Lecture 08: Sep 17, 2018

Linear Regression

- *SLR*
- *MLR*
- *Factors and Design Matrices*

James Balamuta
STAT 385 @ UIUC

Announcements

- **hw03** is due on **Friday, Sep 21st, 2018 @ 6:00 PM**
- **Quiz 04** covers Week 3 contents @ CBTF.
  - Window: Sep 18th - 20th
  - Sign up: https://cbtf.engr.illinois.edu/sched
- **hw01** grade reports released on GitHub.
  - Post on forum detailing how to interpret the grade reports.
  - Got caught using GitHub's web interface? Let's chat.

Last Time

- **Data Structures**
  - 1*D*, 2*D*, and n *D*imensions
  - Homogenous (Same) vs. Heterogenous (Different/Mixed)
- **Coercion**
  - Changing data from one form to the another either implicitly (R) or explicitly (You).
- **Missingness and NULL**
  - The lack of recorded data vs. the lack of an object being created.

Lecture Objectives

- **Fitting** a linear regression model with data
- Constructing **Design Matrices** associated with MLR
- Benefits associated with **using factors**.

SLR

Previously

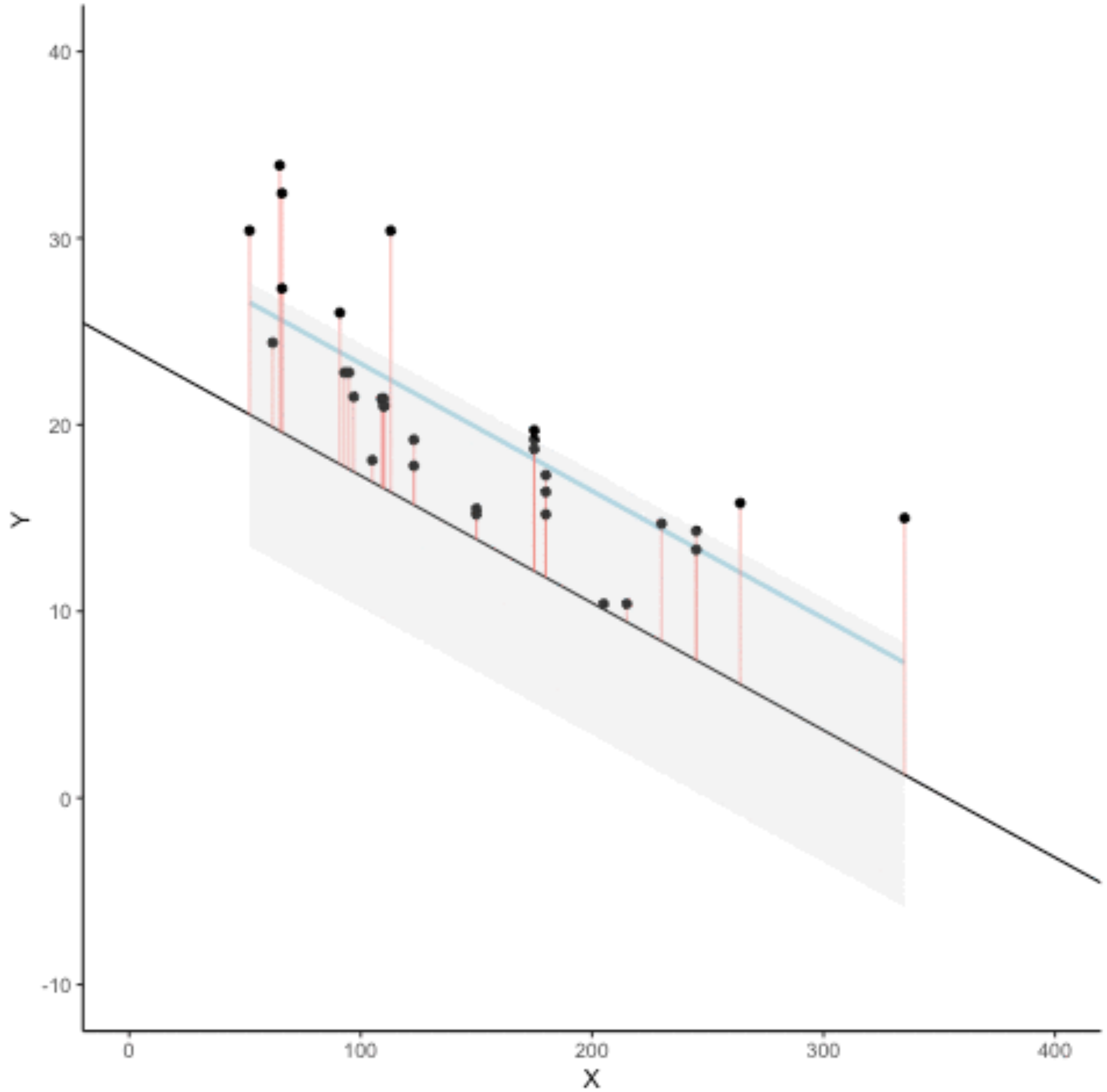
Understand the Algorithm

… weeks of programming saved hours of planning …

- *What* logic is being used?
- *How* does the logic apply in a procedural form?
- *Why* is this logic present?

