved to files of different formats from it.
- There are also file devices such as `pdf` and `postscript`.
- Multiple devices can be open simultaneously, but only one is the *current device*.
  - Switching devices, turning them off, etc. with functions such as `dev.off()`, `dev.cur()`, `dev.next()`, etc.
- Each device has its own graphical parameters. Setting parameters (e.g., with `par()`) affects those of the current device.

Graphics Devices

- Plots are sent to a graphics device, typically a window or file.
- Screen device functions:
    - Mac-OS: `quartz()`. Also allows output to files of different formats.
    - Unix/Linux: `x11()`.
    - Windows OS: `windows()` (or `x11()` or `X11()`).
        - Use `windows(record=TRUE)` to *record* the plots so you can page through them.
- In RStudio, built-in device is `RStudioGD`. Plots can be copied and pasted or saved to files of different formats from it.
- There are also file devices such as `pdf` and `postscript`.
- Multiple devices can be open simultaneously, but only one is the *current device*.
    - Switching devices, turning them off, etc. with functions such as `dev.off()`, `dev.cur()`, `dev.next()`, etc.
- Each device has its own graphical parameters. Setting parameters (e.g., with `par()`) affects those of the current device.

Graphics Devices

- Plots are sent to a graphics device, typically a window or file.
- Screen device functions:
  - Mac-OS: `quartz()`. Also allows output to files of different formats.
  - Unix/Linux: `x11()`.
  - Windows OS: `windows()` (or `x11()` or `X11()`).
    - Use `windows(record=TRUE)` to *record* the plots so you can page through them.
- In RStudio, built-in device is `RStudioGD`. Plots can be copied and pasted or saved to files of different formats from it.
- There are also file devices such as `pdf` and `postscript`.
- Multiple devices can be open simultaneously, but only one is the *current device*.
  - Switching devices, turning them off, etc. with functions such as `dev.off()`, `dev.cur()`, `dev.next()`, etc.
- Each device has its own graphical parameters. Setting parameters (e.g., with `par()`) affects those of the current device.

Graphics Devices

- Plots are sent to a graphics device, typically a window or file.
- Screen device functions:
  - Mac-OS: `quartz()`. Also allows output to files of different formats.
  - Unix/Linux: `x11()`.
  - Windows OS: `windows()` (or `x11()` or `X11()`).
    - Use `windows(record=TRUE)` to *record* the plots so you can page through them.
- In RStudio, built-in device is `RStudioGD`. Plots can be copied and pasted or saved to files of different formats from it.
- There are also file devices such as `pdf` and `postscript`.
- Multiple devices can be open simultaneously, but only one is the *current device*.
  - Switching devices, turning them off, etc. with functions such as `dev.off()`, `dev.cur()`, `dev.next()`, etc.
- Each device has its own graphical parameters. Setting parameters (e.g., with `par()`) affects those of the current device.

Graphics Devices

- Plots are sent to a graphics device, typically a window or file.
- Screen device functions:
  - Mac-OS: `quartz()`. Also allows output to files of different formats.
  - Unix/Linux: `x11()`.
  - Windows OS: `windows()` (or `x11()` or `X11()`).
    - Use `windows(record=TRUE)` to *record* the plots so you can page through them.
- In RStudio, built-in device is `RStudioGD`. Plots can be copied and pasted or saved to files of different formats from it.
- There are also file devices such as `pdf` and `postscript`.
- Multiple devices can be open simultaneously, but only one is the *current device*.
  - Switching devices, turning them off, etc. with functions such as `dev.off()`, `dev.cur()`, `dev.next()`, etc.
- Each device has its own graphical parameters. Setting parameters (e.g., with `par()`) affects those of the current device.

Graphics Devices

- Plots are sent to a graphics device, typically a window or file.
- Screen device functions:
  - Mac-OS: `quartz()`. Also allows output to files of different formats.
  - Unix/Linux: `x11()`.
  - Windows OS: `windows()` (or `x11()` or `X11()`).
    - Use `windows(record=TRUE)` to *record* the plots so you can page through them.
- In RStudio, built-in device is `RStudioGD`. Plots can be copied and pasted or saved to files of different formats from it.
- There are also file devices such as `pdf` and `postscript`.
- Multiple devices can be open simultaneously, but only one is the *current device*.
  - Switching devices, turning them off, etc. with functions such as `dev.off()`, `dev.cur()`, `dev.next()`, etc.
- Each device has its own graphical parameters. Setting parameters (e.g., with `par()`) affects those of the current device.

Graphics Devices

- Plots are sent to a graphics device, typically a window or file.
- Screen device functions:
  - Mac-OS: `quartz()`. Also allows output to files of different formats.
  - Unix/Linux: `x11()`.
  - Windows OS: `windows()` (or `x11()` or `X11()`).
    - Use `windows(record=TRUE)` to *record* the plots so you can page through them.
- In RStudio, built-in device is `RStudioGD`. Plots can be copied and pasted or saved to files of different formats from it.
- There are also file devices such as `pdf` and `postscript`.
- Multiple devices can be open simultaneously, but only one is the *current device*.
  - Switching devices, turning them off, etc. with functions such as `dev.off()`, `dev.cur()`, `dev.next()`, etc.
- Each device has its own graphical parameters. Setting parameters (e.g., with `par()`) affects those of the current device.

