Data Analysis

Sergey Vladimirovich Petropavlovsky

National Research University Higher School of Economics
Master's Program "Big Data Systems"

Fall 2018

Goals and Topics

Major goal – review of basic statistics and data analysis techniques.

- Descriptive statistics (= summarizing and visualizing data).
- Inferential statistics:
  - Confidence intervals
  - Hypothesis testing
  - Simple linear regression
  - Multivariate regression
  - Non-linear regression (?)
- Principal component analysis and extensions:
  - Correspondence analysis (CA), multiple CA, multidimensional scaling (MDS) etc
- Cluster analysis (hierarchical, K-means, probabilistic)
- Factor analysis
- (?) ...depends on the remaining time

Grading

- Formula for the final grade $O_{\text{fin}} \in [0, 10]$:

$$O_{\text{fin}} = 0.7 \times O_{\text{accm}} + 0.3 \times O_{\text{exam}} \tag{1}$$

- $O_{\text{exam}} \in [0, 10]$ is the grade for the final exam
- $O_{\text{accm}} \in [0, 10]$ is the grade accumulated over the module:

$$O_{\text{accm}} = 0.6 \times O_{\text{HA}} + 0.4 \times O_{\text{PE}}$$

- $O_{\text{HA}} \in [0, 10]$ is the grade for the home assignments
- $O_{\text{PE}} \in [0, 10]$ is the grade for the pre-exam test
- $10 \to 5$ point grading system:

O_{fin}, 10-point system	8-10	6-7	4-5	0-3
O_{fin}, regular 5-point system	5	4	3	2 (Fail)

Grading: waiving the final exam

- If $O_{\text{accm}} \geq 7$, the final exam can be waived. In this case,

$$O_{\text{fin}} = O_{\text{accm}}$$

- If you want a higher mark (e.g, $O_{\text{accm}} = 7 = 4$), you need to take the final exam. The overall grade is then given by the regular formula (1)

Home assignments

- Account for 60% of O_{accm}.
- Take a form of a report
- Based on the topics we discuss in class. Just repeat the steps we have made together with your own data sets
- There are some requirements for the reports, e.g.,
  - The working language is English
  - All your conclusions MUST be justified numerically, i.e., by some computed quantities, plots, etc.
  - Each student MUST use a unique data set. A check-in list is on my Google Drive. An invitation to edit that list will be sent to your **group** e-mail address.
- The reports should be submitted to my e-mail address before or on the due date
- Late submission: 25% off **for each weak** after the due date.

Pre-exam

- Accounts for 40% of O_{accm}.
- Taken on the last week before the final exam.
- Contains a few base practice tasks. The tasks are fairly easy for those who have been working throughout the module.

Structure of classes

- First, a theoretical introduction into the subject (“lecture”). It can take more or less time depending on the material.
- Second, we work out the major tasks of the home assignment together demonstrating the process on the screen.
- Third, you are encouraged to start doing your own report.

Textbooks

Unfortunately, there is no sole universal textbook that would suit our needs. Therefore, multiple texts:

- J. Verzani, Using R for Introductory Statistics, Second Edition, Chapman & Hall/CRC The R Series, Taylor & Francis, 2014. URL https://books.google.ru/books?id=O86uAwAAQBAJ
- B. Everitt, T. Hothorn, An introduction to applied multivariate analysis with R, Springer, New York, 2011. URL http://dx.doi.org/10.1007/978-1-4419-9650-3
- F. Husson, S. Le, J. Pages, Exploratory Multivariate Analysis by Example Using R, Second Edition, Chapman & Hall/CRC Computer Science & Data Analysis, CRC Press, 2017. URL https://books.google.com/books?id=nLrODgAAQBAJ.

Software: R. Why?

- R has become popular in data analysis
- Open source, simple to use, numerous packages contributed by a large community of users on any aspect of statistical modeling and data analysis
- If you can have an excellent free product why should you pay for an excellent expensive product (e.g., Matlab, SAS)?
- R is is a collection of statistical routines rather than a programming language.

Getting started with R

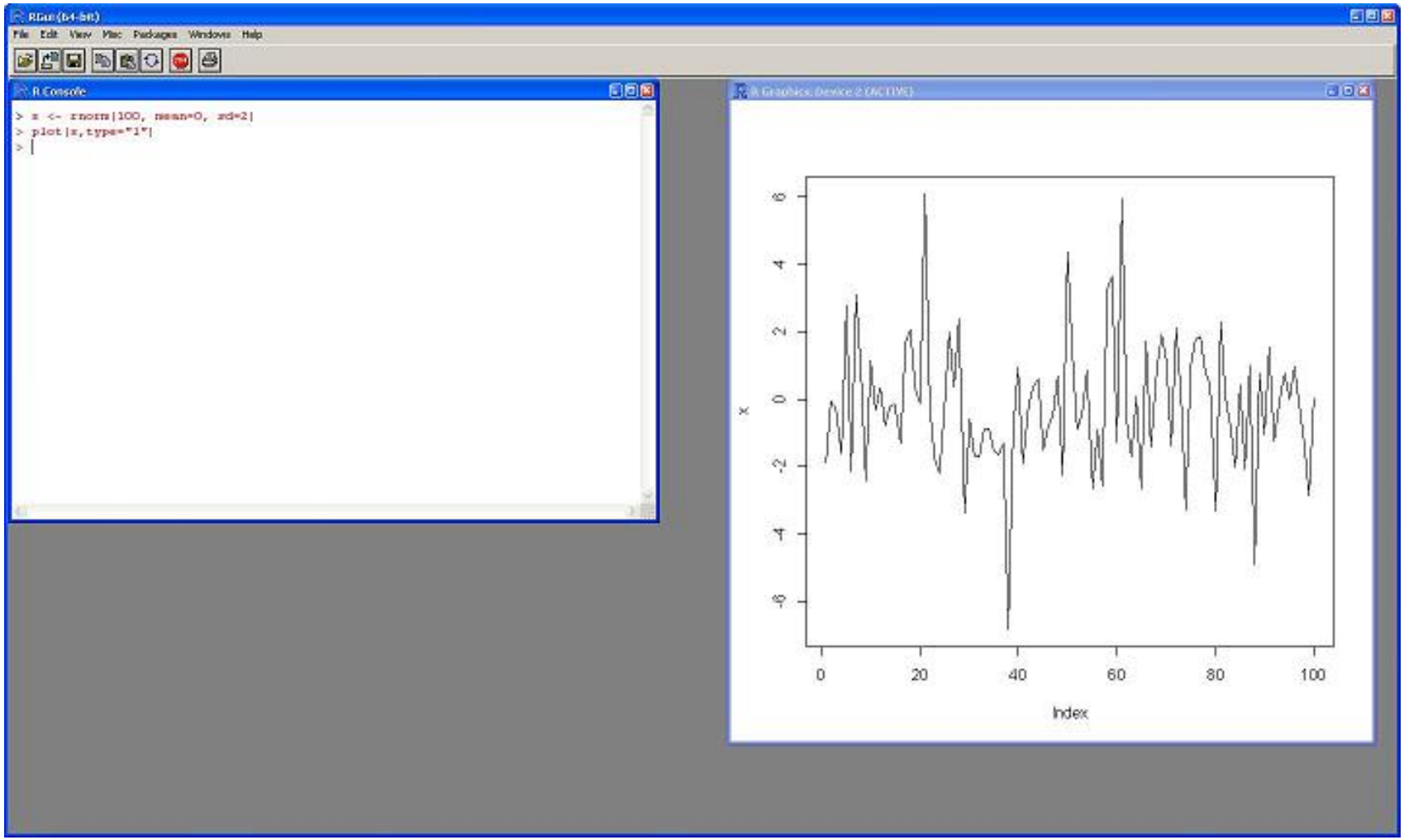


Figure 1: R for Windows

RStudio shell

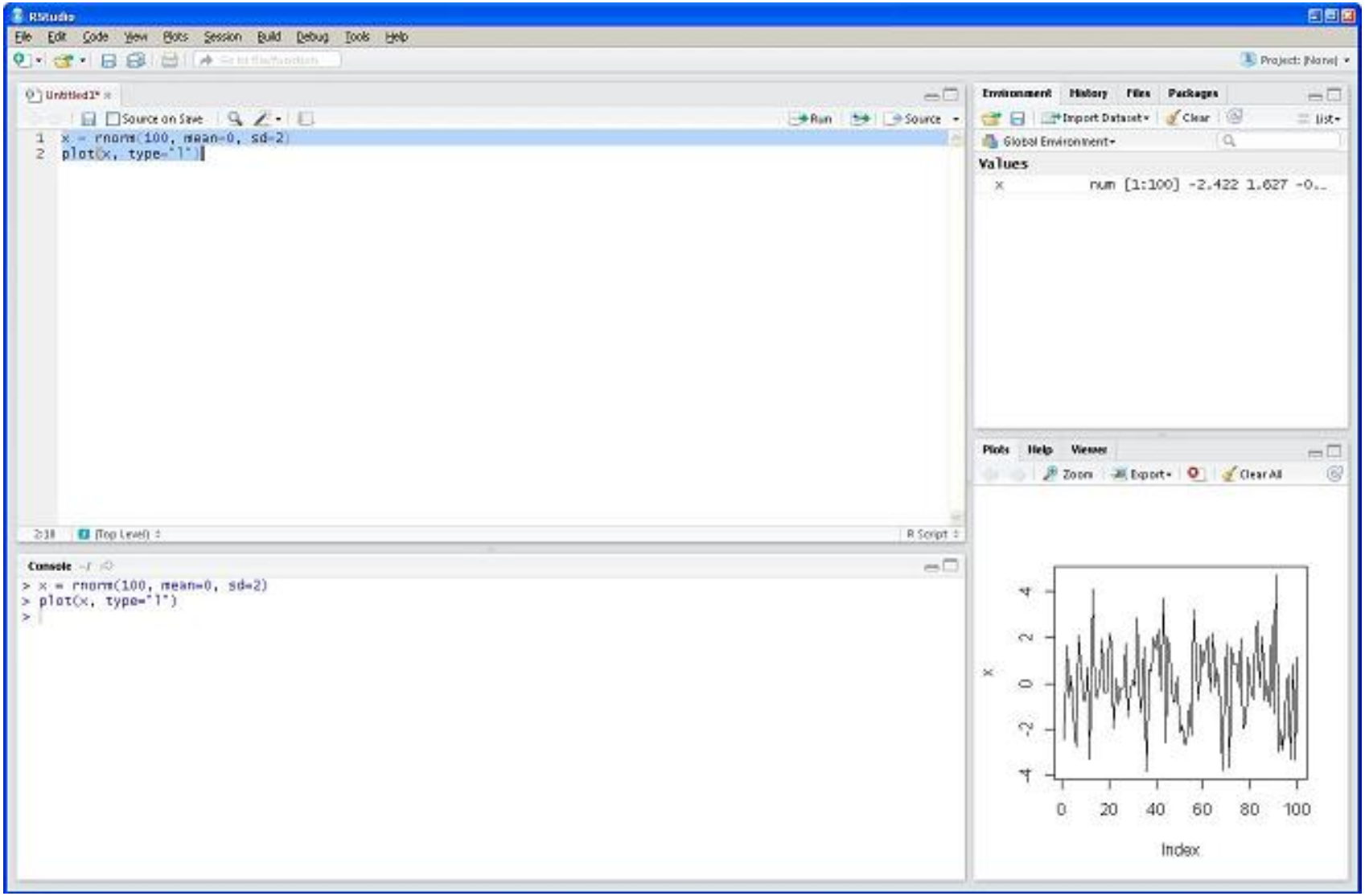


Figure 2: Rstudio

Data objects in `R`

- `R` creates and works with objects that contain data
- The object/data can be of a different structure such as:
  - **data frame**: a table where each column represents a variable and each row a different observation (different time period or unit); a variable can be numerical or a string
  - **matrix**: same as data frame, but all variables/columns have to be of the same type (typically all numbers)
  - **lists**: an object of objects; each element of the list can be, e.g., a data frame, a matrix, and a vector
- **Function**: a set of operations applied to an object and with a set of arguments; e.g., `mean(x, na.rm=T)`
- **Package**: a group of functions with a specific purpose (e.g., `ggplot2`)
- install a package: `install.packages("ggplot2")` (done only once)
- use the package: `library(ggplot2)` or `require(ggplot2)`

Loading data in R

- It is convenient to start a R session by setting the working directory where the data/files are stored, e.g,:
  `setwd('/Users/username/someone/somewhere/')` in Mac/Unix
  `setwd('c:/someone/somewhere/')` in Windows
- Two ways to load a dataset in R:
  - import the data from a local file
  - import the data from an online resource (e.g., Yahoo Finance, FRED, Google Finance, Quandl)

Base function `read.csv()`

- You can load a file from Rstudio via Tools -> Import Dataset and then you are given the option From Text File or From Web URL
- Otherwise, you can type a few lines of code:

```
> splist <- read.csv("List_SP500.csv")
> head(splist,10)
```

	Ticker.symbol	Security	Address.of.Headquarters	Date.first.added
1	MMM	3M Company	St. Paul, Minnesota	
2	ABT	Abbott Laboratories	North Chicago, Illinois	1964-03-31
3	ABBV	AbbVie	North Chicago, Illinois	2012-12-31
4	ACN	Accenture plc	Dublin, Ireland	2011-07-06
5	ATVI	Activision Blizzard	Santa Monica, California	2015-08-31
6	AYI	Acuity Brands Inc	Atlanta, Georgia	2016-05-03
7	ADBE	Adobe Systems Inc	San Jose, California	1997-05-05
8	AAP	Advance Auto Parts	Roanoke, Virginia	2015-07-09
9	AES	AES Corp	Arlington, Virginia	
10	AET	Aetna Inc	Hartford, Connecticut	1976-06-30

- The commands `head( ,n)` and `tail( ,n)` show the first and last n observations.

Type of data: command `str()`

- The `str()` command can be used to evaluate the object structure and the data types:

```
> str(splist)
'data.frame':   505 obs. of  4 variables:
 $ Ticker.symbol          : Factor w/ 505 levels "A","AAL","AAP",..: 308 7 5 8 46 52 9 3 17 18 ...
 $ Security               : Factor w/ 504 levels "3M Company","Abbott Laboratories",..: 1 2 3 4 5 6 7 8 9
 $ Address.of.Headquarters: Factor w/ 264 levels "Akron, Ohio",..: 228 164 164 65 215 8 209 199 5 96 ...
 $ Date.first.added       : Factor w/ 253 levels "","1964-03-31",..: 1 2 177 159 219 240 79 216 1 18 ...
```

- Each variable in the data frame `splist` has a type that can be:
  - numeric: for decimal values
  - integer: for integer values
  - character: for strings of characters
  - Date: for dates
  - factor: represents a type of variable (either numeric, integer, or character) that categorizes the values in a small (relative to the sample size) set of categories (or levels)

Base function `read.csv()` revisited

- The `read.csv()` has the annoying feature that any string is interpreted as a factor
- This can be switched off by adding the argument `stringsAsFactors = FALSE`

```
> splist <- read.csv("List_SP500.csv", stringsAsFactors =
FALSE)
> str(splist)
```

```
'data.frame':   505 obs. of  4 variables:
 $ Ticker.symbol          : chr  "MMM" "ABT" "ABBV" "ACN" ...
 $ Security               : chr  "3M Company" "Abbott Laboratories" "AbbVie" "Accenture plc" ...
 $ Address.of.Headquarters: chr  "St. Paul, Minnesota" "North Chicago, Illinois" "North Chicago, Illinois
 $ Date.first.added       : chr  "" "1964-03-31" "2012-12-31" "2011-07-06" ...
```

- The ticker symbol, security name, and address are all correctly interpreted as `chr`
- The `date.first.added` is also imported as a string but we would like to define it of type `Date`

Converting to `Date`

- The code below is used to define the column/variable `Date.first.added` as a date with command `as.Date()`

```
> splist$Date.first.added <- as.Date(splist$Date.first.added,
format="%Y-%m-%d")
> str(splist)
 'data.frame':   505 obs. of  4 variables:
  $ Ticker.symbol          : chr  "MMM" "ABT" "ABBV" "ACN" ...
  $ Security               : chr  "3M Company" "Abbott Laboratories" "AbbVie" "Accenture plc" ...
  $ Address.of.Headquarters: chr  "St. Paul, Minnesota" "North Chicago, Illinois" "North Chicago, Illinois
  $ Date.first.added       : Date, format: NA "1964-03-31" "2012-12-31" "2011-07-06" ...
```

- `$` sign is used to extract a variable/column in a data frame; `splist$Date.first.added` extract the `Date.first.added` from the data frame object `splist`
- `as.Date()` function converts the `splist$Date.first.added` from character to Date; the role of the argument `format="%Y-%m-%d"` is to specify the format of the date being defined

Saving data files

- The base function `write.csv()` (see `help(write.csv)` for the arguments)

```
index <- read_csv("GSPC.csv")
```

```
> write.csv(index, file = "myfile.csv", row.names = FALSE)
```

- The index object is saved to a file called `myfile.csv` in the working directory

Plotting the data

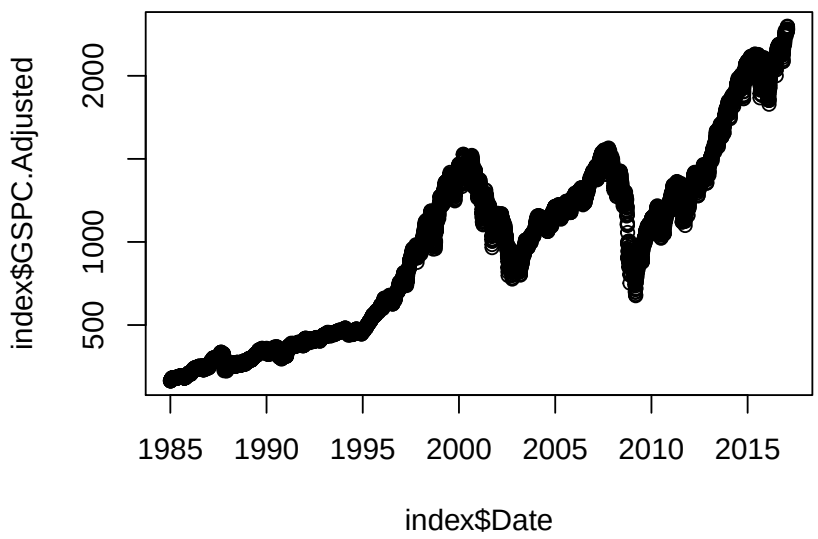
- Visualization is an essential part of data analysis
- It is useful to guide the analysis and to communicate our results
- It is typically easier to process and understand a graph rather than a table with numbers
- The base function in `R` to plot data is `plot()` that takes as arguments:
  - `x,y`: the variables to plot in the x-axis and y-axis
  - `type`: p for points, l for lines, b for both (and more)
  - `xlim,ylim`: the range of the axes
  - `xlab,ylab`: the labels of the axes
  - `main`: string to use for title
  - `col`: color of the point and/or line
  - `pch`: type of point to use

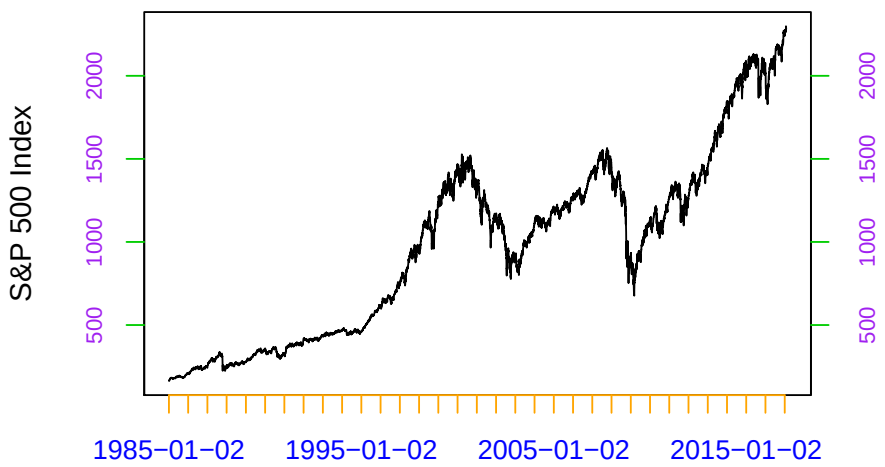
Plotting the data (2)

- The code below produces a time series plot of the S&P 500 Index:
  - The column `index$Date` is defined of class `Date` and used as the x-axis
  - The column `index$GSPC.Adjusted` is used as the y-axis
  - The left plot uses the default settings, the plot on the right has been customized
  - The command `par(mfrow=c(1,2))` is used to plot the two graphs in 1 row and 2 columns.

```
> par(mfrow=c(1,2))
> plot(index$Date, index$GSPC.Adjusted)
> plot(index$Date, index$GSPC.Adjusted, type="l", xlab="", ylab="S&P
500 Index", xaxt="n", yaxt="n")
> ticks <- seq(index$Date[1], index$Date[nrow(index)], by="year")
> axis(1, at=ticks, labels=ticks, cex.axis=0.9, col="orange",
col.axis="blue")
> axis(2, at=seq(0, 2000, 500), labels=seq(0,2000,500),
col.ticks=3,cex.axis=0.75,col.axis="purple")
> axis(4, at=seq(0, 2000, 500), labels=seq(0,2000,500), col.ticks=3,
cex.axis=0.75,col.axis="purple")
```

Plotting the data (3): output





Time series objects

- Variables that are observed over time are called time series data (e.g., stock prices, real GDP, inflation)
- There are several packages that provide an infrastructure to define an object as a time series object
- We will mostly use the `xts` package (that is part of the `quantmod` package for quantitative finance; other packages are `ts` and `zoo`)
- To define an object as a time series we use the command `xts()` that takes two arguments:
  - a data frame
  - a vector of dates (of class `Date`)

```
library(xts)
index.xts <- xts(subset(index, select=-Date),
order.by=index$Date)
```

Time series objects (2)

```
> library(xts)
> index.xts <- xts(subset(index, select=-Date),
order.by=index$Date)
```

	GSPC.Open	GSPC.High	GSPC.Low	GSPC.Close	GSPC.Volume	GSPC.Adjusted
1985-01-02	167.20	167.20	165.19	165.37	67820000	165.37
1985-01-03	165.37	166.11	164.38	164.57	88880000	164.57
1985-01-04	164.55	164.55	163.36	163.68	77480000	163.68
1985-01-07	163.68	164.71	163.68	164.24	86190000	164.24

- The xts package provides functions to extract time series information from the object:

```
> start(index.xts) # start date
> end(index.xts) # end date
> periodicity(index.xts) # periodicity/frequency (daily,
weekly, monthly)
```

```
[1] "1985-01-02"
[1] "2017-01-30"
Daily periodicity from 1985-01-02 to 2017-01-30
```

Time series objects (3)

- There are also functions to aggregate the observations from high frequency (e.g., daily) to lower frequency (e.g., weekly/monthly/quarterly)
- By default the sub-sampling is performed by taking the first observation of the interval (e.g., Monday of each week, 1st of the month)

```
index.weekly <- to.weekly(index.xts)
```

```
           index.xts.Open index.xts.High index.xts.Low index.xts.Close index.xts.Volume index.xts.Adjuste
1985-01-04         167.20         167.20        163.36          163.68        234180000             163.6
1985-01-11         163.68         168.72        163.68          167.91        509830000             167.9
1985-01-18         167.91         171.94        167.58          171.32        634000000             171.3
1985-01-25         171.32         178.16        171.31          177.35        749100000             177.3
```

Time series objects (4)

- Functions `apply.weekly()` and `apply.monthly()` are used when the goal is to apply a function to each week/month in the sample
- In the examples below these functions are applied to subsample the `first()` and `last()` day of the week (notice that the `first()` is equivalent to the `to.weekly()` function)

```
index.weekly <- apply.weekly(index.xts, "first")
```

```
           GSPC.Open GSPC.High GSPC.Low GSPC.Close GSPC.Volume GSPC.Adjusted
1985-01-04    167.20    167.20   165.19     165.37    67820000        165.37
1985-01-11    163.68    164.71   163.68     164.24    86190000        164.24
1985-01-18    167.91    170.55   167.58     170.51   124900000        170.51
```