SLR in Practice

… finding the line of best fit between two variables …

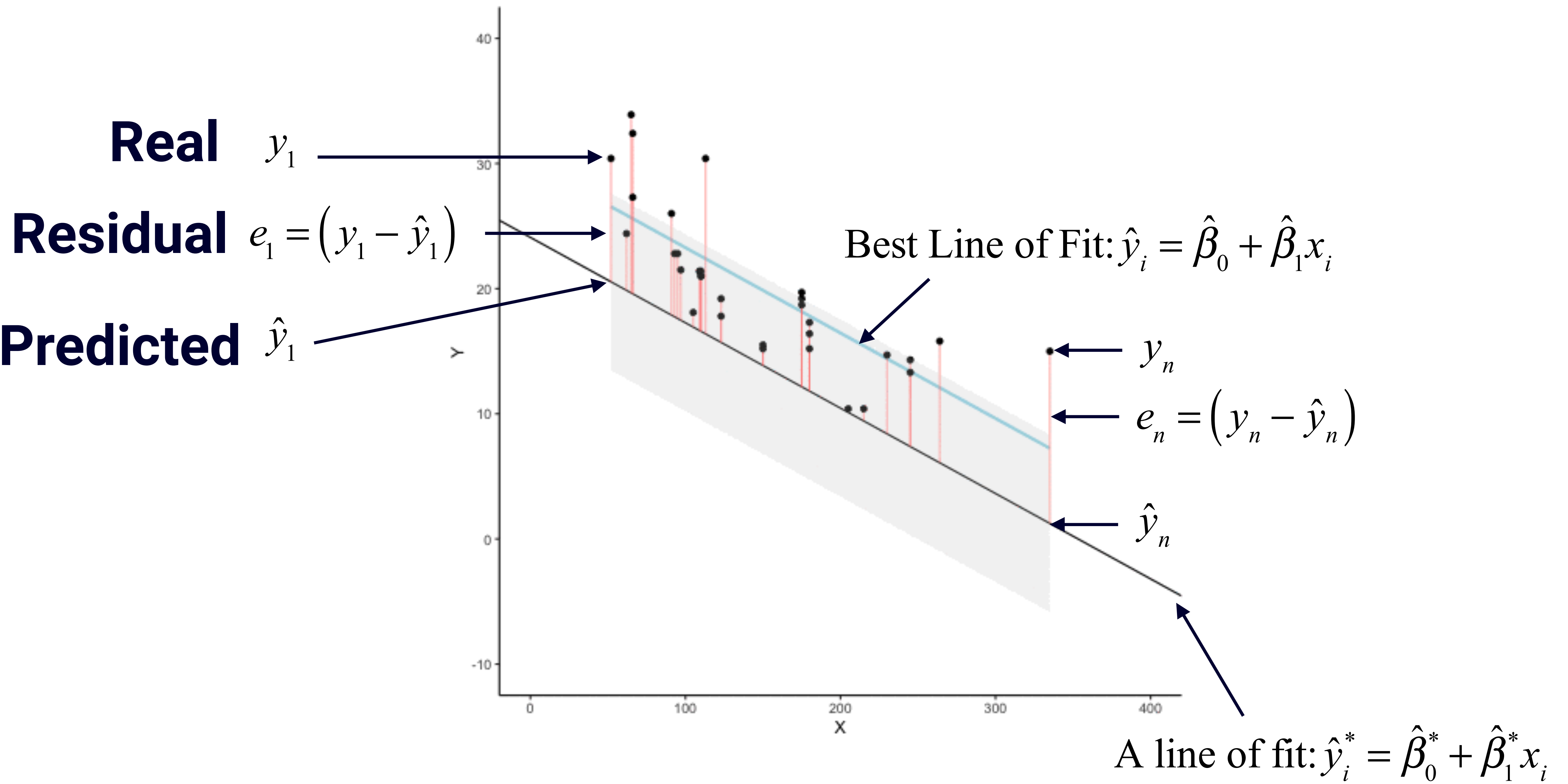


Source

*   The **blue** line represents the optimal line of best fit.
**   The **black** line represents the current line of fit.
*** The **red** lines represent distance from points. The goal is to *minimize* these values.

SLR Labeled

... graph components ...



Simple Linear Regression

... handling two parameters ...

Scalar-form

$$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i$$

Expand Matrix

$$\underbrace{\begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}_{n\times 1}}_{\text{Responses}} = \underbrace{\begin{pmatrix} 1 & x_{1,1} \\ \vdots & \vdots \\ 1 & x_{n,1} \end{pmatrix}_{n\times 2}}_{\text{Design Matrix}} \underbrace{\begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix}_{2\times 1}}_{\text{Parameters}} + \underbrace{\begin{pmatrix} \varepsilon_1 \\ \vdots \\ \varepsilon_n \end{pmatrix}_{n\times 1}}_{\text{Error}}$$