Graphics Devices

- Plots are sent to a graphics device, typically a window or file.
- Screen device functions:
  - Mac-OS: `quartz()`. Also allows output to files of different formats.
  - Unix/Linux: `x11()`.
  - Windows OS: `windows()` (or `x11()` or `X11()`).
    - Use `windows(record=TRUE)` to *record* the plots so you can page through them.
- In RStudio, built-in device is `RStudioGD`. Plots can be copied and pasted or saved to files of different formats from it.
- There are also file devices such as `pdf` and `postscript`.
- Multiple devices can be open simultaneously, but only one is the *current device*.
  - Switching devices, turning them off, etc. with functions such as `dev.off()`, `dev.cur()`, `dev.next()`, etc.
- Each device has its own graphical parameters. Setting parameters (e.g., with `par()`) affects those of the current device.

Graphics Devices

- Plots are sent to a graphics device, typically a window or file.
- Screen device functions:
  - Mac-OS: `quartz()`. Also allows output to files of different formats.
  - Unix/Linux: `x11()`.
  - Windows OS: `windows()` (or `x11()` or `X11()`).
    - Use `windows(record=TRUE)` to *record* the plots so you can page through them.
- In RStudio, built-in device is `RStudioGD`. Plots can be copied and pasted or saved to files of different formats from it.
- There are also file devices such as `pdf` and `postscript`.
- Multiple devices can be open simultaneously, but only one is the *current device*.
  - Switching devices, turning them off, etc. with functions such as `dev.off()`, `dev.cur()`, `dev.next()`, etc.
- Each device has its own graphical parameters. Setting parameters (e.g., with `par()`) affects those of the current device.

Graphics Devices

- Plots are sent to a graphics device, typically a window or file.
- Screen device functions:
  - Mac-OS: `quartz()`. Also allows output to files of different formats.
  - Unix/Linux: `x11()`.
  - Windows OS: `windows()` (or `x11()` or `X11()`).
    - Use `windows(record=TRUE)` to *record* the plots so you can page through them.
- In RStudio, built-in device is `RStudioGD`. Plots can be copied and pasted or saved to files of different formats from it.
- There are also file devices such as `pdf` and `postscript`.
- Multiple devices can be open simultaneously, but only one is the *current device*.
  - Switching devices, turning them off, etc. with functions such as `dev.off()`, `dev.cur()`, `dev.next()`, etc.
- Each device has its own graphical parameters. Setting parameters (e.g., with `par()`) affects those of the current device.

Graphical Parameters

Many aspects of a plot are controlled by a large number of graphical parameters.

- These parameters can be queried or reset with `par()`.
  - See `?par` for a list of graphical parameters.
  - The command `par("param")` queries the value of parameter `param`.
  - E.g., below we set the `col` and `lty` parameters, add a dotted red horizontal line at 0, and then reset the parameters to their previous values.

```
par(c("col","lty"))                         # query the current values ("black" and "solid")
oldParms <- par(col="red",lty="dotted") # set to new values and save the old ones
par(c("col","lty"))                         # ("red" and "dotted")
abline(h=0)                                 # line will be dotted and red
par(oldParms)                               # reset to the original values
par(c("col","lty"))                         # ("black" and "solid")
```

Graphical Parameters

Many aspects of a plot are controlled by a large number of graphical parameters.

- These parameters can be queried or reset with `par()`.
    - See `?par` for a list of graphical parameters.
    - The command `par("param")` queries the value of parameter `param`.
    - E.g., below we set the `col` and `lty` parameters, add a dotted red horizontal line at 0, and then reset the parameters to their previous values.

```
par(c("col","lty"))                    # query the current values ("black" and "solid")
oldParms <- par(col="red",lty="dotted") # set to new values and save the old ones
par(c("col","lty"))                    # ("red" and "dotted")
abline(h=0)                            # line will be dotted and red
par(oldParms)                          # reset to the original values
par(c("col","lty"))                    # ("black" and "solid")
```

Graphical Parameters

Many aspects of a plot are controlled by a large number of graphical parameters.

- These parameters can be queried or reset with `par()`.
    - See `?par` for a list of graphical parameters.
    - The command `par("param")` queries the value of parameter `param`.
    - E.g., below we set the `col` and `lty` parameters, add a dotted red horizontal line at 0, and then reset the parameters to their previous values.

```
par(c("col","lty"))                        # query the current values ("black" and "solid")
oldParms <- par(col="red",lty="dotted") # set to new values and save the old ones
par(c("col","lty"))                        # ("red" and "dotted")
abline(h=0)                                # line will be dotted and red
par(oldParms)                              # reset to the original values
par(c("col","lty"))                        # ("black" and "solid")
```

Graphical Parameters

Many aspects of a plot are controlled by a large number of graphical parameters.