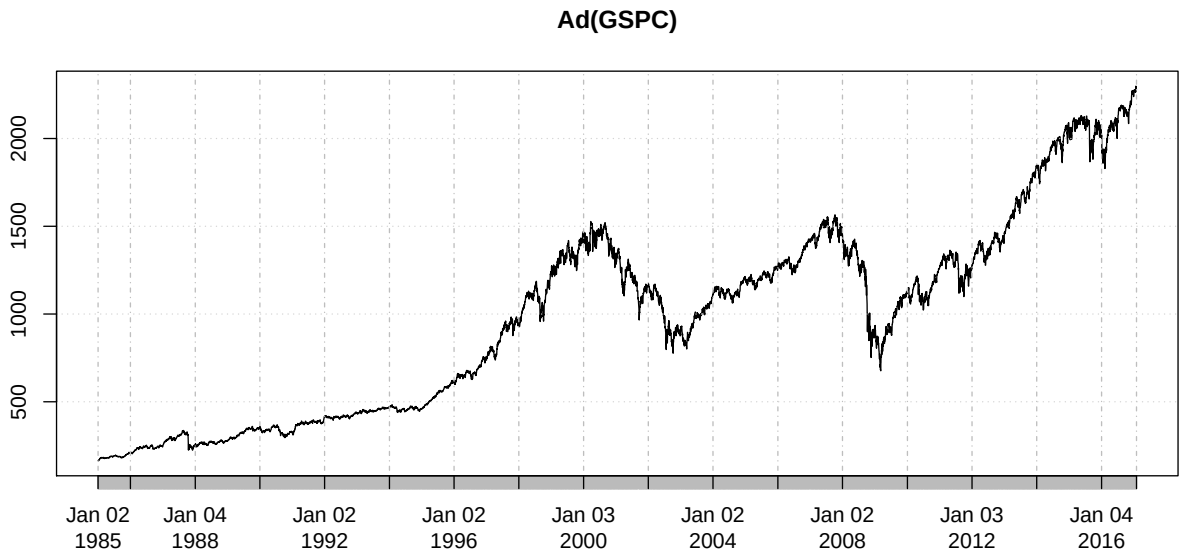
```
index.weekly <- apply.weekly(index.xts, "last")
```

```
           GSPC.Open GSPC.High GSPC.Low GSPC.Close GSPC.Volume GSPC.Adjusted
1985-01-04    164.55    164.55   163.36     163.68    77480000        163.68
1985-01-11    168.31    168.72   167.58     167.91   107600000        167.91
1985-01-18    170.73    171.42   170.66     171.32   104700000        171.32
```

Plotting time series data

- Once the object is defined as `xts`, plotting is done by `plot.xts()` and:
  - No need to specify the x-axis since for a time series the default x-axis is time
  - the graphics is improved relative to the base plot, with grids and careful rendering of the dates
  - Using `plot()` on an `xts` object actually calls `plot.xts()`

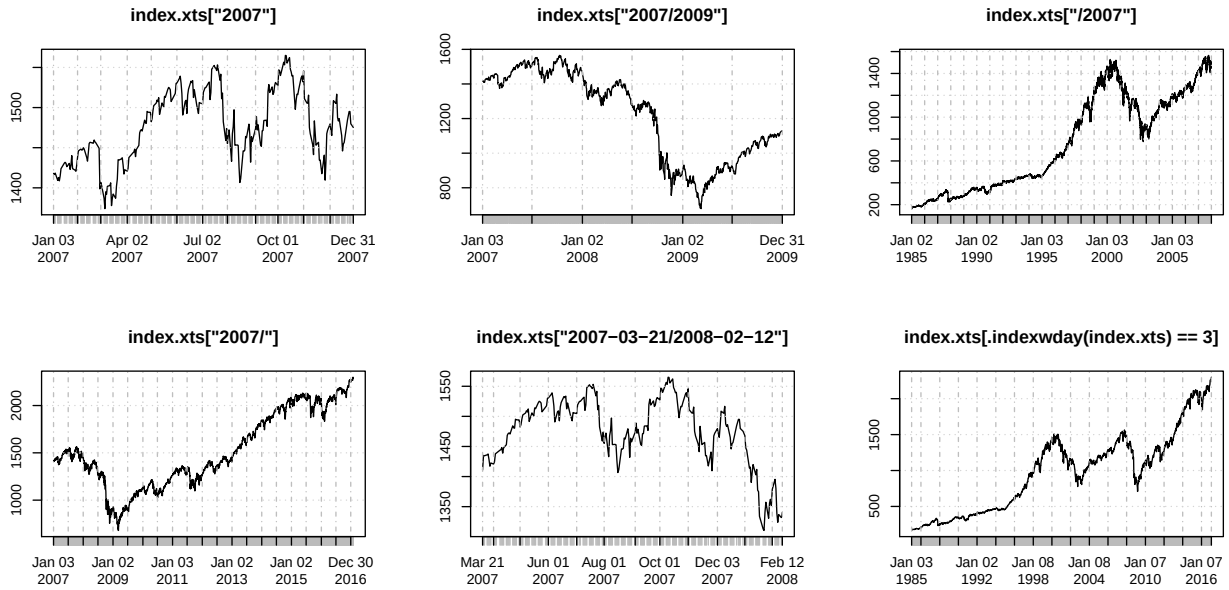
```
> GSPC <- getSymbols("GSPC", from="1985-01-01",
auto.assign=FALSE)
> plot(Ad(GSPC))
```



More on `xts`

- The `xts` package provides its own syntax to subset the time series object
- Below are some examples:

```
> par(mfrow=c(2,3)) # organize the plots in 2 rows and 3 columns
> plot(index.xts['2007'])
> plot(index.xts['2007/2009'])
> plot(index.xts['/2007'])
> plot(index.xts['2007/'])
> plot(index.xts['2007-03-21/2008-02-12'])
> plot(index.xts[.indexwday(index.xts)==3])
```



Reading data from the online databases

- `R` can directly download data files from online sources
- The code below shows how to download a dataset from Yahoo Finance by specifying the url (S&P index at the daily frequency from March 1950 until today)
- The url is used in the `read.csv()` function that reads the file and assigns it to the object data

```
> url <-
'http://chart.finance.yahoo.com/table.csv=GSPC&a=0&b=3&c=1950&d=11&e=31&f=
> data <- read.csv(url)
> head(data, 3)
```

	Date	Open	High	Low	Close	Volume	Adj.Close
1	2016-12-01	2200.2	2277.5	2187.4	2238.8	3710578000	2238.8
2	2016-11-01	2128.7	2214.1	2083.8	2198.8	4468273300	2198.8
3	2016-10-03	2164.3	2169.6	2114.7	2126.1	3672334700	2126.1

Reading data from the online databases (2)

- The data are not sorted from the oldest to newest date
- If the object is defined as `xts` the time-ordering will happen automatically
- In the `order.by` argument we provide the `Date` variable transformed to type `Date` using the `as.Date()` command

```
data.xts <- xts(data[,-1], order.by =as.Date(data$Date))
head(data.xts, 3)
```

```
            Open  High   Low Close  Volume Adj.Close
1950-01-03 16.66 17.09 16.66 17.05 1926600     17.05
1950-02-01 17.05 17.32 16.99 17.22 1750500     17.22
1950-03-01 17.24 17.56 17.07 17.29 1710000     17.29
```

`getSymbols()` from `quantmod` package

- There are several packages that provide functions to download economic and financial data by only specifying the ticker and time period (and frequency for some functions)
- `getSymbols()` function from package `quantmod()`
- Features of `getSymbols()`:
  - Sources: Yahoo Finance, Google Finance, OANDA (fx rates), FRED (argument src)
  - Download multiple tickers in one call
  - Select the time period (with arguments `from` and `to`)
  - Output is a `xts` object
  - Yahoo Finance: downloads open, high, low, close, volume, and adjusted close at the daily frequency
  - You can convert to weekly or monthly use `to.weekly()`/`to.monthly()` or the `apply.weekly()`/`apply.monthly()`

`getSymbols()` with one ticker

- By default, the function creates a `xts` object with the name of the ticker
- When downloading only one ticker, setting `auto.assign=FALSE` allows to assign the output to an object with another name (in the example below `data`)

```
> library(quantmod)
> getSymbols("GSPC", src = "yahoo", from = "1990-01-01")
[1] "GSPC"
> tail(GSPC, 2)
> data <- getSymbols("GSPC", src = "yahoo", from =
"1990-01-01", auto.assign = FALSE)
```

	GSPC.Open	GSPC.High	GSPC.Low	GSPC.Close	GSPC.Volume	GSPC.Adjusted
2017-01-31	2274.0	2279.1	2267.2	2278.9	4087450000	2278.9
2017-02-01	2285.6	2289.1	2272.4	2279.6	3916610000	2279.6

`getSymbols()` with many ticker

- A vector of symbols can be passed to `getSymbols()` to download data for multiple assets
- The function will create a `xts` object for each ticker (using as name the ticker; the `auto.assign` options does not work for more than one ticker)
- When you download more than 5 symbols you will see a message `pausing 1 second between requests for more than 5 symbols`

```
> library(quantmod)
> getSymbols(c("GSPC","DJI"), src="yahoo", from="1990-01-01")
> periodicity(GSPC)
> periodicity(DJI)
[1] "GSPC" "DJI"
Daily periodicity from 1990-01-02 to 2017-02-01
Daily periodicity from 1990-01-02 to 2017-02-01
```

`getSymbols()`: environment

- getSymbols() allows you also to specify an environment (argument `env`)
- The environment = a folder in the `R` global environment where the objects are stored
- Steps:
  - create a new environment with `new.env()` command (called `myenv` below)
  - call the `getSymbols()` function and set the `env=` argument to the new environment you created
  - the `ls()` command below lists the objects in the new environment myenv

```
splist     <- read.csv("List_SP500.csv", stringsAsFactors = FALSE)
myenv      <- new.env()
getSymbols(splist$Ticker.symbol[1:10], env=myenv, from="2010-01-01", src="yahoo")
```

```
 [1] "MMM"  "ABT"  "ABBV" "ACN"  "ATVI" "AYI"  "ADBE" "AAP"  "AES"  "AET"
```

```
ls(myenv)
```

getSymbols(): environment (2)

```
splist     <- read.csv("List_SP500.csv", stringsAsFactors = FALSE)
myenv      <- new.env()
getSymbols(splist$Ticker.symbol[1:10], env=myenv, from="2010-01-01", src="yahoo")
```

```
 [1] "MMM"  "ABT"  "ABBV" "ACN"  "ATVI" "AYI"  "ADBE" "AAP"  "AES"  "AET"
```

```
ls(myenv)
```

```
 [1] "AAP"  "ABBV" "ABT"  "ACN"  "ADBE" "AES"  "AET"  "ATVI" "AYI"  "MMM"
```

`quantmod` package

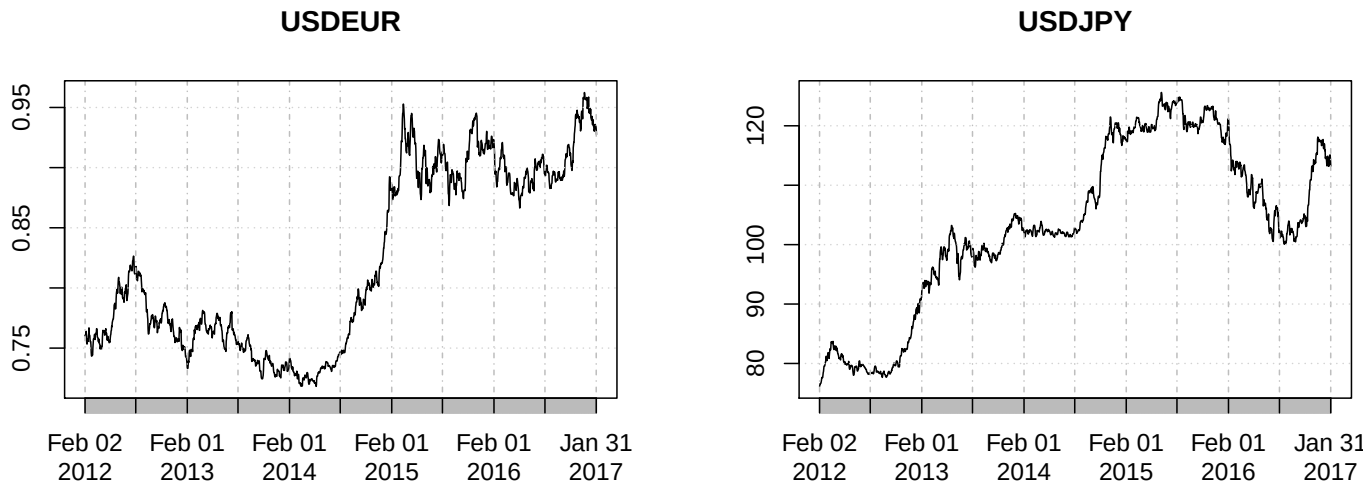
- The object created contains all the information, but for our analysis we might need only some of the columns
- The package has function to extract the open price `Op()`, the closing price `Cl()`, the highest intra-day price `Hi()`, the lowest `Lo()`, the volume `Vo()`, and the adjusted closing price `Ad()`
- `OpCl()` calculates the open-to-close daily return, `ClCl()` for the close-to-close return, and `LoHi()` for the low-to-high difference (also called the intra-day range)

```
> data.new <- merge(Ad(GSPC),Ad(DJI))
                           GSPC.Adjusted DJI.Adjusted
1990-01-02                        359.69       2810.1
1990-01-03                        358.76       2809.7
1990-01-04                        355.67       2796.1
```

Oanda

- Daily exchange rates for a wide range of currency pairs
- Limit of 5 year history
- Command `oanda.currencies` gives you the symbols for 191 currencies

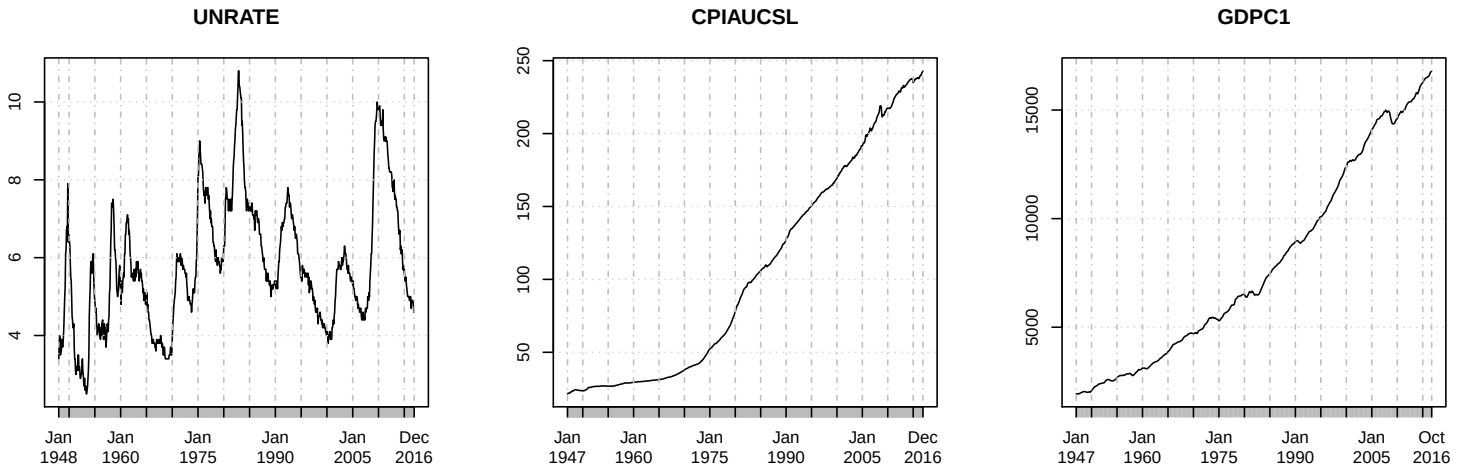
```
>getSymbols(c("USD/EUR", "USD/JPY"), src="oanda", from="2012-01-01")
[1] "USDEUR" "USDJPY"
>par(mfrow=c(1,2)); plot(USDEUR); plot(USDJPY)
```



FRED

- Federal Reserve Economic Data (FRED) can be used to download macroeconomic time series for the US economy and also international
- Visit FRED to find the symbol of the variable(s) you are interested to download
- Some examples:

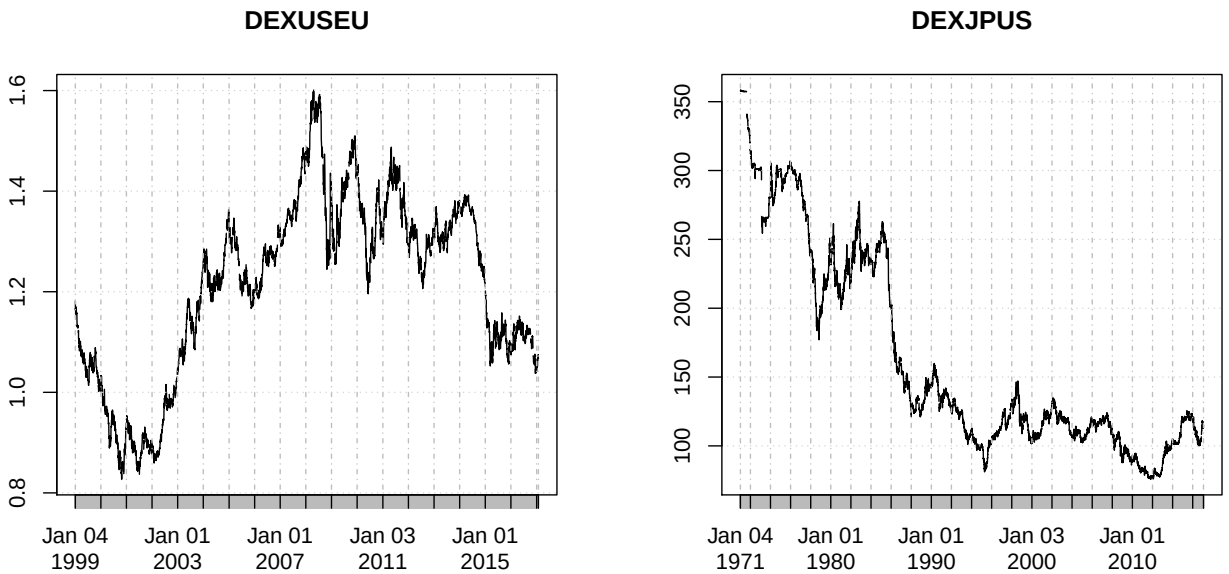
```
>library(quantmod)
>macrodata <- getSymbols(c('UNRATE','CPIAUCSL','GDPC1'), src="FRED")
>macrodata <- merge(UNRATE, CPIAUCSL, GDPC1)
>par(mfrow=c(1,3))
>plot(UNRATE); plot(CPIAUCSL); plot(GDPC1))
```



FRED (2)

- Exchange rates are also available from FRED (no restriction on the time period)

```
# DEXUSEU: U.S. Dollars to One Euro
# DEXJPUS: Japanese Yen to One U.S. Dollar
>macrodata <- getSymbols(c('DEXUSEU','DEXJPUS'), src="FRED",
from="1975-01-01")
> par(mfrow=c(1,2))
> plot(DEXUSEU); plot(DEXJPUS)
```



Quandl

- Quandl works as an aggregator of open databases and offers also subscription-based access to some selected datasets
- The macroeconomic variables retrieved from FRED can also be obtained from Quandl:

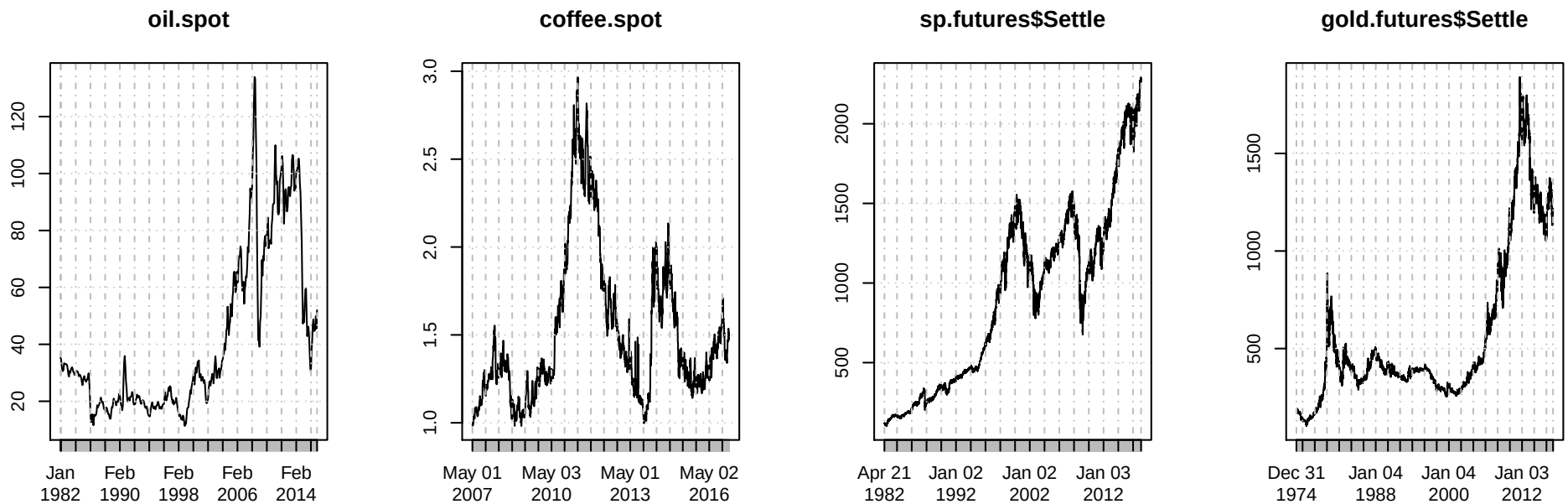
```
>library(Quandl)
>macrodata <- Quandl(c("FRED/UNRATE", 'FRED/CPIAUCSL', "FRED/GDPC1"),
start_date="1950-01-02", type="xts")
>head(macrodata)
```

	FRED.UNRATE - VALUE	FRED.CPIAUCSL - VALUE	FRED.GDPC1 - VALUE
1950-02-01	6.4	23.61	NA
1950-03-01	6.3	23.64	NA
1950-04-01	5.8	23.65	2147.6
1950-05-01	5.5	23.77	NA
1950-06-01	5.4	23.88	NA
1950-07-01	5.0	24.07	2230.4

Quandl (2)

- Quandl has many more datasets, e.g. commodity spot and futures prices

```
>oil.spot <- Quandl("COM/WLD_CRUDE_WTI", type="xts")
>coffee.spot <- Quandl("COM/COFFEE_BRZL", type="xts")
>sp.futures <- Quandl("CHRIS/CME_SP1", type="xts")
>gold.futures <- Quandl("CHRIS/CME_GC3", type="xts")
>par(mfrow=c(1,4));plot(oil.spot); plot(coffee.spot);
plot(sp.futures$Settle); plot(gold.futures$Settle)
```



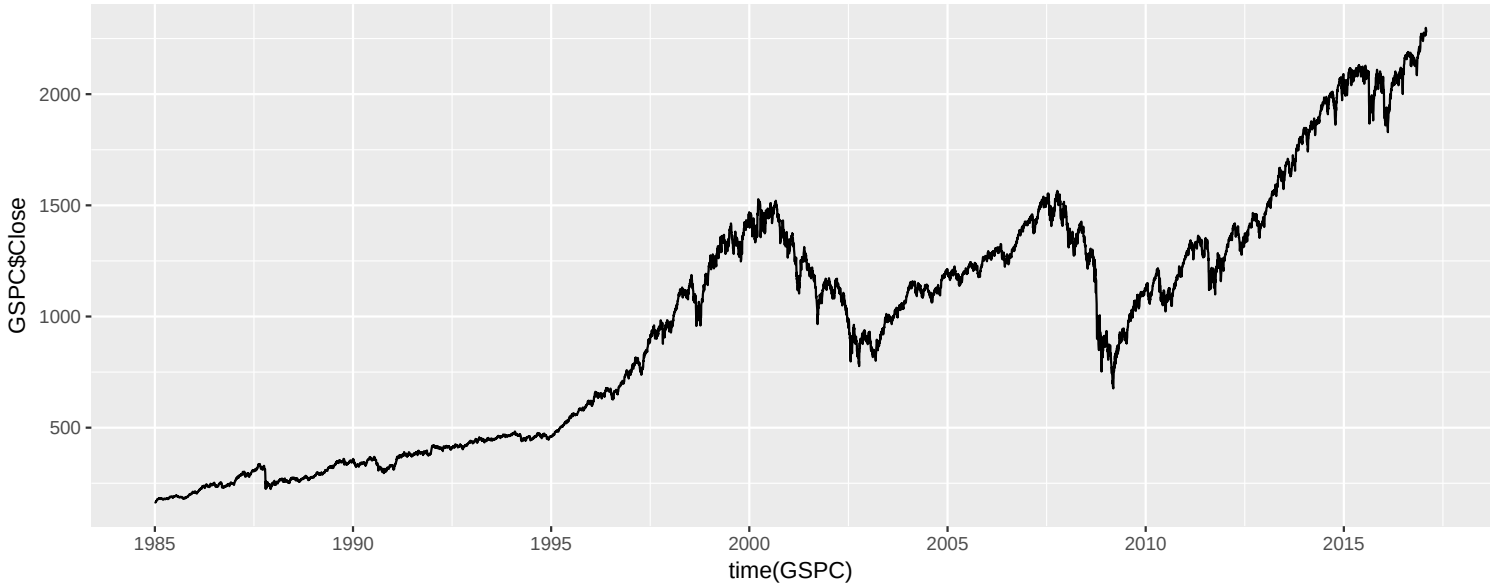
Advanced graphics: `ggplot2` package

- The base plotting functions are easy to use and convenient for quick plotting
- However, they lack elegance and it is difficult to produce high-quality graphics
- The package `ggplot2` offers an alternative set of function to make graphs
- There are two ways of producing plots with `ggplot2`:
  1. `qplot()` is a wrapper function similar to `plot()` that uses underlying `ggplot2` plotting functions
  2. Using the grammar of graphics that is composed of:
     - ★ `ggplot()`: creating the graph and specifying the data frame that contains the variables to plot
     - ★ `geom_xxx()`: the type of plot that is needed; point, line, histogram, boxplot, etc
     - ★ `aes()`: the x and y variables to plot
     - ★ `theme()`: the overall look of the plot; `theme_bw()`, `theme_classic()`, `theme_dark()` etc.

