

# Basics of Data Frames

STAT 133

Gaston Sanchez

Department of Statistics, UC–Berkeley

`gastonsanchez.com`

`github.com/gastonstat/stat133`

Course web: `gastonsanchez.com/stat133`

# Data Frames

## Data Frame

A `data.frame` is the primary data structure that R provides for handling tabular data sets (eg spreadsheet like).

## Function `data.frame()`

The `data.frame()` function allows us to create data frames

# Creating a Data Frame

```
# data frame
df <- data.frame(
  name = c('Anakin', 'Padme', 'Luke', 'Leia'),
  gender = c('male', 'female', 'male', 'female'),
  height = c(1.88, 1.65, 1.72, 1.50),
  weight = c(84, 45, 77, 49)
)
```

by default, `data.frame()` converts strings into factors

# Simple data frame df

```
df
```

```
##      name gender height weight
## 1 Anakin   male   1.88     84
## 2 Padme  female   1.65     45
## 3 Luke    male   1.72     77
## 4 Leia   female   1.50     49
```

# Inspecting Data Frames

- ▶ dimensions (number of rows and columns)
- ▶ type of data in each column
- ▶ row names and column names
- ▶ missing data
- ▶ overall summary of each variable

# Overall structure

A summary of the structure can be obtained with `str()`

```
# structure of a data frame  
str(df)  
  
## 'data.frame': 4 obs. of 4 variables:  
## $ name : Factor w/ 4 levels "Anakin","Leia",...: 1 4 3 2  
## $ gender: Factor w/ 2 levels "female","male": 2 1 2 1  
## $ height: num 1.88 1.65 1.72 1.5  
## $ weight: num 84 45 77 49
```

# Function `str()`

`str()` applied on data frames provides:

- ▶ number of rows
- ▶ number of variables
- ▶ name of each column
- ▶ mode (i.e. type) of each column (e.g. num, int, chr, factor)
- ▶ number of levels for factor variables

`str()` is good for visual inspection, but doesn't give you direct access to the displayed information.

# Basic Information of Data Frames

Function	Description
<code>dim()</code>	dimensions (rows and columns)
<code>nrow()</code>	number of rows
<code>ncol()</code>	number of columns
<code>names()</code>	name of columns
<code>colnames()</code>	name of columns
<code>rownames()</code>	names of rows
<code>dimnames()</code>	list with names of rows and columns



# Basic Information of Data Frames

```
dim(df) # dimensions in a two element vector
```

```
## [1] 4 4
```

```
nrow(df)
```

```
## [1] 4
```

```
ncol(df)
```

```
## [1] 4
```

# Basic Information of Data Frames

```
colnames(df)
```

```
## [1] "name" "gender" "height" "weight"
```

```
rownames(df)
```

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

```
dimnames(df) # names in a list
```

```
## [[1]]
```

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

```
##
```

```
## [[2]]
```

```
## [1] "name" "gender" "height" "weight"
```

# Function `object.size()`

To know how much memory space is allocated for a data frame (or any other R object) we use `object.size()`

```
object.size(df)
```

```
## 2136 bytes
```

# Functions head() and tail

Inspect the first and last rows, respectively:

```
# first 3 rows
```

```
head(df, n = 3)
```

```
##      name gender height weight
## 1 Anakin   male   1.88     84
## 2 Padme  female   1.65     45
## 3  Luke   male   1.72     77
```

```
# last 2 rows
```

```
tail(df, n = 2)
```

```
##      name gender height weight
## 3 Luke   male   1.72     77
## 4 Leia  female   1.50     49
```

# Function summary()

There's also the function `summary()`, which provides a descriptive summary of each column

```
summary(df)
```

##	name	gender	height	weight
##	Anakin:1	female:2	Min. :1.500	Min. :45.00
##	Leia :1	male :2	1st Qu.:1.613	1st Qu.:48.00
##	Luke :1		Median :1.685	Median :63.00
##	Padme :1		Mean :1.688	Mean :63.75
##			3rd Qu.:1.760	3rd Qu.:78.75
##			Max. :1.880	Max. :84.00

# Elementary Manipulations

# Accessing Elements

There are many different ways in which the elements of a "data.frame" can be accessed (i.e. retrieved, selected).