- These parameters can be queried or reset with `par()`.
  - See `?par` for a list of graphical parameters.
  - The command `par("param")` queries the value of parameter `param`.
  - E.g., below we set the `col` and `lty` parameters, add a dotted red horizontal line at 0, and then reset the parameters to their previous values.

```
par(c("col","lty"))                        # query the current values ("black" and "solid")
oldParms <- par(col="red",lty="dotted") # set to new values and save the old ones
par(c("col","lty"))                        # ("red" and "dotted")
abline(h=0)                                # line will be dotted and red
par(oldParms)                              # reset to the original values
par(c("col","lty"))                        # ("black" and "solid")
```

Graphical Parameters

- Graphical parameters can be changed with `par()` and the new value will persist.
- Most plotting functions also accept graphical parameters optionally.
  - These settings will be temporary to the function being executed. E.g.:

```
plot(residuals(m1)~fitted(m1),col="black",pch="x")
abline(h=0,lty="dotted",col="red")
```

Graphical Parameters

Some important graphical parameters:

`col`: the plotting color
`lty`: the line type
`lwd`: the line width
`pch`: the point marker
`pty`: plotting region shape
`main, sub`: title, subtitle
`new`: wipe/retain previous plot

`mfrow/mfcol`: plots/page
`ask`: hit return for next plot?
`cex`: text expansion factor
`mar/mai`: margin dimensions
`oma/omi`: outer margin dimensions
`xlim/ylim`: axis limits
`xlab/ylab`: axis labels

- Click here for a handy cheatsheet on graphical parameters.

Graphical Parameters

`col`, `lty`, `pch` can take numeric values or character strings.

- Colors 1–8 for `col` and some of the 657 color names that R knows:

```
palette() # default mapping of colors 1-8
```

```
[1] "black"   "red"     "green3"  "blue"    "cyan"    "magenta" "yellow"
[8] "gray"
```

```
cl <- colors(); length(cl)
```

```
[1] 657
```

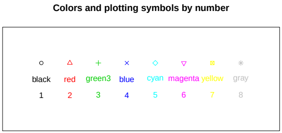
```
cl[1:12]
```

```
 [1] "white"         "aliceblue"     "antiquewhite"  "antiquewhite1"
 [5] "antiquewhite2" "antiquewhite3" "antiquewhite4" "aquamarine"
 [9] "aquamarine1"   "aquamarine2"   "aquamarine3"   "aquamarine4"
```

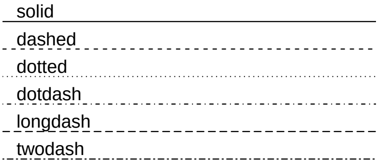
- In R, colors can also be specified with hexadecimal codes representing concentrations of red, green and blue (#rrggbb).
- See this cheatsheet for an explanation, all color names, and more on color in R.

Graphical Parameters

- Color and plotting symbol types 1–8 for `col` and `pch` and the corresponding color names:

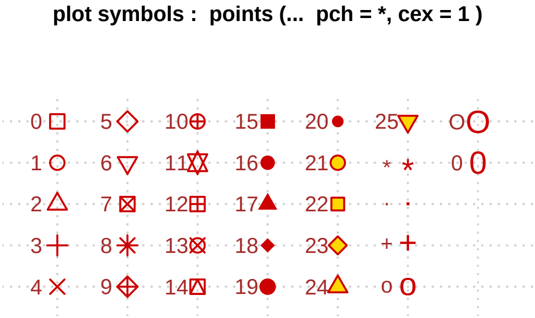


- Line types 1–8 for `lty` and the corresponding names:



Graphical Parameters

- More choices for `pch` (using `col`="red" and `bg`="gold"):



The `plot()` Function

- R is an object-oriented language and `plot()` is a generic function. That means it looks at its argument, and determines what to do with it based on its class.
  - E.g., `plot(Nile)` checks to see that `Nile` is of class `ts` (a time series object) and passes the job on to the `plot.ts()` function.
  - E.g., if `tab` is a contingency table object (class `table`) then `plot(tab)` passes the job on to the `plot.table()` function, which creates a mosaic plot.
- Thus, `plot()` can produce many different kinds of plots depending on what argument(s) you pass it.

The `plot()` Function

- R is an object-oriented language and `plot()` is a generic function. That means it looks at its argument, and determines what to do with it based on its class.
  - E.g., `plot(Nile)` checks to see that `Nile` is of class `ts` (a time series object) and passes the job on to the `plot.ts()` function.
  - E.g., if `tab` is a contingency table object (class `table`) then `plot(tab)` passes the job on to the `plot.table()` function, which creates a mosaic plot.
- Thus, `plot()` can produce many different kinds of plots depending on what argument(s) you pass it.

The `plot()` Function

- R is an object-oriented language and `plot()` is a generic function. That means it looks at its argument, and determines what to do with it based on its class.
  - E.g., `plot(Nile)` checks to see that `Nile` is of class `ts` (a time series object) and passes the job on to the `plot.ts()` function.
  - E.g., if `tab` is a contingency table object (class `table`) then `plot(tab)` passes the job on to the `plot.table()` function, which creates a mosaic plot.
- Thus, `plot()` can produce many different kinds of plots depending on what argument(s) you pass it.

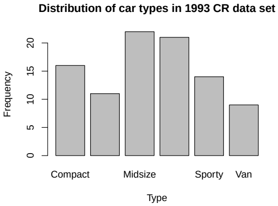
The `plot()` Function

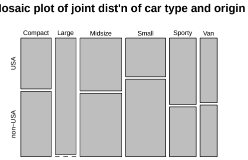
- R is an object-oriented language and `plot()` is a generic function. That means it looks at its argument, and determines what to do with it based on its class.
  - E.g., `plot(Nile)` checks to see that `Nile` is of class `ts` (a time series object) and passes the job on to the `plot.ts()` function.
  - E.g., if `tab` is a contingency table object (class `table`) then `plot(tab)` passes the job on to the `plot.table()` function, which creates a mosaic plot.
- Thus, `plot()` can produce many different kinds of plots depending on what argument(s) you pass it.