Matrix-form

$$Y_{n\times 1} = X_{n\times 2}\beta_{2\times 1} + \varepsilon_{n\times 1}$$

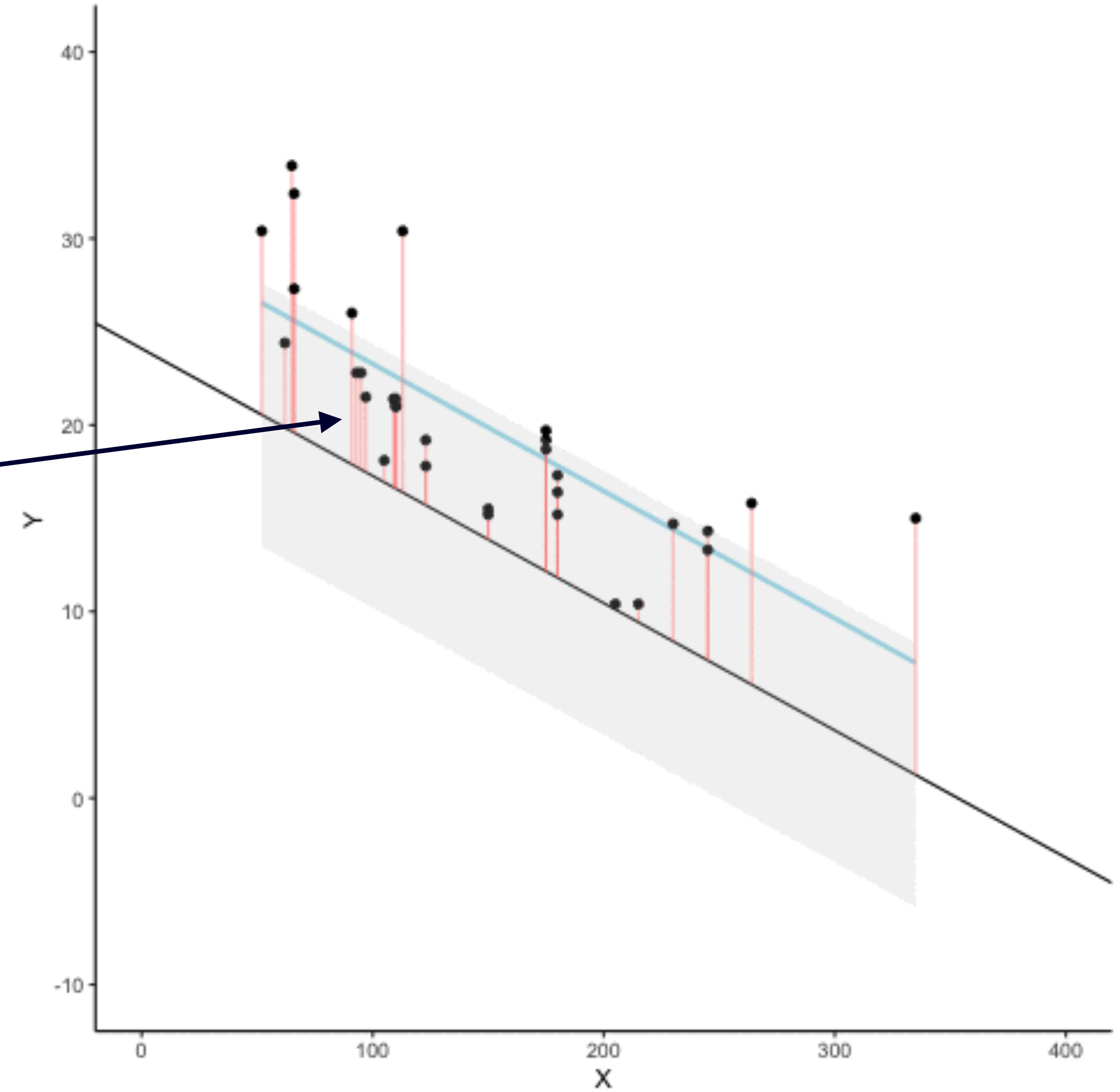
---

n is the number of observations, there are 2 variables, ***X*** provides the design matrix for the variables, ***y*** is response vector, β is the parameter or coefficient vector and $\boldsymbol{\varepsilon}$ is the random error vector

Estimating Parameters

… goal is to **minimize the *Residual* *Sum* *Squared*** (*RSS*/red lines) …

$$\hat{\beta} = \underset{\beta_0,\beta_1 \in \mathbb{R}}{\arg\min} \sum_{i=1}^{n} \left(y_i - \underbrace{\left(\beta_0 + \beta_1 x_i\right)}_{\hat{y}_i} \right)^2$$



Analytical Solutions

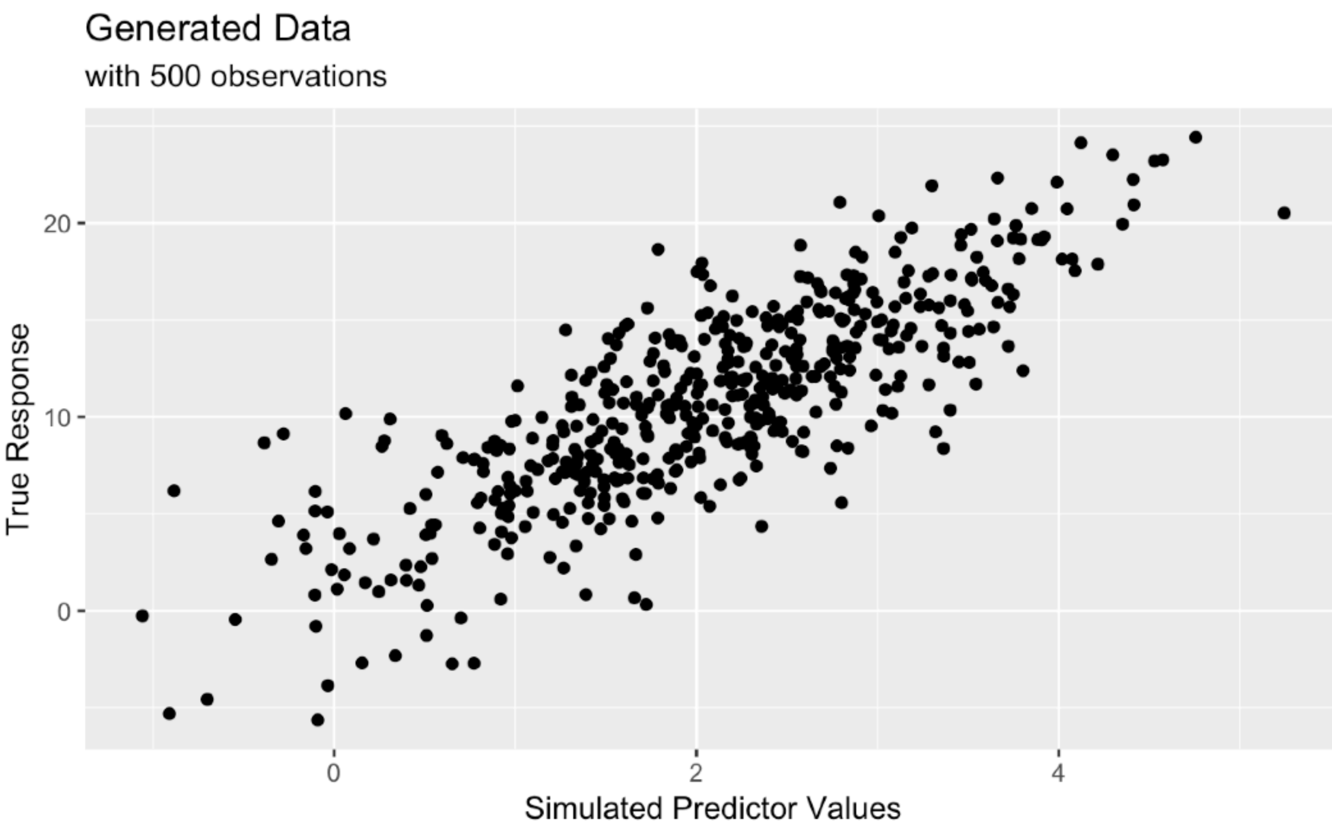
$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$$

