Week-3: Variables & their types

NM2207: Computational Media Literacy

Narayani Vedam, Ph.D.

Department of Communications and New Media

National University of Singapore | Faculty of Arts & Social Sciences

This week

Table of contents

I. Introduction to variables (click here)

II. Data structures (click here)

III. Exploring a data-set (click here)

I. Variables

What are variables?

- In programming, a **variable** is a name given to locations in the memory of the computer
- They allow the user to *store* and *manipulate* data
- R has six primitive types of variables,

> 1. character
> 2. double (real numbers)
> 3. integer
> 4. complex
> 5. logical
> 6. raw

Note: Think of them to be wallets that hold data instead of currency!

Variables (continued)

- Each variable has four attributes

> 1. an identifier (or name)
> 2. location
> 3. type
> 4. value

- The *identifier* and *value* are assigned by the user
  - Remember to always give names that are meaningful and related to the data
  - Avoid using space between characters/words in the name
- The *value* assigned by the user determines it *type*
- The *location* and *type* are automatically assigned, unless the user desires to override them

Variables: Classification

Variables can be broadly classified into two kinds,

1. **Numeric variables**
   a. Continuous
   b. Discrete
2. **Non-numeric variables/Categoric**
   a. Ordinal
   b. Nominal
   c. Binary

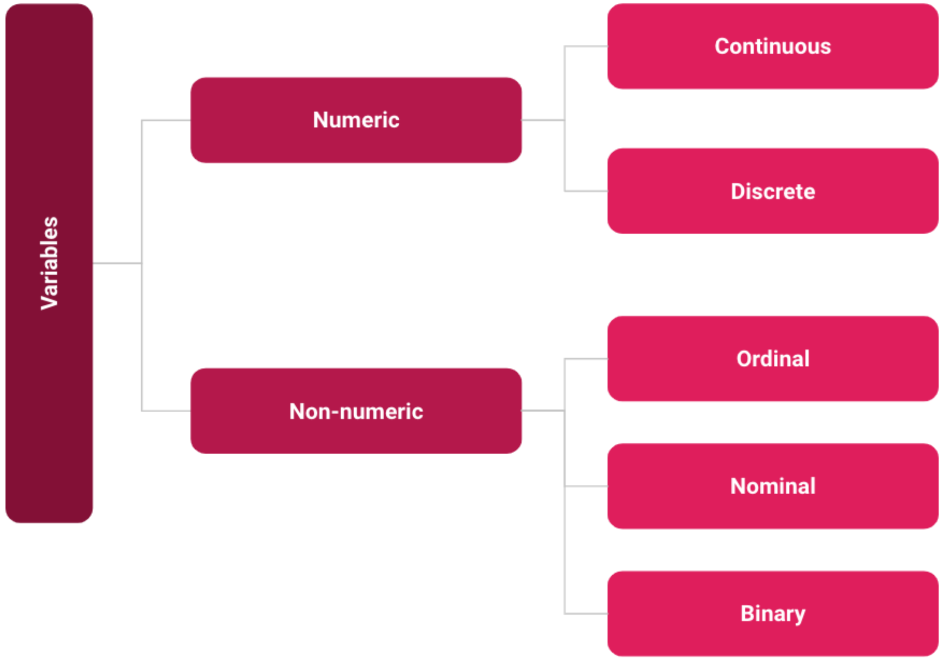


Figure: Classification of variables

Variables: Numeric

Numeric variables could be one of the following,

1. **integer**: are whole numbers, *without* decimal values

   Examples

   1000, 100, 200, 0, 1, 121353346970980,-1,-10

2. **double**: are numbers *with* decimal values

   Examples

   1.1, 0.89, 200.0, 0.001, 1000.5090809090068,-0.98,-200.0

3. **complex**: are *imaginary* numbers

   Examples

   1i, 2+10i, -200, 0.001, 1000.5090809090068-896.43124319i

Numeric variables: classification

> They can be **continuous**

- Of *type* double
- Can have infinite decimal values

Examples

Weight, Temperature, Height

> They can also be **discrete**

- Of *type* integer or double
- They are finite

Examples

Population, Number of occurrences

<!--

Non-numeric variables: Categoric

Non-numeric variables are also called categoric variables. Additionally, they can be,