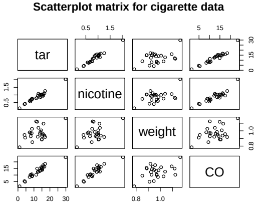
The `plot()` Function—Examples

- The generic nature of `plot()`.

```
# Get some data sets:
source("https://tinyurl.com/une4s3g/getData_3.R"); data(Cars93,package="MASS")
# plotting a factor gives a bar chart of freq distribution:
plot(Cars93$Type,main="Distribution of car types in 1993 CR data set",xlab="Type",ylab="Frequency")
# plotting a table gives a mosaic plot:
plot(table(Cars93$Type,Cars93$Origin),main="Mosaic plot of joint dist'n of car type and origin")
# plotting a data frame gives a matrix of pairwise plots (a scatterplot matrix in this case):
plot(cigData,main="Scatterplot matrix for cigarette data")
```





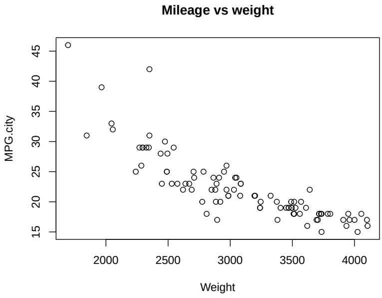


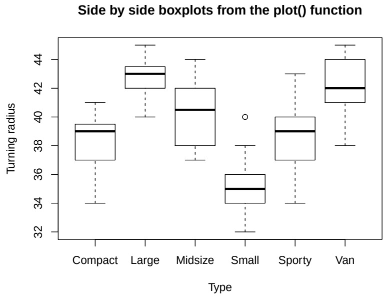
The `plot()` Function—Examples

Plotting `y` vs `x`:

- `plot(y~x,data=myDFrame)` and `plot(myDFrame$x,myDFrame$y)` produce same result.
- Gives a scatterplot if `x, y` both continuous.
- Gives side-by-side boxplots if `y` is continuous and `x` is a factor.

```
plot(MPG.city ~ Weight,data=Cars93,main="Mileage vs weight") # a scatter plot
plot(Turn.circle~Type,data=Cars93,xlab="Type",ylab="Turning radius",
     main="Side by side boxplots from the plot() function") # side-by side box plots
```

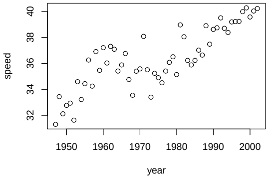


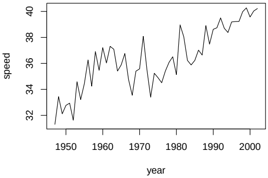


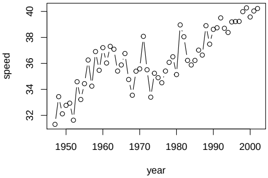
The `plot()` Function—Examples

- `plot()` takes a `type=` argument with several choices. Most important are `p`=points, `l`=lines, `b`=both (see also type `o`), or `n`=neither are plotted. Lines are useful for time series, but data should be sorted by the x-variable.
- Here, are data on the average speed of Tour de France winners over time.

```
plot(speed~year,data=tdf,type="p") # just plot points. This is the default
plot(speed~year,data=tdf,type="l") # plot lines connecting the data values.
plot(speed~year,data=tdf,type="b") # plot both points and lines.
```





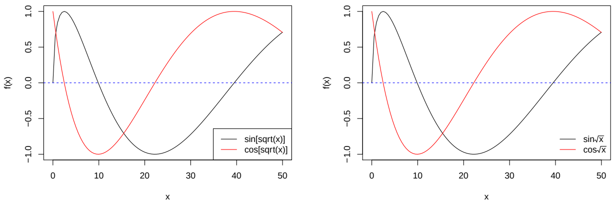


The `plot()` Function—Examples

- Plotting a function is easy with `plot()`.
- Here we also see how to add a curve, a reference line, a legend, and how to render math notation (type `?plotmath` at console for more).

```
sinRoot <- function(x){ sin(sqrt(x))}; cosRoot <- function(x){ cos(sqrt(x))}
plot(sinRoot,0,50,ylab="f(x)",xlab="x") # plots a function between (in this case) 0 and 50
curve(cosRoot,0,50,add=TRUE,col="red") ; abline(h=0,col="blue",lty="dashed")
legend("bottomright",col=c("black","red"),lty=c(1,1),legend=c("sin[sqrt(x)]","cos[sqrt(x)]"  ))

# try again with mathematical notation in legend
plot(sinRoot,0,50,ylab="f(x)",xlab="x") # plots a function between (in this case) 0 and 50
curve(cosRoot,0,50,add=TRUE,col="red") ; abline(h=0,col="blue",lty="dashed")
legend("bottomright",col=c("black","red"),lty=c(1,1),bty="n",
       legend=c(expression(plain(sin)*sqrt(x)),expression(plain(cos)*sqrt(x))))
```