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n} (x_i - \bar{x})^2} = \frac{S_{xy}}{S_{xx}}$$

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

Simulating Data for SLR

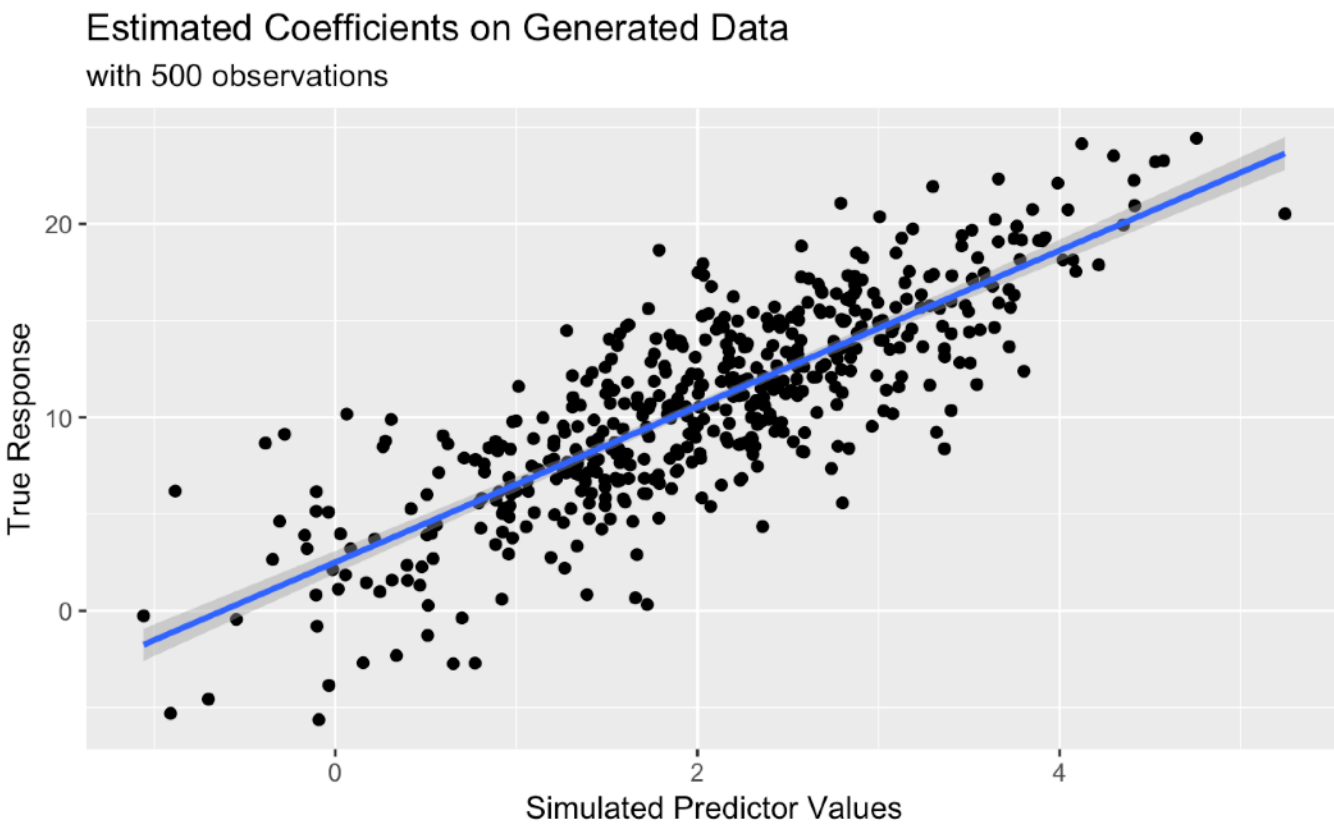
… generating a test data set …




```
# Set seed for reproducibility
set.seed(9123)

# Number of observations
n = 500

# Generate a single predictor
x_i = rnorm(n, mean = 2)

# Design matrix w/ intercept: n x 2
X = cbind(1, x_i)

# True beta values: 2 x 1
beta = c(2.5, 4)

# Response variable with error: n x 1
y = X[, 1] * beta[1] +
    X[, 2] * beta[2] +
    rnorm(n, sd = 2) # error term
```

Analytical SLR Solution

... solving using equations ...