ggplot2 package (2)

- ggplot2 does not recognize the time series properties of xts objects and we have to specify the x axis
- The time(GSPC) command is used to extract the date associated with each observation

```
>GSPC <- getSymbols("GSPC", from="1985-01-01", auto.assign=FALSE)
>names(GSPC) <- sub('GSPC.'," ", names(GSPC))
>library(ggplot2)
>qplot(time(GSPC), GSPC$Close, geom="line")
```

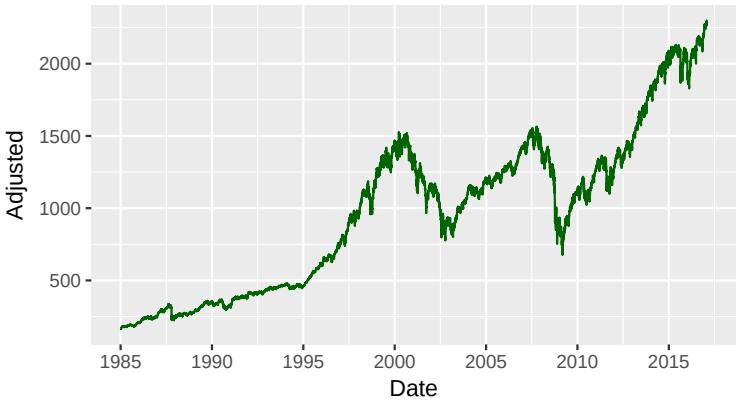


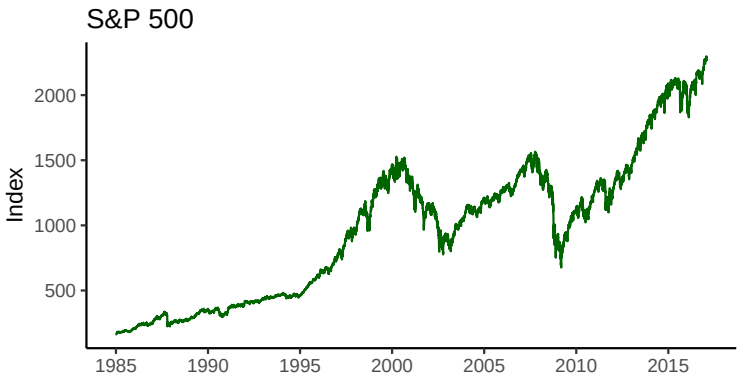
ggplot2 package (3)

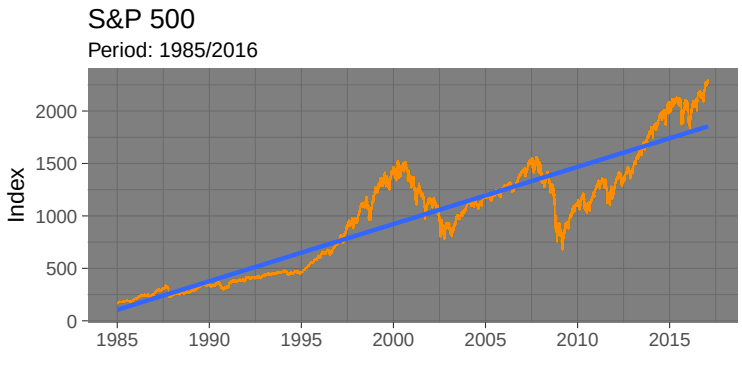
- The plots can be customized with themes, line colors and types, label names etc
- The `par(mfrow=c(2,2))` does not work with `ggplot` and we should use the `grid.arrange()` function from package `gridExtra`

```
>plot1 <- ggplot(GSPC.df, aes(Date, Adjusted)) +
geom_line(color="darkgreen")
>plot2 <- plot1 + theme_bw()
>plot3 <- plot2 + theme_classic() + labs(x="", y="Index", title="S&P
500")
>plot4 <- plot3 + geom_line(color="darkorange") +
geom_smooth(method="lm") + theme_dark() + labs(subtitle="Period:
1985/2016", caption="Source:  Yahoo")
>library(gridExtra)
>grid.arrange(plot1, plot2, plot3, plot4, ncol=2)
```

ggplot2 package (4)







Source: Yahoo

Covariance and correlation between two or more assets

- Do the S&P 500 and NIKKEI move together?

```
>Ret <- subset(GN.df, select=c("SPret","NIKret"))
>cov(Ret, use='complete.obs')
        SPret NIKret
SPret  17.385 13.746
NIKret 13.746 39.669

>cor(Ret, use='complete.obs')
         SPret  NIKret
SPret  1.00000 0.52342
NIKret 0.52342 1.00000
```

Functions in R

- A function is a set of operations applied to some data.
- The syntax is as follows:

```
myfunction <- function(inputs)
{
   #
   # operations
   #
  return(output)
}
```

- The `function()` can include several arguments
- The `return()` is the output of the function that can include only one object, although that object might include several elements
- To call the function you type in R the name of the function with the appropriate arguments: `myfunction()`

A function to calculate the sample average

- The built-in function `mean`:

```
>mean(GSPC$ret.log, na.rm=T)
```

- Below is the code that defines a new function called `mymean()`:

```
# Y is the input, Ybar the output of the function
mymean <- function(Y)
{
  Y = na.omit(Y)
  Ybar <- sum(Y) / length(Y)
 return(Ybar)
}
```

Loops in R

- A common loop is the `for loop` which has the following syntax:

```
>for (i in 1:N)
{
  # write your commands here
}
```

- Example:

```
for (i in 1:3)
{
  print(i)
}
[1] 1
[1] 2
[1] 3
```

mysum() example

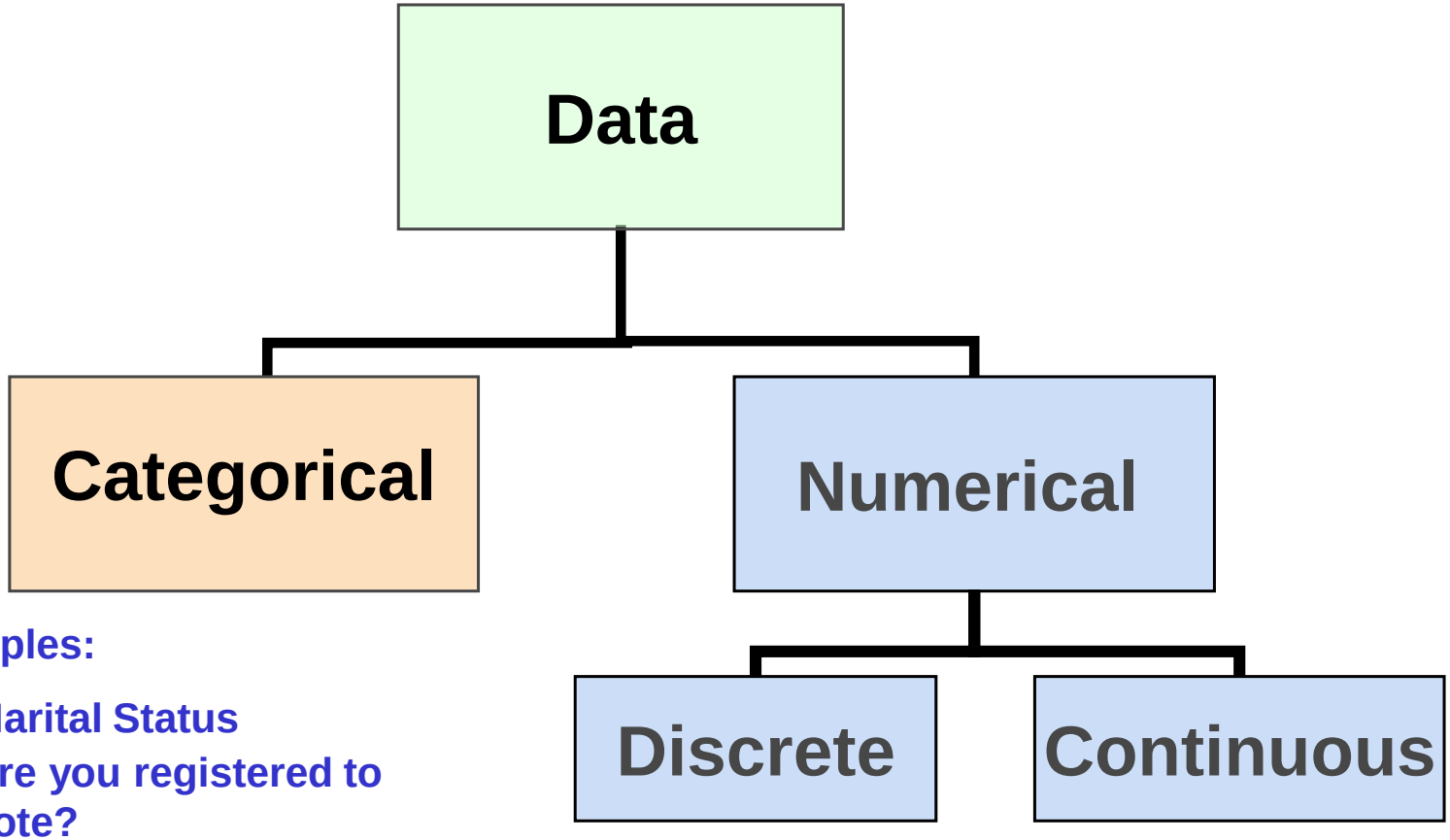
```
mysum <- function(Y)
{
Y = na.omit(Y)
N = length(Y) # define N as the number of elements of Y
sumY = 0 # initialize the variable that will store the sum of Y
for (i in 1:N)
{
sumY = sumY + as.numeric(Y[i]) # current sum is equal to
previous sum
} # plus the i-th value of Y
return(sumY) # as.numeric():  makes sure to transform
} # from other classes to a number
>mysum(GSPC$ret.log)
```

Descriptive and Inferential Statistics

Two branches of statistics:

- Descriptive statistics
  - Collecting, summarizing, and processing data to transform data into information
- Inferential statistics
  - provide the bases for predictions, forecasts, and estimates that are used to transform information into knowledge

Types of Data



Examples:

- Marital Status
- Are you registered to vote?
- Eye Color

(Defined categories or groups)

Examples:

- Number of Children
- Defects per hour

(Counted items)

Examples:

- Weight
- Voltage

(Measured characteristics)

Graphical Presentation of Data

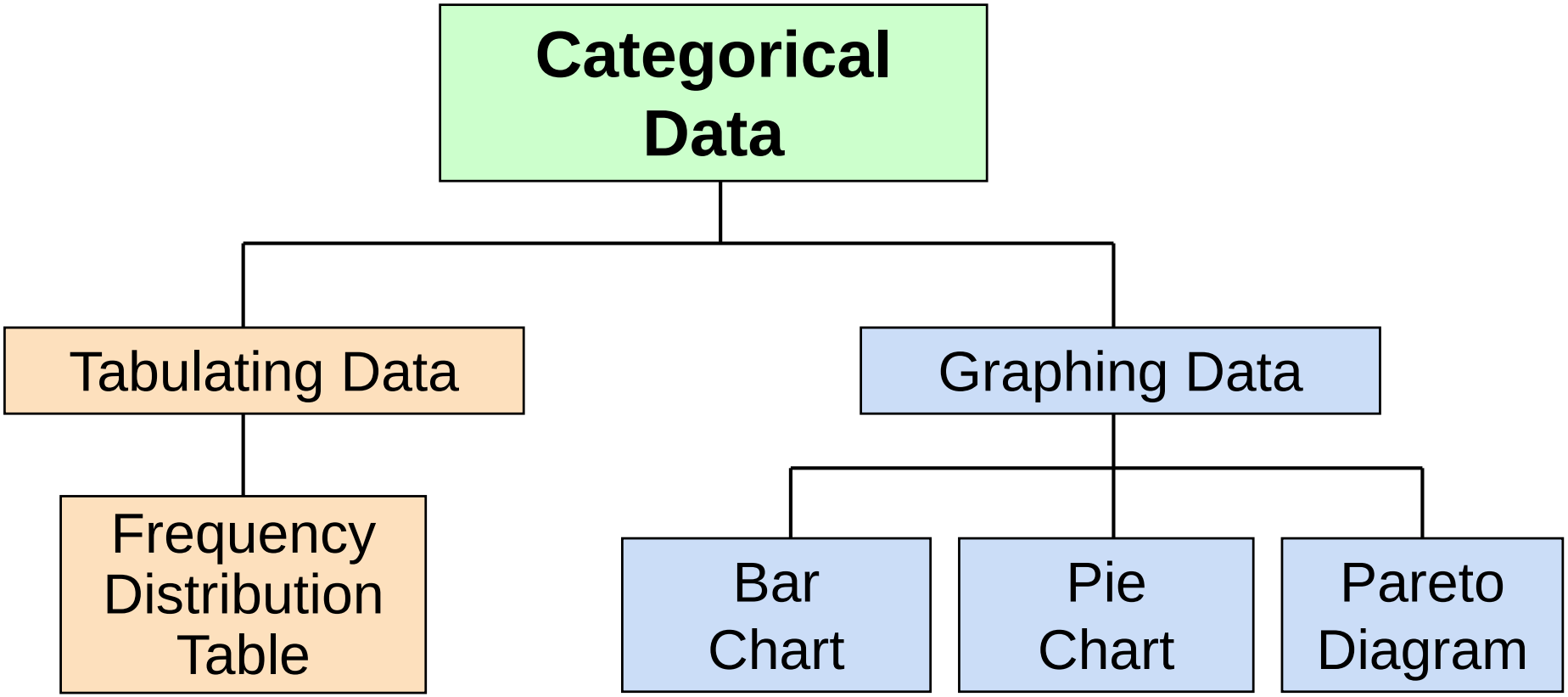
Categorical Variables

- Frequency distribution
- Bar chart
- Pie chart
- Pareto diagram

Numerical Variables

- Line chart
- Frequency distribution
- Histogram and ogive
- Stem-and-leaf display
- Scatter plot

Tables and Graphs for Categorical Variables



The Frequency Distribution Table

Summarize data by category

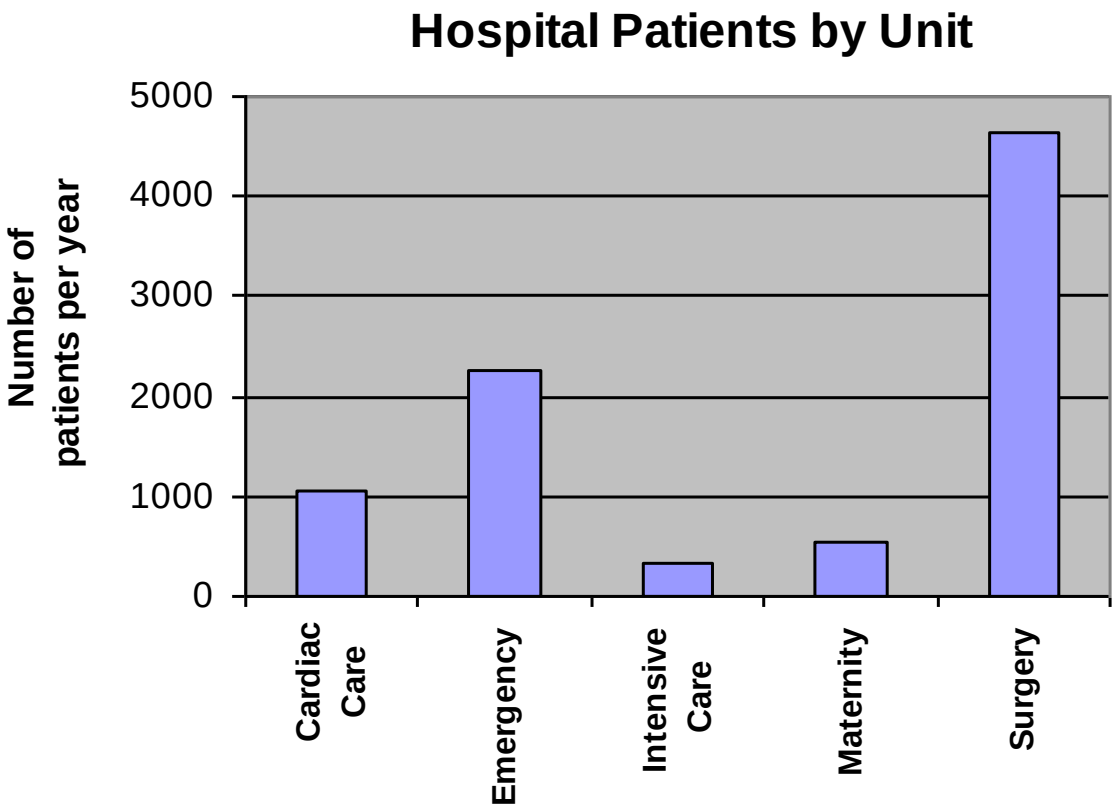
Example: Hospital Patients by Unit

Hospital Unit	Number of Patients
Cardiac Care	1,052
Emergency	2,245
Intensive Care	340
Maternity	552
Surgery	4,630

(Variables are categorical)

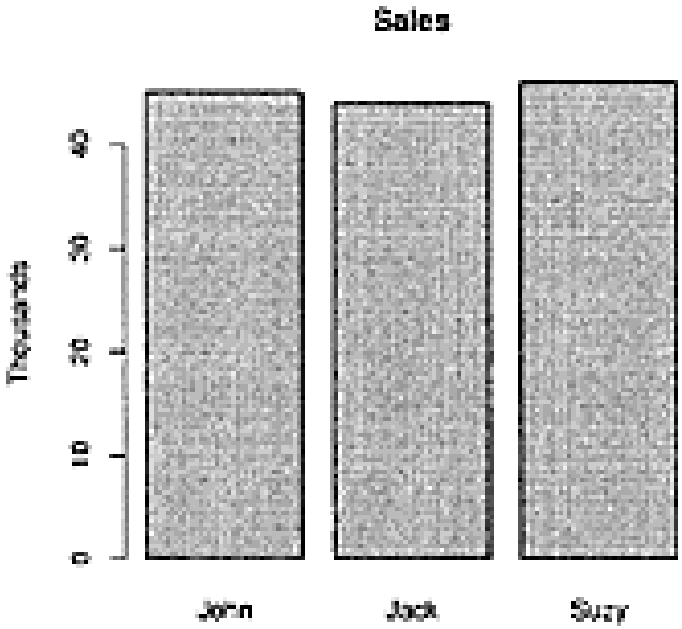
Bar chart

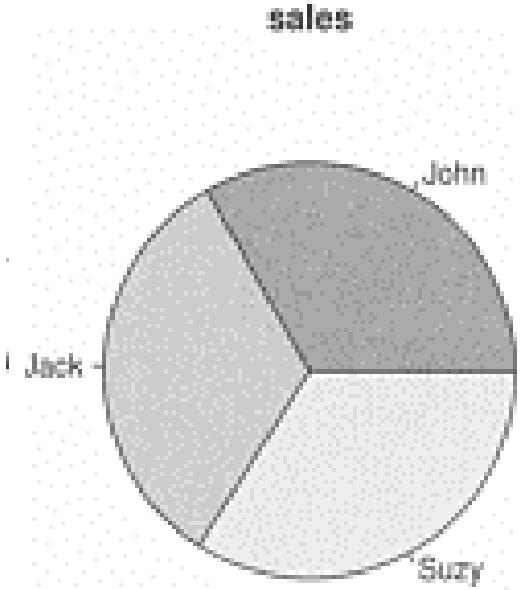
Hospital Unit	Number of Patients
Cardiac Care	1,052
Emergency	2,245
Intensive Care	340
Maternity	552
Surgery	4,630



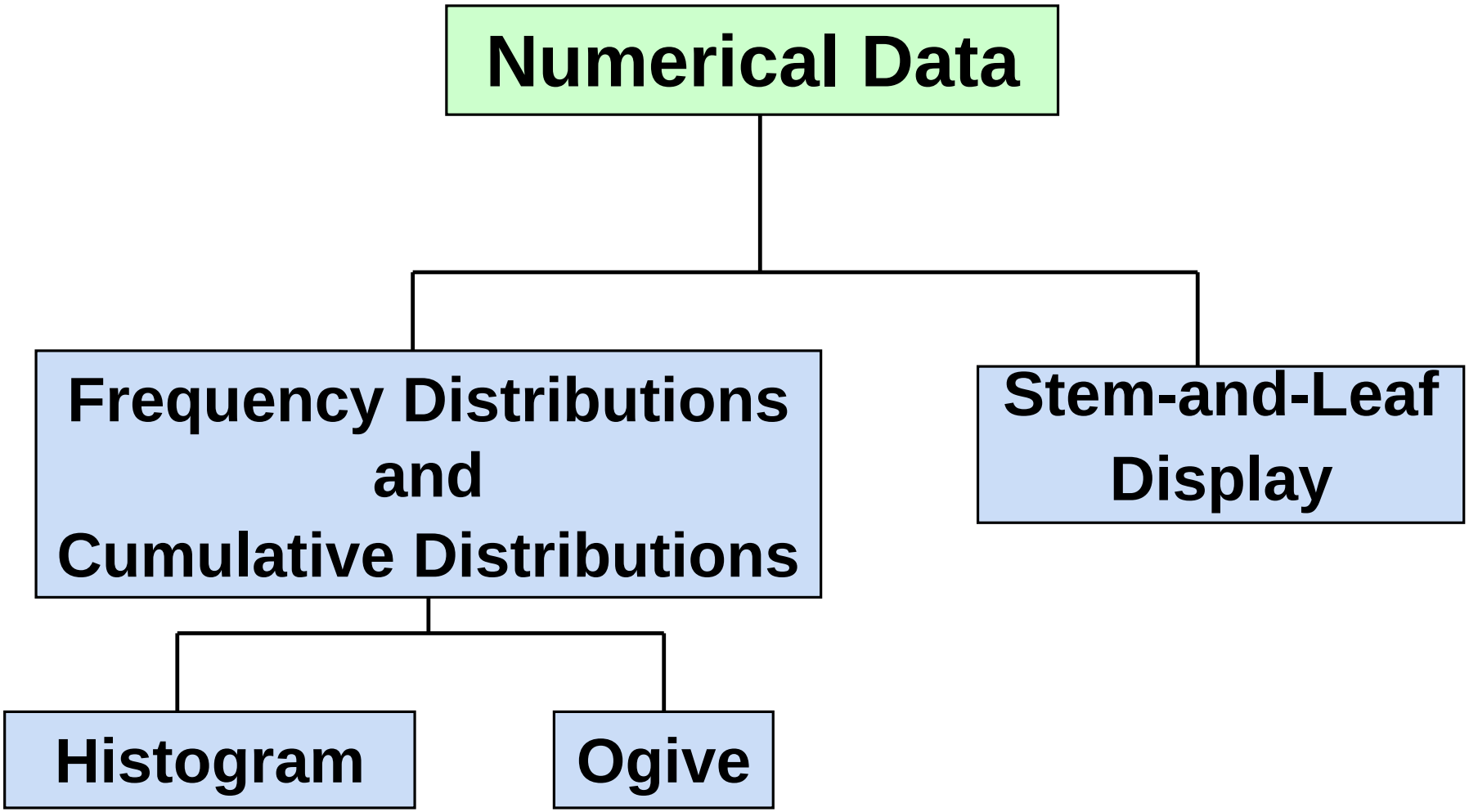
Bar charts in R

```
> sales = c(45,44,46) # quarterly sales
> names(sales) = c("John","Jack","Suzy") # include names
> barplot(sales, main="Sales", ylab="Thousands") # basic
barplot
> pie(sales, main="sales")
```