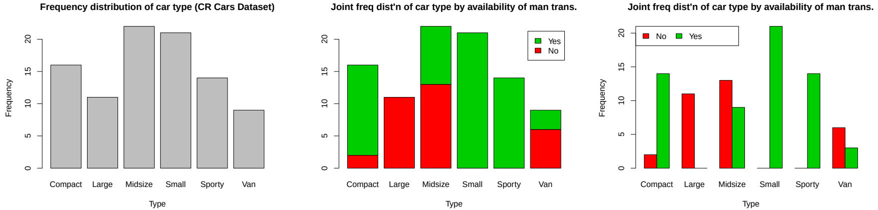
Bar charts with `barplot()`—Freq Distributions

- Bar charts are used both for plotting (joint) frequency distributions and summary statistics for multiple groups.
- `barplot()` can do both. First, frequency distributions:

```
barplot(table(Cars93$Type),ylab="Frequency",xlab="Type")
title(main="Frequency distribution of car type (CR Cars Dataset)")

# Now a two-way freq distribution with stacked bars:
carTab <- table(Cars93$Man.trans.avail,Cars93$Type)
barplot(carTab,legend.text=TRUE,col=2:3,xlab="Type")
title(main="Joint freq dist'n of car type by availability of man trans.")

# Now a two-way freq distribution with clustered bars:
barplot(carTab,legend.text=TRUE,col=2:3,xlab="Type",ylab="Frequency",
        args.legend=list(x="topleft",ncol=3),beside=TRUE)
title(main="Joint freq dist'n of car type by availability of man trans.")
```



Bar charts with barplot()—Group Statistics

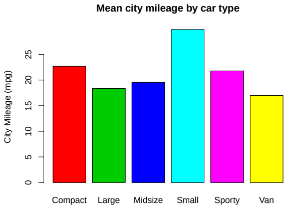
- Example: mean city mileage by car type. Error bars are hard(ish) with barplot() so use barplot2() from gplots package.

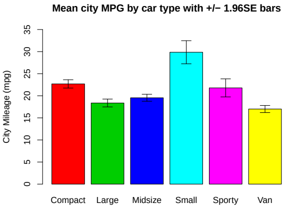
```
# First compute the means for each car Type:
(mean.arr <- tapply(Cars93$MPG.city,Cars93$Type,mean))
```

```
 Compact    Large  Midsize    Small   Sporty      Van
22.68750 18.36364 19.54545 29.85714 21.78571 17.00000
```

```
# Then plot them:
barplot(mean.arr, legend.text=FALSE, col=2:7, main="Mean city mileage by car type",
        xlab="Type", ylab="City Mileage (mpg)")

# To add error bars, must compute SEs:
se.arr <- tapply(Cars93$MPG.city,Cars93$Type,function(x) sqrt(var(x)/length(x)))
gplots::barplot2(mean.arr, legend.text=FALSE, col=2:7, ylim=c(0,35), plot.ci=TRUE, xlab="Type",
  ci.l=mean.arr-1.96*se.arr, ci.u=mean.arr+1.96*se.arr, ci.width=0.3, ylab="City Mileage (mpg)")
title(main="Mean city MPG by car type with +/- 1.96SE bars")
```



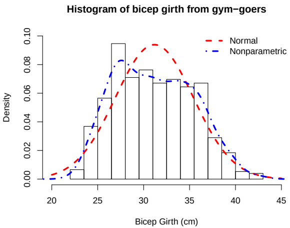


Histograms with hist()

- Histograms can be plotted with `hist()`. Binning scheme matters (a lot) and trial and error is necessary.
- Use density scale rather than counts to compare to a fitted density. Here we overlay a normal and a kernel density estimate.

```
hist(bodyData$bicep_girth,xlab="Bicep Girth (cm)", main="Histogram of bicep girth from gym-goers")

# Now use different bins, switch to a probability density scale, and overlay densities
hist(bodyData$bicep_girth,xlab="Bicep Girth (cm)",breaks=seq(from=22,to=43,by=1.5),
     main="Histogram of bicep girth from gym-goers",freq=FALSE,ylim=c(0,.1),xlim=c(20,45))
curve(dnorm(x, mean=mean(bodyData$bicep_girth),sd=sd(bodyData$bicep_girth)),
      col = 2, lty = 2, lwd = 3, add = TRUE)
lines(density(bodyData$bicep_girth), col = 4, lty = 4, lwd = 3)
legend("topright",lty=c(2,4),col=c(2,4),lwd=3,legend=c("Normal","Nonparametric"),bty="n")
```

