

# *Introduction to Data Wrangling*

# Data Wrangling

- Real-world data is NOT clean!!
- We have to process it. More efficiency, better time use.
- This is where dplyr comes in: aka A Grammar of Data Manipulation.
- Hadley Wickham, 2014



# Installing Package dplyr

You can use the `install.packages` command to install the `dplyr` package.

After installing the package, you can load it into the workspace using the `library` command. Note that while you only need to install a package once, you need to load it into the workspace whenever you want to access it.

Alternatively, recall that `dplyr` gets loaded automatically once you call `tidyverse` package, so you can skip this step.

```
> install.packages("dplyr")  
> library(dplyr)
```

```
Attaching package: 'dplyr'
```

```
The following object is masked from 'package:MASS':
```

```
  select
```

```
The following objects are masked from 'package:stats':
```

```
  filter, lag
```

```
The following objects are masked from 'package:base':
```

```
  intersect, setdiff, setequal, union
```

What is this telling us?

If we just type the object name, we get the object in the *last package loaded*. i.e., order matters. What is we want `select` from the `MASS` package?

```
> MASS::select
```

**DEE-PLIER**



**DIP-LER**

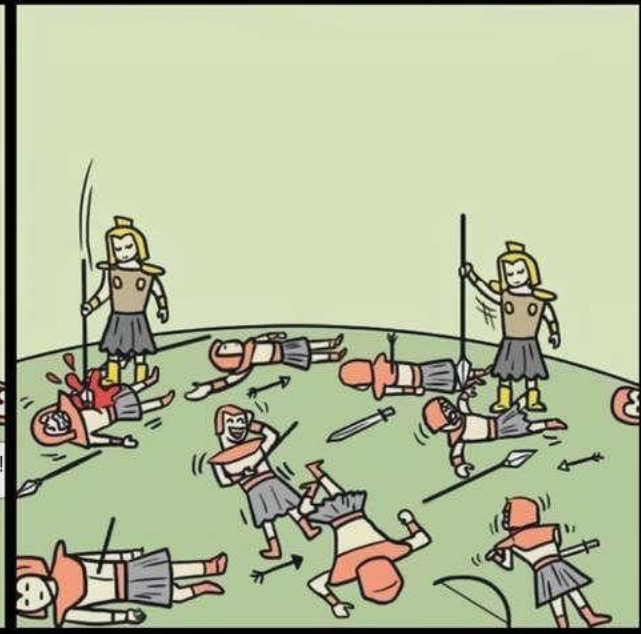
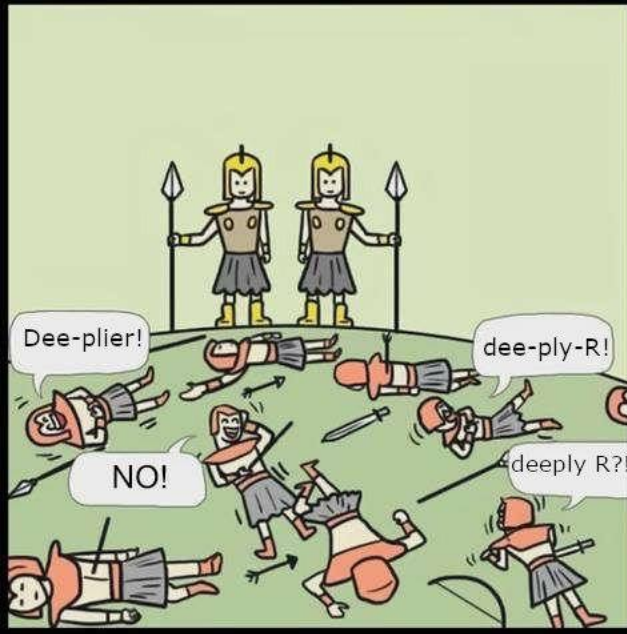
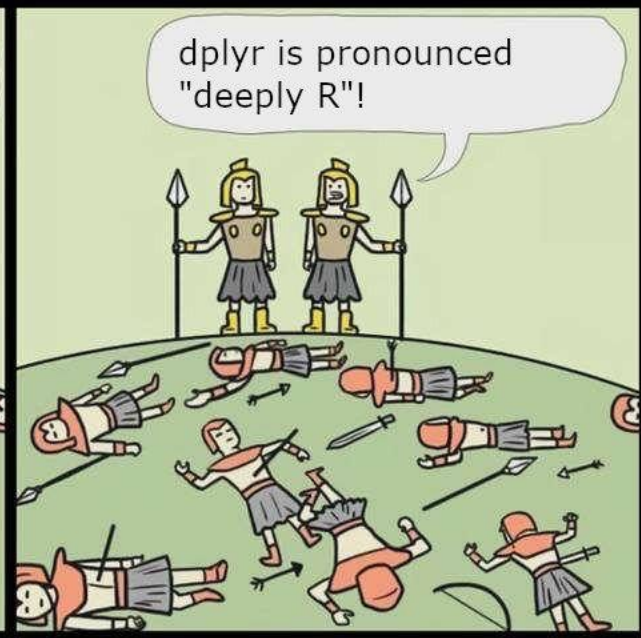
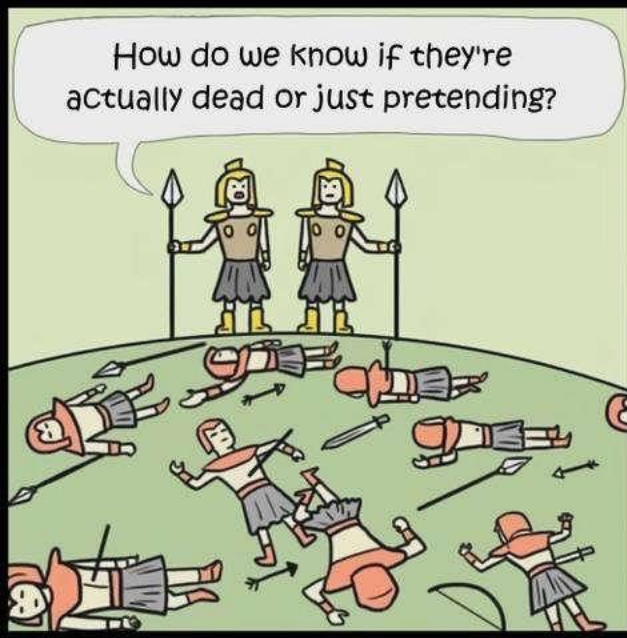