Graphs to Describe Numerical Variables



Frequency distribution

Example: A manufacturer of insulation randomly selects 20 winter days and records the daily high temperature

24, 35, 17, 21, 24, 37, 26, 46, 58, 30,
32, 13, 12, 38, 41, 43, 44, 27, 53, 27

Frequency distribution (2)

- Sort raw data in ascending order:

  12, 13, 17, 21, 24, 24, 26, 27, 27, 30, 32, 35, 37, 38, 41, 43, 44, 46, 53, 58

- Find range: **58 - 12 = 46**
- Select number of classes: **5 (usually between 5 and 15)**
- Compute interval width: **10** **(46/5 then round up)**
- Determine interval boundaries: **10 but less than 20, 20 but less than 30, . . . , 60 but less than 70**
- Count observations & assign to classes

Frequency distribution (3)

Data in ordered array:

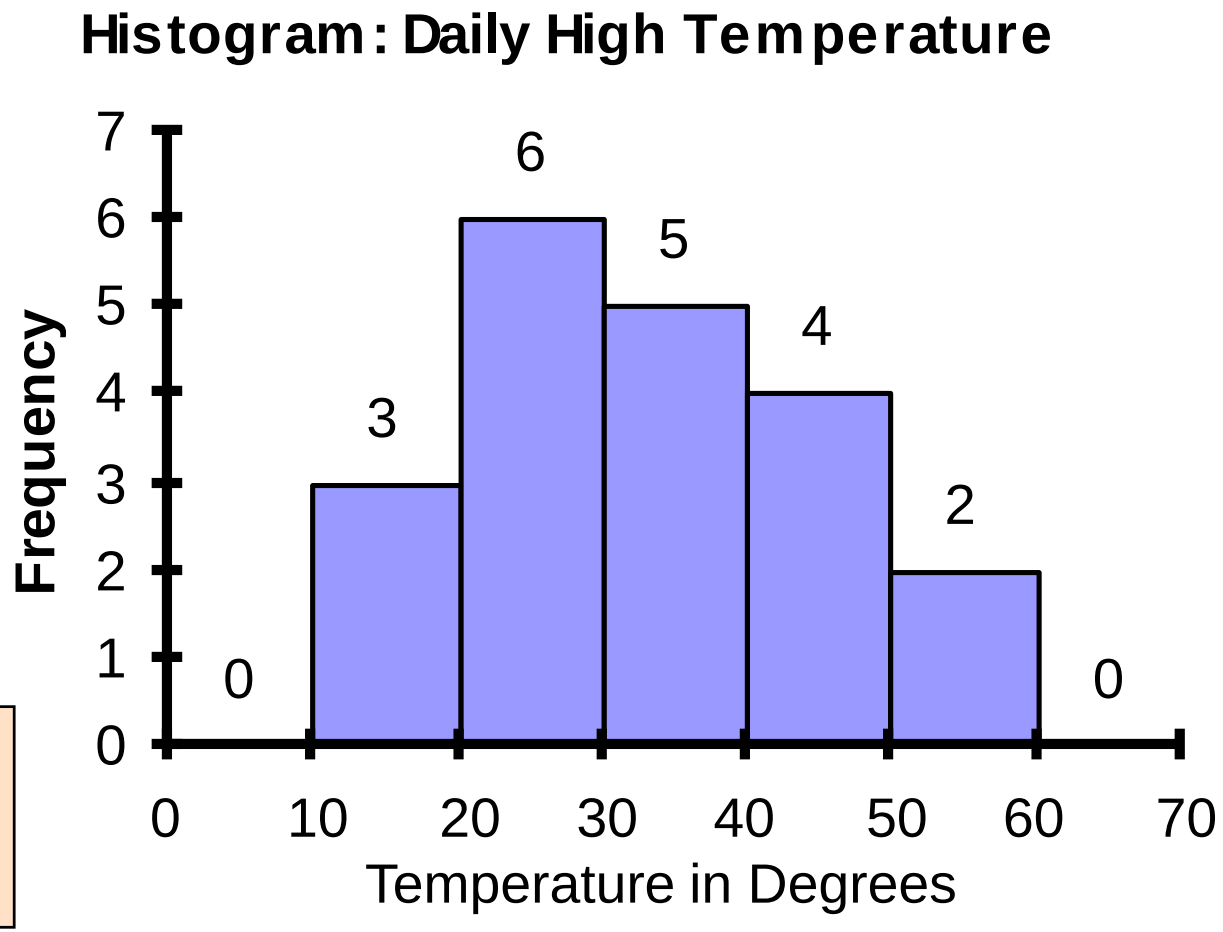
12, 13, 17, 21, 24, 24, 26, 27, 27, 30, 32, 35, 37, 38, 41, 43, 44, 46, 53, 58

Interval	Frequency	Relative Frequency	Percentage
10 but less than 20	3	.15	15
20 but less than 30	6	.30	30
30 but less than 40	5	.25	25
40 but less than 50	4	.20	20
50 but less than 60	2	.10	10
Total	20	1.00	100

Histograms: output

Interval	Frequency
10 but less than 20	3
20 but less than 30	6
30 but less than 40	5
40 but less than 50	4
50 but less than 60	2

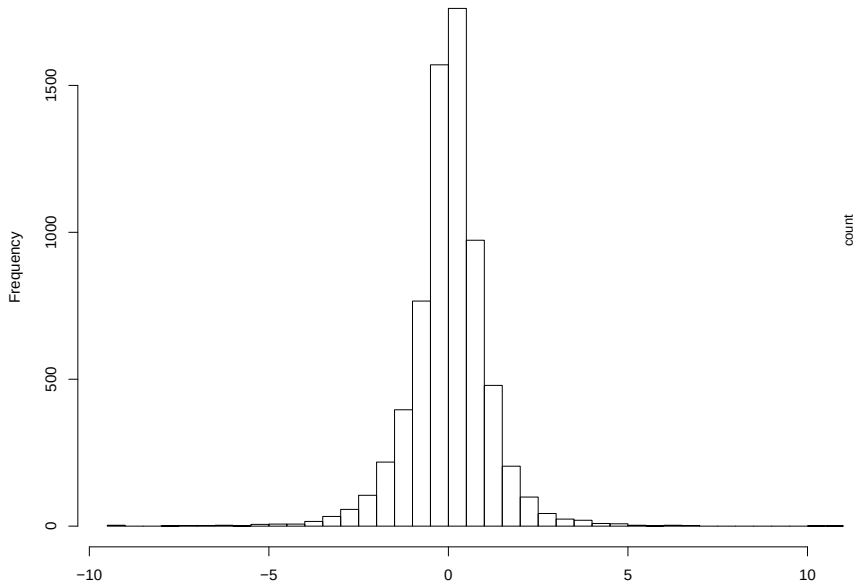
(No gaps between bars)

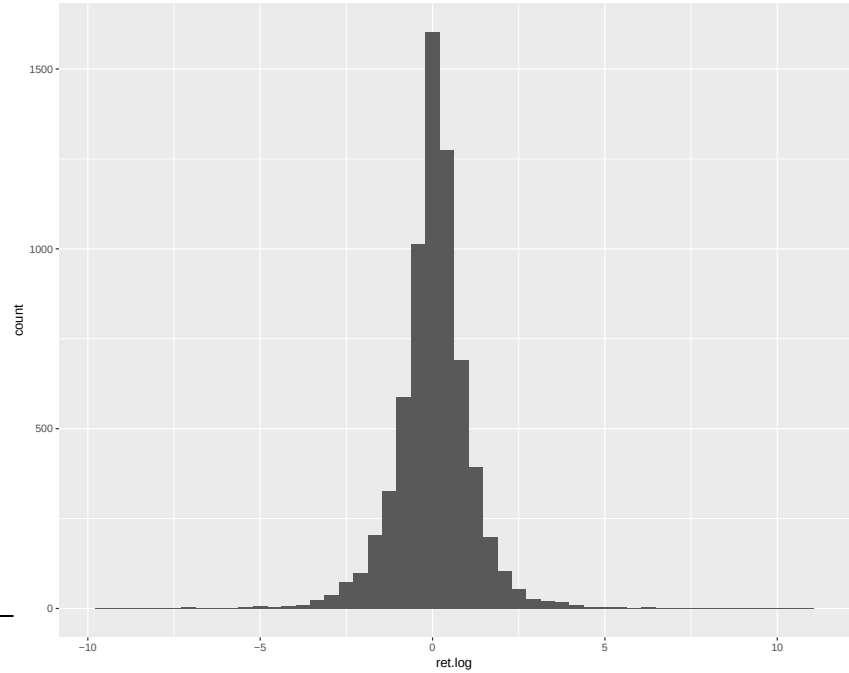


Histograms in R

- A very useful tool to explore the empirical distribution of the data
- Allows to assess (visually) the characteristics of the data such as normality, fat tails, asymmetry etc.

```
>hist(GSPC$ret.log, breaks=50, xlab="", main="") # base function
>qplot(ret.log, data=GSPC, geom="histogram", bins=50) # ggplot function
```

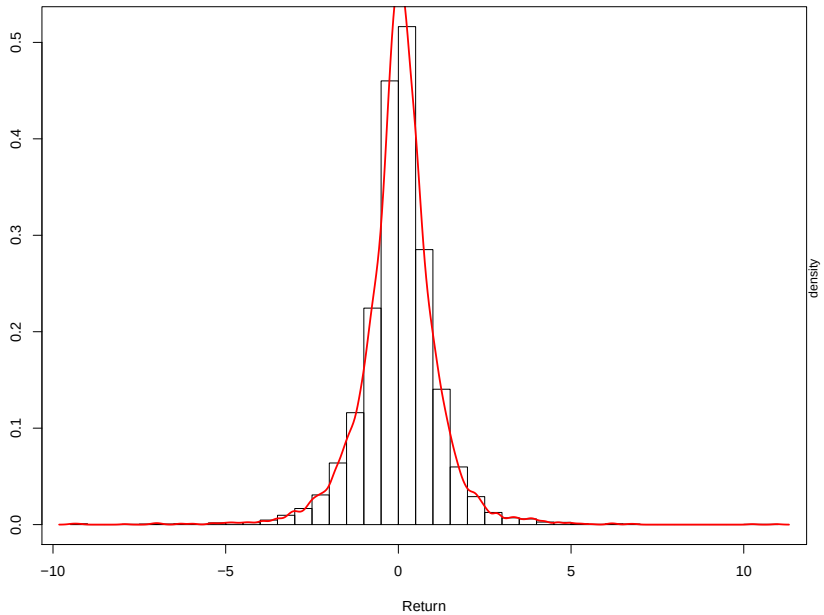


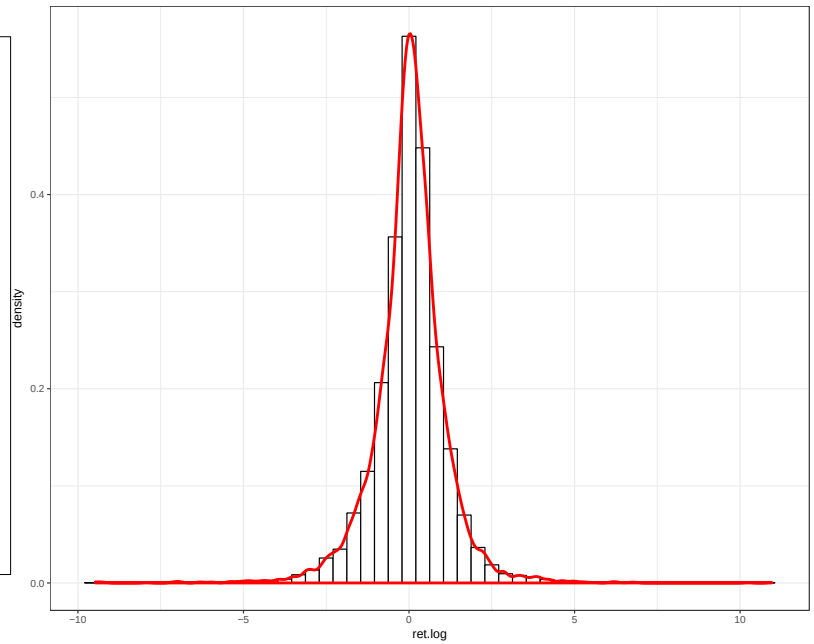


Plotting the density

- One can overlap a non-parametric estimate of the density that goes through the histogram bars

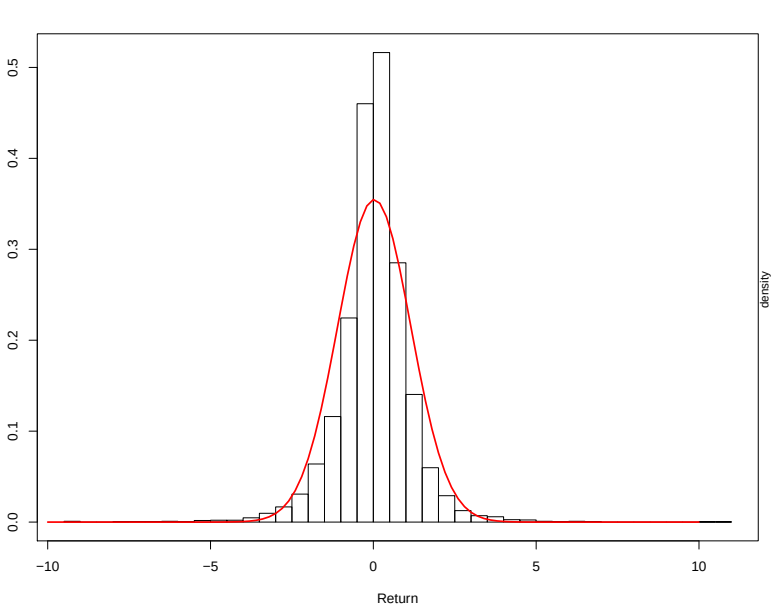
```
# base function
>hist(GSPC$ret.log, breaks=50, main="", xlab="Return", ylab="",prob=TRUE)
>lines(density(GSPC$ret.log,na.rm=TRUE),col=2,lwd=2)
>box()
# ggplot function
>ggplot(GSPC, aes(ret.log)) + geom_histogram(aes(y = ..density..), bins=50, color="black",
fill="white") + geom_density(color="red", size=1.2) + theme_bw()
```

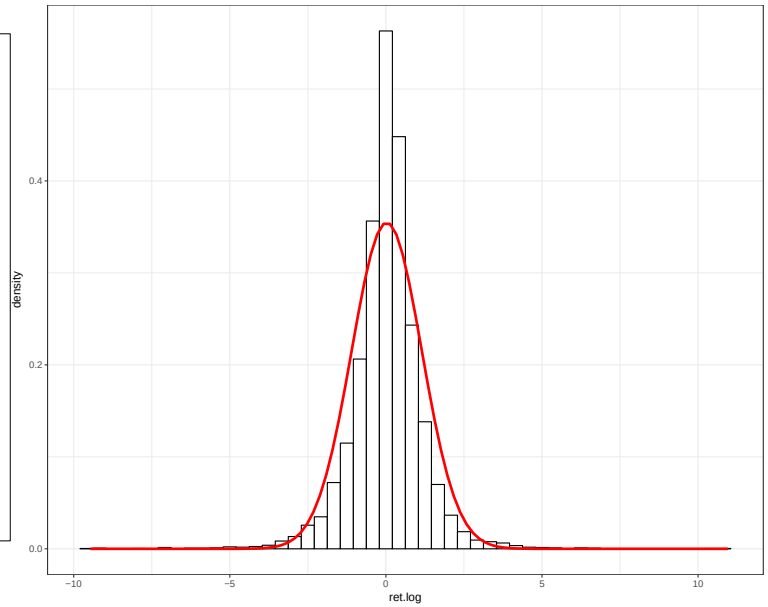




Comparing the histogram to a distribution

```
# base function
>hist(GSPC$ret.log, breaks=50, main="", xlab="Return",
ylab="",prob=TRUE)
>curve(dnorm(x, mean(GSPC$ret.log, na.rm=T), sd(GSPC$ret.log,
na.rm=T)), from=-10, to=10, add=TRUE, col="red",lwd=2)
>box()
# ggplot function
>ggplot(GSPC, aes(ret.log)) + geom_histogram(aes(y = ..density..),
bins=50, color="black", fill="white") + stat_function(fun = dnorm,
colour = "red", args = list(mean(GSPC$ret.log, na.rm=T),
sd(GSPC$ret.log, na.rm=T)), size=1.2) + theme_bw()
```





Stem-and-Leaf Diagram

- A simple way to see distribution details in a data set

METHOD: Separate the sorted data series into leading digits (**the stem**) and the trailing digits (**the leaves**)

Data in ordered array:

21, 24, 24, 26, 27, 27, 30, 32, 38, 41

- Here, use the 10's digit for the stem unit:
  - 21 is shown as
  - 38 is shown as

Stem	Leaf
2	1
3	8

Stem-and-Leaf Diagram (2)

Data in ordered array:

21, 24, 24, 26, 27, 27, 30, 32, 38, 41

- Completed stem-and-leaf diagram:

Stem	Leaves
2	1 4 4 6 7 7
3	0 2 8
4	1

Using other stem units

- Using the 100's digit as the stem:
  - Round off the 10's digit to form the leaves

- 613 would become
- 776 would become
- . . .
- 1224 becomes

Stem	Leaf
6	1
7	8
12	2

Using other stem units (2)

- Using the 100's digit as the stem:
  - The completed stem-and-leaf display:

Data:

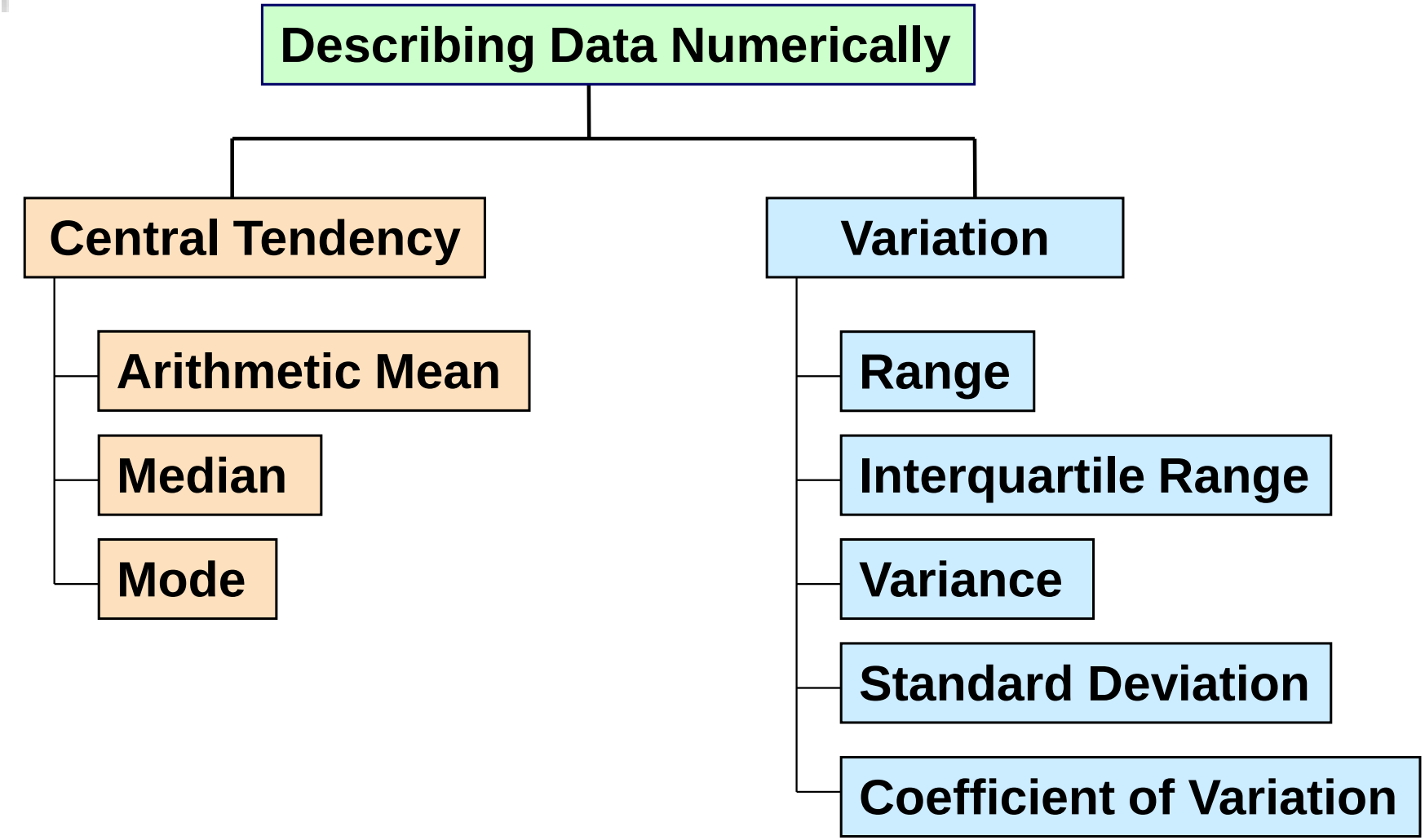
613, 632, 658, 717,
722, 750, 776, 827,
841, 859, 863, 891,
894, 906, 928, 933,
955, 982, 1034,
1047,1056, 1140,
1169, 1224

Stem	Leaves
6	1 3 6
7	2 2 5 8
8	3 4 6 6 9 9
9	1 3 3 6 8
10	3 5 6
11	4 7
12	2

Stem-and-Leaf in R

```
> x = scan()
1:2 3 16 23 14 12 4 13 2 0 0 0 6 28 31 14 4 8 2 5
21:
Read 20 items
> stem(x)
            The decimal point is 1 digit(s) to the right of the |
            0 | 000222344568
            1 | 23446
            2 | 38
            3 | 1
```

Describing Data Numerically



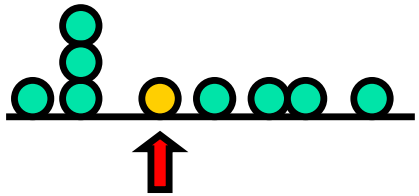
Measures of Central Tendency

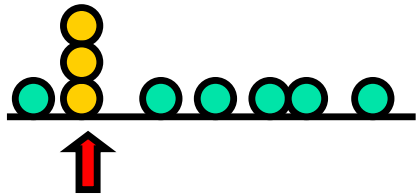
Overview

Central Tendency

Mean | **Median** | **Mode**

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}$$





Arithmetic average

Midpoint of ranked values

Most frequently observed value

Arithmetic Mean

- The arithmetic mean (mean) is the most common measure of central tendency
  - For a population of N values:

$$\mu = \frac{\sum_{i=1}^{N} x_i}{N} = \frac{x_1 + x_2 + \cdots + x_N}{N}$$

Population values

Population size

  - For a sample of size n:

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n} = \frac{x_1 + x_2 + \cdots + x_n}{n}$$

Observed values

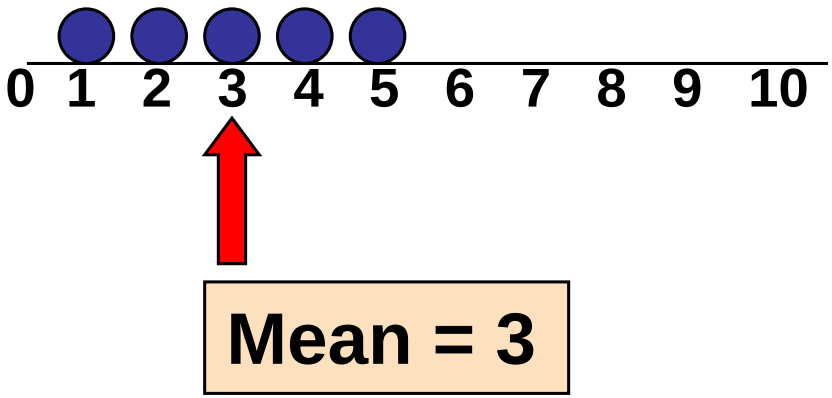
Sample size

Arithmetic Mean in R

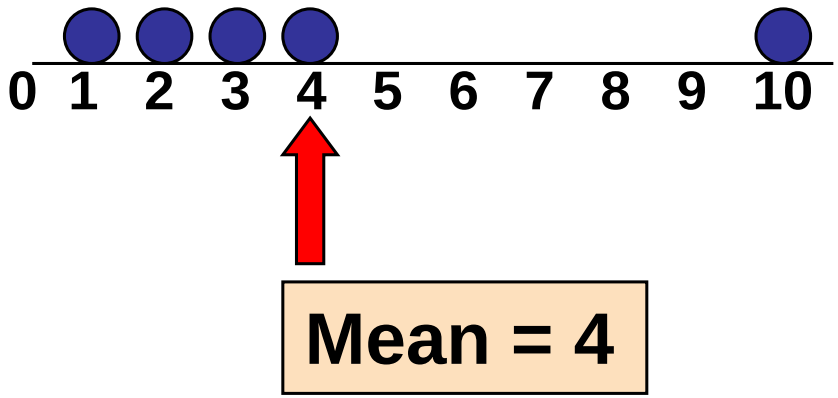
```
> x = scan()
1:2 3 16 23 14 12 4 13 2 0 0 0 6 28 31 14 4 8 2 5
21:
Read 20 items
> sum(x)/length(x)
[1] 9.35
> mean(x)
[1] 9.35
```