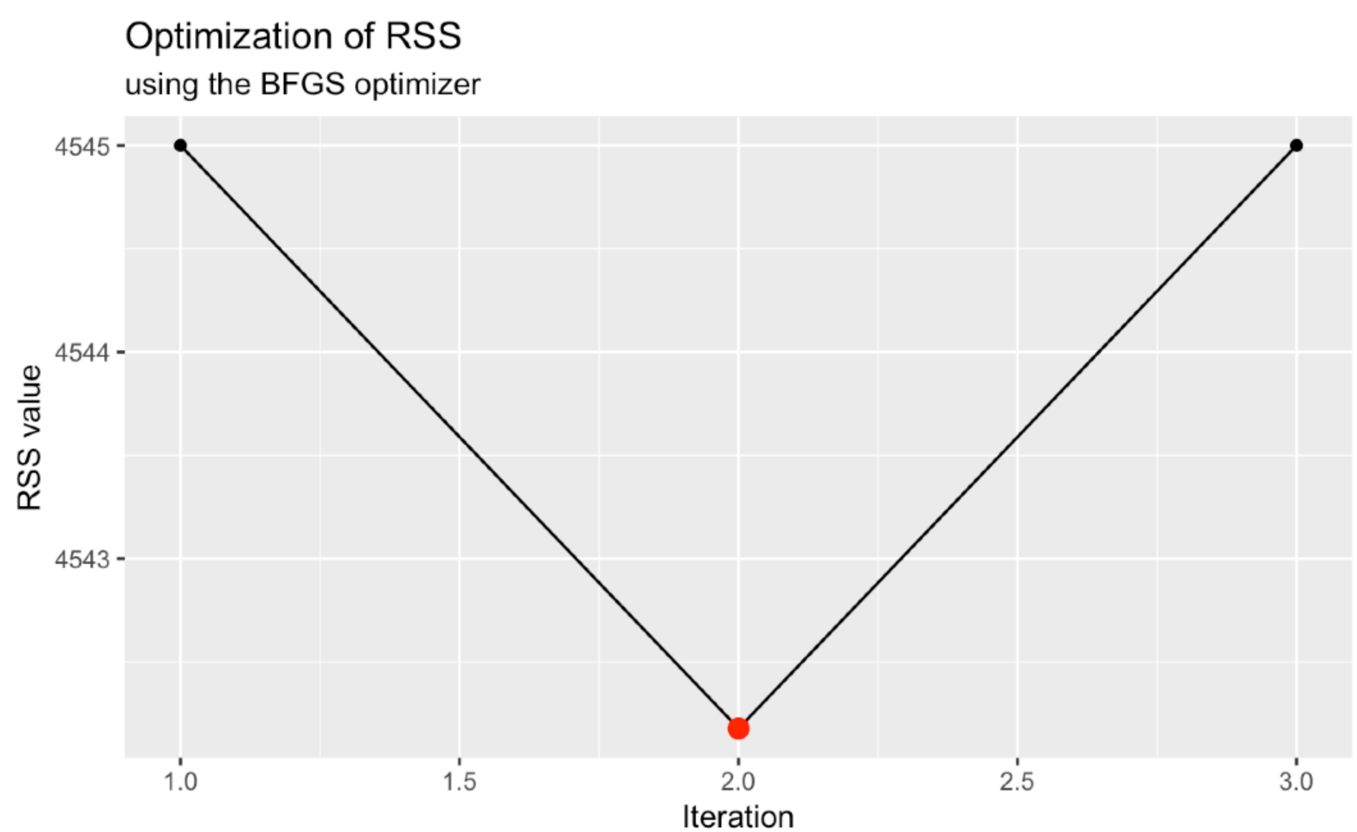



```r
# Mean of Response (y)
y_mu = mean(y)

# Mean of Predictor (x)
x = X[, 2]
x_mu = mean(x)

# Estimate Slope
beta1_hat =
     sum((x - x_mu) * (y - y_mu)) /
      sum((x - x_mu) ^ 2)

# Estimate Intercept
beta0_hat =
     y_mu - beta1_hat * x_mu

# Check Parameter Difference
cbind(c(beta0_hat, beta1_hat), beta)
#        beta_hat  beta
# [1,] 2.511657   2.5
# [2,] 4.027985   4.0
```

```r
# Write the cost function to minimize
min_rss_slr = function(par, X, y) {
  rss = sum((y - (X[,1]*par[1] + X[, 2] * par[2]))^2)
  return(rss)
}

# Initial beta parameters values
beta_init = c(0, 0)

# Perform the minimization
model_opt = optim(par = beta_init,
                  fn = min_rss_slr, method = "BFGS",
                 control = list(trace = TRUE),
                  X = X, y = y)
# initial  value 4545.000693
# final  value   4542.177432
# converged

# Check parameter difference
cbind(model_opt$par, beta)
#        beta_hat  beta
# [1,] 2.511657  2.5
# [2,] 4.027985  4.0
```

Optimizing RSS

… minimizing point distance …



MLR

Multiple Linear Regression

... scaling SLR to multiple predictors ...

Scalar-form

$$y_i = \beta_0 + \beta_1 x_{i,1} + \beta_2 x_{i,2} + \cdots + \beta_{p-1} x_{i,p-1} + \varepsilon_i$$

Expand Matrix

$$\underbrace{\begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}_{n\times 1}}_{\text{Responses}} = \underbrace{\begin{pmatrix} 1 & x_{1,1} & \cdots & x_{1,p-1} \\ \vdots & \vdots & & \vdots \\ 1 & x_{n,1} & \cdots & x_{n,p-1} \end{pmatrix}_{n\times p}}_{\text{Design Matrix}} \underbrace{\begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_{p-1} \end{pmatrix}_{p\times 1}}_{\text{Parameters}} + \underbrace{\begin{pmatrix} \varepsilon_1 \\ \vdots \\ \varepsilon_n \end{pmatrix}_{n\times 1}}_{\text{Error}}$$

Matrix-form

$$Y_{n\times 1} = X_{n\times p}\beta_{p\times 1} + \varepsilon_{n\times 1}$$

---

n is the number of observations, ***p* is the number of variables**, *X* is where predictors are stored, ***y*** is response vector, β is the parameter or coefficient vector and ε is the random error vector

SLR vs MLR

$$\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_i$$

$$Y_{n\times1} = X_{n\times2}\hat{\beta}_{2\times1}$$

$$\underbrace{\begin{pmatrix} \hat{y}_1 \\ \vdots \\ \hat{y}_n \end{pmatrix}_{n\times1}}_{\text{Responses}} = \underbrace{\begin{pmatrix} 1 & x_{1,1} \\ \vdots & \vdots \\ 1 & x_{n,1} \end{pmatrix}_{n\times2}}_{\text{Design Matrix}} \underbrace{\begin{pmatrix} \hat{\beta}_0 \\ \hat{\beta}_1 \end{pmatrix}_{2\times1}}_{\text{Parameters}}$$

$$\hat{\beta} = \underset{\beta_0,\beta_1\in\mathbb{R}}{\arg\min} \sum_{i=1}^{n}\left(y_i - \left(\beta_0 + \beta_1 x_i\right)\right)^2$$

$$\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_{i,1} + \underline{\hat{\beta}_2 x_{i,2} + \cdots + \hat{\beta}_{p-1} x_{i,p-1}}$$