1. **character**: single alphabet or numeral

   Examples

   'A', 'v', 'X', 'P', 'd', '1', '2'

2. **character**: string of alphabets or numerals

   Examples

   "hello", "world", "hi there", "20"

3. **logical**: either `TRUE` or `FALSE`

   Note: Character variables can be enclosed either within single or double quotes. However, the practice is to use single quotes for a character and double quotes for a string of characters.

Categoric variables: Ordinal

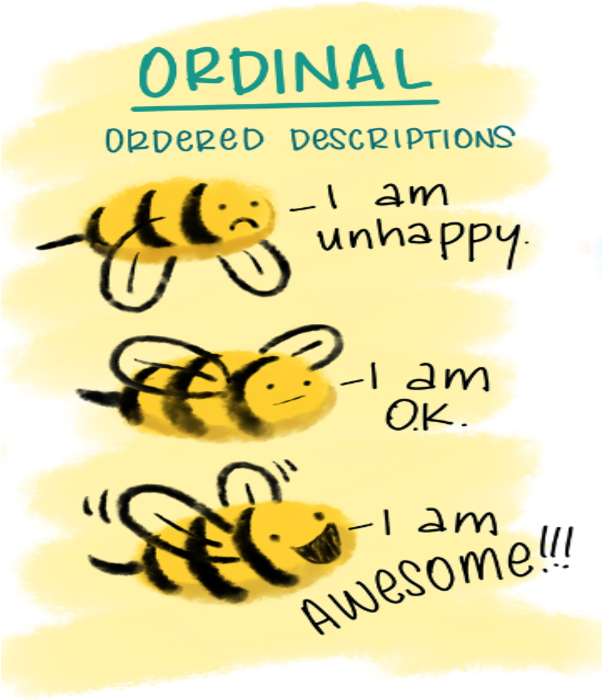
> Categoric variables can be ordinal

- They can be of type - **character**
- They have *natural ordering*

Examples

Survey responses ("Strongly agree", "Agree", "Neutral", "Disagree", "Strongly disagree")

Survey responses ("Low", "Medium", "High")



Source: https://allisonhorst.com/everything-else

Categoric variables: Nominal

> Categoric variables can be nominal

- They can be of either types - **character** or **logical**
- They *do not* have a natural ordering

<u>Examples</u>

Gender ("Male", "Female")

Color ("Red", "Blue", "Green")

Source: https://allisonhorst.com/everything-else

Categoric variables: Binary

> Categoric variables can also be binary

- They can be of either types - **character**, **logical** or **numeric**
- They *do not* have a natural ordering
- They take only two mutually exclusive values

<u>Examples</u>

"Yes", "No"

"TRUE", "FALSE"

1, 0

Source: https://allisonhorst.com/everything-else

Working with variables in R

a. character

```
x <- 'A'
```

b. character (string)

```
x <- "Apple"
```

c. logical

```
x <- FALSE
```

d. integer

```
x <- 5L
```

e. numeric (double)

```
x <- 5
```

f. complex

```
x <- 1i
```

Notice how numeric assignments in d and e differ

Working with variables in R

```r
x <- 'A'
typeof(x)
```

```
## [1] "character"
```

```r
x <- "Apple"
typeof(x)
```

```
## [1] "character"
```

```r
x <- FALSE
typeof(x)
```

```
## [1] "logical"
```

```r
x <- 5L
typeof(x)
```

```
## [1] "integer"
```

```r
x <- 5
typeof(x)
```

```
## [1] "double"
```

```r
x <- 1i
typeof(x)
```

```
## [1] "complex"
```