Arithmetic Mean: Sensitivity to Outliers

- The most common measure of central tendency
- Mean = sum of values divided by the number of values
- Affected by extreme values (outliers)



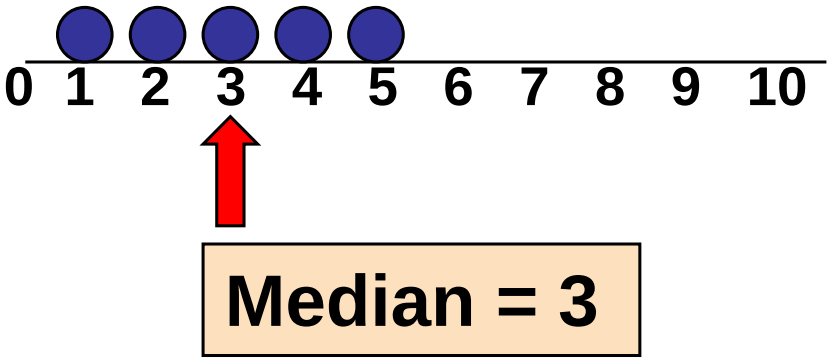
$$\frac{1+2+3+4+5}{5} = \frac{15}{5} = 3$$

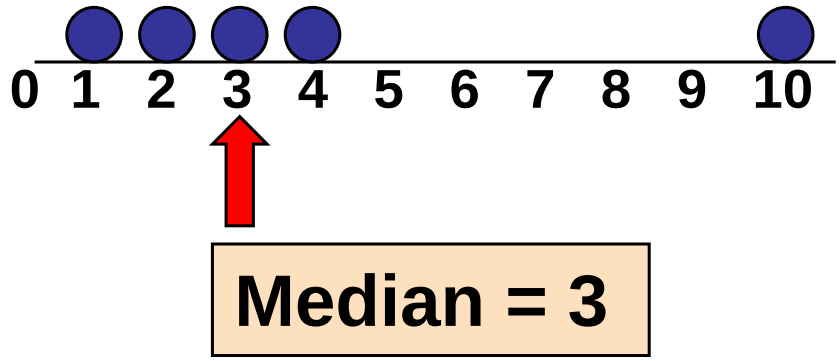


$$\frac{1+2+3+4+10}{5} = \frac{20}{5} = 4$$

Median

- In an ordered list, the median is the “middle” number (50% above, 50% below)





- Not affected by extreme values

Finding the Median

- The location of the median:

$$\text{Median position} = \frac{n+1}{2} \text{ position in the ordered data}$$

  - If the number of values is odd, the median is the middle number
  - If the number of values is even, the median is the average of the two middle numbers

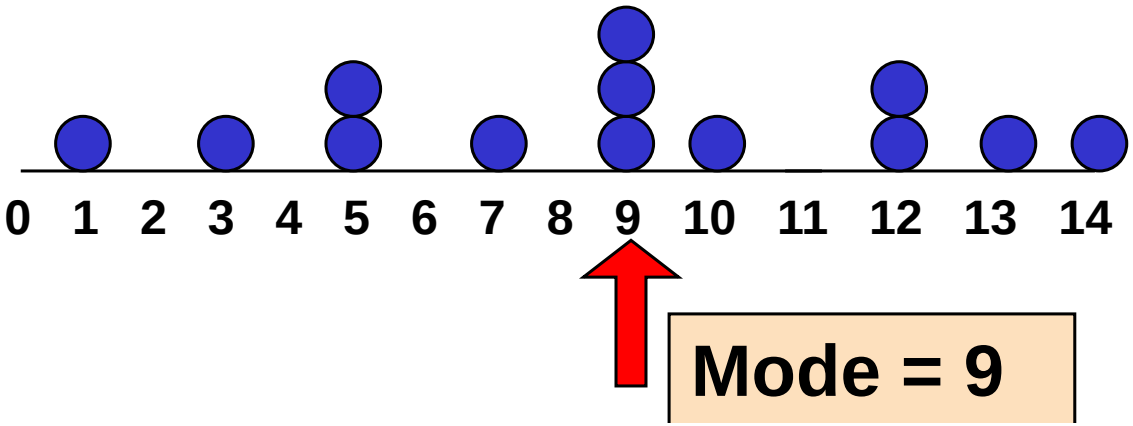
- Note that $\frac{n+1}{2}$ is not the *value* of the median, only the *position* of the median in the ranked data

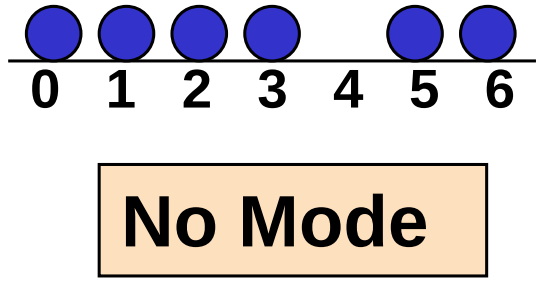
Median in R

```
> bar = c(50,60,100,75,200) # bar patrons worth in 1000s
> bar.with.gates = c(bar,50000) # after Bill Gates enters
> mean(bar)
[1] 97
> mean(bar.with.gates) # mean is sensitive to large values
[1] 8414
> median(bar)
[1] 75
> median(bar.with.gates) # median is resistant
[1] 87.5
```

Mode

- A measure of central tendency
- Value that occurs most often
- Not affected by extreme values
- Used for either numerical or categorical data
- There may may be no mode
- There may be several modes





Mode in R

There is no built-in function, so:

```
> x=c(72,75,84,84,98,94,55, 62)
> which(table(x) == max(table(x)))
84
5
```

Review Example

House Prices:	
	$2,000,000
	500,000
	300,000
	100,000
	100,000
Sum	3,000,000

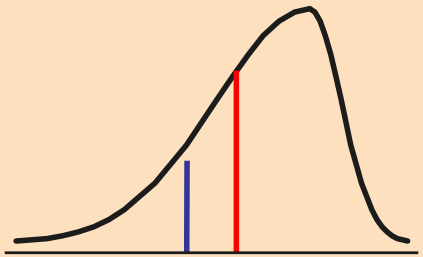
- **Mean:** ($3,000,000/5)
  = **$600,000**
- **Median:** middle value of ranked data
  = **$300,000**
- **Mode:** most frequent value
  = **$100,000**

Shape of a Distribution

- Describes how data are distributed
- Measures of shape
  - Symmetric or skewed

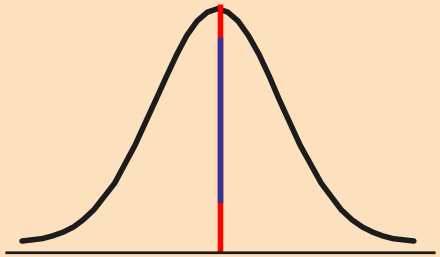
Left-Skewed

Mean < Median



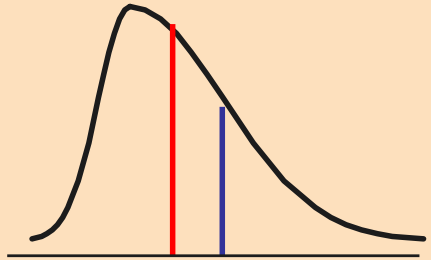
Symmetric

Mean = Median



Right-Skewed

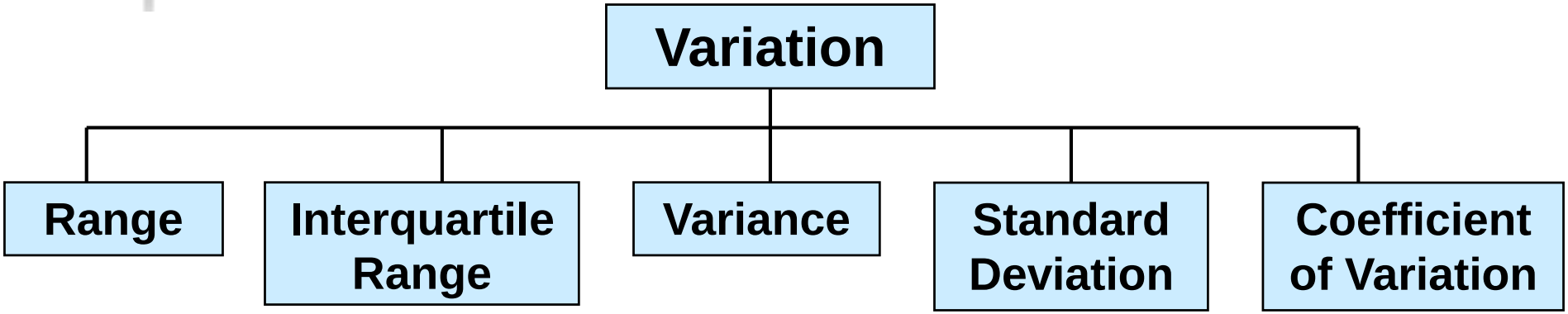
Median < Mean



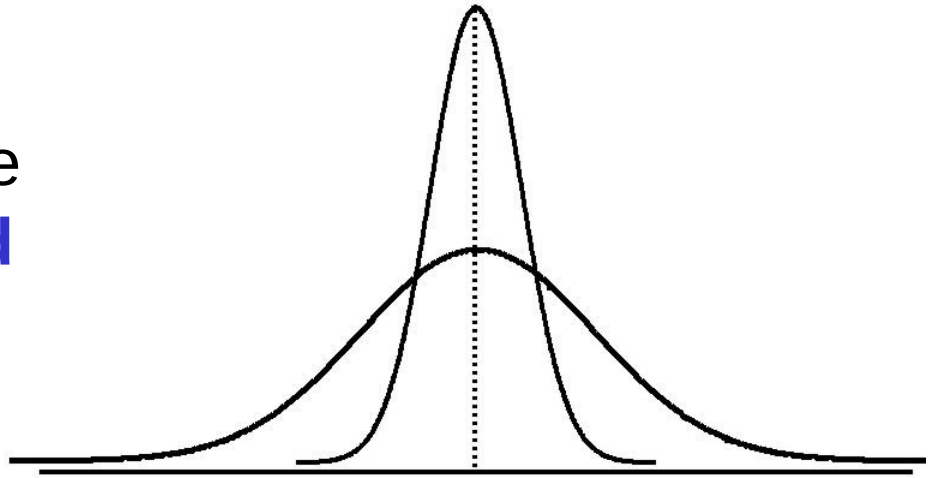
Shape of a Distribution: R example

```
> income=cfb$INCOME
> mean(income)
[1] 63403
> median(income)
[1] 38033
> sum(income <= mean(income))/length(income)*100
[1] 70.5
```

Measures of Variability



- Measures of variation give information on the **spread** or **variability** of the data values.



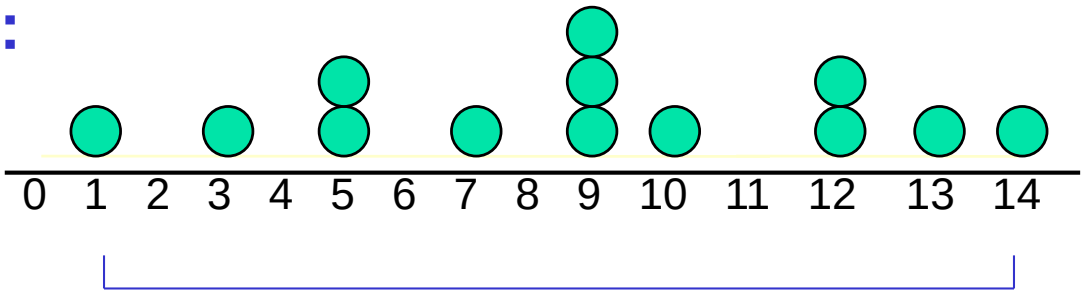
Same center, different variation

Range

- Simplest measure of variation
- Difference between the largest and the smallest observations:

$$\text{Range} = X_{\text{largest}} - X_{\text{smallest}}$$

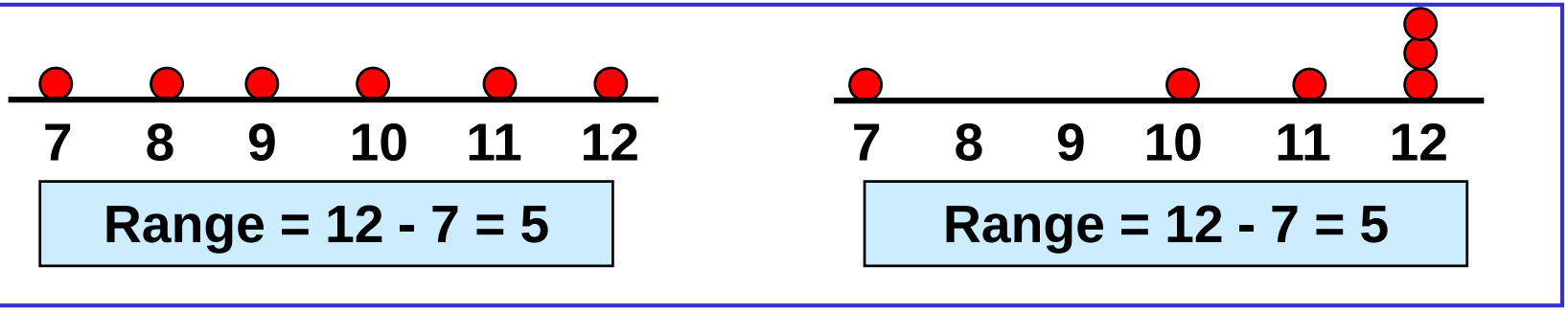
Example:



Range = 14 - 1 = 13

Disadvantages of the Range

- Ignores the way in which data are distributed



- Sensitive to outliers

1,1,1,1,1,1,1,1,1,1,1,2,2,2,2,2,2,2,2,3,3,3,3,4,5

Range = 5 - 1 = 4

1,1,1,1,1,1,1,1,1,1,1,2,2,2,2,2,2,2,2,3,3,3,3,4,120

Range = 120 - 1 = 119

Range in R

```
diff(range(x))
```

Sample Variance

- Average (approximately) of squared deviations of values from the mean
  - Sample variance:

$$s^2 = \frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n-1}$$

Where $\bar{X}$ = arithmetic mean

n = sample size

X_i = i^{th} value of the variable X

Sample Standard Deviation

- Most commonly used measure of variation
- Shows variation about the mean
- Has the same units as the original data

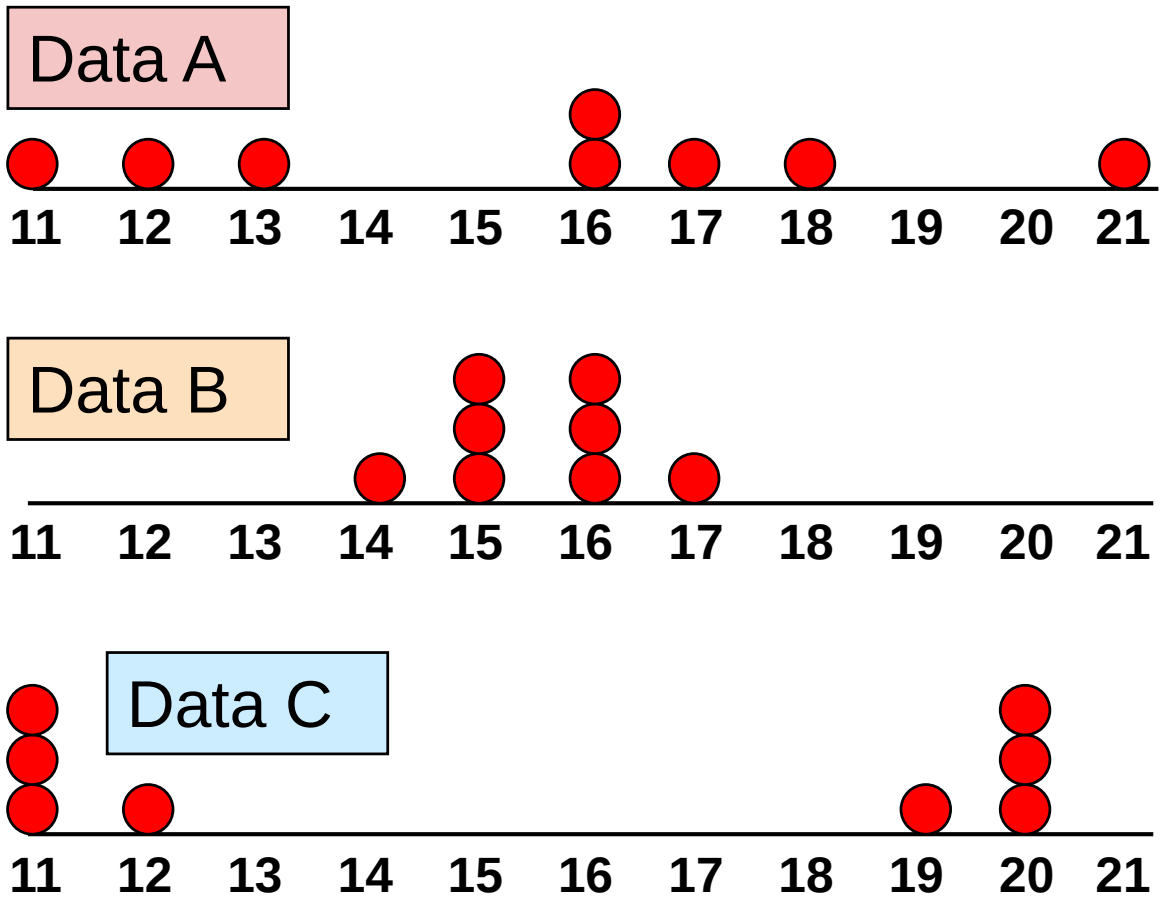
  - Sample standard deviation:

$$S = \sqrt{\frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n-1}}$$

Sample Variance and Sample Standard Deviation in R

```
> test.scores = c(80,85,75,77,87,82,88)
> test.scores.b = c(100,90,50,57,82,100,86)
> mean(test.scores)
[1] 82
> mean(test.scores.b) # means are similar
[1] 80.71
> var(test.scores) # built-in var function
[1] 24.67
> var(test.scores.b) # larger, as anticipated
[1] 394.2
> sd(test.scores)
[1] 4.967
```

Comparing Standard Deviations



Mean = 15.5
s = 3.338

Mean = 15.5
s = 0.926

Mean = 15.5
s = 4.570

Interquartile Range

- Can eliminate some outlier problems by using the **interquartile range**

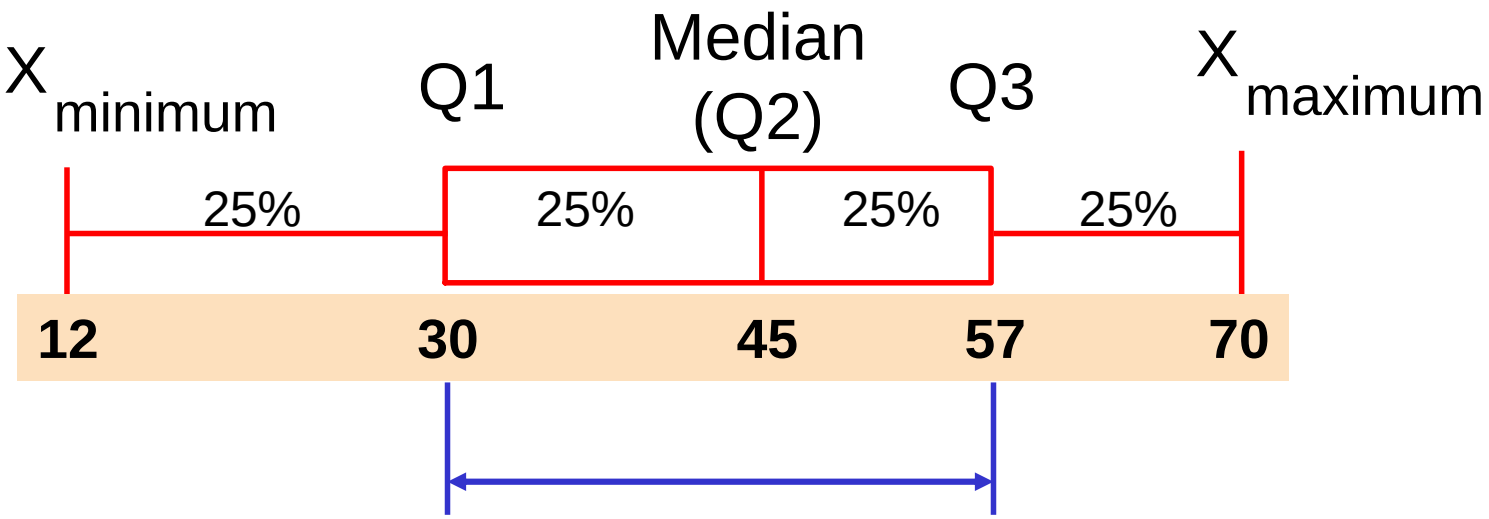
- Eliminate high- and low-valued observations and calculate the range of the middle 50% of the data

- Interquartile range = 3^{rd} quartile – 1^{st} quartile

$$IQR = Q_3 - Q_1$$

Interquartile Range (2)

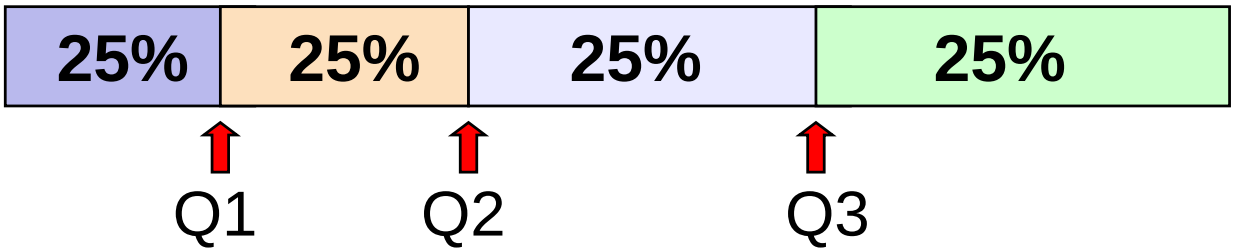
Example:



Interquartile range
= 57 – 30 = 27

Quartiles

- Quartiles split the ranked data into 4 segments with an equal number of values per segment



- The first quartile, Q_1, is the value for which 25% of the observations are smaller and 75% are larger
- Q_2 is the same as the median (50% are smaller, 50% are larger)
- Only 25% of the observations are greater than the third quartile

Formulae for Quartiles

Find a quartile by determining the value in the appropriate position in the ranked data, where

First quartile position: $\mathbf{Q_1 = 0.25(n+1)}$

Second quartile position: $\mathbf{Q_2 = 0.50(n+1)}$
(the median position)

Third quartile position: $\mathbf{Q_3 = 0.75(n+1)}$

where **n** is the number of observed values

Quartiles: example

- Example: Find the first quartile

Sample Ranked Data: **11 12 13 16 16 17 18 21 22**

(n = 9)

Q_1 = is in the 0.25(9+1) = 2.5 position of the ranked data

so use the value half way between the 2nd and 3rd values,

so **$Q_1 = 12.5$**

Quartiles in R

```
> x = 0:5 # 0,1,2,3,4,5 > length(x)
[1] 6
> median(x)
[1] 2.5
> quantile(x,.25)
25%
1.25
> quantile(x,c(0.25,0.5,0.75)) # more than 1 at a time
25% 50% 75%
1.25 2.50 3.75
> quantile(x) # default gives quartiles
0% 25% 50% 75% 100%
0.00 1.25 2.50 3.75 5.00
```

Difference between quartiles and proportions

```
> sum(exec.pay > 100)/length(exec.pay) # proportion more [1]
0.09045 # 9% make more than 1 million
> quantile(exec.pay,0.9) # 914,000 dollars is 90 percentile
90%
91.4
> quantile(exec.pay,0.99) # 9 million is top 1 percentile
997,906.6
> sum(exec.pay <= 10)/length(exec.pay)
[1] 0.1457 # 14 percent make 100,000 or less
> quantile(exec.pay,.10) # the 10 percentile is 90,000
10%
9
```