$$\hat{Y}_{n\times1} = X_{n\times p}\hat{\beta}_{\underline{p\times1}}$$

$$\underbrace{\begin{pmatrix} \hat{y}_1 \\ \vdots \\ \hat{y}_n \end{pmatrix}_{n\times1}}_{\text{Responses}} = \underbrace{\begin{pmatrix} 1 & x_{1,1} & \cdots & x_{1,p-1} \\ \vdots & \vdots & & \vdots \\ 1 & x_{n,1} & \cdots & x_{n,p-1} \end{pmatrix}_{n\times p}}_{\text{Design Matrix}} \underbrace{\begin{pmatrix} \hat{\beta}_0 \\ \hat{\beta}_1 \\ \vdots \\ \hat{\beta}_{p-1} \end{pmatrix}_{p\times1}}_{\text{Parameters}}$$

$$\hat{\beta} = \underset{\beta\in\mathbb{R}^p}{\arg\min} \left\|\mathbf{y}_{n\times1} - X_{n\times p}\beta_{p\times1}\right\|^2$$

Derivations of MLR

http://thecoatlessprofessor.com/statistics/multiple-linear-regression-proofs/

STAT 420 and STAT 425 will focus more on the derivations.

Fitting MLR

... using linear regression ...

Formula
The underlying model for the data

Data
Data to regress over

$$\texttt{model\_fit = lm(} \underbrace{\texttt{y} \sim \texttt{x1 + x2}}_{\text{Formula}} \texttt{,} \underbrace{\texttt{data = data\_set}}_{\text{Data}} \texttt{)}$$

Inside the *formula*, model terms can be combined using **+** to give:

$$y_i = \beta_0 + \beta_1 x_{i,1} + \beta_2 x_{i,2} + \varepsilon_i$$

---

* β_0 represents the intercept term and is automatically included.

** The intercept can be removed by subtracting 1, e.g. y ~ x - 1

Example MLR Fit

... fitting data to mtcars ...

```
# Construct a multiple linear regression (3 variables: wt, qsec, intercept)
model_fit = lm(mpg ~ wt + qsec, data = mtcars)

# View the estimated parameters
model_fit
# Call:
# lm(formula = mpg ~ wt + qsec, data = mtcars)
#
# Coefficients:
# (Intercept)          wt        qsec
#     19.7462     -5.0480      0.9292
```

```r
# How formulas generate a design matrix

# Constructing a design matrix with the intercept
X = model.matrix(mpg ~ wt + qsec, data = mtcars)


# Building a design matrix without the intercept
X_noint = model.matrix(mpg ~ wt + qsec - 1, data = mtcars)
```

Intercept

(Intercept)	wt	qsec
1	2.62	16.46
1	2.875	17.02
1	2.32	18.61

X

No Intercept

wt	qsec
2.62	16.46
2.875	17.02
2.32	18.61

X_noint

Inference

… understanding the regression model …

```
summary(lm(mpg ~ wt + qsec, data = mtcars))
```

```
Call:
lm(formula = mpg ~ wt + qsec, data = mtcars)

Residuals:
    Min      1Q  Median      3Q     Max
-4.3962 -2.1431 -0.2129  1.4915  5.7486

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  19.7462     5.2521   3.760 0.000765 ***
wt           -5.0480     0.4840 -10.430  2.52e-11 ***
qsec          0.9292     0.2650   3.506 0.001500 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.596 on 29 degrees of freedom
Multiple R-squared: 0.8264,    Adjusted R-squared: 0.8144
F-statistic: 69.03 on 2 and 29 DF,  p-value: 9.395e-12
```

Residuals: $\mathbf{e} = \mathbf{y} - \mathbf{X}\hat{\beta}$, $e_i = y_i - \left(\sum_{j=0}^{p-1} x_{i,j}\hat{\beta}_j\right)$

Residual quantiles: summary(residuals(model))

Estimate: $\hat{\beta} = \left(X^T X\right)^{-1} X^T \mathbf{y}$

Std. Error: $s.e.\left(\hat{\beta}_j\right) = \sqrt{\operatorname{cov}(\hat{\beta}_j)} = \sqrt{\sigma^2_{RSS}\left(X^T X\right)^{-1}_{jj}}$

t value: $\mathbf{t}_{\hat{\beta}} = \frac{\hat{\beta} - \beta_0}{s.e.(\hat{\beta})}, \beta_0 = 0$

Pr(>|t|): $\text{pvalue} = 2 \cdot P\left(T \le -\left|\mathbf{t}_{\hat{\beta}}\right|\right) \sim t_{n-p}$

Residual standard error: $\hat{\sigma}_{RSS} = \sqrt{\frac{\mathbf{e}^T\mathbf{e}}{n-p}} = \sqrt{\frac{1}{n-p}\sum_{i=1}^{n} e_i^2}$

Degrees of freedom: $df = n - p$

Multiple R-squared: $R^2 = 1 - \frac{RSS}{TSS}$

Adjusted R-squared: $R^2_{adj} = 1 - \frac{\left(1-R^2\right)\left(n-1\right)}{n-p}$

F-statistic: $F = \frac{\left(\sum_{i=1}^{n}\left(y_i - \bar{y}\right)^2 - \sum_{i=1}^{n} e_i^2\right) / \left(p-1\right)}{\sum_{i=1}^{n} e_i^2 / \left(n-p\right)} = \frac{\left(TSS - RSS\right)/\left(p-1\right)}{RSS / \left(n-p\right)}$

$df_{num} = p - 1$

$df_{denom} = n - p$

p-value: $P\left(F > f\right) \sim F_{df_{num}, df_{denom}}$

Predicting the Future

… with confidence I can predict what happened yesterday …

```
# Values to use for prediction
future_car = data.frame(wt = 0.5, qsec = 12)

# Make prediction
y_hat = predict(model_fit, new_data = future_car)

head(y_hat)
#      Mazda RX4    Mazda RX4 Wag         Datsun 710
#       21.81511          21.04822           25.32728
#   Hornet 4 Drive  Hornet Sportabout        Valiant
#       21.58057           18.19611           21.06859
```