- ▶ accessing individual cells
- ▶ accessing sets of cells
- ▶ accessing entire rows
- ▶ accessing entire columns

# Notation System Reminder

## Notation system to extract values from data frames

- ▶ to extract values use brackets: [ ]
- ▶ inside the brackets specify indices for rows and columns
- ▶ each index is separated by comma
- ▶ row indices can be numbers or logicals
- ▶ column indices can be numbers, logicals, or names



# Single Cells

Using row and column indices to access a single cell

```
# first cell 1,1
```

```
df[1, 1]
```

```
## [1] Anakin
```

```
## Levels: Anakin Leia Luke Padme
```

```
# cell 3,4
```

```
df[3, 4]
```

```
## [1] 77
```

```
# last cell
```

```
df[4, 4]
```

```
## [1] 49
```

# Various Cells

Using vectors of row and column indices to access various cells

```
# various adjacent cells
```

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

```
##      gender height weight
```

```
## 1    male    1.88     84
```

```
## 2 female    1.65     45
```

```
## 3    male    1.72     77
```

```
# various adjacent cells
```

```
# (permuted order)
```

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

```
##      height gender
```

```
## 4    1.50 female
```

```
## 3    1.72   male
```

```
## 2    1.65 female
```

```
## 1    1.88   male
```

# Various Cells

Using vectors of row and column indices to access various cells

```
# non-adjacent cells  
df[c(2, 4), c(1, 3)]
```

```
##      name height  
## 2 Padme    1.65  
## 4  Leia    1.50
```

# Various Cells

Using excluding indices

```
# excluding various adjacent cells
```

```
df[-c(1:2), -c(2:3)]
```

```
##   name weight
```

```
## 3 Luke     77
```

```
## 4 Leia     49
```

# Retrieving Rows

## Selecting rows

```
# first row
```

```
df[1, ]
```

```
##      name gender height weight  
## 1 Anakin   male   1.88     84
```

```
# rows 1 to 3
```

```
df[1:3, ]
```

```
##      name gender height weight  
## 1 Anakin   male   1.88     84  
## 2 Padme female   1.65     45  
## 3  Luke   male   1.72     77
```

# Retrieving Rows

Selecting rows (by excluding indices)

```
# all rows except first one
```

```
df[-1, ]
```

```
##      name gender height weight
```

```
## 2 Padme female   1.65     45
```

```
## 3 Luke   male   1.72     77
```

```
## 4 Leia female   1.50     49
```

```
# rows except 2 to 3
```

```
df[-c(2:3), ]
```

```
##      name gender height weight
```

```
## 1 Anakin   male   1.88     84
```

```
## 4  Leia female   1.50     49
```

# Retrieving Columns

## Selecting columns

```
# 1st column (as a vector)
df[, 1]

## [1] Anakin Padme Luke Leia
## Levels: Anakin Leia Luke Padme
```

## Using argument drop=FALSE

```
# 1st column (as a column)
df[, 1, drop = FALSE]

##      name
## 1 Anakin
## 2 Padme
## 3 Luke
## 4 Leia
```

# Retrieving Columns

```
# columns 1 to 3
```

```
df[ , 1:3]
```

```
##      name gender height
```

```
## 1 Anakin   male   1.88
```

```
## 2 Padme  female   1.65
```

```
## 3 Luke    male   1.72
```

```
## 4 Leia   female   1.50
```

```
# columns 2, 4
```

```
df[ , c(2,4)]
```

```
##      gender weight
```

```
## 1    male      84
```

```
## 2 female      45
```

```
## 3    male      77
```

```
## 4 female      49
```



# Retrieving Columns

Selecting columns by excluding indices

```
# all columns but 2, 4,  
df[ , -c(2,4)]
```

```
##      name height  
## 1 Anakin   1.88  
## 2 Padme    1.65  
## 3 Luke     1.72  
## 4 Leia     1.50
```