Need for data types

Example: Cat lovers

A survey asked respondents their name and number of cats. The instructions said to enter the number of cats as a numerical value.

```r
cat_lovers <- read_csv("cat-lovers.csv")
```

```
## # A tibble: 60 × 3
##    name           number_of_cats handedness
##    <chr>          <chr>          <chr>
##  1 Bernice Warren 0              left
##  2 Woodrow Stone  0              left
##  3 Willie Bass    1              left
##  4 Tyrone Estrada 3              left
##  5 Alex Daniels   3              left
##  6 Jane Bates     2              left
##  7 Latoya Simpson 1              left
##  8 Darin Woods    1              left
##  9 Agnes Cobb     0              left
## 10 Tabitha Grant  0              left
## # ℹ 50 more rows
```

Example: Cat lovers

```
mean(cat_lovers$number_of_cats)
```

```
## Warning in mean.default(cat_lovers$number_of_cats): argument is not numeric or
## logical: returning NA
```

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

Example: Cat lovers (Continued)

```
?mean
```

mean {base} R Documentation

Arithmetic Mean

Description

Generic function for the (trimmed) arithmetic mean.

Usage

```
mean(x, ...)

## Default S3 method:
mean(x, trim = 0, na.rm = FALSE, ...)
```

Arguments

`x`	An R object. Currently there are methods for numeric/logical vectors and date, date-time and time interval objects. Complex vectors are allowed for `trim = 0`, only.
`trim`	the fraction (0 to 0.5) of observations to be trimmed from each end of `x` before the mean is computed. Values of trim outside that range are taken as the nearest endpoint.
`na.rm`	a logical evaluating to `TRUE` or `FALSE` indicating whether `NA` values should be stripped before the computation proceeds.
`...`	further arguments passed to or from other methods.

Value

If `trim` is zero (the default), the arithmetic mean of the values in `x` is computed, as a numeric or complex vector of length one. If `x` is not logical (coerced to numeric), numeric (including integer) or complex, `NA_real_` is returned, with a warning.

If `trim` is non-zero, a symmetrically trimmed mean is computed with a fraction of `trim` observations deleted from each end before the mean is computed.

References

Example: Cat lovers - Coercion

```
mean(as.integer(cat_lovers$number_of_cats))
```

```
## Warning in mean(as.integer(cat_lovers$number_of_cats)): NAs introduced by
## coercion
```

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

Example: Cat lovers - Investigation

```
cat_lovers$number_of_cats
```

```
##  [1] "0"
##  [2] "0"
##  [3] "1"
##  [4] "3"
##  [5] "3"
##  [6] "2"
##  [7] "1"
##  [8] "1"
##  [9] "0"
## [10] "0"
## [11] "0"
## [12] "0"
## [13] "1"
## [14] "3"
## [15] "3"
## [16] "2"
## [17] "1"
## [18] "1"
## [19] "0"
## [20] "0"
## [21] "1"
```

Example: Cat lovers - Investigation (Continued)

```
as.integer(cat_lovers$number_of_cats)
```

```
## Warning: NAs introduced by coercion
```

```
##  [1]  0  0  1  3  3  2  1  1  0  0  0  0  1  3  3  2  1  1  0  0  1  1  0  0  4
## [26]  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  3  3  2  1 NA  0  0
## [51]  1  0  1 NA  1  1  1  0  0  2
```

II. Data Structures

What are data structures

- They are a collection of values, of one or more types
- Data structures in R include the below types,
  - (atomic) vectors
  - lists
  - matrix
  - factors
- Among them, lists and vectors are the most common and basic data structures in R
- They are pretty much the workhorses of R

Note: Imagine a wallet that can store currencies of one or more nationalities

Vectors & their types

- A vector is a collection of elements of the <u>same</u> type
- They could be of type, character, logical, integer or double
- Let us create an empty vector `x`

```
# An empty vector
x <- vector()
```

- By default, the *type* of an empty vector is <u>logical</u>

```
# Type of the empty vector
typeof(x)
```

```
## [1] "logical"
```

