Naming identifiers

Allowed: letters, digits, period (.) and underscore (_).
Must start with letter or period. If starts with period, cannot be followed by digit.
Cannot use keywords such as TRUE, IF, etc.

variable <- "value"
typeof(variable) -> "logical"

Useful Facts

As you create (i.e., assign) objects in an R session, the objects will appear in the **Environment.**
Use **?** to look for help in R.
To install a package, do **install.packages("name of the package")**

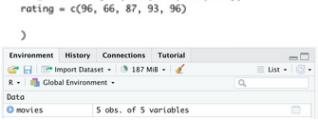
Dataframes

part of dplyr, inside tidyverse package.
a table with **columns representing variables** and **rows representing observations.**
(think of a 2x2 matrix array, **All vectors in data frame must have the same number of elems.**)
Can also use **data.frame(), a part of R's standard library**

```
movies <- data_frame(

  title = c("Groundhog Day", "Mission: Impossible", "Inception", "Terminator 2", "Back to the Future"),
  year = c(1993, 1996, 2010, 1991, 1985),
  duration = c(101, 110, 148, 137, 116),
  box_office = c(105, 458, 837, 521, 389),
  rating = c(96, 66, 87, 93, 96)

)
```



get column values: **name_of_data_frame$name_of_column**

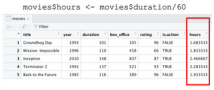
```
> movies$rating
[1] 96 66 87 93 96
```

Suppose you are asked to create a vector whose elements are TRUE if each of the corresponding movies is longer than 2 hours, and FALSE otherwise

```
> movies$duration > 120
[1] FALSE FALSE  TRUE  TRUE FALSE
```

create new column: **name_of_dataframe$new_col <- movies$duration/60**

```
movies$hours <- movies$duration/60
```



NULL VALUES

By default, if there is a missing value, the mean will evaluate to **NA**

```
x <- c(1, 2, 3, 4, NA)
```

```
> 1+NA
[1] NA
> NA > 3
[1] NA
> NA != TRUE
[1] NA
```

```
> mean(x)
[1] NA
```

na.rm() is a logical value indicating whether NA values should be ignored. default is FALSE

```
> mean(x, na.rm = TRUE)
[1] 2.5
```

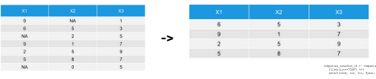
is.na(x) indicates which elements are missing.

```
> is.na(x)
[1] FALSE FALSE FALSE FALSE  TRUE
```

sum(is.na(x)) adds up all the elements that evaluate to 1, thus giving the **number of missing elements.** is.na() also works with data frames.

```
> sum(is.na(x))
[1] 1
```

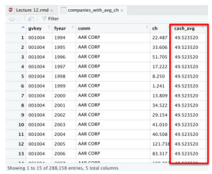
data_omit <- na.omit(data) any rows with NA are deleted



->

group_by()
Creates groups of observations based on one or more variables

Ex. to get average cash for each firm, enter:
companies_with_avg_ch <- companies %>%
group_by(gvkey) %>%
mutate(cash_avg = mean(ch, na.rm = TRUE))



group_by() WITH TWO VARIABLES

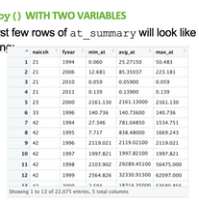
- To do this, enter into the Console:

```
at_summary <- companies %>%
  filter(!is.na(naicsh), !is.na(fyear)) %>%
  group_by(naicsh, fyear) %>%
  summarise(
     min_at = min(at, na.rm = TRUE),
     avg_at = mean(at, na.rm = TRUE),
     max_at = max(at, na.rm = TRUE))
```

It is always a good practice to remove observations with NA values in the columns that will be used as group_by variables, before doing a group_by.

group_by () WITH TWO VARIABLES

- The first few rows of `at_summary` will look like the following:



Operations & Data Structures

AND: &
OR: |
NOT: ! Missing Values: **NA**

Vectors
Must be of same data type.
city <- c("Vancouver", "Victoria", "Delta", "Surrey")

length(x)

```
> city <- c("Vancouver", "Victoria", "Delta", "Surrey")
> length(city)
[1] 4
```