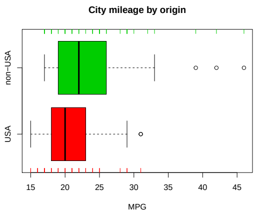


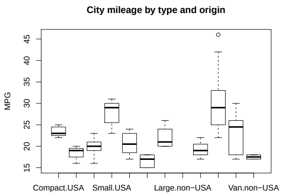
Boxplots with `boxplot()`

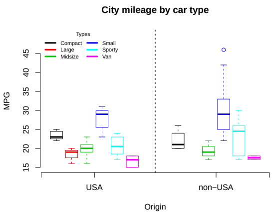
- Formula syntax in `boxplot()` is convenient for grouped boxplots, but labels can get crowded.
- `boxplot(y ~ f)` for boxplots of `y` at each level of `f` or `boxplot(y ~ f + g)` at each combination of `f` and `g`.

```r
boxplot(MPG.city~Origin, data=Cars93, main="City mileage by origin", xlab="MPG", horizontal=T, col=2:3)
rug(Cars93$MPG.city[Cars93$Origin=="USA"],side=1,col=2); rug(Cars93$MPG.city[Cars93$Origin!="USA"],side=3,col=3)
boxplot(MPG.city~Type+Origin, data=Cars93, main="City mileage by type and origin", ylab="MPG")

# fixing crowding of x-axis labels can be challenging
boxplot(MPG.city~Type+Origin,data=Cars93, main="City mileage by car type",
        ylab="MPG", border=1:6, pars=list(axes=F, ylim=c(15,50)), xlab="Origin", at=c(1:6,9:14))
abline(v=7.5,lty=2); axis(2,at=seq(from=15,to=45,by=5))
axis(1,at=c(0,3.5,11.5,15), labels=c("",levels(Cars93$Origin),""))
legend("topleft", bty="n", col=1:6, legend=levels(Cars93$Type), cex=.70, ncol=2, lty=rep(1,6), lwd=3, title="Types")
```



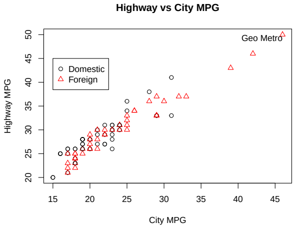


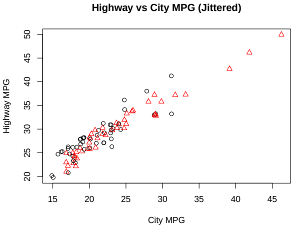


Labelling Points

- Groups can be distinguished with graphical parameters. E.g., give `col` or `pch` a vector of values of same length as the data.
- Labels for points can be done with `text()` function.

```
plot(MPG.highway~MPG.city, data=Cars93, pch= as.integer(Origin), col=as.integer(Origin),
     xlab="City MPG",ylab="Highway MPG",main="Highway vs City MPG")
with(Cars93,text(x=MPG.city[MPG.city>45], y=MPG.highway[MPG.city>45], labels=Make[MPG.city>45], adj=c(1,1))) # adj offsets labels
legend(x=15, y=45, legend=c("Domestic","Foreign"), pch=1:2, col=1:2)
# Many values overplotted. Better to jitter the points:
plot(jitter(MPG.highway,1.5) ~ jitter(MPG.city,1.5), data=Cars93, pch= as.integer(Origin), col=as.integer(Origin),
     xlab="City MPG",ylab="Highway MPG",main="Highway vs City MPG (Jittered)")
```





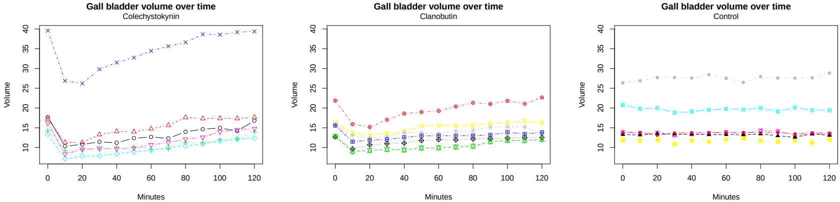
Profile Plots for Longitudinal Data

- Longitudinal studies involve data over time for multiple individuals, possibly in different (treatment) groups.
- Task is much better handled by functions in `lattice` or `ggplot2`.

```
gall$trtfac <- factor(gall$trt,labels=c("Colechystokynin","Clanobutin","Control")); head(gall,3)
```

```
  trt dogno min volume          trtfac
1   1     1   0  17.70 Colechystokynin
2   1     1  10  10.35 Colechystokynin
3   1     1  20  10.78 Colechystokynin
```

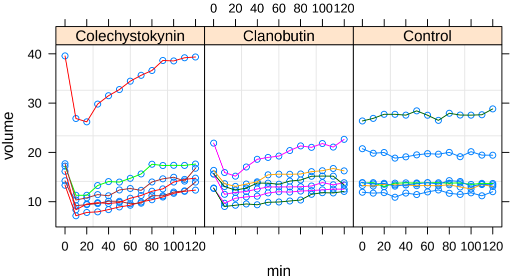
```
for (i in levels(gall$trtfac)){
  galli <- gall[gall$trtfac==i,]
  plot(volume ~ min, data=gall, type="n", xlab="Minutes", ylab="Volume", main="Gall bladder volume over time")
  mtext(i,side=3,line=0.5)
  for (j in unique(galli$dogno)){
    lines(galli$min[galli$dogno==j], galli$volume[galli$dogno==j], lty=j, pch=j, col=j, type="b") }}
```



Profile Plots for Longitudinal Data

- The `nlme` package implements profile plots via the `plot()` function applied to a `groupedData` object.
- The work is done by the `xyplot()` function of the `lattice` package.
- Later we'll see how to use `ggplot2` for this task.

```
library(nlme)
gall2 <- groupedData(volume~min|dogno,data=gall) # a data frame with a formula attached
plot(gall2,outer= ~trtfac,aspect="fill",key=FALSE)
```



Adding Fitted Lines/Curves to a Plot

- Here we add a linear fit with 95% CI and PI and a lowess fitted curve.
- Important to add features in the proper order to avoid overplotting.