IQR in R

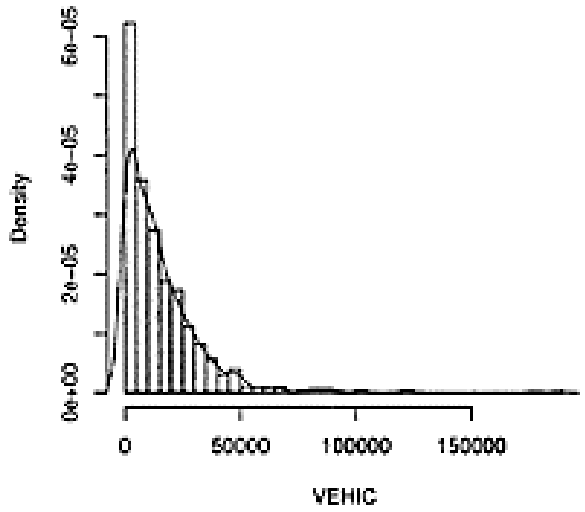
```
> IQR(exec.pay)
[1] 27.5
> sd(exec.pay)
[1] 207.0
```

Numeric summaries of the data in R

```
> summary(exec.pay)
Min.  1st Qu.  Median Mean 3rd Qu.  Max.
0.0 14.0 27.0 59.9 41.5 2510.0
```

Another example:

```
> attach(cfb) # it is a data frame
> summary(VEHIC)
Min.  1st Qu.  Median Mean 3rd Qu.  Max.
0 3880 11000 15400 21300 188000
> hist(VEHIC,breaks="Scott",prob=TRUE)
> lines(density(VEHIC))
> detach(cfb)
```



Summary statistics

- The function `summary()` provides a few summary statistics of the distribution of a data object
- Of course, the variable needs to be of type `numerical`

```
>summary(GSPC$ret.simple)
    Index              ret.simple
Min.   :1990-01-02   Min.   :-9.0350
1st Qu.:1996-09-30   1st Qu.:-0.4607
Median :2003-07-14   Median : 0.0489
Mean   :2003-07-13   Mean   : 0.0334
3rd Qu.:2010-04-22   3rd Qu.: 0.5628
Max.   :2017-02-01   Max.   :11.5800
                     NA's   :1
```

```
>summary(GSPC$ret.log)
    Index              ret.log
Min.   :1990-01-02   Min.   :-9.4695
1st Qu.:1996-09-30   1st Qu.:-0.4618
Median :2003-07-14   Median : 0.0489
Mean   :2003-07-13   Mean   : 0.0271
3rd Qu.:2010-04-22   3rd Qu.: 0.5612
Max.   :2017-02-01   Max.   :10.9572
                     NA's   :1
```

Package `fBasics`

- Package `fBasics` provides the `basicStats()` function with a more comprehensive set of descriptive statistics compared to `summary()`

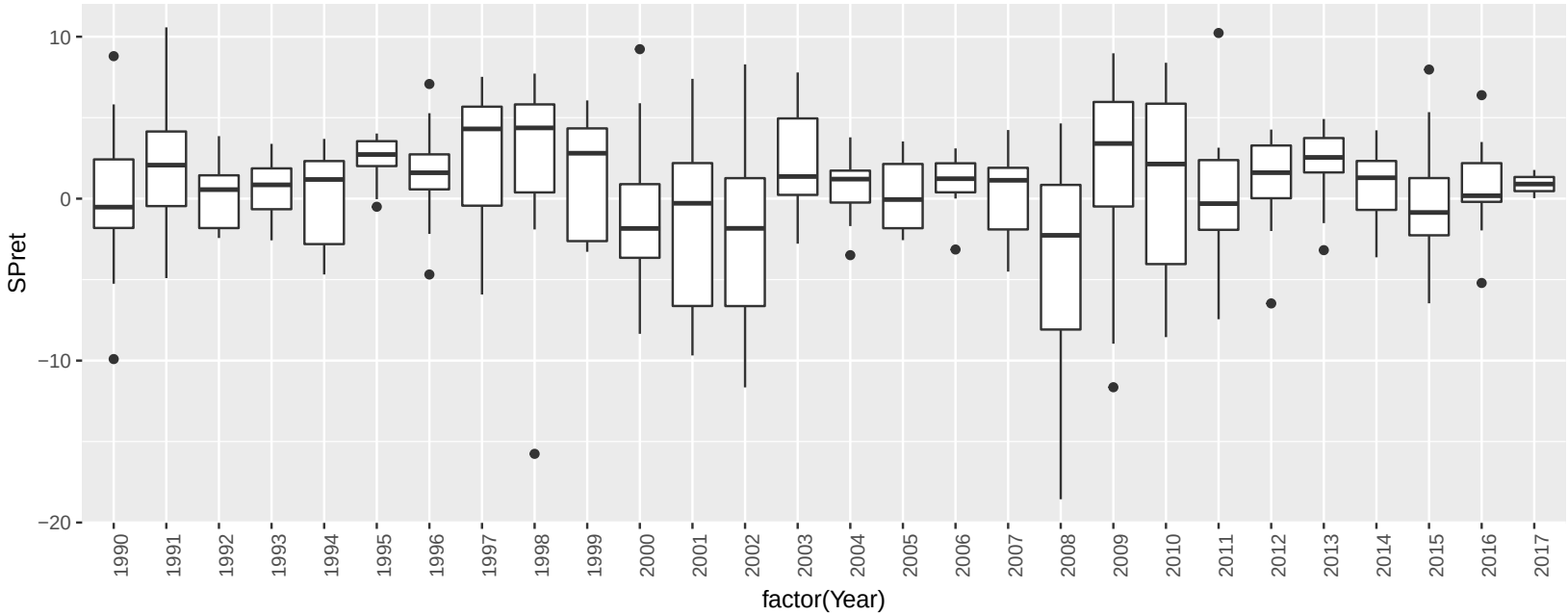
```
>fBasics::basicStats(GSPC$ret.log)
```

	ret.log
nobs	6826.000000
NAs	1.000000
Minimum	-9.469512
Maximum	10.957197
1. Quartile	-0.461792
3. Quartile	0.561214
Mean	0.027055
Median	0.048885
Sum	184.649080
SE Mean	0.013610
LCL Mean	0.000374
UCL Mean	0.053735
Variance	1.264258
Stdev	1.124392
Skewness	-0.244446
Kurtosis	8.704038

Boxplot

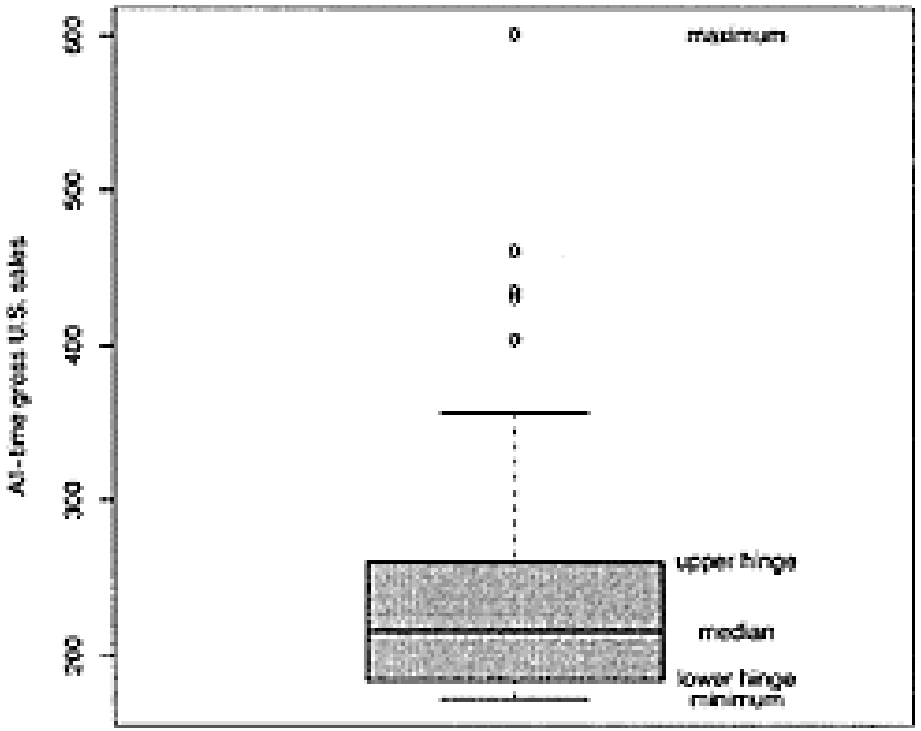
- A boxplot: the box represents the 25%, 50%, and 75% quantile and the whiskers extend to the highest value within 1.5 times the interquartile range; the dots are the min/max

```
>ggplot(GN.df, aes(factor(Year), SPret)) + geom_boxplot() +
theme(axis.text.x = element_text(angle = 90))
```



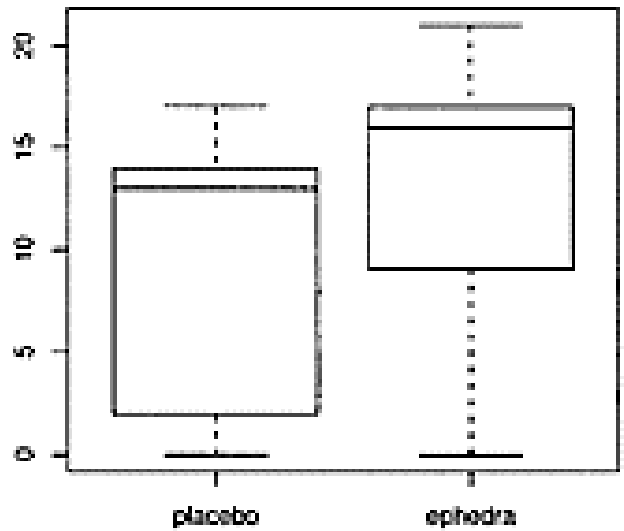
Boxplot: built-in function

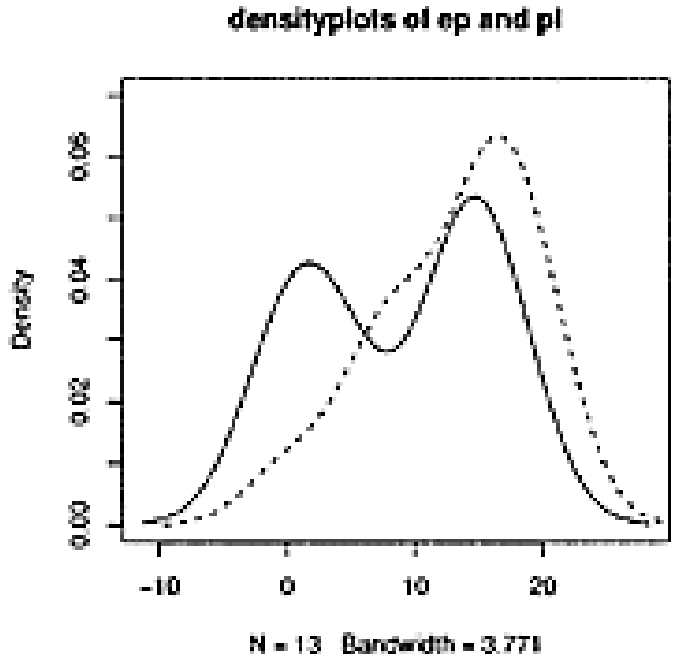
```
> attach(alltime.movies)
> out=boxplot(Gross,ylab="All-time gross sales") # object out
contains outliers !
> text(rep(1.3,5),f,labels=c("minimum","lower hinge", +
"median","upper hinge","maximum"))
```



Bivariate data: side-by-side boxplots

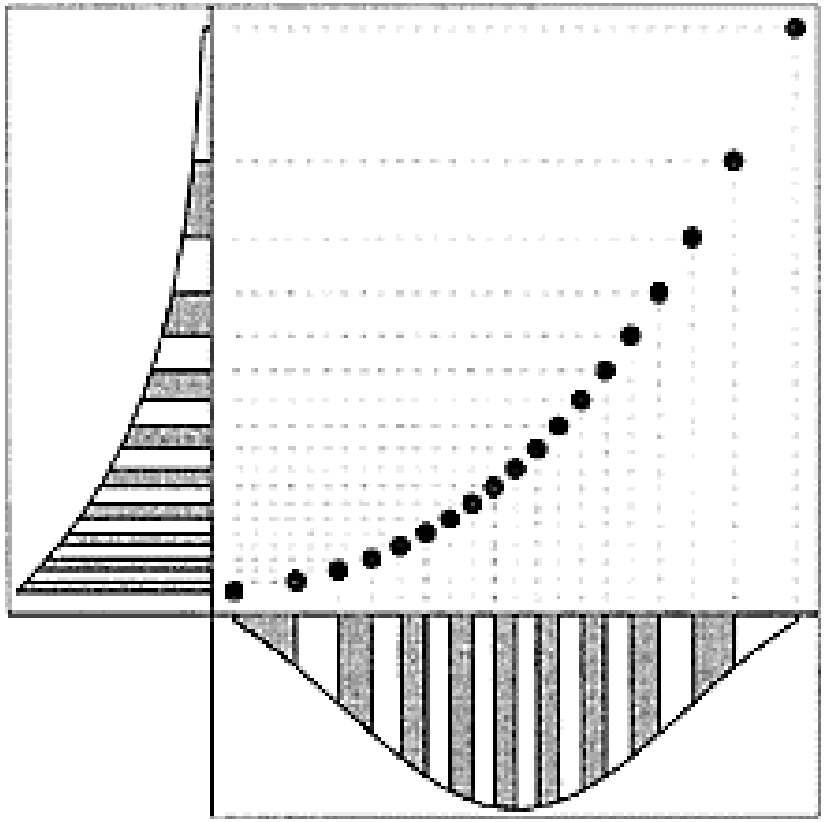
```
> pl = c(0, 0, 0, 2, 4, 5, 14, 14, 14, 13, 17, 17, 15)
> ep = c(0, 6, 7, 9, 11, 13, 16, 16, 16, 17, 18, 20, 21)
> boxplot(pl,ep, names=c("placebo","ephedra"))
> plot(density(pi),ylim=c(0,0.07), main="densityplots of ep and
pi")
> lines(density(ep), lty=2)
```





Bivariate data: quantile-quantile plots

- A quantile-quantile plot (q-q plot) plots the quantiles of one distribution against the quantiles of the other as points
- If the distributions have similar shapes, the points will fall roughly along a straight line.
- If the distributions are different, the points will not lie near a line revealing the domain(s) where the distributions differ.



q-q plots

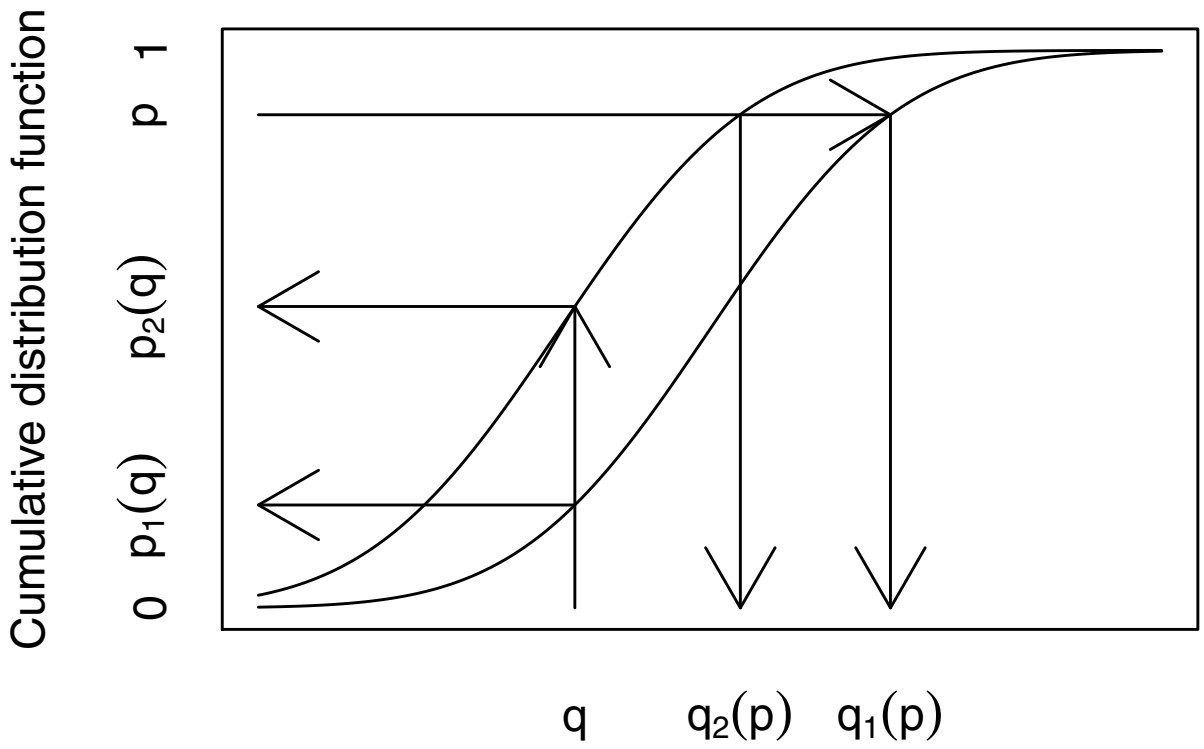
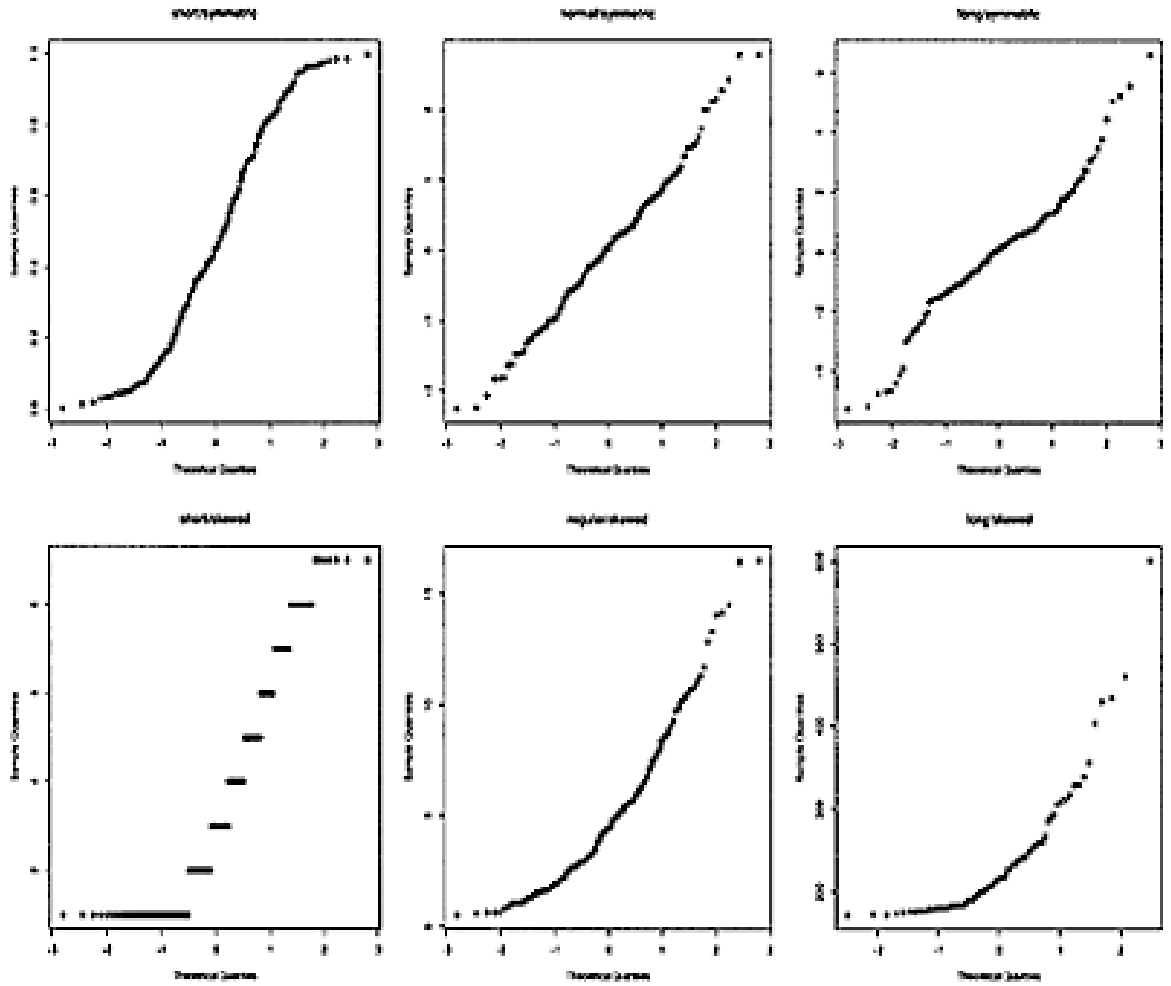


Figure: Quantiles $q_1(p)$, $q_2(p)$ coming from two different distributions

Normal quantile plot

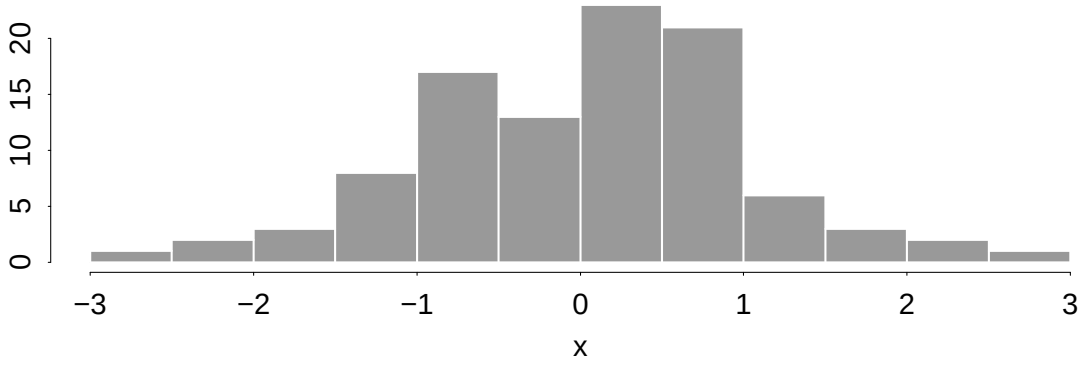
- A normal quantile plot depicts the quantiles of a data set against the quantiles of the Normal distribution.