# Accessing Single Columns

Besides using numeric indices, we can also access a single column using its name and following different syntax options:

- ▶ `df[ , "name"]`
- ▶ `df[["name"]]`
- ▶ `df$name` or `df$"name"`
- ▶ `df["name"]`

# Column by Name

```
# equivalent ways to retrieve one column
```

```
df[, "name"]
```

```
## [1] Anakin Padme Luke Leia
```

```
## Levels: Anakin Leia Luke Padme
```

```
df[["gender"]]
```

```
## [1] male female male female
```

```
## Levels: female male
```

```
df$height
```

```
## [1] 1.88 1.65 1.72 1.50
```

# Columns by Name

Retrieve various columns by name:

```
# vector of names  
df[, c("name", "gender", "height")]
```

```
##      name gender height  
## 1 Anakin   male   1.88  
## 2 Padme  female   1.65  
## 3 Luke    male   1.72  
## 4 Leia   female   1.50
```

# Adding New Elements

A typical data frame modification consists in adding new elements, that is, new rows and columns.

# Adding One Column

Adding a single column to a data frame

```
# adding 'a_vector' as a 'new' column  
df$eyecolor <- c('blue', 'brown', 'blue', 'brown')
```

```
df
```

```
##      name gender height weight eyecolor  
## 1 Anakin   male   1.88     84     blue  
## 2 Padme female   1.65     45    brown  
## 3 Luke    male   1.72     77     blue  
## 4 Leia   female   1.50     49    brown
```

# Adding One Column

Using *column binding* `cbind()` to add a column to a data frame

```
haircolor <- c('blond', 'brown', 'blond', 'brown')  
  
# binding a column  
df <- cbind(df, haircolor)  
  
df  
  
##      name gender height weight eyecolor haircolor  
## 1 Anakin   male   1.88    84      blue      blond  
## 2 Padme  female   1.65    45     brown     brown  
## 3 Luke    male   1.72    77      blue     blond  
## 4 Leia   female   1.50    49     brown     brown
```

# Adding One Column

Remember the recycling rule:

```
# "human" will be recycled!
```

```
df$species <- "human"
```

```
df
```

```
##      name gender height weight eyecolor haircolor species
## 1 Anakin   male   1.88    84     blue     blond   human
## 2 Padme female   1.65    45    brown    brown   human
## 3 Luke    male   1.72    77     blue     blond   human
## 4 Leia    female   1.50    49    brown    brown   human
```



# Adding Several Columns

Equivalent ways to add several columns to a data frame

```
# adding vectors x and y
```

```
df[, c("x", "y")] <- cbind(1:4, 5:8)
```

```
df
```

```
##      name gender height weight eyecolor haircolor species x y
## 1 Anakin   male   1.88    84      blue      blond   human 1 5
## 2 Padme female   1.65    45     brown     brown   human 2 6
## 3 Luke    male   1.72    77      blue      blond   human 3 7
## 4 Leia    female   1.50    49     brown     brown   human 4 8
```

# Adding Several Columns

Equivalent ways to add several columns to a data frame

```
# adding vectors u and v  
uv <- cbind(u = 1:4, v = 5:8)  
  
df <- cbind(df, uv)
```

# Removing Columns

Removing columns with the NULL object

```
# removing x and y
```

```
df$x <- NULL
```

```
df$y <- NULL
```

```
df
```

```
##      name gender height weight eyecolor haircolor species
## 1 Anakin  male   1.88    84      blue      blond    human
## 2 Padme  female  1.65    45     brown     brown    human
## 3 Luke   male   1.72    77      blue      blond    human
## 4 Leia   female  1.50    49     brown     brown    human
```

# Removing Columns

## Removing columns by reassignment

```
# removing columns 5, 6, ...
```

```
df <- df[ , 1:4]
```

```
df
```

```
##      name gender height weight
## 1 Anakin   male   1.88     84
## 2 Padme female   1.65     45
## 3 Luke    male   1.72     77
## 4 Leia   female   1.50     49
```