Vectors & their types

One can be more explicit and create vectors of the needed *type*

- Different ways to create vectors of *type*, **logical**

```
# Different ways to create vectors: method .
x<-vector("logical",length=5)

# Different ways to create vectors: method .
x<-logical(5)

# Different ways to create vectors: method .
x<-c(TRUE,FALSE,TRUE,FALSE,TRUE)

typeof(x)
```

```
## [1] "logical"
```

- Different ways to create vectors of *type*, **character**

```
# Different ways to create vectors: method .
x<-vector("character",length=5)

# Different ways to create vectors: method .
x<-character(5)

# Different ways to create vectors: method .
x<-c('A','b','r','q')

typeof(x)
```

```
## [1] "character"
```

Vectors & their types

One can be more explicit and create vectors of the needed *type*

- Different ways to create vectors of *type*, **integer**

```
# Different ways to create vectors: method .
x<-vector("integer",length=5)

# Different ways to create vectors: method .
x<-integer(5)

# Different ways to create vectors: method .
x<-c(1,2,3,4,5)

# Different ways to create vectors: method .
x<-seq(from=1,to=5,by=0.1)

# Different ways to create vectors: method .
x<-1:5

typeof(x)
```

```
## [1] "integer"
```

- Different ways to create vectors of *type*, **double**

```
# Different ways to create vectors: method .
x<-vector("double",length=5)

# Different ways to create vectors: method .
x<-double(5)

# Different ways to create vectors: method .
x<-c(1.787,0.63573,2.3890)

typeof(x)
```

```
## [1] "double"
```

Vectors: mixed types

If elements of the vector differ in their types,

- R will convert the vector to accommodate all the element types
- This automatic conversion by R from one type to another is called **coercion**
- When R converts the type based on its contents it is called, **implicit** coercion
- Forcing conversion manually is called, **explicit** coercion

Implicit coercion

Example 1: Automatic conversion of *types* by R from double to character

```r
# Create a vector
x <- c(1.8)
# Check the type of x
typeof(x)
```

```
## [1] "double"
```

```r
# Add a character to the vector
x <- c(x,'a')
# Check the type of x
typeof(x)
```

```
## [1] "character"
```

Note: how `c()` is used to insert elements to a vector

Implicit coercion

Example 2: Automatic conversion of *types* by R from logical to double

```r
# Create a vector
x <- c(TRUE)
# Check the type of x
typeof(x)
```

```
## [1] "logical"
```

```r
# Add a number to the vector
x <- c(x,2)
# Check the type of x
typeof(x)
```

```
## [1] "double"
```

Note: how `c()` is used to insert elements to a vector

Implicit coercion

Example 3: Automatic conversion of *types* by R from character to character

```r
# Create a vector
x <- c('a')
# Check the type of x
typeof(x)
```

```
## [1] "character"
```

```r
# Add a logical value to the vector
x <- c(x,TRUE)
# Check the type of x
typeof(x)
```

```
## [1] "character"
```

Note: how `c()` is used to insert elements to a vector

Implicit coercion

Example 4: Automatic conversion of *types* by R from integer to double

```
# Create a vector
x <- c(1L)
# Check the type of x
typeof(x)
```

```
## [1] "integer"
```

```
# Add a number to the vector
x <- c(x,2)
# Check the type of x
typeof(x)
```

```
## [1] "double"
```

Note: how `c()` is used to insert elements to a vector

Explicit coercion

Example 1: Explicit coercion from integer to character

```
# Create a vector
x <- c(1L)
# Check the type of x
typeof(x)
```

```
## [1] "integer"
```

```
# Convert the vector to type character
x <- as.character(x)
# Check the type of x
typeof(x)
```

```
## [1] "character"
```

Note: how `as.character()` is used for conversion

Explicit coercion

Example 2: Explicit coercion from character to double

```r
# Create a vector
x <- c('A')
# Check the type of x
typeof(x)
```

```
## [1] "character"
```

```r
# Convert the vector to type double
x <- as.numeric(x)
# Check the type of x
typeof(x)
```

```
## [1] "double"
```