Subsetting
R IS ONE INDEXED.

```
> expensive_city <- city[1:2]
> expensive_city
[1] "Vancouver" "Victoria"
```

Use - to remove elements, : to return range. can assign subsets to new vectors.

city[2] returns "Victoria"
city[1:3] returns a vector "Vancouver" "Victoria" "Delta"
city[c(1,2,4)] returns specified elements "Vancouver", "Victoria", "Surrey"
city[-3] returns "Vancouver" "Victoria" "Surrey"

Can update the value(s) of the subset of a vector.

```
> expensive_city[2] <- "Toronto"
> expensive_city
[1] "Vancouver" "Toronto"
```

Can also add new elements to an existing vector.

```
> length(expensive_city)
[1] 2
> expensive_city[3:4] <- c("New York","Los Angeles")
> expensive_city
[1] "Vancouver"   "Toronto"     "New York"    "Los Angeles"
```

If you perform an operation with vectors, the operation will be applied to each element of the vector.

```
> city_temp + 2
[1] 27 22 29 32
```

Function	What it returns
max(x)	the maximum value of x
min(x)	the minimum value of x
range(x)	the minimum and maximum values of x
length(x)	the number of elements in x
sum(x)	the sum of the elements in x
mean(x)	the average value of the elements in x
sd(x)	the standard deviation of x
var(x)	the variance of x
sqrt(x)	the square root of x

round(value_to_round, decimal_places_to_round_to) ex. round(x/y, 3))

Data Wrangling

The process of transforming and mapping data from one raw data form into another, with the intent of making the data more useful (e.g., for analytics).
Packages used: **dplyr, magrittr** shipped with **tidyverse**

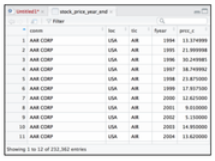
dplyr

Function	How it works
select()	picks variables based on their names
filter()	picks observations based on their values
mutate()	adds new variables that are functions of existing variables
summarise()	reduces multiple values to a summary statistics
arrange()	changes the order of rows
group_by()	allows a user to perform the above functions on a subset of the data

UNIVERSITY OF BRITISH COLUMBIA SAUDER SCHOOL OF BUSINESS

Loading rds files (singular r objects)

```
> companies <- readRDS("~/North American Stock Market 1994-2018.rds")
```



select(data_object_name, variable_name(s))
keeps only the variables you specify from the dataset

price_yr_end <- select(companies, conm, loc, tic, fyear, prcc_c)

filter(data_object_name, condition(s))
finds rows/cases where conditions specified are true. If true, those rows are kept. Otherwise, they are dropped.

companies_canadian_large <- filter(companies, loc=="CAN", at > 1000)

mutate(data_object_name, new_variable_name=function_name(existing_variable_names)
Using this function, you can create multiple new variables by separating them with commas.

(new variable called full_ch which is is ch multiplied by 1 mill)
companies %>% mutate(full_ch = 1000000*ch) is identical to mutate(companies, full_ch = 1000000*ch)

summarise(data_object_name, new_variable_name = aggregate_function(existing_variable_names))
used to create an aggregate statistic over all of the observations (or subgroups, if specified)

Suppose you want to find the maximum and minimum cash recorded for Canadian companies in 2010.

```
max_min_cash_Canadian_2010 <- companies %>%
    filter(loc=="CAN" & fyear==2010) %>%
    summarise(   max_cash = max(ch, na.rm = TRUE),
                 min_cash = min(ch, na.rm = TRUE))
```

max_min_cash_Canadian_2010 ->

	max_cash	min_cash
1	17368	0

Common Tasks

If the objective is to perform aggregate operations over groups and to display one aggregate result per group, you should use group_by() with summarise().

The syntax is:

```
data_object %>%
group_by(group_variable(s)) %>%
summarise(new_variable =
aggregate_function(existing_variable(s)))
```

There are four main data types in Excel

Logical Type: `TRUE` or `FALSE`

Number Type: Whole numbers (e.g., `1`, `-13`, or `678`) or decimal numbers (`0.45`, `45.6778`, or `-55.53`).

Text Type (string of characters): Data which cannot be resolved as number or logical type is stored as text type (up to 32,768 characters). All characters can be stored as text (`A-Z`, `a-z`, `0-9`, `!@#$` etc.)