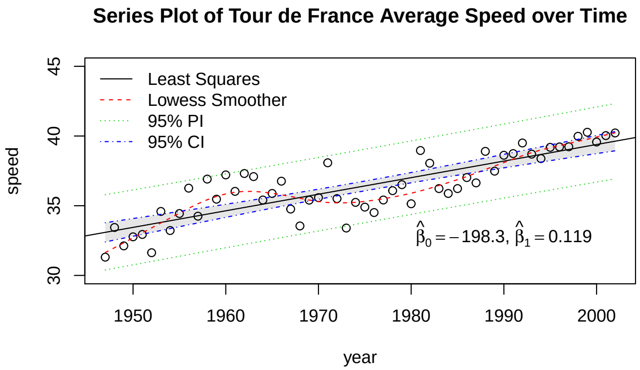
```
# Create a plot of the data but omit the points initially by setting type="n":
plot(speed~year, data=tdf, type="n", ylim=c(30,45), main="Series Plot of Tour de France Average Speed over Time")

# Fit a simple linear regression to get least squares fitted line, and 95% CI and PI limits
tdf.lm1 <- lm(speed~year,data=tdf)
tdf.lm1.pi <- predict(tdf.lm1, interval="prediction")
tdf.lm1.ci <- predict(tdf.lm1, interval="confidence")

# Now add a 95% prediction interval with a shaded region to the plot:
polygon(c(tdf$year,rev(tdf$year)), c(tdf.lm1.ci[,2],rev(tdf.lm1.ci[,3])), border=NA, col=gray(.9))
# Then add other features:
lines(speed ~ year, data=tdf, type="p") ; abline(tdf.lm1) # adds points and a straight line fit
lines(lowess(tdf$year,tdf$speed, f=1/3), col=2, lty=2)    # adds a lowess fit
lines(tdf$year, tdf.lm1.pi[,2], lty=3, col=3); lines(tdf$year, tdf.lm1.pi[,3], lty=3, col=3) #PI limits
lines(tdf$year, tdf.lm1.ci[,2], lty=4, col=4); lines(tdf$year, tdf.lm1.ci[,3], lty=4, col=4) #CI limits
legend("topleft", c("Least Squares","Lowess Smoother","95% PI","95% CI"), col=1:4, lty=1:4, bty="n") # a legend
text(1990,33,expression(paste(hat(beta)[0]==-198.3,", ", hat(beta)[1]==0.119))) # parm estimates
```

Adding Fitted Lines/Curves to a Plot

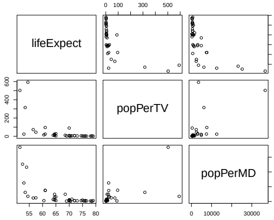
- Results from the code on the previous slide:

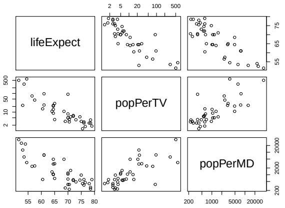


Scatterplot Matrices

- The `pairs()` function produces scatterplot matrices. It is illustrated below on some country-level data on life expectancy, access to healthcare, and access to technology.

```
# basic illustration of pairs():
pairs(~lifeExpect+popPerTV+popPerMD,data=tvData)
# log scaling of axes:
pairs(~lifeExpect+popPerTV+popPerMD,data=tvData,log=2:3,
      main="Log scale for popPerTV and popPerMD")
```

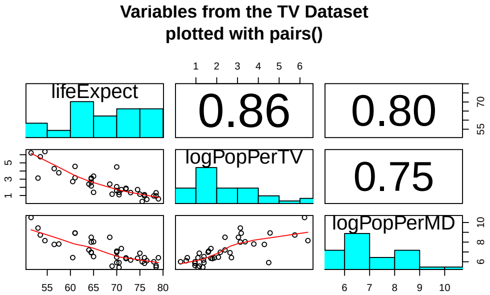




Scatterplot Matrices

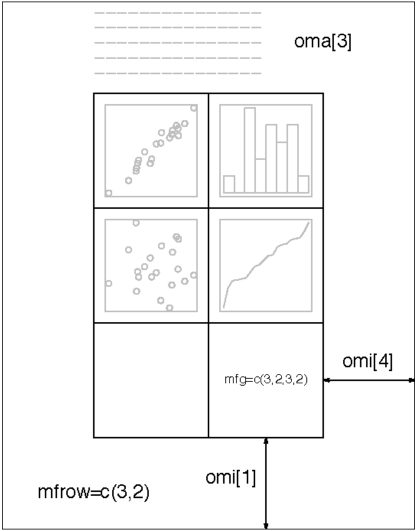
- Can make better use of the space in a scatterplot matrix by using the diagonal cells and the upper or lower triangle to display other information.
- Can be done with `lower.panel`, `upper.panel`, `diag.panel` arguments.
- The `panel.cor()` and `panel.hist()` functions used below are from the `pairs()` help page; `panel.smooth()` is built-in.

```
tvData$logPopPerTV <- log(tvData$popPerTV); tvData$logPopPerMD <- log(tvData$popPerMD)
pairs(~lifeExpect+logPopPerTV+logPopPerMD, data=na.omit(tvData),
      lower.panel=panel.smooth, upper.panel=panel.cor, diag.panel=panel.hist,
      main="Variables from the TV Dataset\nplotted with pairs()")
```