# Adding Rows

- ▶ Another operation is adding rows
- ▶ This can be done with *row binding* `rbind()`
- ▶ When adding rows to a data frame, we need to take into account the mode of each column
- ▶ If all columns have the same mode, then we can add a vector
- ▶ If columns have different modes, then we need to add data.frames

# Adding Rows

Be careful when adding vector rows to data frames!

```
# new vector
```

```
newone <- c("Han", 'male', 1.8, 80)
```

```
# trying to add a vector to data frame
```

```
rbind(df, newone)
```

```
## Warning in '[<-.factor'('*tmp*', ri, value = "Han"):  
invalid factor level, NA generated
```

```
##      name gender height weight  
## 1 Anakin  male   1.88     84  
## 2 Padme female   1.65     45  
## 3 Luke   male   1.72     77  
## 4 Leia  female   1.5     49  
## 5 <NA>   male   1.8     80
```

# Adding Rows

Since columns in `df` are of different modes, we must create a new row "vector" in `data.frame` format

```
# creating a data frame "vector"
```

```
han <- data.frame(  
  name = "Han",  
  gender = 'male',  
  height = 1.8,  
  weight = 80)
```

```
han
```

```
##   name gender height weight  
## 1  Han   male    1.8     80
```

# Adding Rows

Use row binding `rbind()` to add one or more rows:

```
# adding 'han' with rbind()  
df <- rbind(df, han)
```

df

##		name	gender	height	weight
## 1	Anakin	male	1.88	84	
## 2	Padme	female	1.65	45	
## 3	Luke	male	1.72	77	
## 4	Leia	female	1.50	49	
## 5	Han	male	1.80	80	



# Arranging Columns

A less common, but equally important type of `data.frame` modification involves rearranging or moving its columns.

The common approach to move columns is to define a vector with the column names in the desired order, and then redefine the current data frame.

# Rearranging columns

```
# rearranging columns
```

```
df[, c(1, 4, 3, 2)]
```

```
##      name weight height gender
## 1 Anakin      84    1.88   male
## 2 Padme      45    1.65 female
## 3 Luke       77    1.72   male
## 4 Leia       49    1.50 female
## 5 Han        80    1.80   male
```

# Column Names

## Changing column names

```
# change first column  
names(df)[1] <- "Name"
```

```
# change weight  
names(df)[4] <- "wgt"
```

```
df
```

```
##      Name gender height wgt  
## 1 Anakin   male   1.88  84  
## 2 Padme female   1.65  45  
## 3 Luke    male   1.72  77  
## 4 Leia   female   1.50  49  
## 5 Han     male   1.80  80
```

# Column Names

## Changing column names

```
# rename first column  
names(df)[1] <- "name"  
  
# rename weight  
names(df)[4] <- "weight"
```

df

```
##      name gender height weight  
## 1 Anakin   male   1.88     84  
## 2 Padme female   1.65     45  
## 3 Luke    male   1.72     77  
## 4 Leia   female   1.50     49  
## 5 Han     male   1.80     80
```

# Exercise

## Creating new column

```
# height x weight
```

```
df$htwt <- df$height * df$weight
```

```
# gender and species
```

```
df$new <- paste(df$name, df$gender, sep = "_")
```

```
df
```

##	name	gender	height	weight	htwt	new
## 1	Anakin	male	1.88	84	157.92	Anakin_male
## 2	Padme	female	1.65	45	74.25	Padme_female
## 3	Luke	male	1.72	77	132.44	Luke_male
## 4	Leia	female	1.50	49	73.50	Leia_female
## 5	Han	male	1.80	80	144.00	Han_male

# Subsetting

# Subsetting

Subsetting using comparisons (logical vectors TRUE, FALSE)

```
df$name == "Luke"

## [1] FALSE FALSE  TRUE FALSE FALSE

# Luke's info
df[df$name == "Luke", ]

##   name gender height weight   htw  new
## 3 Luke   male   1.72    77 132.44 Luke_male
```