Error Type: Values returned when Excel encounters an error in evaluating the contents of a cell (`#DIV/0`, `#N/A`, `#NAME?`, `#NULL!`, `#NUM!`, `#REF!`, and `#VALUE!`)

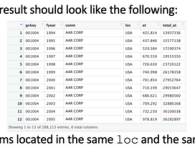
sum() FUNCTION
sum() is another aggregate function that can be used within summarise() or mutate(). Returns the total values of the vector passed as an argument.

Ex . Calculate the sum of total assets of all firms by country of headquarters (loc) and fiscal year (fyear). Call this new variable tot_at. Retain all the observations in the original data.

Can use the sum() function with mutate().

```
tot_at <- companies %>%
  filter(!is.na(fyear), !is.na(loc)) %>%
  group_by(fyear, loc) %>%
  mutate(total_at = sum(at, na.rm = TRUE))%>%
  select(gvkey, fyear, conm, loc, at, total_at)
```

Your result should look like the following:

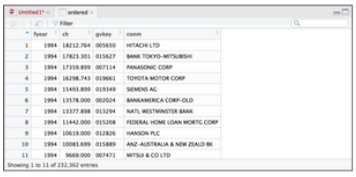


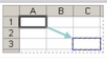
All firms located in the same `loc` and the same `fyear` are grouped together as a subset, and then R returns the total of `at` for each subset – the country/year combination.

arrange() FUNCTION
orders the rows by variable(s) specified, in ascending order.

Ex. Arrange observations in companies in ascending order of fyear and descending order of ch within each fyear.

```
ordered <- companies %>%
  filter(!is.na(fyear), !is.na(ch)) %>%
  arrange(fyear, desc(ch)) %>%
  select(fyear, ch, gvkey, conm)
```



FOR A FORMULA BEING COPIED:	IF THE REFERENCE IS:	IT CHANGES TO:
	A1 (absolute column and absolute row)	A1 (the reference is absolute)
	A$1 (relative column and absolute row)	C$1 (the reference is mixed)
	$A1 (absolute column and relative row)	$A3 (the reference is mixed)
	A1 (relative column and relative row)	C3 (the reference is relative)

MAGRITTR

pipe operator

When the pipe operator (i.e., %>%) is used, the dplyr functions (like select() and filter()) take the data from the previous step as their input.

companies %>% filter(loc=="CAN") is identical to **filter(companies, loc=="CAN")** examples:

inner_join(dataset_x, dataset_y) FUNCTION

returns all **rows** from x where there are **matching values in y**, and **all columns from x and y.**

merged1 <- inner_join(example1,example2)
To specify this match more clearly (which is good practice), you can do the following:

```
merged1 <- inner_join(example1,example2, by = c("gvkey", "fyear"))
```

left_join(dataset_, dataset_y) FUNCTION

returns all rows left_join(x, y) **returns all rows from x**, and all columns from x and y

merged2 <- left_join(example1,example2)

```
merged2 <- left_join(example1,example2)
```

- R will return `Joining, by = c("gvkey", "fyear")`

	gvkey	fyear	conm	ni	at	sale
1	001004	2014	AAR CORP	10.200	NA	NA
2	001004	2015	AAR CORP	47.700	NA	NA
3	001004	2016	AAR CORP	56.500	1504.100	1767.600
4	001045	2014	AMERICAN AIRLINES GROUP INC	2882.000	NA	NA
5	001045	2015	AMERICAN AIRLINES GROUP INC	7610.000	NA	NA
6	001045	2016	AMERICAN AIRLINES GROUP INC	2676.000	51274.000	40180.000

Observations in the "left" dataset with no matching observations in the other dataset will be retained in the resultant dataset. Observations in the "right" dataset which do not have matching datasets, will not be returned.

Non One To One Matching

When you merge two datasets whose key variables uniquely identify each observation within the dataset, this kind of merge is called a one-to-one merge.

• However, the attribute(s) (i.e., column(s)) used as a basis of matching the observations in two data sets do not have to be key variables for both datasets.