**DEEPLY-R**



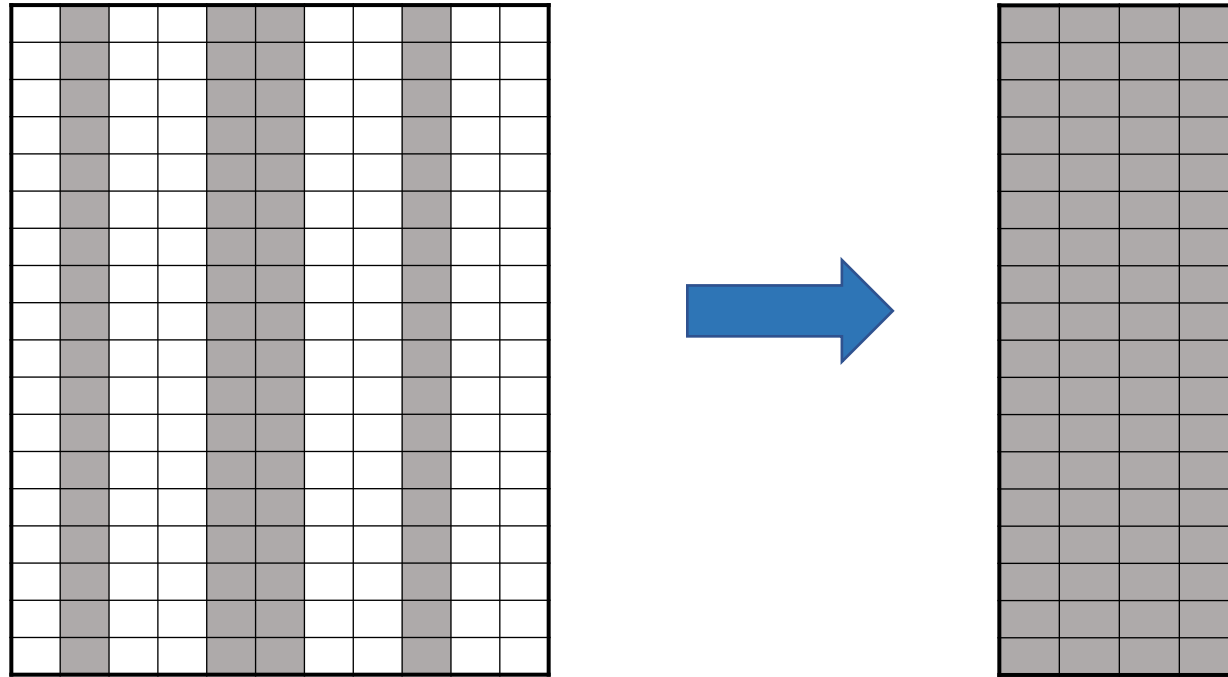
**DEE PEE  
EL WHY ARE**





# Our 1<sup>st</sup> function!: The `select` Function

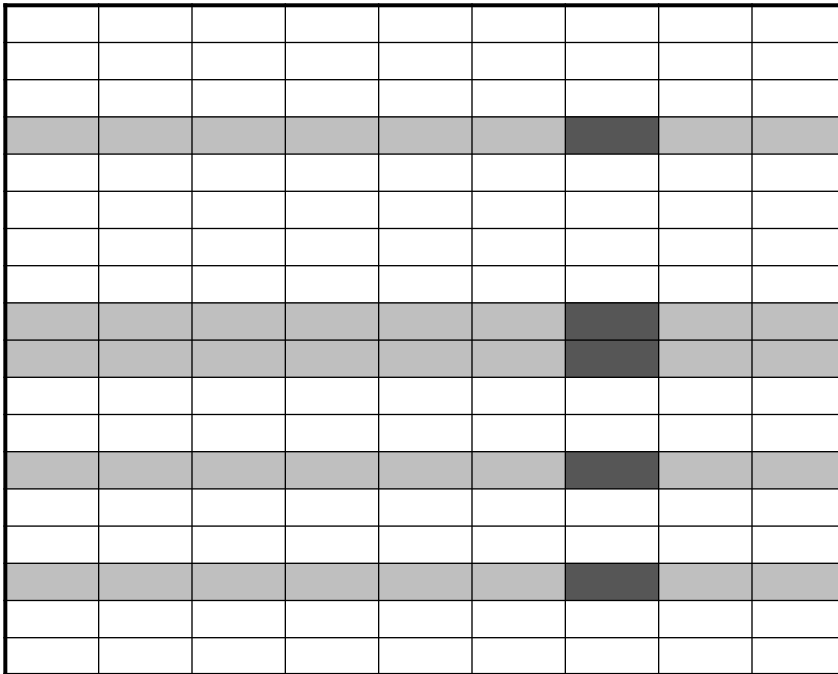
The `select` function allows you to create a new data frame that is a subset of an existing data frame by choosing a set of the columns of the original data frame.