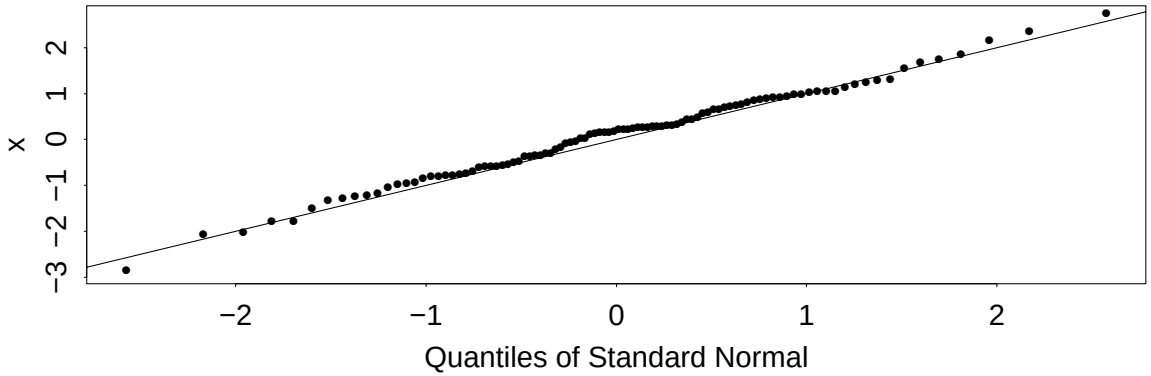


q-q plots in R

- The following R commands create the normal QQ-plot:

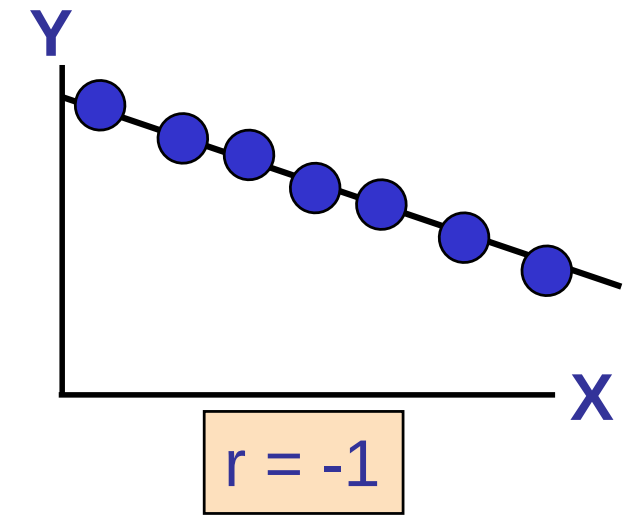
```
> x=rnorm(100)
> par(mfcol=c(2,1))
> hist(x)
> qqnorm(x) # built-in
function
> qqline(x) # straight
line
> prob=(c(1:100)-0.5)/100
> z=qnorm(prob) #
theoretical quantiles
> y=sort(x)
> cor(y,z)
```

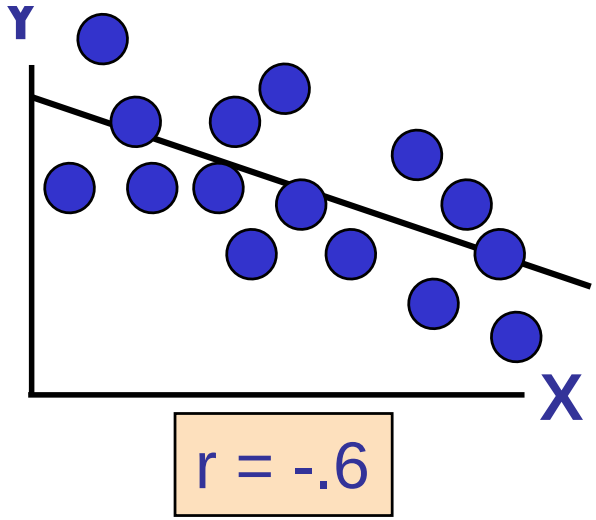


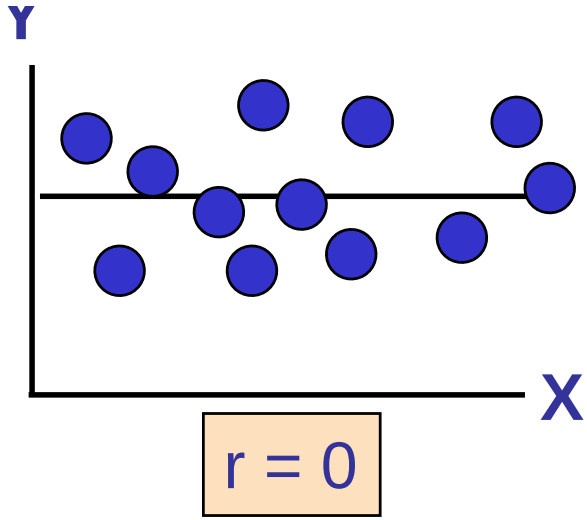


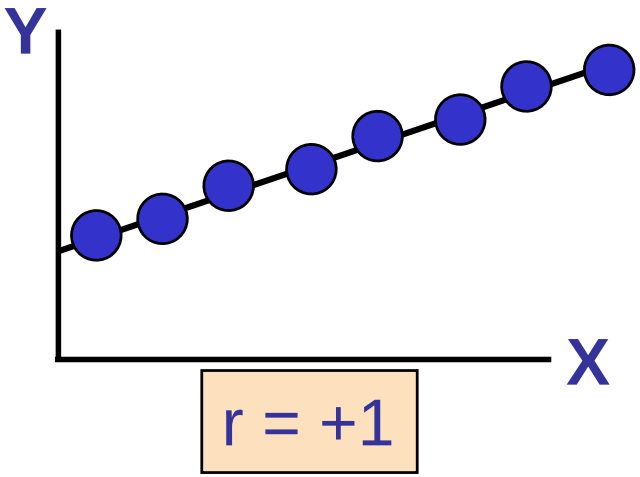
$$\text{cor(y,z)} = 0.9961$$

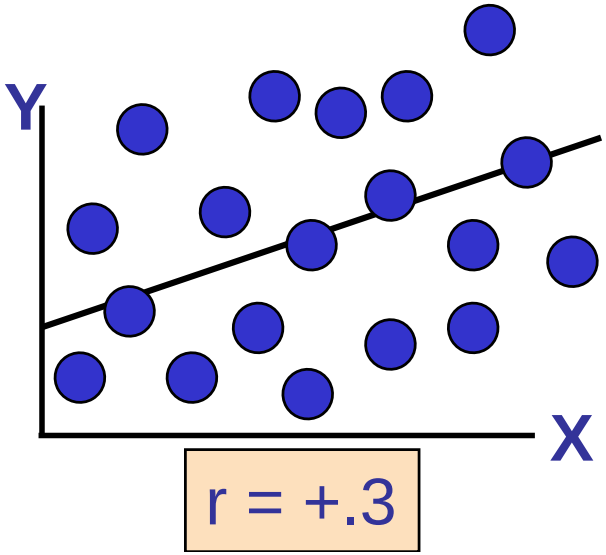
Scatter Plots of Bivariate Data

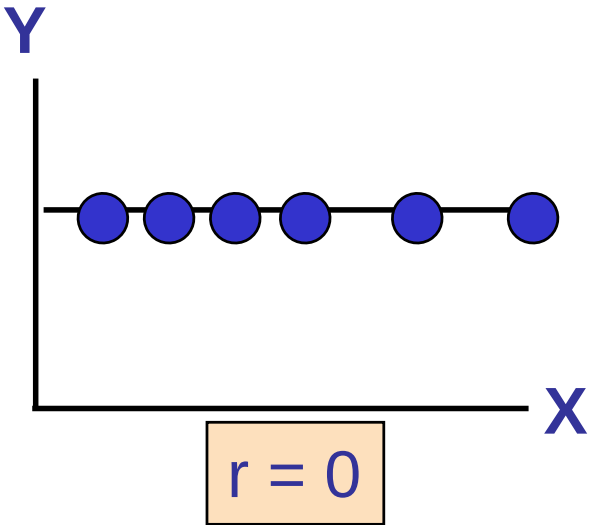












Scatter plots in R, base function

```
> attach(homedata)
> plot(y1970, y2000) # make the scatterplot
> detach(homedata) # tidy up
```

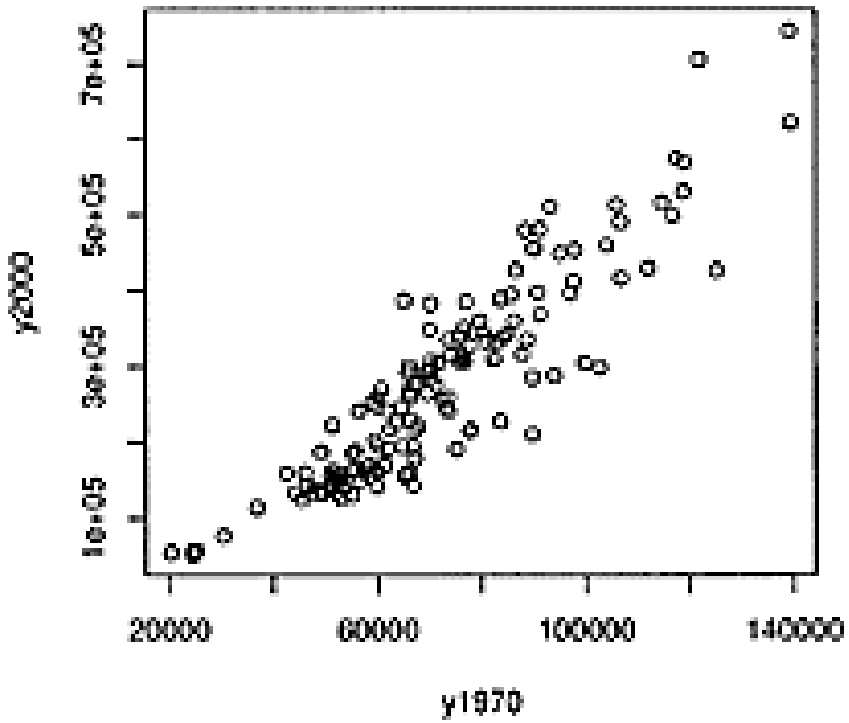
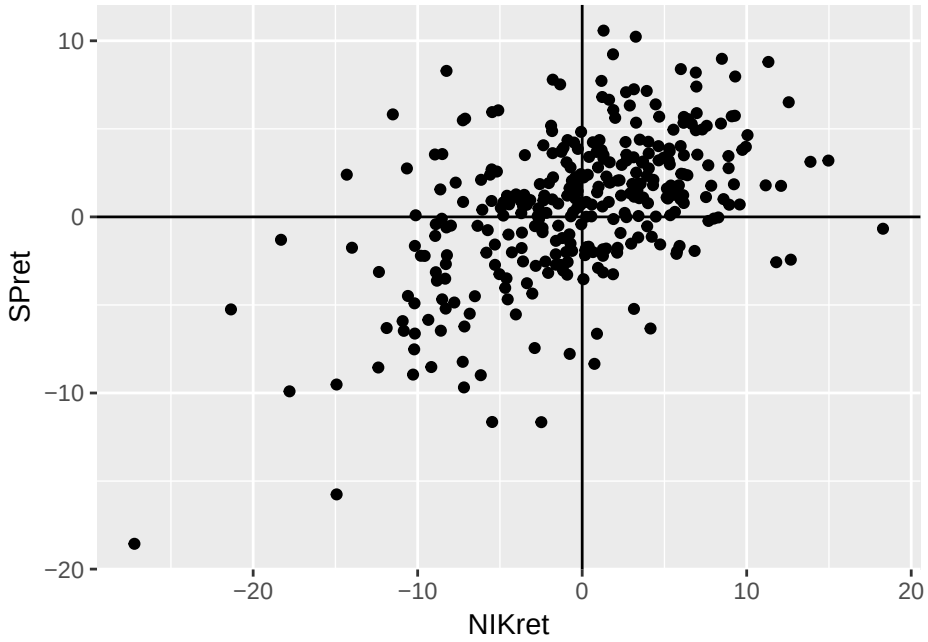


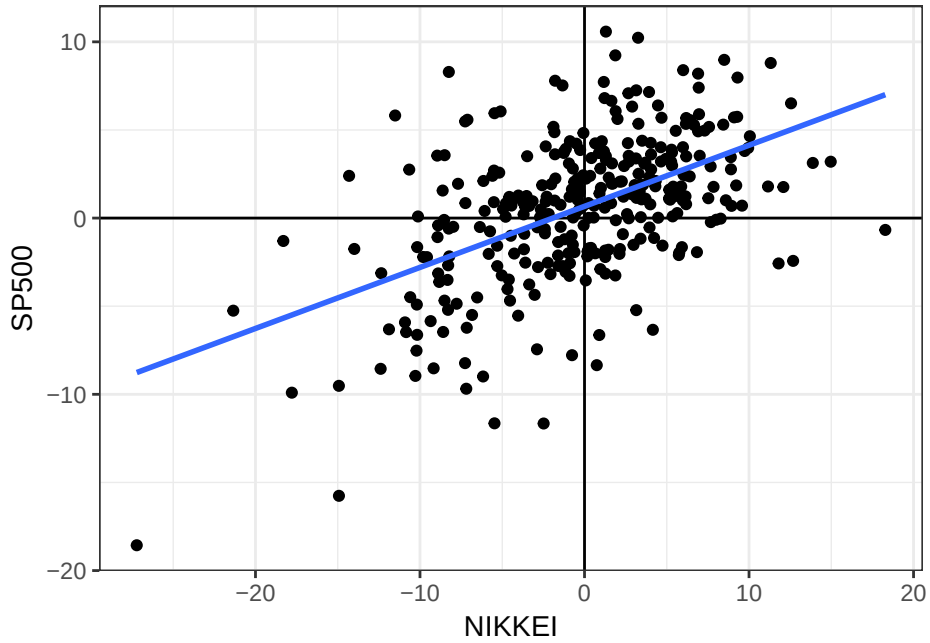
Figure 3.6 Assessed values of homes in Maplewood, N.J. in 1970 and 2000

Scatter plots by `ggplot2` package

- A scatter plot of two variables can be easily produced in `ggplot2`

```
>data <- getSymbols(c("GSPC", "N225"), from="1990-01-01")
>price <- merge(Ad(to.monthly(GSPC)), Ad(to.monthly(N225)))
>ret <- 100 * diff(log(price))
>GN.df <- data.frame(Date=time(price), Year = year(time(price)), coredata(merge(price,ret)))
>names(GN.df) <- c("Date","Year", "SP", "NIK", "SPret","NIKret")
>plot1 <- ggplot(GN.df, aes(NIKret, SPret)) + geom_point() + geom_vline(xintercept = 0) +
geom_hline(yintercept = 0)
>plot2 <- plot1 + geom_smooth(method="lm", se=FALSE) + theme_bw() + labs(x="NIKKEI", y="SP500")
>grid.arrange(plot1, plot2, ncol=2)
```

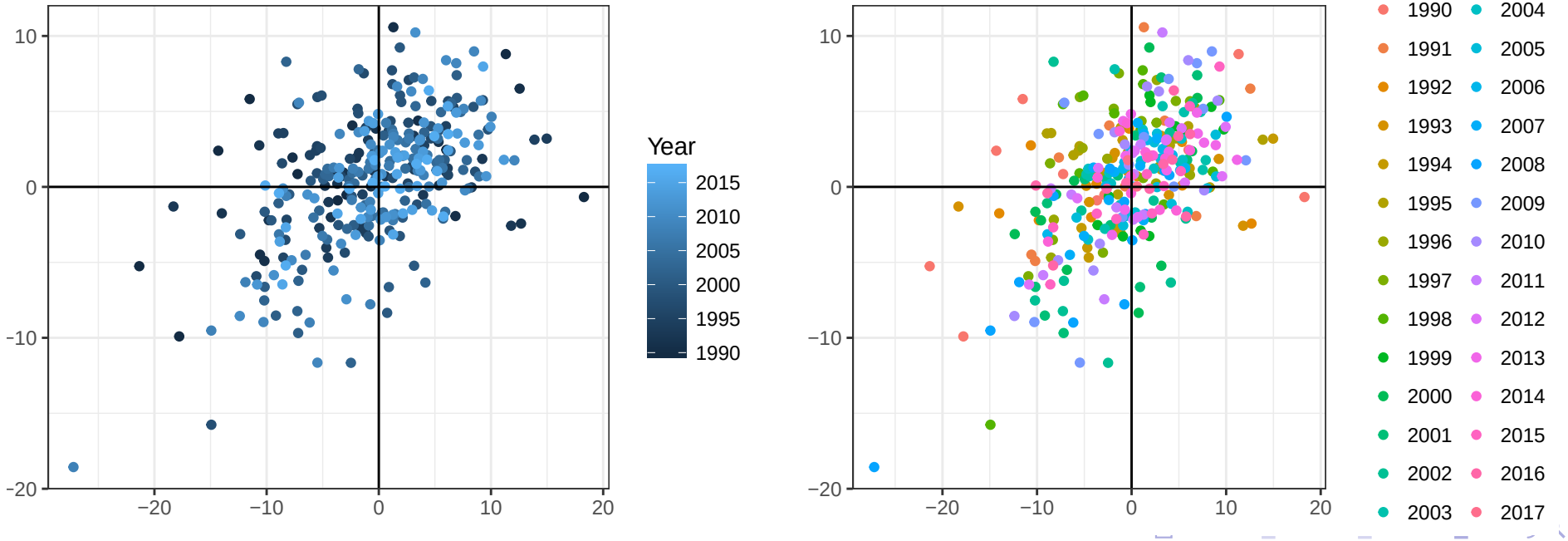




Scatter plots by `ggplot2` package

- More sophisticated graphics is available, e.g., we want to have the dots in the scatter plot depend on a variable (e.g., `Year`) this can be done easily by adding the argument `color=Year` in the aesthetics

```
>GN.df$Year <- year(time(price))
>plot1 <- ggplot(GN.df, aes(NIKret, SPret, color=Year)) + geom_point() + geom_vline(xintercept
= 0) + geom_hline(yintercept = 0) + theme_bw() + labs(x="", y="")
>plot2 <- ggplot(GN.df, aes(NIKret, SPret, color=factor(Year))) + geom_point() +
geom_vline(xintercept = 0) + geom_hline(yintercept = 0) + theme_bw() + labs(x="", y="")
>grid.arrange(plot1, plot2, ncol=2)
```



The Sample Covariance

- The covariance measures the strength of the linear relationship between **two variables**

- The population covariance:

$$\mathrm{Cov}(x,y) = \sigma_{xy} = \frac{\sum_{i=1}^{N}(x_i - \mu_x)(y_i - \mu_y)}{N}$$

- The sample covariance:

$$\mathrm{Cov}(x,y) = s_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{n-1}$$

  - Only concerned with the strength of the relationship
  - No causal effect is implied

Interpreting Covariance

- **Covariance** between two variables:

Cov(x,y) > 0 $\longrightarrow$ x and y tend to move in the same direction

Cov(x,y) < 0 $\longrightarrow$ x and y tend to move in opposite directions

Cov(x,y) = 0 $\longrightarrow$ x and y are independent

Pearson's Coefficient of Correlation

- Measures the relative strength of the linear relationship between two variables

- Population correlation coefficient:

$$\rho = \frac{\mathrm{Cov}(x, y)}{\sigma_X \sigma_Y}$$

- Sample correlation coefficient:

$$r = \frac{\mathrm{Cov}(x, y)}{s_X s_Y}$$

Properties of Correlation Coefficient

- Unit free
- Ranges between –1 and 1
- The closer to –1, the stronger the negative linear relationship
- The closer to 1, the stronger the positive linear relationship
- The closer to 0, the weaker any positive linear relationship

For `R` command for computing coefficient of correlation, see Slide 108.

Spearman's Coefficient of Correlation

- The data are ordered from smallest to largest, and a data point's rank is its position after sorting
- The Spearman rank correlation is the Pearson correlation coefficient computed with the ranked data.

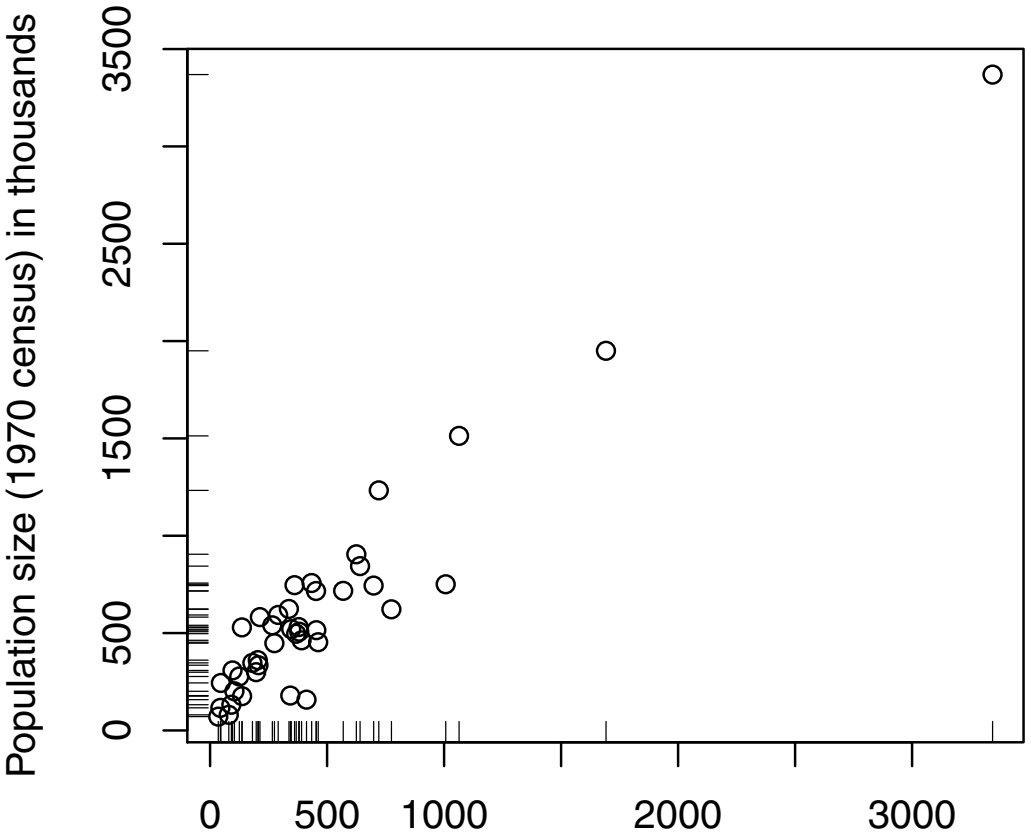
```
# Dow Jones example, r = 0.01029
> cor(max.temp,DJA,method="spearman")
[1] 0.1316
```

Example of multivariate data

Table 1.5: `USairpollution` data. Air pollution in 41 US cities.

	SO2	temp	manu	popul	wind	precip	predays
Albany	46	47.6	44	116	8.8	33.36	135
Albuquerque	11	56.8	46	244	8.9	7.77	58
Atlanta	24	61.5	368	497	9.1	48.34	115
Baltimore	47	55.0	625	905	9.6	41.31	111
Buffalo	11	47.1	391	463	12.4	36.11	166
Charleston	31	55.2	35	71	6.5	40.75	148
Chicago	110	50.6	3344	3369	10.4	34.44	122
Cincinnati	23	54.0	462	453	7.1	39.04	132
Cleveland	65	49.7	1007	751	10.9	34.99	155
Columbus	26	51.5	266	540	8.6	37.01	134
Dallas	9	66.2	641	844	10.9	35.94	78
Denver	17	51.9	454	515	9.0	12.95	86
Des Moines	17	49.0	104	201	11.2	30.85	103
Detroit	35	49.9	1064	1513	10.1	30.96	129

Plotting joint and marginal distributions together

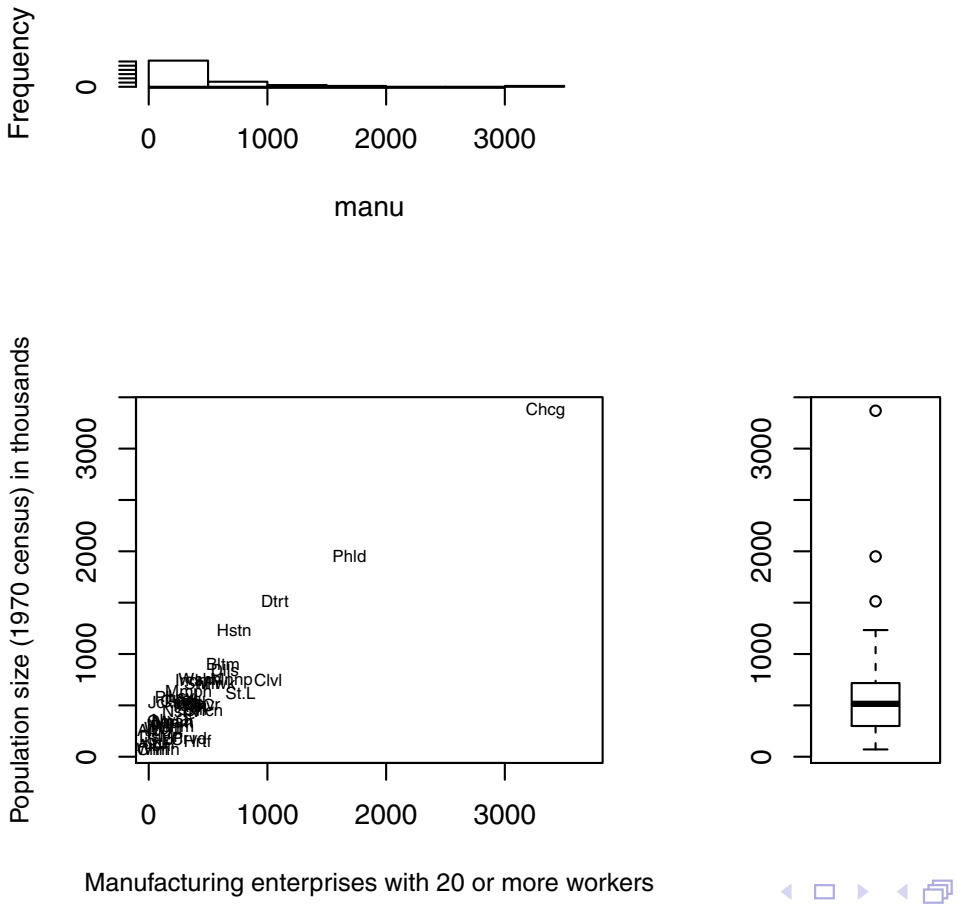


- Plotting marginal and joint distributions together is usually good data analysis practice
- The marginal distributions are shown as rug plots on each axis (produced by `rug()`)

```
> rug(USairpollution$manu, side = 1)
> rug(USairpollution$popul, side = 2)
```

Plotting joint and marginal distributions together (2)

- Here the marginal distribution of `manu` is given as a histogram and that of `popul` as a boxplot.