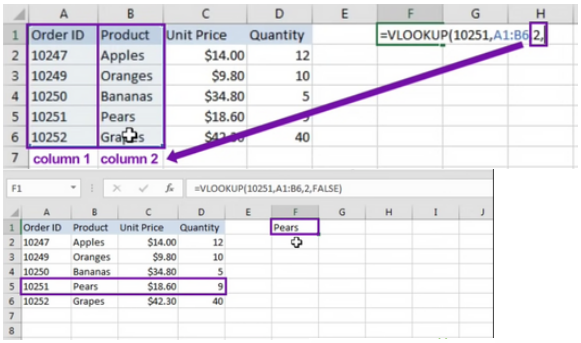
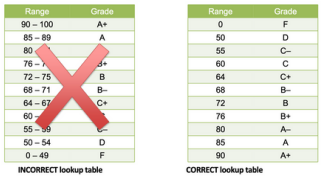
• In such situations, an observation in one dataset can match with multiple observations in the other datasets through non-one-to-one matching.

• However, the variable or variable(s) to perform the merge should uniquely identify at least one of the datasets.

You want to merge the two datasets, so that each observation will have the following variables: `gvkey, fyear, naicsh`, and `NAICS_description`. This will attempt to add the industry description to every observation in the Example3 dataset.

To do this, enter into the Console:

```
merged3 <- left_join(example3, NAICS_2_6_digit_codes, by = c("naicsh" = "NAICS"))
```

What	Syntax	Description	Example
IF	=IF(logical_test, [value_if_true], [value_if_false])	Logical Operator	=IF(B2>D2, "Good", "Bad")
MAX	=MAX(number1, [number2], ...)	Values can be numbers (e.g. 10), a cell reference that contains numbers (e.g. A8), a cell range/array that contains numbers (e.g. A8:A10), or a combination of the three.	
MIN	=MIN(number1, [number2], ...)		
SUM	=SUM(number1, [number2], ...)		
AND	=AND(logical1, [logical2], ...)	Used to evaluate if all logicals are TRUE or FALSE	
OR	=OR(logical1, [logical2], ...)	Used to evaluate if at least one of the logicals are TRUE or FALSE	
COUNTIFS	=COUNTIFS(criteria_range1, criteria1, ...)	Count values in a specific range that meet certain criteria. At least 1 criterion is needed, but up to 127 criteria can be tested. **Criteria range** refers to range of cells you want to test for Criteria can be in the form of a number, cell reference, or text. Do NOT use formulas/functions for criteria..	Ex. How many NY airports had at least 10 different airlines and at least 1 million passenger departures in 2014? 1) airlines >=10 2) passenger departures >=1000000 **=COUNTIFS(D3:D7,">=10",C3:C7,">=1000000")** returns amount that fulfills the criteria.
SUMIFS	=SUMIFS(sum_range, criteria_range1, criteria1, ...) **sum_range**: the cells you want to sum **criteria_range**: the values you're checking **criteria1**: what do you want it to equal to?	Adds values in a specified range or multiple ranges that meet specified criteria. At least 1 criterion is needed, but up to 127 criteria can be tested. Same as COUNTIFS, but sums it up afterward. IMPORTANT: your sum_range must be of the same size and shape with your criteria_range. Ex. if you select C3:C7 as your sum_range, you can use D3:D7 as your criteria range because both are of the same size (5 cells going downward), but you cannot use D3:D8 as your criteria range.	Ex. What are the total passenger departures at NY airports that had at least 10 different airlines and at least 1 million passenger departures in 2014? Our sum_range is passenger departures (C3:C7). **=SUMIFS(C3:C7,D3:D7,">=10",C3:C7,">=1000000")**
VLOOKUP	=VLOOKUP(lookup_value, table_array, col_index_num, [range_lookup]) =VLOOKUP(VALUE, table, index_number, approx_match) **value** The value to look for in the first column of the table. **table:** Two or more columns of data that's sorted in ascending order **index_number:** The column number in the table from which the matching value must be returned. The first number is one. **approximate_match:** Optional. FALSE = EXACT MATCH, TRUE = APPROXIMATE MATCH (if omitted, TRUE is DEFAULT	The VLOOKUP function looks up a value in the first column of a range of cells, and then returns a value from any cell on the same row of the range. exact matches, or approximate matches: the nearest value that is less than or equal to the search value. For example, you can use VLOOKUP to assign a grade of B+ for students who receive a grade from 76 to 79.	 If the VLOOKUP is used to find an exact match, the rows can be in **any** order. VLOOKUP starts with the first value in the first column, and works its way down until it finds a match. Therefore, to avoid mistakes, make sure that **each value in the first column is unique**. That is, that there are no duplicates in the first column. **Approximate Matches** • If you want to look up a value in a range (e.g. 76 to 79), you **MUST** arrange the lookup table so that the data are sorted from the lowest to the highest. • You **MUST only include the lowest value in the range** (in this case, 76). It is called the **breakpoint.** • Do **NOT** use the complete range in Column 1 (i.e. NOT 76 – 79). 
LEFT	=LEFT(text,num_chars)	LEFT function returns the first character(s) in a string of text based on the specified number of characters.	**Faculty is = RIGHT(A2,1)** **Status is =LEFT(A2,1)** **Year Entered is =MID(A2,5,2** **FULL NAME is =CONCATENATE(C2," ",B2).) OR =C2&" "&B2**
RIGHT	=RIGHT(text,num_chars)	return the last character(s) in a string of text based on the specified number of characters.	
MID	=MID(text,start_num,num_chars) **start_num:** the (one indexed, inclusive) number to start at. **num_chars:** the number of characters to include.	MID function returns character(s) in a string of text, starting at a specified position and based on the specified number of characters.	