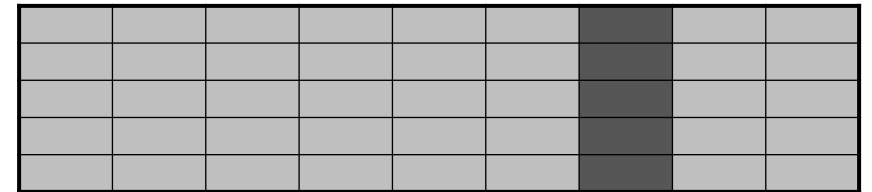


The general syntax is `select(data_frame, Var_name_1, Var_name_2, ...)`. Note that the entire column is transferred to the new data frame by the `select` function.

# The `filter` Function

The `filter` function allows you to create a new data frame that is a subset of an existing data frame by choosing a set of the rows of the original data frame based on a collection of specified conditions.




The general syntax is `filter(data_frame, Condition,...)`. The condition can include logical operators for the row selection. Note that the entire row is transferred to the new data frame by the `filter` function.



# Examples from the Cars93 Data Frame

Recall the `Cars93` data frame contains data for 93 cars sold in the U.S. during the year 1993. The first six rows of the data frame can be viewed using the `head` function.

```
> head(Cars93)
```

	Manufacturer	Model	Type	Min.Price	Price	Max.Price	MPG.city	MPG.highway	AirBags	DriveTrain
1	Acura	Integra	Small	12.9	15.9	18.8	25	31	None	Front
2	Acura	Legend	Midsize	29.2	33.9	38.7	18	25	Driver & Passenger	Front
3	Audi	90	Compact	25.9	29.1	32.3	20	26	Driver only	Front
4	Audi	100	Midsize	30.8	37.7	44.6	19	26	Driver & Passenger	Front
5	BMW	535i	Midsize	23.7	30.0	36.2	22	30	Driver only	Rear
6	Buick	Century	Midsize	14.2	15.7	17.3	22	31	Driver only	Front
	Cylinders	EngineSize	Horsepower	RPM	Rev.per.mile	Man.trans.avail	Fuel.tank.capacity	Passengers	Length	
1	4	1.8	140	6300	2890	Yes	13.2	5	177	
2	6	3.2	200	5500	2335	Yes	18.0	5	195	
3	6	2.8	172	5500	2280	Yes	16.9	5	180	
4	6	2.8	172	5500	2535	Yes	21.1	6	193	
5	4	3.5	208	5700	2545	Yes	21.1	4	186	
6	4	2.2	110	5200	2565	No	16.4	6	189	
	Wheelbase	Width	Turn.circle	Rear.seat.room	Luggage.room	Weight	Origin	Make		
1	102	68	37	26.5	11	2705	non-USA	Acura Integra		
2	115	71	38	30.0	15	3560	non-USA	Acura Legend		
3	102	67	37	28.0	14	3375	non-USA	Audi 90		
4	106	70	37	31.0	17	3405	non-USA	Audi 100		
5	109	69	39	27.0	13	3640	non-USA	BMW 535i		
6	105	69	41	28.0	16	2880	USA	Buick Century		



# The select Function: Cars93 Example

```
> Cars93_Ex1<-select(Cars93,Type,EngineSize,DriveTrain,MPG.city)
> head(Cars93_Ex1)
```

	Type	EngineSize	DriveTrain	MPG.city
1	Small	1.8	Front	25
2	Midsize	3.2	Front	18
3	Compact	2.8	Front	20
4	Midsize	2.8	Front	19
5	Midsize	3.5	Rear	22
6	Midsize	2.2	Front	22

## Notes:

- An arbitrary number of variable names can be passed into the function.
- The order of the variables in the new data frame matches the order they are entered into the function.
- The variable names are not in quotations in the function input.

# The filter Function: Cars93 Example

```
> Cars93_Ex2<-filter(Cars93, Type=="Small")
```

```
> head(Cars93_Ex2)
```

	Manufacturer	Model	Type	Min.Price	Price	Max.Price	MPG.city	MPG.highway	AirBags	DriveTrain	Cylinders
1	Acura	Integra	Small	12.9	15.9	18.8	25	31	None	Front	4
2	Dodge	Colt	Small	7.9	9.2	10.6	29	33	None	Front	4
3	Dodge	Shadow	Small	8.4	11.3	14.2	23	29	Driver only	Front	4
4	Eagle	Summit	Small	7.9	12.2	16.5	29	33	None	Front	4
5	Ford	Festiva	Small	6.9	7.4	7.9	31	33	None	Front	4
6	Ford	Escort	Small	8.4	10.1	11.9	23	30	None	Front	4
	EngineSize	Horsepower	RPM	Rev.per.mile	Man.trans.avail	Fuel.tank.capacity	Passengers	Length	Wheelbase		
1	1.8	140	6300	2890	Yes	13.2	5	177	102		
2	1.5	92	6000	3285	Yes	13.2	5	174	98		
3	2.2	93	4800	2595	Yes	14.0	5	172	97		
4	1.5	92	6000	2505	Yes	13.2	5	174	98		
5	1.3	63	5000	3150	Yes	10.0	4	141	90		
6	1.8	127	6500	2410	Yes	13.2	5	171	98		
	Width	Turn.circle	Rear.seat.room	Luggage.room	Weight	Origin	Make				
1	68	37	26.5	11	2705	non-USA	Acura Integra				
2	66	32	26.5	11	2270	USA	Dodge Colt				
3	67	38	26.5	13	2670	USA	Dodge Shadow				
4	66	36	26.5	11	2295	USA	Eagle Summit				
5	63	33	26.0	12	1845	USA	Ford Festiva				
6	67	36	28.0	12	2530	USA	Ford Escort				

The == is a test for equality and we need to enter the “Small” in quotations since we are filtering for rows where the Type variable is equal to “Small”.