Plotting joint and marginal distributions together (3)

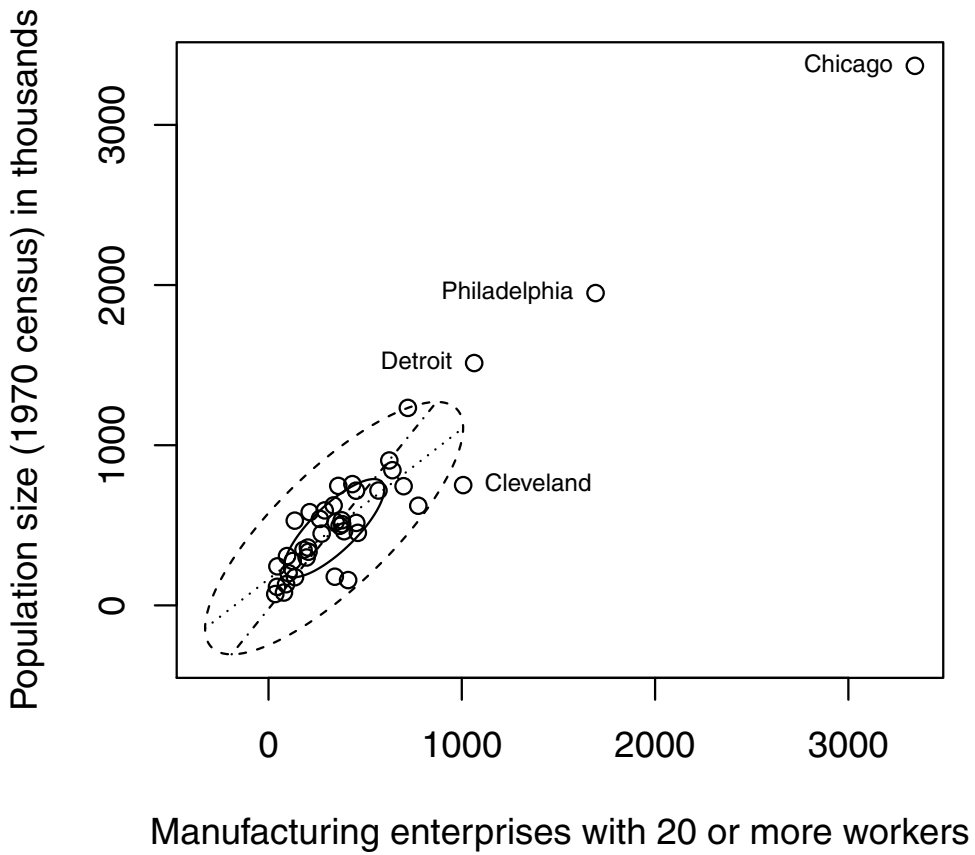
- Code for plots in Slide 120

```
> layout(matrix(c(2, 0, 1, 3), nrow = 2, byrow = TRUE),
+ widths = c(2, 1), heights = c(1, 2), respect = TRUE)
> xlim <- with(USairpollution, range(manu)) * 1.1
> plot(popul ~ manu, data = USairpollution, cex.lab = 0.9,
+ xlab = mlab, ylab = plab, type = "n", xlim = xlim)
> with(USairpollution, text(manu, popul, cex = 0.6,
+ labels = abbreviate(row.names(USairpollution))))
> with(USairpollution, hist(manu, main = "", xlim = xlim))
> with(USairpollution, boxplot(popul))
```

- Here the marginal distribution of `manu` is given as a histogram and that of `popul` as a boxplot
- The `with()` command allows to avoid extracting variables from the data frames.
- The calls to `hist()` and `boxplot()` are evaluated inside the data frame

Bivariate boxplot

- A two-dimensional analogue of boxplot for univariate data
- Consists of a pair of concentric ellipses, one of which (the “hinge”) includes 50% of the data and the other (called the “fence”) delineates outliers
- Regression lines of both y on x and x on y are shown
- The acute angle between the regression lines is small for large correlation and vice versa.



Bivariate boxplot: output

- Cleveland, Detroit, Philadelphia and Chicago are outliers
- The angle between the regression lines is small => the correlation is strong enough

```
> outcity <- match(lab <- c("Chicago", "Detroit",
+ "Cleveland", "Philadelphia"), rownames(USairpollution))
> x <- USairpollution[, c("manu", "popul")]
> bvbox(x, mtitle = "", xlab = mlab, ylab = plab)
> text(x$manu[outcity], x$popul[outcity], labels = lab,
+ cex = 0.7, pos = c(2, 2, 4, 2, 2))
```

Command `bvbox` comes from `MVA` package!

Bivariate boxplot: removing outliers

- Correlation when Cleveland, Detroit, Philadelphia and Chicago are in:

```
> with(USairpollution, cor(manu, popul))
[1] 0.9553
```

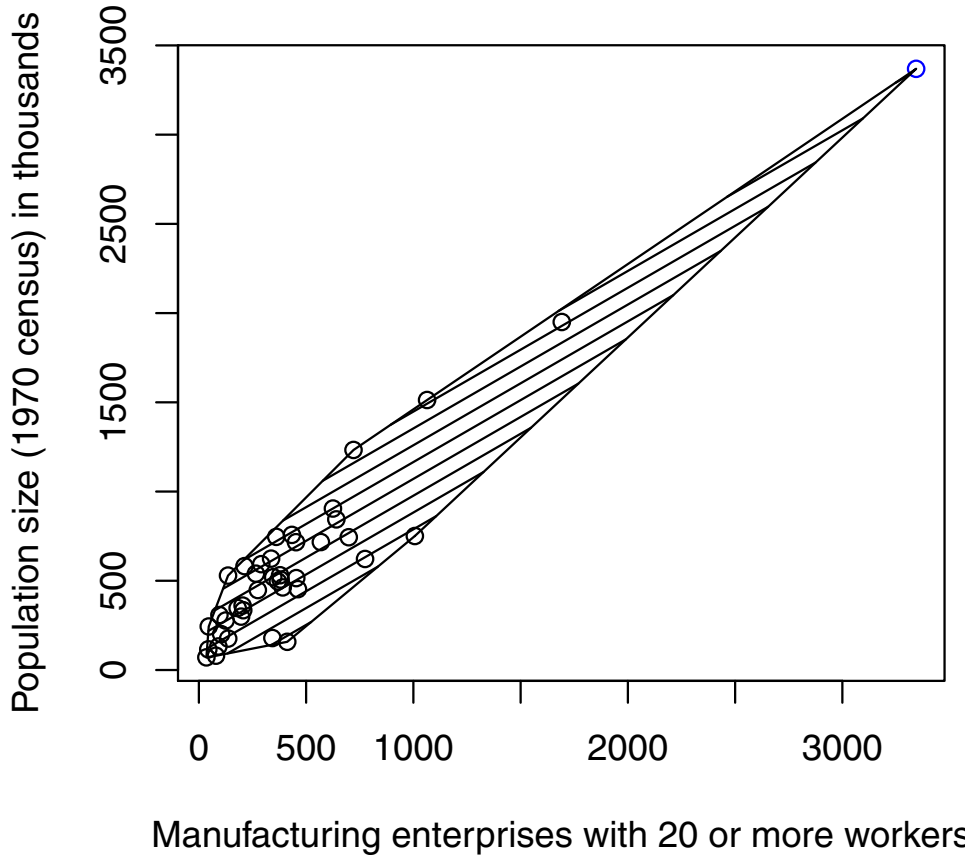
- Correlation when Cleveland, Detroit, Philadelphia and Chicago are out:

```
> outcity <- match(c("Chicago", "Detroit",
+ "Cleveland", "Philadelphia"),
+ rownames(USairpollution))
> with(USairpollution, cor(manu[-outcity], popul[-outcity]))
[1] 0.7956
```

- The `match()` function identifies rows of the data frame `USairpollution` corresponding to the cities of interest

The convex hull of the bivariate data

- Another approach to deal with the problem of calculating correlation without the distortion caused by outliers.
- The convex hull of a set of bivariate observations consists of the vertices of the smallest convex polyhedron in variable space within which all data points lie.
- Removal of the points lying **on the convex hull** can eliminate isolated outliers while preserving the general shape of the bivariate distribution



Convex hull: code

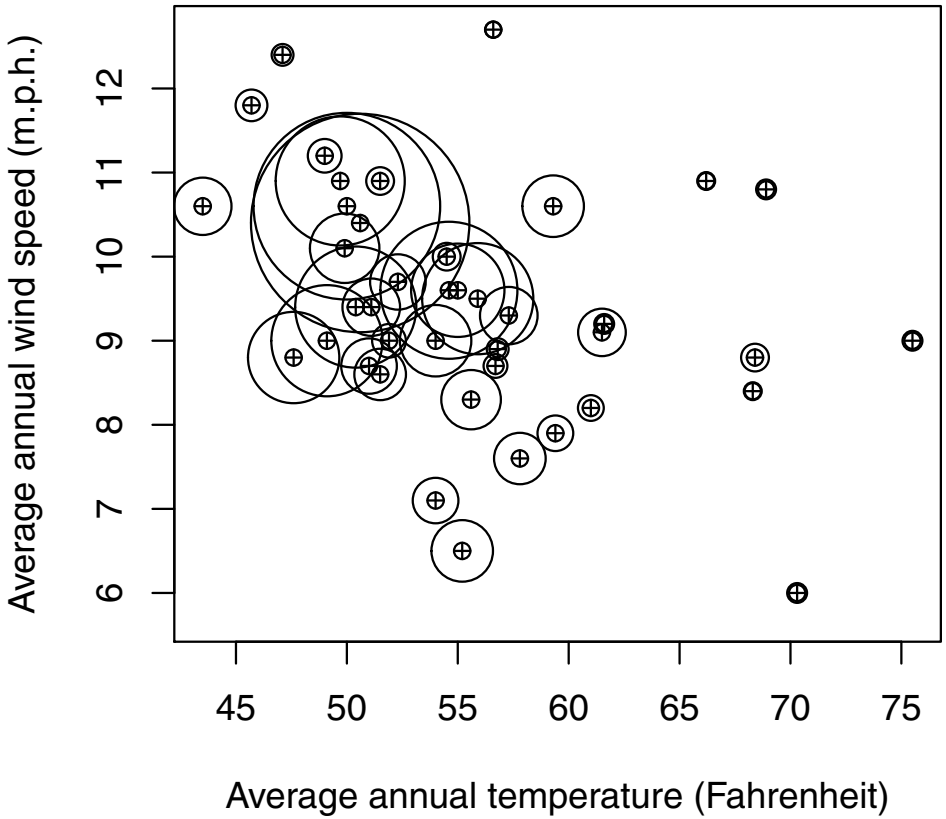
```
> (hull <- with(USairpollution, chull(manu, popul)))
[1] 9 15 41 6 2 18 16 14 7
> with(USairpollution,
+ plot(manu, popul, pch = 1, xlab = mlab, ylab = plab))
> with(USairpollution,
+ polygon(manu[hull], popul[hull], density = 15, angle = 30))
```

Command `chull` comes from `MVA` package!

- Removing points defining the convex hull:

```
> with(USairpollution, cor(manu[-hull],popul[-hull]))
[1] 0.9225
```

Three variables: bubble plot



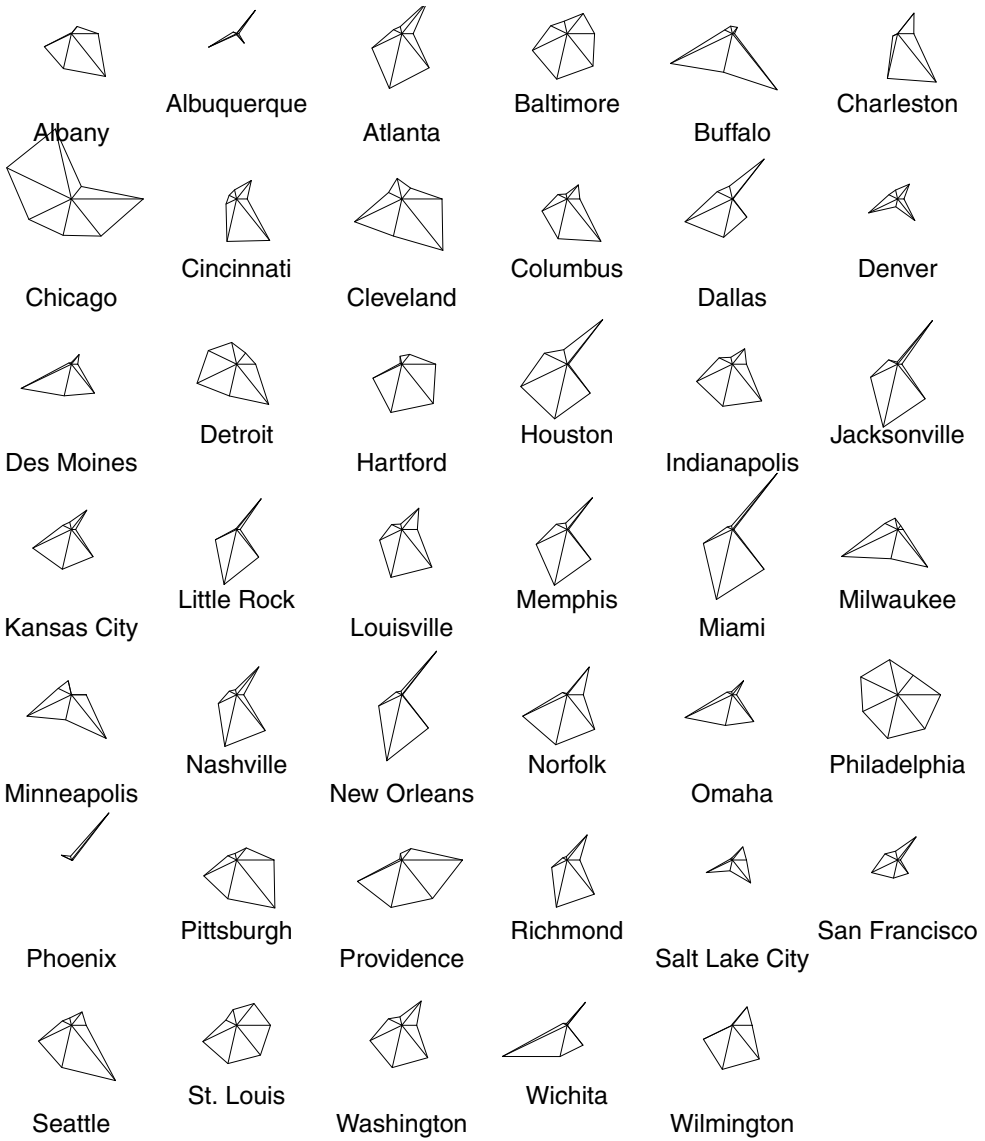
- Three variables are displayed.
- The two are used to form the scatterplot.
- The third variable is represented by circles with radii proportional to its values and centered at the appropriate point of the scatterplot.

Bubble plot: script

- Cities with moderate annual temperatures and annual wind speeds tend to suffer the greatest air pollution.
- Code producing the figure in Slide 127:

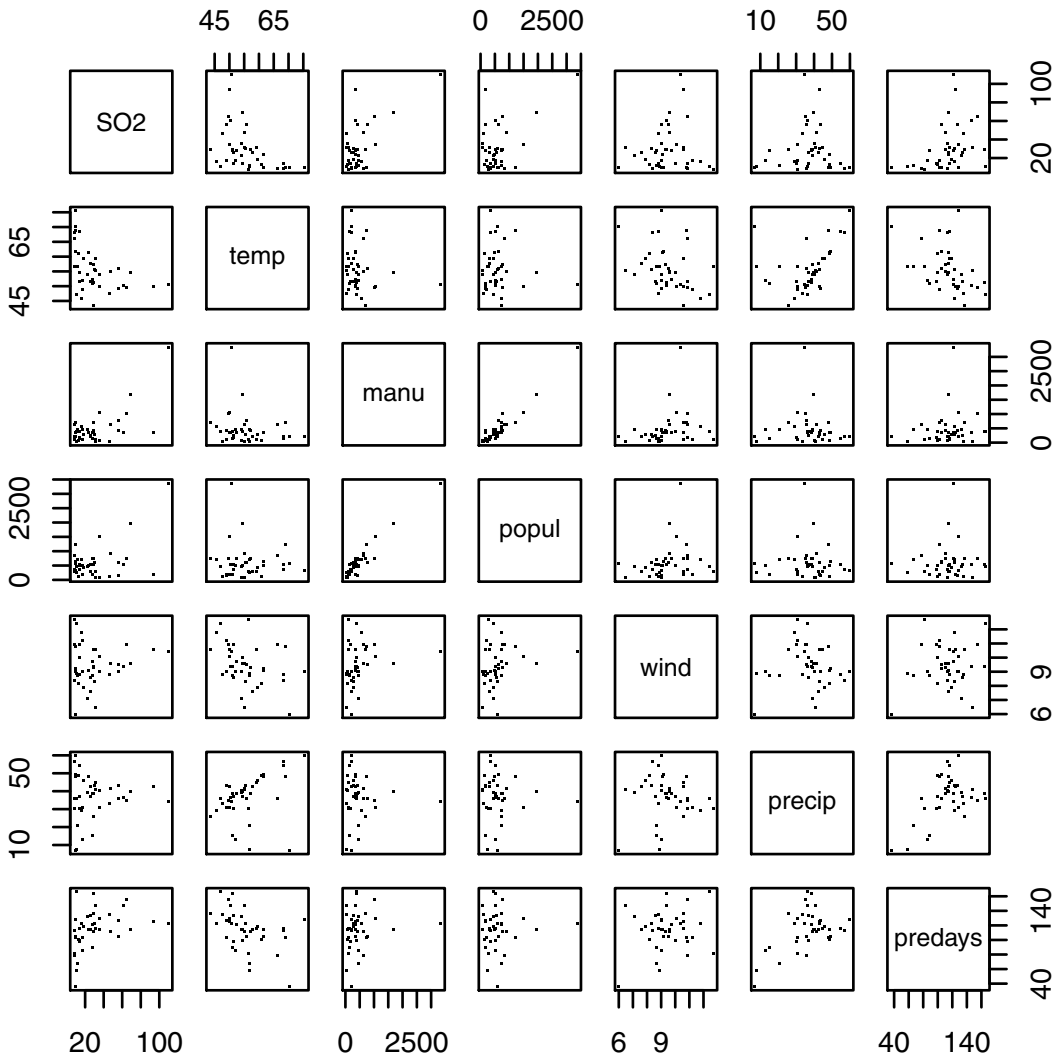
```
> ylim <- with(USairpollution, range(wind)) * c(0.95, 1)
> plot(wind   temp, data = USairpollution,
+ xlab = "Average annual temperature (Fahrenheit)",
+ ylab = "Average annual wind speed (m.p.h.)", pch = 10,
+ ylim = ylim)
> with(USairpollution, symbols(temp, wind, circles = SO2,
+ inches = 0.5, add = TRUE))
```

Multiple variables: glyph plot



- The seven variables for each city are represented as a seven-sided star.
- Some stars, e.g., New Orleans, Miami, Jacksonville, and Atlanta, have similar shapes.
- Yet hard to observe and interpret.
- `> stars(USairpollution, cex = 0.55)`
- `stars` comes from `MVA` package.

Scatter plot matrix



- Sulphur dioxide concentration is the response variable, the remaining variables are explanatory
- A strong linear relationship between `SO2` and `manu` and between `SO2` and `popul`, but the (3; 4) panel shows that `manu` and `popul` are themselves very highly related and thus predict `SO2` in the same way

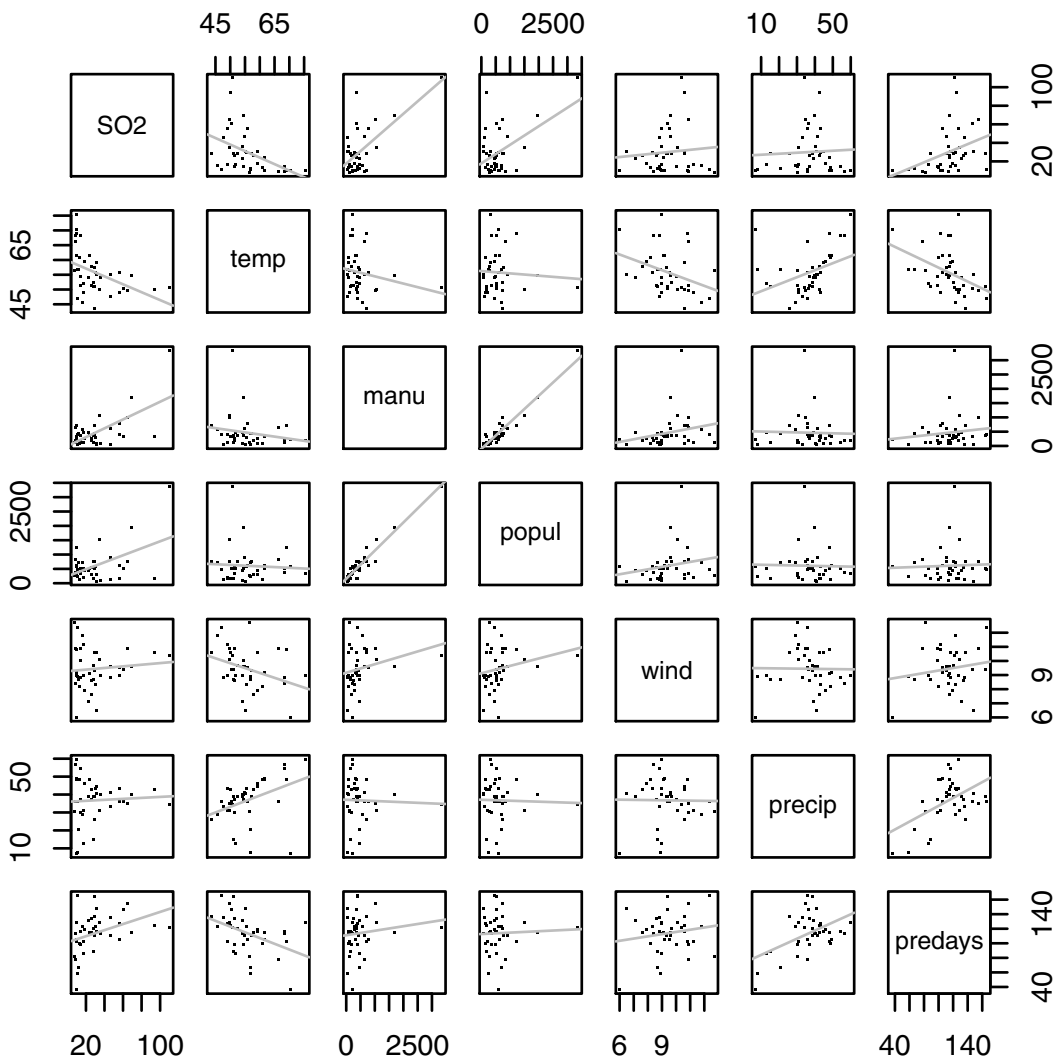
Scatter plot matrix: analysis

- Correlation matrix:

```
> round(cor(USairpollution), 4)
                SO2    temp    manu   popul    wind  precip predays
SO2          1.0000 -0.4336  0.6448  0.4938  0.0947  0.0543  0.3696
temp        -0.4336  1.0000 -0.1900 -0.0627 -0.3497  0.3863 -0.4302
manu         0.6448 -0.1900  1.0000  0.9553  0.2379 -0.0324  0.1318
popul        0.4938 -0.0627  0.9553  1.0000  0.2126 -0.0261  0.0421
wind         0.0947 -0.3497  0.2379  0.2126  1.0000 -0.0130  0.1641
precip       0.0543  0.3863 -0.0324 -0.0261 -0.0130  1.0000  0.4961
predays      0.3696 -0.4302  0.1318  0.0421  0.1641  0.4961  1.0000
```

- Correlation between `SO2` and `precip` is very small and that for `SO2` and `predays` is moderate. But the respective panels of the scatterplot indicate that the Pearson's correlation coefficient might assess only the linear relationship!

Scatter plot matrix: regression lines added




```
> pairs(USairpollution,
+ panel = function (x, y, ...)
{
+ points(x, y, ...)
+ abline(lm(y   x), col =
"grey")
+ }, pch = ".", cex = 1.5)
```

Acknowledgments

Slides are partially adapted from **Sebastiano Manzan** and from those accompanying Newbold's textbook

References

See Chapters 1-4 of [1] and Chapter 2 of [2] for more.

[1] J. Verzani.
Using R for Introductory Statistics, Second Edition.
Chapman & Hall/CRC The R Series. Taylor & Francis, 2014.

[2] Brian Everitt and Torsten Hothorn.
An introduction to applied multivariate analysis with R.
Springer, New York, 2011.