	A	B	C	D	E	F	G
1	StudentNo	LastName	FirstName	Status	Year Entered	Faculty	Full Name
2	D20511A	Hadfield	Chris	D	11	A	Chris Hadfield
3	D38512C	Gretzky	Wayne	D	12	C	Wayne Gretzky
4	I49511A	Winfrey	Oprah	I	11	A	Oprah Winfrey
5	I55313A	Lin	Jeremy	I	13	A	Jeremy Lin
6	I52312C	Clinton	Hillary	I	12	C	Hillary Clinton

CONCATENATE	=CONCATENATE(text1,[text2],...) =C2&" "&B2 (alternate)	CONCATENATE function join two or more strings of text into one. Up to 255 strings of text can be joined. Instead of using the CONCATENATE function, we could also use the ampersand (&) operator to join multiple strings of text into one.	

	A	B	C	D	E
1	Course Code/No	Short Course Name	Char Count	Trimmed Course Code	BUSI Course Code
2	COMM 293	FINANCIAL ACCTNG	16	COMM 293	BUSI 293
3	COMM 294	MANAGERIAL ACCTG	16	COMM 294	BUSI 294
4	COMM 329	ORGANIZTL BHVR	14	COMM 329	BUSI 329
5	COMM 335	INFO SYS MNGT	13	COMM 335	BUSI 335
6	COMM 393	COMMERCIAL LAW	14	COMM 393	BUSI 393

Trimmed Course Code is =TRIM(A2)
Char Count is =LEN(A2)
BUSI Course Code is =SUBSTITUTE(D2,"COMM","BUSI").

New Course Code is =REPLACE(A9,6,1,2)

LEN	=LEN(text)	Returns the number of characters in a text string. Spaces are counted, too.	
TRIM	=TRIM(text)	TRIM function removes all spaces from text strings, EXCEPT for single spaces between words.	
SUBSTITUTE	=SUBSTITUTE(text,old_text,new_text,[instance_num]) **Instance_num:** optional, If you use instance number (1, 2, 3, etc.), Excel will only substitute the new_text for the old_text in that specific instance.	Substitutes a new text string for an old text string. Ex. Dean wants to change COMM 335 to COMM 235. If you decide to use SUBSTITUTE function to change the course number, instance_number is required because you only want to change the number 3 the first time Excel finds the number 3. The instance number is, therefore, 1.	
REPLACE	**=REPLACE(old_text,start_num,num_chars,new_text)**	**REPLACE** function replaces an old text string with a new text string, starting from a specific location of text string and based on a specified number of characters.	
FIND	**=FIND(find_text,within_text,[start_num])** **Find_text:** refers to the text you want to find. If find_text is not in the within_text, the FIND function will return the error **#VALUE!** **Start_num:** is optional. Start_num refers to the character position at which FIND function will start looking for the text.	**The FIND function does NOT allow wildcard characters.** **The FIND function is case sensitive.**	
SEARCH	=SEARCH(find_text,within_text,[start_num])	The SEARCH function **allows wildcard characters and is NOT case sensitive.**	