# Combining the `filter` and `select` Functions

```
> Cars93_Ex3<-select(filter(Cars93, Type=="Small"),Model,EngineSize,Horsepower,MPG.highway)
> head(Cars93_Ex3)
```

	Model	EngineSize	Horsepower	MPG.highway
1	Integra	1.8	140	31
2	Colt	1.5	92	33
3	Shadow	2.2	93	29
4	Summit	1.5	92	33
5	Festiva	1.3	63	33
6	Escort	1.8	127	30

If I wanted to include the value 150,  
I would use

`Horsepower>=150`

```
> Cars93_Ex4<-select(filter(Cars93, Horsepower>150),Model,EngineSize,Horsepower,MPG.highway)
> head(Cars93_Ex4)
```

	Model	EngineSize	Horsepower	MPG.highway
1	Legend	3.2	200	25
2	90	2.8	172	26
3	100	2.8	172	26
4	535i	3.5	208	30
5	LeSabre	3.8	170	28
6	Roadmaster	5.7	180	25

In this example the functions are *nested*, and the operations are completed from the inside out. We will see shortly another way to perform this nesting that is more readable.

# Combining the `filter` and `select` Functions

```
> Cars93_Ex5<-select(filter(Cars93, Horsepower>120 & Type=="Small"),  
+ Model,EngineSize,Horsepower,MPG.highway)  
> head(Cars93_Ex5)
```

	Model	EngineSize	Horsepower	MPG.highway
1	Integra	1.8	140	31
2	Escort	1.8	127	30
3	Elantra	1.8	124	29

Here we have used the `&` operator which means we are filtering for `Small` cars that **also** have more than 120 `Horsepower`. That is, **both** conditions must be satisfied to pass through the filter. We note that only 3 cars in the data set satisfy this requirement.

```
> Cars93_Ex6<-select(filter(Cars93, Type=="Midsize"|Cylinders==4),  
+ Model,Type,EngineSize,Cylinders,Horsepower,MPG.highway)  
> head(Cars93_Ex6)
```

	Model	Type	EngineSize	Cylinders	Horsepower	MPG.highway
1	Integra	Small	1.8	4	140	31
2	Legend	Midsize	3.2	6	200	25
3	100	Midsize	2.8	6	172	26
4	535i	Midsize	3.5	4	208	30
5	Century	Midsize	2.2	4	110	31
6	Riviera	Midsize	3.8	6	170	27

Here we have used the `|` operator which means we are filtering for cars that are `Midsize` **or** that have 4 `Cylinders`. That is, if either (or both) conditions are met the row passes through the filter.

# The %in% Operator

A long list of criteria can be cumbersome to type. We can use the %in% operator to check whether a value is in a list of possible values.

```
> Cars93_Ex7<-select(filter(Cars93, Type %in% c("Sporty","Compact") & Horsepower >=120),  
+ Model,Type,EngineSize,Cylinders,Horsepower,MPG.highway)  
> head(Cars93_Ex7)
```

	Model	Type	EngineSize	Cylinders	Horsepower	MPG.highway
1	90	Compact	2.8	6	172	26
2	Camaro	Sporty	3.4	6	160	28
3	Corvette	Sporty	5.7	8	300	25
4	LeBaron	Compact	3.0	4	141	28
5	Stealth	Sporty	3.0	6	300	24
6	Prelude	Sporty	2.3	4	160	31

Here, the cars are returned that have a Type of either Sporty or Compact and have Horsepower greater than 120.

# The %>% Operator

When we have several operations to complete that need to be nested, using the *pipe operator* %>% can result in more readable code. That is, pipe-forwarding is an alternative to nesting.

```
> Cars93_Ex8<-Cars93 %>%  
+ filter(Type %in% c("Sporty","Compact") & Horsepower >=120) %>%  
+ select(Model,Type,EngineSize,Cylinders,Horsepower,MPG.highway)  
> head(Cars93_Ex8)
```