Note: how `as.numeric()` is used for conversion

Accessing elements of a vector

Consider the following vector,

```r
# Create a vector
x <- c(1,10,9,8,1,3,5)
```

a. Accessing the elements by index: one element

```r
# One index
x[3]
```

```
## [1] 9
```

b. Accessing the elements by index: many elements

```r
# Consecutive indices
x[2:4]
```

```
## [1] 10  9  8
```

```r
# Non-consecutive indices
x[c(1,3,5)]
```

```
## [1] 1 9 1
```

Accessing elements of a vector

Consider the following vector,

```
# Create a vector
x <- c(1,10,9,8,1,3,5)
```

c. Accessing elements using a logical vector

```
# Use of logical vector
x[c(TRUE,FALSE,FALSE,TRUE,FALSE,FALSE,TRUE)]
```

```
## [1] 1 8 5
```

d. Accessing elements using conditional operator

```
# Use of conditional operator
x[x<10]
```

```
## [1] 1 9 8 1 3 5
```

Examining vectors

a. Number of elements in a vector

```
# length of the vector
length(x)
```

```
## [1] 7
```

b. *type* of the object

```
# class of the vector
typeof(x)
```

```
## [1] "double"
```

c. Display the structure of the object

```
# structure of the vector
str(x)
```

```
##  num [1:7] 1 10 9 8 1 3 5
```

- It is of class num, **numeric**
- It has 7 elements, [1:7]

Lists

- A list is a special vector whose elements can be of varied types

a. Creating a list

```
# Initialise a list
x<-list(1,"a",0.289,TRUE)
```

b. Conversion to a list: vector to list

```
# Initialise a vector
x<-c(1,2,3,4,10)
# Convert to a list
x<-as.list(x)
```

Lists

- Elements of a list can be named
- A list is printed on the console along with the name
- Each element of the list starts on a new line

Example

```
# Initialise a named list
my_pie = list(type="key lime", diameter=7, is
my_pie
```

```
## $type
## [1] "key lime"
##
## $diameter
## [1] 7
##
## $is.vegetarian
## [1] TRUE
```

Lists

a. Print names of the list

```r
# Elicit names of the list
names(my_pie)
```

```
## [1] "type"            "diameter"        "is.vege
```

b. Access elements of the list by their name

```r
# Retrieve the element named type
my_pie$type
```

```
## [1] "key lime"
```

c. Access elements of the list using `[]`: returns a list

```r
# Retrieve a truncated list
my_pie["type"]
```

```
## $type
## [1] "key lime"
```

d. Access elements of the list using `[[]]`: returns the element

```r
# Retrieve the element named type
my_pie[["type"]]
```

```
## [1] "key lime"
```

Data-set overview

Vector

Data Frame

List

	country	year	strike.volume	unemployment
1	Australia	1951	296	1.3
2	Australia	1952	397	2.2
3	Australia	1953	360	2.5
4	Australia	1954	3	1.7
5	Australia	1955	326	1.4
6	Australia	1956	352	1.8
7	Australia	1957	195	2.3
8	Australia	1958	133	2.7
9	Australia	1959	109	2.6
10	Australia	1960	208	2.5

III. Exploring a data-set

Let us explore a data-set

<u>Example data-set</u>

- Let us consider a data-set, **Lending club**
- The data-set is from a platform that enables loans between individuals
- Not all requests are treated equally;
  - Approval of loan is dependent on the borrower's ability to repay
- The data-set has a compilation of all sanctioned loans
  - There are no loan applications

Lending club: packages

- Install the package containing the data-set

```r
# Install package
install.packages("openintro")
```

- Load the installed package

```r
# Load package
library(openintro)
```

```
## Loading required package: airports
```

```
## Loading required package: cherryblossom
```

```
## Loading required package: usdata
```

Lending club: packages

- Load `tidyverse` package to manipulate the data-set

```r
# Load package
library(tidyverse)
```