Factors

Previously

Hierarchy of Data Types*

* **raw** type is missing

Variable Data

- Numerical
  - Continuous
    - **complex**
      3 + 4i, 4.2 - i, 3.6i, 99 + 0i
      Real and imaginary
      - **numeric**
        3.14, 10.5, 0.0, -4.8
        Decimals
  - Discrete
    - **integer**
      6, 12, 0, -4
      Whole Numbers
      - **logical**
        TRUE (1), FALSE (0)
        Boolean values
- Categorical
  - **character**
    "toad", "got pie?","orange", "stat 385"
    Strings
    - Nominal
      - **factor**
        “AB”, "A", "B", “O”
        Blood Type
    - Ordinal
      - **ordered factor**
        “A”, “B”, “C”, “D”, “F”
        Letter grades

Augmented

Verifying Class in R

subject_heights

id	sex	height
1	M	6.1
2	F	5.5
3	F	5.2
…	…	…
55	M	5.9

```
id = c(1, 2, 3, 55)

class(id)
# [1] "numeric"

sex = c("M", "F", "F", "M")

class(sex)
# [1] "character"

height = c(6.1, 5.5, 5.2, 5.9)

class(height)
# [1] "height"
```

Class Difference

```r
# Default way
# Treat as factors
subject_heights = data.frame(
  id     = c(1, 2, 3, 55),
  sex    = c("M", "F", "F", "M"),
  height = c(6.1, 5.5, 5.2, 5.9),
  stringsAsFactors = TRUE
)
summary(subject_heights)
```

```
       id          sex       height     
 Min.   : 1.00   F:2   Min.   :5.200  
 1st Qu.: 1.75   M:2   1st Qu.:5.425  
 Median : 2.50         Median :5.700  
 Mean   :15.25         Mean   :5.675  
 3rd Qu.:16.00         3rd Qu.:5.950  
 Max.   :55.00         Max.   :6.100  
```

```r
# Opt out of factors
# Treat as characters
subject_heights_nofct = data.frame(
  id     = c(1, 2, 3, 55),
  sex    = c("M", "F", "F", "M"),
  height = c(6.1, 5.5, 5.2, 5.9),
  stringsAsFactors = FALSE
)
summary(subject_heights_nofct)
```

```
       id            sex                height     
 Min.   : 1.00   Length:4           Min.   :5.20
 1st Qu.: 1.75   Class :character   1st Qu.:5.42
 Median : 2.50   Mode  :character   Median :5.70
 Mean   :15.25                      Mean   :5.67
 3rd Qu.:16.00                      3rd Qu.:5.95
 Max.   :55.00                      Max.   :6.10
```

stringsAsFactors: An unauthorized biography

Roger Peng 2015/07/24

Recently, I was listening in on the conversation of some colleagues who were discussing a bug in their R code. The bug was ultimately traced back to the well-known phenomenon that functions like 'read.table()' and 'read.csv()' in R convert columns that are detected to be character/strings to be factor variables. This lead to the spontaneous outcry from one colleague of

> Why does stringsAsFactors not default to FALSE????

The argument 'stringsAsFactors' is an argument to the 'data.frame()' function in R. It is a logical that indicates whether strings in a data frame should be treated as factor variables or as just plain strings. The argument also appears in 'read.table()' and related functions because of the role these functions play in reading in table data and converting them to data frames. By default, 'stringsAsFactors' is set to TRUE.

This argument dates back to May 20, 2006 when it was originally introduced into R as the 'charToFactor' argument to 'data.frame()'. Soon afterwards, on May 24, 2006, it was changed to 'stringsAsFactors' to be compatible with S-PLUS by request from Bill Dunlap.

Most people I talk to today who use R are completely befuddled by the fact that 'stringsAsFactors' is set to TRUE by default. First of all, it should be noted that before the 'stringsAsFactors' argument even existed, the behavior of R was to coerce all character strings to be factors in a data frame. If you didn't want this behavior, you had to manually coerce each column to be character.

So here's the story:

In the old days, when R was primarily being used by statisticians and statistical types, this setting strings to be factors made total sense. In most tabular data, if there were a column of the table that was non-numeric, it almost certainly encoded a categorical variable. Think sex (male/female), country (U.S./other), region (east/west), etc. In R, categorical variables are represented by 'factor' vectors and so character columns got converted factor.

Why do we need factor variables to begin with? Because of modeling functions like 'lm()' and 'glm()'. Modeling functions need to treat expand categorical variables into individual dummy variables, so that a categorical variable with 5 levels will be expanded into 4 different columns in your modeling matrix. There's no way for R to know it should do this unless it has some extra information in the form of the factor class. From this point of view, setting 'stringsAsFactors = TRUE' when reading in tabular data makes total sense. If the data is just going to go into a regression model, then R is doing the right thing.

https://simplystatistics.org/2015/07/24/stringsasfactors-an-unauthorized-biography/

Just say ***YES*** to stringsAsFactors = FALSE

… or feel the wraith of factors …

```
# Factors
# Codifying character values as numbers

sex = c("M", "F", "F", "M")     # Character vector
sex
# [1] "M" "F" "F" "M"

sex_factor = factor(sex)    # Convert from character to factor
# [1] M F F M                # No quotations around character!
# Levels: F M               # Lists the character mapped to numbers.

as.numeric(sex_factor)     # Access internal level / number representation
# [1] 2 1 1 2
```

sex		levels
"M"	→	2
"F"	→	1
"F"	→	1
...		...
"M"	→	2

Levels	
id	**sex**
1	"F"
2	"M"

* Storing data as factors instead of character data on disk will result in lower file sizes.
** Computationally, there are advantages by transforming character values to factors.

Usefulness of Factors

- Creating a **design matrix** for *linear regression* as they provide a *codified* **dummy variable** structure. e.g. FALSE (0) or TRUE (1)

sex	height
"M"	6.1
"F"	5.5
"F"	5.2
...	...
"M"	5.9

subject_data

sex_F	height
0	6.1
1	5.5
1	5.2
...	...
0	5.9

Design Matrix

- Poor if character values (e.g. levels) need to be manipulated or there are too many unique values (e.g. text messages on a phone.)