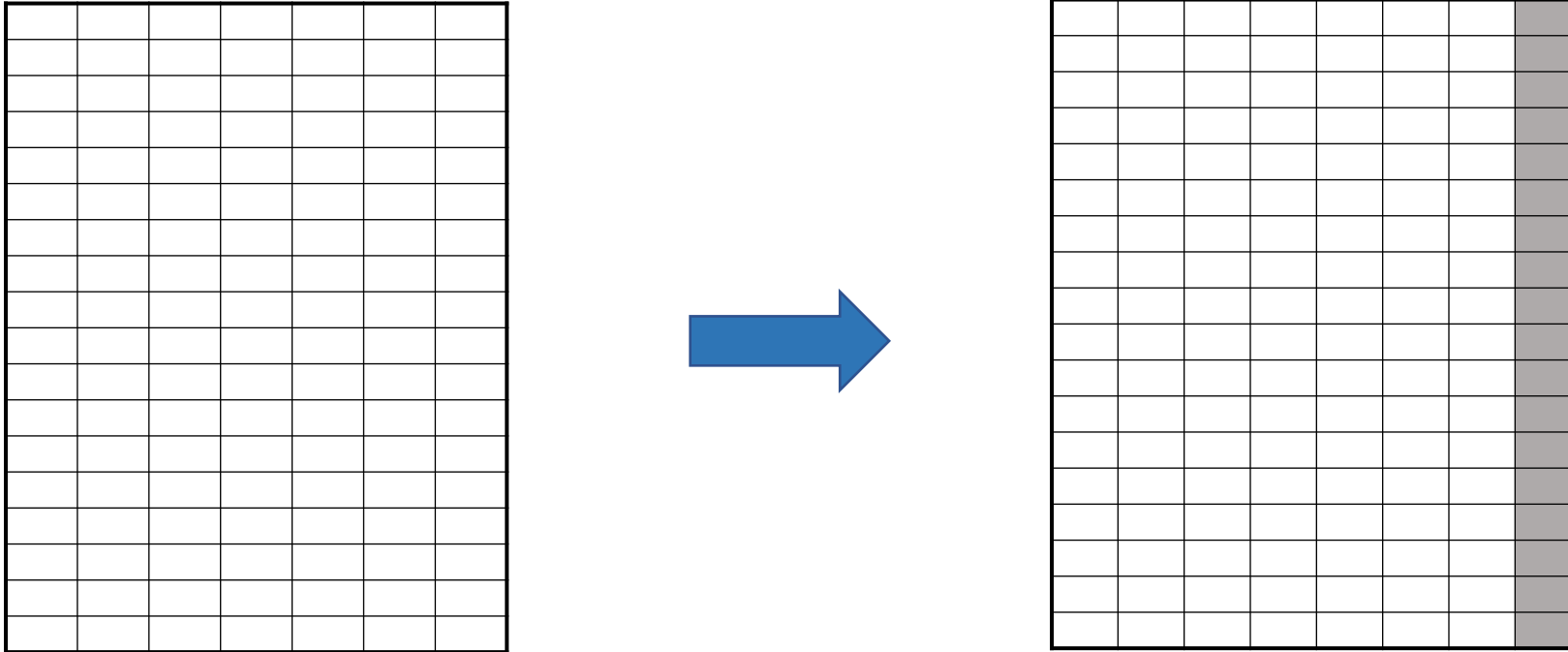
	Model	Type	EngineSize	Cylinders	Horsepower	MPG.highway
1	90	Compact	2.8	6	172	26
2	Camaro	Sporty	3.4	6	160	28
3	Corvette	Sporty	5.7	8	300	25
4	LeBaron	Compact	3.0	4	141	28
5	Stealth	Sporty	3.0	6	300	24
6	Prelude	Sporty	2.3	4	160	31

The *pipeline* above accomplishes the same set of operations as the code from the last slide.

```
> Cars93_Ex7<-select(filter(Cars93, Type %in% c("Sporty","Compact") & Horsepower >=120),  
+ Model,Type,EngineSize,Cylinders,Horsepower,MPG.highway)
```

# The mutate Function

The `mutate` function allows you to create a new data frame consisting of the original data frame with a column appended on the right end.



The general syntax is `mutate(data_frame, Var_Name = function(...))`. The new column will have the name given in `Var_Name` and be computed from the function provided.



# Example of the mutate Function

We now add a column to the `Cars93` data frame containing the horsepower per liter of engine size. That is, we define the new variable `HPpLiter` by the function

$$\text{HPpLiter} = \frac{\text{Horsepower}}{\text{EngineSize}}$$

```
> Cars93_Ex9<-select(mutate(Cars93,HPpLiter=Horsepower/EngineSize) ,  
+ Model,Type,EngineSize,Cylinders,Horsepower,MPG.highway,HPpLiter)  
> head(Cars93_Ex9)
```