G	H	I
Old Course Code	Formula Used	New Course Code
COMM 335	=SUBSTITUTE(G2,3,2,1)	COMM 235
COMM 335	=SUBSTITUTE(G3,3,2,2)	COMM 325
COMM 335	=SUBSTITUTE(G4,3,2)	COMM 225

	A	B
8	Old Course Code	New Course Code
9	COMM 335	COMM 235
10	COMM 396	COMM 296
11	COMM 397	COMM 297

	A	B	C	D
1	Within Text	Formula Used	Result	Description
2	Mississippi	=FIND("issi",A2)	2	Starts from the first character ("M") and returns the starting position of the first "issi" found.
3	Mississippi	=FIND("issi",A3,3)	5	Starts from the third character (the first "s") and returns the starting position of the first "issi" found.
4	Mississippi	=FIND("ISSI",A4)	#VALUE!	Results in #VALUE! error value because there is no series of characters of "ISSI" in the text string. Recall that FIND is **case sensitive.**

	A	B	C
1	Wildcard Char	Description	Example
2	?	Any single character. One ? is equal to one character.	Use **n?n** to search for any text that has any two characters between the first and second n. Examples: noun or noon. If you use n??n to search for **non** , however, it won't work since there is only one character in between the first and second n .
3	*	Any number of characters.	Use **n*n** to search for any text that has any series of characters beginning with n and ending with n. Examples: noun, noon, non, mining, business in China, and even just nn (zero is also considered a valid number).
4	~ followed by ?	To look for ? question mark.	Example: To look for "What?", use "What~?".
5	~ followed by *	To look for * asterisk.	Example: To look for "5*hotel", use "5~*hotel".
6	~ followed by ~	To look for ~ tilde.	Example: To look for "East~West", use "East~~West".

	A	B	C	D
1	Within Text	Formula Used	Result	Description
2	Mississippi	=SEARCH("issi",A2)	2	Starts from the first character ("M") and returns the starting position of the first "issi" found.
3	Mississippi	=SEARCH("issi",A3,3)	5	Starts from the third character (the first "s") and returns the starting position of the first "issi" found.
4	Mississippi	=SEARCH("i*i",A4)	2	Starts from the first character ("M") and returns the starting position of a series of characters that begins with "i" and ends with "i" with any number of characters in between.
5	Mississippi	=SEARCH("i??i",A5)	2	Starts from the first character ("M") and returns the starting position of a series of characters that begins with "i" and ends with "i" with 2 characters in between (number of ? = number of characters in between).
6	Mississippi	=SEARCH("i*i",A6,6)	8	Starts from the sixth character (the third "s") and returns the starting position of a series of characters that begins with "i" and ends with "i" with any number of characters in between.
7	Mississippi	=SEARCH("issi",A7,SEARCH("issi",A7)+1)	5	Works similarly to the second example above. Starts from the third character since the start_num **SEARCH("issi",A7)+1** returns the number 3. Returns the starting position of the first "issi" found after the third character (the first "s").
8	Mississippi	=SEARCH("i?i",A8)	#VALUE!	Results in #VALUE! error value because there is no series of characters that begins with "i" and ends with "i" that has only 1 character in between.

Wildcard characters

- ? is used to find any single character. One ? is equal to one character.
- * is used to find any number of characters.
- ~ is used to find an actual ?, *, or ~ inside a text string.

=MID(A2,FIND(",",A2)+2,2).

	A	B
1	City, Province/State, Country	Province/State
2	Austin, TX, United States	TX
3	Edmonton, AB, Canada	AB
4	Halifax, NS, Canada	NS
5	Kansas City, MO, United States	MO
6	San Francisco, CA, United States	CA
7	Seattle, WA, United States	WA
8	Toronto, ON, Canada	ON
9	Vancouver, BC, Canada	BC

- Some **common logical operators**:
  - = Equal to
  - <> Not equal to
  - < Less than
  - > Greater than
  - <= Less than or equal to
  - >= Greater than or equal to