Your Turn

create the design matrix for where students sit in class

dist	side
"back"	"right"
"back"	"left"
"front"	"middle"
"back"	"middle"
"front"	"right"
"front"	"left"

→

dist_back	side_middle	side_right

Why are *side_left* or *dist_front* not included as variables?

Limitations of Factors

... no math support ...

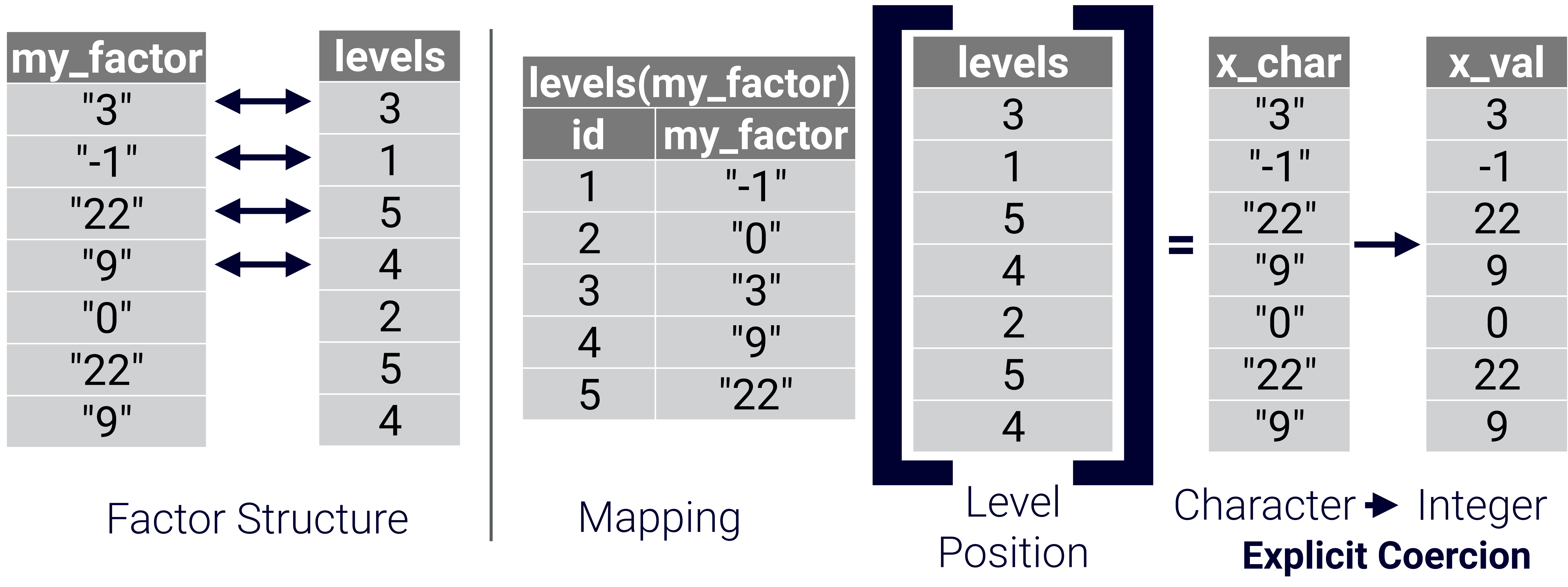
```
x = c(3L, -1L, 22L, 9L, 0L, 22L, 9L) # Create Integer Vector

my_factor = as.factor(x)             # Cast integer to factor

my_factor + 10                       # Error in an unexpected way
# Warning in Ops.factor(my_factor, 10) : '+' not meaningful for factors
# [1] NA NA NA NA NA NA NA

min(my_factor)                       # Show stopping error
# Error in Summary.factor(c(3L, 1L, 5L, 4L, 2L, 5L, 4L), na.rm = FALSE) :
#  'min' not meaningful for factors
```

Fun with Factors

... viewing levels, changing values, and ...

```
levels(my_factor)                    # List of Levels
# [1] "-1" "0"  "3"  "9"  "22"

my_factor[1] = 9                     # Modify with a pre-existing level
# [1] 9  -1 22 9  0  22 9
# Levels: -1 0 3 9 22

my_factor[2] = 18                    # Error if level isn't present already
# Warning in `[<-.factor`(`*tmp*`, 2, value = 18) :
#  invalid factor level, NA generated
```

```
# Recovering Values from a Factor
# Translating a factor's levels to an atomic vector

# Extract out names of levels by element-wise position.
# For details, see ?`[.factor`
x_char = levels(my_factor)[my_factor]     # Levels treated as positions
x_char
# [1] "3"  "-1" "22" "9"  "0"  "22" "9"          # Character vector extracted.

# Coerce to the appropriate type. (Integer for this example...)
x_val = as.integer(x_char)
x_val
# [1]  3 -1 22  9  0 22  9                      # Integer vector returned so data is recovered!
```



Ordered Factors

... making factors have precedence ...

```
yields = c("hi", "low", "med", "low", "med", "low")   # Vector of Character Values

yields_fct = factor(yields)                            # Create factor
# [1] hi  low med low med low
# Levels: hi low med

yields_ordered = factor(yields, ordered = TRUE) # Add ordered component
# [1] hi  med low med low med low
# Levels: hi < low < med

# Correct ordering from low to high
yields_fixed_order = factor(yields, levels = c("low", "med", "hi"),  ordered = TRUE)
# [1] hi  med low med low med low
# Levels: low < med < hi
```

Your Turn

Determine whether the following should be either a factor or an ordered factor

Months: (Jan, Feb, … , Nov, Dec)

Colors: (red, orange, … , black, green)

Alphabet: (a, b, … , y, z)

Recap

- **SLR and MLR**
  - Estimating a linear regression with 2 parameters vs. p parameters
  - Underlying design matrix construction
- **Factors**
  - Provide a mapping of levels to indicator variables inside the design matrix.