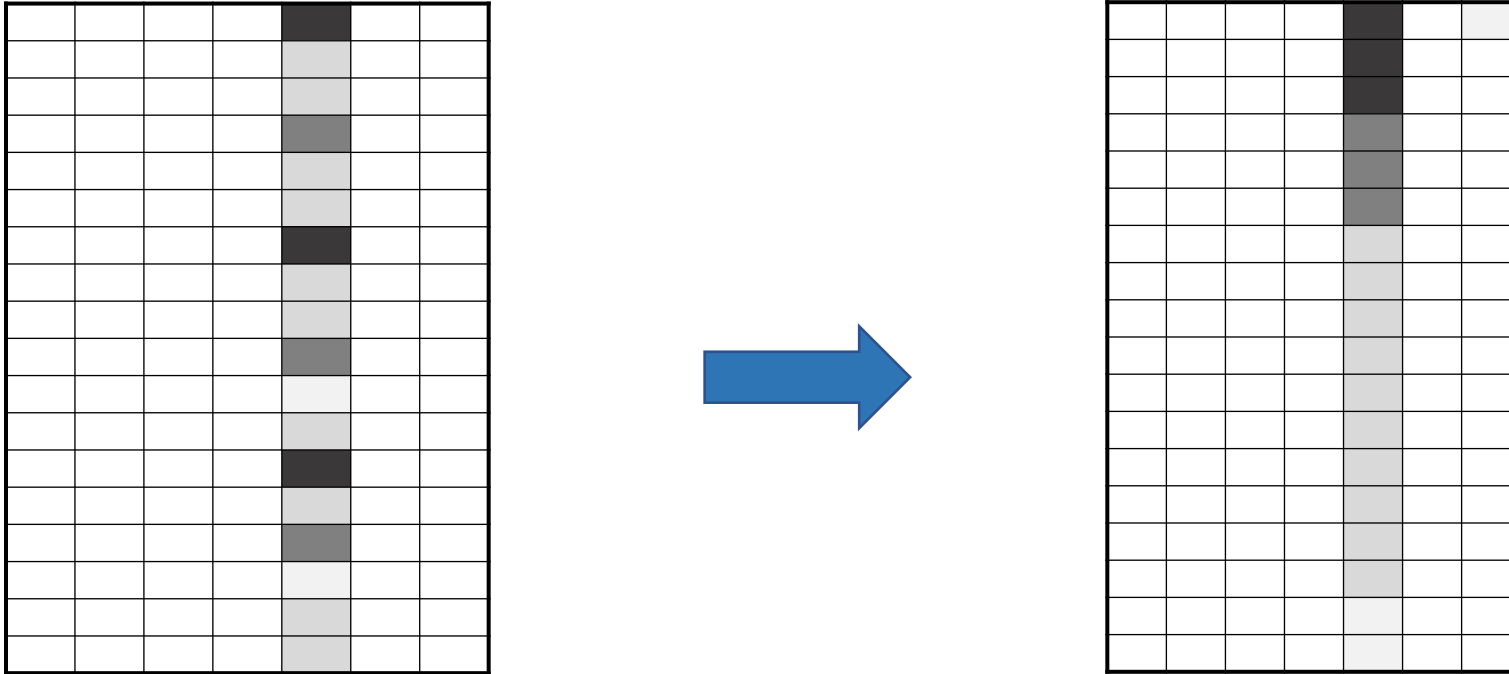
	Model	Type	EngineSize	Cylinders	Horsepower	MPG.highway	HPpLiter
1	Integra	Small	1.8	4	140	31	77.77778
2	Legend	Midsize	3.2	6	200	25	62.50000
3	90	Compact	2.8	6	172	26	61.42857
4	100	Midsize	2.8	6	172	26	61.42857
5	535i	Midsize	3.5	4	208	30	59.42857
6	Century	Midsize	2.2	4	110	31	50.00000

We could have also written this using the equivalent pipeline

```
> Cars93_Ex9<-Cars93 %>%  
+ mutate(HPpLiter=Horsepower/EngineSize) %>%  
+ select(Model,Type,EngineSize,Cylinders,Horsepower,MPG.highway,HPpLiter)
```

# The `arrange` Function

The `arrange` function allows you to sort a data frame by variable producing a new data frame ordered by that variable.



The general syntax is `arrange(data_frame, Var_Name1, Var_Name2, ...)`. The rows are sorted by the variables selected in the order they are entered. That is, on `Var_Name1` first, with ties broken by `Var_Name2`, etc. The default is ascending order, and descending order can be selected by using `desc(Var_Name1)` when passing in the variable name.

# Example of the arrange Function

In the previous example we used the `mutate` function to add a column to the `Cars93` data frame containing the horsepower per liter. We now sort this new data frame first by the number of cylinders and then by the engine's horsepower.

```
> Cars93_Ex10<-arrange(Cars93_Ex9,Cylinders,desc(Horsepower))
> head(Cars93_Ex10)
```

	Model	Type	EngineSize	Cylinders	Horsepower	MPG.highway	HPpLiter
1	Justy	Small	1.2	3	73	37	60.83333
2	Swift	Small	1.3	3	70	43	53.84615
3	Metro	Small	1.0	3	55	50	55.00000
4	535i	Midsize	3.5	4	208	30	59.42857
5	626	Compact	2.5	4	164	34	65.60000
6	Prelude	Sporty	2.3	4	160	31	69.56522

This could have been accomplished without the previous work using the pipeline

```
> Cars93_Ex10<-Cars93 %>%
+ mutate(HPpLiter=Horsepower/EngineSize) %>%
+ select(Model,Type,EngineSize,Cylinders,Horsepower,MPG.highway,HPpLiter) %>%
+ arrange(Cylinders,desc(Horsepower))
```

Observe that we used `desc(Horsepower)` to sort on this variable from highest to lowest.

# The rename Function

The `rename` function allows you to rename columns in a data frame.

The syntax is `rename(data_frame, New_Name1=Old_Name1, New_Name2=Old_Name2 ...)`. The variable named `Old_Name` is renamed with the value entered for `New_Name` in the output data frame. Rows are not affected, and the column order remains unchanged.

Example: We can rename two of the columns in the data frame from the previous example using the `rename` function as follows:

```
> Cars93_Ex11<-rename(Cars93_Ex10,HP=Horsepower,MPG_Highway=MPG.highway)
> head(Cars93_Ex11)
```

	Model	Type	EngineSize	Cylinders	HP	MPG_Highway	HPpLiter
1	Justy	Small	1.2	3	73	37	60.83333
2	Swift	Small	1.3	3	70	43	53.84615
3	Metro	Small	1.0	3	55	50	55.00000
4	535i	Midsize	3.5	4	208	30	59.42857
5	626	Compact	2.5	4	164	34	65.60000
6	Prelude	Sporty	2.3	4	160	31	69.56522

# More Examples with the `mutate` Function

Suppose we are interested in the engines of the various cars. We previously used the `mutate` function to add a column containing the horsepower per liter of engine size.

```
> head(Cars93_Ex11)
```

	Model	Type	EngineSize	Cylinders	HP	MPG_Highway	HPpLiter
1	Justy	Small	1.2	3	73	37	60.83333
2	Swift	Small	1.3	3	70	43	53.84615
3	Metro	Small	1.0	3	55	50	55.00000
4	535i	Midsize	3.5	4	208	30	59.42857
5	626	Compact	2.5	4	164	34	65.60000
6	Prelude	Sporty	2.3	4	160	31	69.56522

What if we wanted the data in the `HPpLiter` column to be rounded to the nearest tenth? (Note that we could have done this when we first added the column, but we forgot). We can use the `mutate` function to accomplish this as well.

```
> Cars93_Ex12<-mutate(Cars93_Ex11,HPpLiter=round(HPpLiter,1))
```

Observe that we have entered a variable name that is already in the data frame. How does the `mutate` function process this request?