# Subsetting

Subsetting using comparisons (logical vectors TRUE, FALSE)

```
df$gender == "male"
```

```
## [1]  TRUE FALSE  TRUE FALSE  TRUE
```

```
# male subjects
```

```
df[df$gender == "male", ]
```

```
##      name gender height weight  htwt      new
## 1 Anakin  male   1.88     84 157.92 Anakin_male
## 3  Luke   male   1.72     77 132.44  Luke_male
## 5   Han   male   1.80     80 144.00   Han_male
```



# Subsetting

## Subsetting with composed statements

```
# male with height > 1.75
```

```
df[df$gender == "male" & df$height > 1.75, ]
```

```
##      name gender height weight   hwt      new
## 1 Anakin  male   1.88     84 157.92 Anakin_male
## 5   Han   male   1.80     80 144.00   Han_male
```

# Subsetting

Subsetting statements can become very verbose

```
# male with height > 1.75 and weight > 80
```

```
df[df$gender == "male"  
   & df$height > 1.75  
   & df$weight > 80, ]
```

```
##      name gender height weight  htwt      new  
## 1 Anakin  male   1.88     84 157.92 Anakin_male
```

# Subsetting with subset()

To reduce verbose subsetting statements we can use subset()

```
# male with height > 1.75
```

```
subset(df, gender == "male" & height > 1.75)
```

```
##      name gender height weight  htwt      new
## 1 Anakin  male   1.88     84 157.92 Anakin_male
## 5   Han   male   1.80     80 144.00   Han_male
```

```
# male with height > 1.75 and weight > 80
```

```
subset(df, gender == "male" & height > 1.75 & weight > 80)
```

```
##      name gender height weight  htwt      new
## 1 Anakin  male   1.88     84 157.92 Anakin_male
```

# Subsetting with subset()

subset() also allows you to select columns according to a specified condition

```
# name and weight of male subjects  
subset(df,  
        gender == "male",  
        select = c(name, weight))
```

```
##      name weight  
## 1 Anakin      84  
## 3   Luke      77  
## 5    Han      80
```

# Subsetting with subset()

subset() also allows you to select columns according to a specified condition

```
# excluding height of male subjects
```

```
subset(df,  
       gender == "male",  
       select = -height)
```

##	name	gender	weight	htwt	new
## 1	Anakin	male	84	157.92	Anakin_male
## 3	Luke	male	77	132.44	Luke_male
## 5	Han	male	80	144.00	Han_male

# Ordering Rows

# Sorting rows

Remember `sort()` and `order()`

```
# sort() sorts the values
```

```
sort(df$weight)
```

```
## [1] 45 49 77 80 84
```

```
# order() gives you the position
```

```
order(df$weight)
```

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

When sorting rows, we want to work with the ordered positions

# Sorting rows

```
# sorting rows by weight
```

```
df[order(df$weight), ]
```

##	name	gender	height	weight	htwt	new
## 2	Padme	female	1.65	45	74.25	Padme_female
## 4	Leia	female	1.50	49	73.50	Leia_female
## 3	Luke	male	1.72	77	132.44	Luke_male
## 5	Han	male	1.80	80	144.00	Han_male
## 1	Anakin	male	1.88	84	157.92	Anakin_male



# Sorting rows

```
# sorting subjects by height  
df[order(df$height), c('name', 'height')]
```

```
##      name height  
## 4    Leia   1.50  
## 2   Padme   1.65  
## 3    Luke   1.72  
## 5     Han   1.80  
## 1 Anakin   1.88
```

# Sorting rows

```
# sorting subjects by height in decreasing order  
ht_sort <- order(df$height, decreasing = TRUE)  
df[ht_sort, c('name', 'height')]
```

```
##      name height  
## 1 Anakin   1.88  
## 5   Han    1.80  
## 3   Luke   1.72  
## 2 Padme   1.65  
## 4   Leia   1.50
```