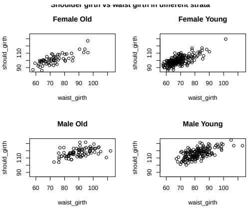
Multiple Plots per Page

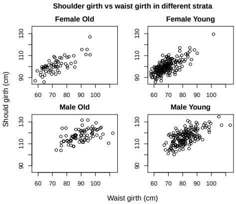
- Placing multiple plots in an $R \times C$ grid on a page can be done with `par(mfrow=c(R,C))` (see also `mfcol`).
- The page layout is displayed to the right. For multiple plots/page, usually must adjust the margins (`mar`) and outer margins (`oma`). This can be tricky.



Multiple Plots per Page

- This example using `mfrow` shows the need to adjust the margins. The plot on the top is run without the adjustments to `mar` and `oma` on the third line of code.






```
bodyData$over34 <- factor(bodyData$age>34, labels=c("Young","Old"))
bodyData$sexAge <- factor(paste(bodyData$genderFac,bodyData$over34))
op <- par(mfrow=c(2,2),mar=c(2,2,2,2)+.1,oma=c(3,3,1.5,1))
for (lev in levels(bodyData$sexAge)){
  plot(should_girth~waist_girth,data=bodyData,type="n",main=lev)
  points(should_girth~waist_girth,data=bodyData[bodyData$sexAge==lev,])
}
title(main="Shoulder girth vs waist girth in different strata",outer=TRUE)
mtext("Should girth (cm)",side=2,outer=TRUE,line=1);
mtext("Waist girth (cm)",side=1,outer=TRUE,line=1)
par(op)
```

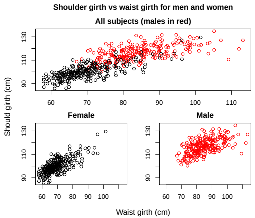
Multiple Plots per Page

- A more flexible arrangement is possible with the command `layout(mat)` where `mat` is a matrix specifying the desired arrangement.

```
(layoutMat <- rbind(c(1,1),c(2,3)))
```

```
     [,1] [,2]
[1,]    1    1
[2,]    2    3
```

```
layout(layoutMat); op <- par(mar=c(2,2,2,2)+.1,oma=c(3,3,1.5,1))
plot(should_girth~waist_girth,data=bodyData,main="All subjects (males in red)",
     col=as.integer(genderFac))
for (lev in levels(bodyData$genderFac)){
  plot(should_girth~waist_girth,data=bodyData,type="n",main=lev)
  points(should_girth~waist_girth,data=bodyData[bodyData$genderFac==lev,],
         col=as.integer(genderFac)) }
title(main="Shoulder girth vs waist girth for men and women",outer=TRUE)
mtext("Should girth (cm)",side=2,outer=TRUE,line=1)
mtext("Waist girth (cm)",side=1,outer=TRUE,line=1); par(op)
```



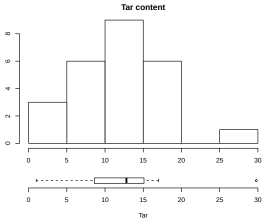
Multiple Plots per Page

- Another example. Here the top plot takes 75% of the page.

```
(layoutMat <- rbind(matrix(1,nrow=3,ncol=2),c(2,2)))
```

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

```
layout(layoutMat)
op <- par(mar=c(2,2,2,2)+.1,oma=c(1,1,1,1))
hist(cigData$tar,main="Tar content",xlim=c(0,30))
par(mar=c(4,2,1,2)+.1)
boxplot(cigData$tar,horizontal=TRUE,xlab="Tar",pars=list(bty="l"),
        frame=FALSE,ylim=c(0,30)) # ylim becomes xlim when horiz=T
par(op)
```



Resources for Graphics in R

Friendly, M. (2018). Data Visualization in R, SCS Short Course. http://www.datavis.ca/courses/RGraphics/

- Fantastic resource. Session 2 slides focus on Base Graphics. Session 1 slides point the way to many more important resources.

Tierney, L. (2019). STAT:4580 Data Visualizations and Data Technology. Course Notes.

- Another great resource on data visualization methods and the tools to implement them. Much content on R graphics systems, especially `ggplot2`.

RStudio. Data Visualization with ggplot2:: Cheat Sheet. (All RStudio cheat sheets in a single PDF at this link.)

Thank You!

- If you need assistance with R or with selecting or implementing data visualizations to better understand your data, contact the SCC!
- We can help!

www.stat.uga/consulting

Finally...

- Holiday wishes, shamelessly stolen from the is.R() tumblr site:

```
library(ggplot2)
Turkey <- read.csv("http://pages.iu.edu/~cdesante/turkey.csv")
ggplot(data = Turkey) + geom_tile(aes(x = Happy, y = Thanksgiving, fill=Turkey.Colors,
width=1))+ scale_fill_identity() + theme_bw()
```