# More Examples with the mutate Function

Viewing the function output produces

```
> head(Cars93_Ex12)
```

	Model	Type	EngineSize	Cylinders	HP	MPG_Highway	HPpLiter
1	Justy	Small	1.2	3	73	37	60.8
2	Swift	Small	1.3	3	70	43	53.8
3	Metro	Small	1.0	3	55	50	55.0
4	535i	Midsize	3.5	4	208	30	59.4
5	626	Compact	2.5	4	164	34	65.6
6	Prelude	Sporty	2.3	4	160	31	69.6

The `mutate` function did not append a new column on the end of the data frame, but rather **replaced** the existing column with the matching name with the updated information.

Suppose that engines producing 65 horsepower or more per liter of displacement can be considered high-performance while those producing less than this value are regular performance. How can we add a column indicating this to our data frame?

# More Examples with the mutate Function

We can combine the `mutate` function with the `ifelse` function to obtain this result

```
> Cars93_Ex13<-mutate(Cars93_Ex12,Performance=ifelse(HPpLiter>=65,"High","Regular"))
> head(Cars93_Ex13)
```

	Model	Type	EngineSize	Cylinders	HP	MPG_Highway	HPpLiter	Performance
1	Justy	Small	1.2	3	73	37	60.8	Regular
2	Swift	Small	1.3	3	70	43	53.8	Regular
3	Metro	Small	1.0	3	55	50	55.0	Regular
4	535i	Midsize	3.5	4	208	30	59.4	Regular
5	626	Compact	2.5	4	164	34	65.6	High
6	Prelude	Sporty	2.3	4	160	31	69.6	High

The `mutate` function appends the result of the `ifelse` function on the right side of the data frame. Note that the format of the `ifelse` function is `ifelse(condition, what if true, what if false)`. We can obtain further information using

```
> table(Cars93_Ex13$Performance)
```

High	Regular
16	77



# More Examples with the mutate Function

The `ifelse` statements can also be nested. As an example, suppose we have three classifications for the `Performance` variable.

```
> Cars93_Ex14<-Cars93_Ex13 %>%  
+ mutate(Performance=ifelse(HPpLiter>=65,"High",ifelse(HPpLiter>=55,"Regular","Low")))  
> head(Cars93_Ex14)
```

	Model	Type	EngineSize	Cylinders	HP	MPG_Highway	HPpLiter	Performance
1	Justy	Small	1.2	3	73	37	60.8	Regular
2	Swift	Small	1.3	3	70	43	53.8	Low
3	Metro	Small	1.0	3	55	50	55.0	Regular
4	535i	Midsize	3.5	4	208	30	59.4	Regular
5	626	Compact	2.5	4	164	34	65.6	High
6	Prelude	Sporty	2.3	4	160	31	69.6	High

Here the `mutate` function **replaces** the existing `Performance` column with the updated values which has `High` for engines with 65 `HPpLiter` or more, `Regular` for those satisfying  $55 \leq \text{HPpLiter} < 65$  and `Low` performance for engines with `HPpLiter` < 55. We can get counts using

```
> table(Cars93_Ex14$Performance)
```

High	Low	Regular
16	48	29

# Tibbles, Row Names, and `dplyr`

The package `dplyr` is part of the `Tidyverse`, a collection of packages “designed to work together to make common data science operations more user friendly.”

There are several differences between how the tidyverse views data objects and how traditional R views them. Two important distinctions

- The data frame has been replaced by the tibble as the basic data object.
- Row names have been deprecated.

**NOTE:** A large number of tidyverse functions will work with both tibbles and dataframes, and the data structure of the output will be identical to the input. However, there are some functions that will return a tibble (without row names), whether or not a tibble or dataframe is provided.

# Tibbles

A core component of the tidyverse is the tibble. Tibbles are a modern rework of the standard `data.frame`, with some internal improvements to make code more reliable. They are similar to data frames, but do not follow all of the same rules. For example, tibbles can have numbers/symbols for column names, which is not allowed in base R.

If you use a function from the `dplyr` package that returns a tibble as output, you can convert it to a data frame using the `data.frame` function. For example

```
> Result
```

```
# A tibble: 3 x 4
```

	cyl	N	Mean_HP	Mean_mpg
	<dbl>	<int>	<dbl>	<dbl>
1	4	11	82.6	26.7
2	6	7	122.	19.7
3	8	14	209.	15.1

```
> Result<-data.frame(Result)
```

```
> Result
```

	cyl	N	Mean_HP	Mean_mpg
1	4	11	82.63636	26.66364
2	6	7	122.28571	19.74286
3	8	14	209.21429	15.10000

# rownames and dplyr

**Important:** tidyverse is very opinionated about row names. These packages insist that all column data (e.g. data.frame) be treated equally, and that special designation of a column as rownames should be deprecated. Package `tibble` provides simple utility functions to handle rownames: `rownames_to_column` and `column_to_rownames`.

```
> head(mammals)
```

	body	brain
Arctic fox	3.385	44.5
Owl monkey	0.480	15.5
Mountain beaver	1.350	8.1
Cow	465.000	423.0
Grey wolf	36.330	119.5
Goat	27.660	115.0

```
> has_rownames(mammals)
```

```
[1] TRUE
```

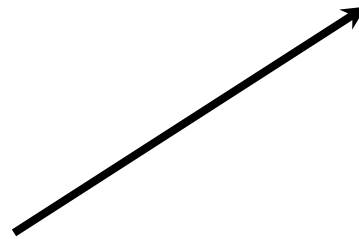
```
> mammals_rn<-rownames_to_column(mammals, "Mammal")
```

```
> has_rownames(mammals_rn)
```

```
[1] FALSE
```

```
> head(mammals_rn)
```

	Mammal	body	brain
1	Arctic fox	3.385	44.5
2	Owl monkey	0.480	15.5
3	Mountain beaver	1.350	8.1
4	Cow	465.000	423.0
5	Grey wolf	36.330	119.5
6	Goat	27.660	115.0



# rownames and dplyr

What if you want the row names to function as rownames? For example, what if you have an R function that uses the row names of an object to perform some kind of task?

There are two ways to handle this: 1. Make a new data frame with the row names.

```
> head(mammals_rn, 5)
```

	Mammal	body	brain
1	Arctic fox	3.385	44.5
2	Owl monkey	0.480	15.5
3	Mountain beaver	1.350	8.1
4	Cow	465.000	423.0
5	Grey wolf	36.330	119.5

```
> mammals_2<-column_to_rownames(mammals_rn, "Mammal")
```

```
> head(mammals_2, 5)
```

	body	brain
Arctic fox	3.385	44.5
Owl monkey	0.480	15.5
Mountain beaver	1.350	8.1
Cow	465.000	423.0
Grey wolf	36.330	119.5

2. Pass in the argument to the function by first applying the `column_to_rownames` function on the data frame.

```
> function(mammals_rn, optional_arg1,... )
```

```
> function(column_to_rownames(mammals_rn,"Mammal"), optional_arg1, ... )
```

# The summarize Function

- The `summarize` function allows you to produce a data frame with user chosen statistics calculated from the columns of the input data frame.
- Often used with the `group_by` function which allows the statistics to be computed for particular groups in the input data frame.

Example: We have the data on the engines in the `Cars93_Ex14` data frame.

```
> head(Cars93_Ex14)
```

	Model	Type	EngineSize	Cylinders	HP	MPG_Highway	HPpLiter	Performance
1	Justy	Small	1.2	3	73	37	60.8	Regular
2	Swift	Small	1.3	3	70	43	53.8	Low
3	Metro	Small	1.0	3	55	50	55.0	Regular
4	535i	Midsize	3.5	4	208	30	59.4	Regular
5	626	Compact	2.5	4	164	34	65.6	High
6	Prelude	Sporty	2.3	4	160	31	69.6	High

We can use the `summarize` function to obtain information about the engines grouped by, for example, the car `Type` or the `Cylinders` variable.

# The summarize Function

An example of the `summarize` function is provided in the following pipeline:

```
> Engine_Summary<-Cars93_Ex14 %>%  
+ group_by(Type) %>%  
+ summarise(Num=n(),Min_Size=min(EngineSize),Max_Size=max(EngineSize),Ave_HP=mean(HP),  
+ Median_MPG_HWY=median(MPG_Highway),Num_High_Performance=sum(Performance=="High"))  
`summarise()` ungrouping output (override with `.groups` argument)  
> Engine_Summary<-data.frame(Engine_Summary)  
> Engine_Summary
```

	Type	Num	Min_Size	Max_Size	Ave_HP	Median_MPG_HWY	Num_High_Performance
1	Compact	16	2.0	3.0	131.0000	30.0	4
2	Large	11	3.3	5.7	179.4545	26.0	0
3	Midsize	22	2.0	4.6	173.0909	26.5	4
4	Small	21	1.0	2.2	91.0000	33.0	5
5	Sporty	14	1.3	5.7	160.1429	28.5	3
6	Van	9	2.4	4.3	149.4444	22.0	0

The code above creates five new variables of interest and calculates them for the groups determined by the `group_by` function. The `n()` function gives the number of units in the group. The output of the function is a tibble we convert to a data frame using the `data.frame` function.



# Example of the summarize Function

A second example of the `summarize` function is provided in the following pipeline:

```
> Engine_Summary_2<-Cars93_Ex14 %>%  
+ group_by(Cylinders) %>%  
+ summarise(Num=n(),Min_Size=min(EngineSize),Max_Size=max(EngineSize),Ave_HP=mean(HP),  
+ Median_MPG_HWY=median(MPG_Highway),Num_High_Performance=sum(Performance=="High"))  
`summarise()` ungrouping output (override with `.groups` argument)  
> Engine_Summary_2<-data.frame(Engine_Summary_2)  
> Engine_Summary_2
```

	Cylinders	Num	Min_Size	Max_Size	Ave_HP	Median_MPG_HWY	Num_High_Performance
1	3	3	1.0	1.3	66.0000	43.0	0
2	4	49	1.3	3.5	113.4694	31.0	10
3	5	2	2.4	2.5	138.5000	24.5	1
4	6	31	2.8	5.7	175.5806	26.0	4
5	8	7	4.5	5.7	234.7143	25.0	0
6	rotary	1	1.3	1.3	255.0000	25.0	1

The same five new variables of interest are calculated but this time the engines are grouped by the `Cylinders` variable. Again, we which we convert the tibble output to a data frame using the `data.frame` function.