Lending club: overview

An overview of the contents of the data-set

```r
# Catch a glimpse of the data-set
glimpse(loans_full_schema)
```

```
## Rows: 10,000
## Columns: 55
## $ emp_title                  <chr> "global config engineer ", "warehouse…
## $ emp_length                 <dbl> 3, 10, 3, 1, 10, NA, 10, 10, 10, 3, 1…
## $ state                      <fct> NJ, HI, WI, PA, CA, KY, MI, AZ, NV, I…
## $ homeownership              <fct> MORTGAGE, RENT, RENT, RENT, RENT, OWN…
## $ annual_income              <dbl> 90000, 40000, 40000, 30000, 35000, 34…
## $ verified_income            <fct> Verified, Not Verified, Source Verifi…
## $ debt_to_income             <dbl> 18.01, 5.04, 21.15, 10.16, 57.96, 6.4…
## $ annual_income_joint        <dbl> NA, NA, NA, NA, 57000, NA, 155000, NA…
## $ verification_income_joint  <fct> , , , , Verified, , Not Verified, , ,…
## $ debt_to_income_joint       <dbl> NA, NA, NA, NA, 37.66, NA, 13.12, NA,…
## $ delinq_2y                  <int> 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0…
## $ months_since_last_delinq   <int> 38, NA, 28, NA, NA, 3, NA, 19, 18, NA…
## $ earliest_credit_line       <dbl> 2001, 1996, 2006, 2007, 2008, 1990, 2…
## $ inquiries_last_12m         <int> 6, 1, 4, 0, 7, 6, 1, 1, 3, 0, 4, 4, 8…
## $ total_credit_lines         <int> 28, 30, 31, 4, 22, 32, 12, 30, 35, 9,…
```

Lending club: Numeric variables

Let us select some variables of *type* Numeric,

a. pass the data-set through a pipe operator (`%>%`)

b. `select()` variables of interest

- `paid_total`
- `term`
- `interest_rate`
- `annual_income`
- `paid_late_fees`
- `debt_to_income`

```
loans <- loans_full_schema %>% # <-- pipe ope
  select(paid_total, term, interest_rate,
         annual_income,paid_late_fees,debt_to_
glimpse(loans)
```

```
## Rows: 10,000
## Columns: 6
## $ paid_total     <dbl> 1999.330, 499.120, 28
## $ term           <dbl> 60, 36, 36, 36, 36,
## $ interest_rate  <dbl> 14.07, 12.61, 17.09,
## $ annual_income  <dbl> 90000, 40000, 40000,
## $ paid_late_fees <dbl> 0, 0, 0, 0, 0, 0, 0,
## $ debt_to_income <dbl> 18.01, 5.04, 21.15,
```

Lending club: Numeric variables

Variable	Description
paid_total	Repaid loan amount, in US dollars
term	The length of the loan, which is always set as a whole number of months
interest_rate	Interest rate on the loan, in an annual percentage
annual_income	Borrower's annual income, including any second income, in US dollars
paid_late_fees	Penalty paid for defaulting on payment of interest, in US dollars
debt_to_income	Debt-to-income ratio

Lending club: classification of Numeric variables

Variable	Type
paid_total	Continuous
term	Discrete
interest_rate	Continuous
annual_income	Continuous
paid_late_fees	Continuous
debt_to_income	Continuous

Lending club: overview revisited

```r
# overview of the data-set
glimpse(loans_full_schema)
```

```
## Rows: 10,000
## Columns: 55
## $ emp_title                  <chr> "global config engineer ", "warehouse…
## $ emp_length                 <dbl> 3, 10, 3, 1, 10, NA, 10, 10, 10, 3, 1…
## $ state                      <fct> NJ, HI, WI, PA, CA, KY, MI, AZ, NV, I…
## $ homeownership              <fct> MORTGAGE, RENT, RENT, RENT, RENT, OWN…
## $ annual_income              <dbl> 90000, 40000, 40000, 30000, 35000, 34…
## $ verified_income            <fct> Verified, Not Verified, Source Verifi…
## $ debt_to_income             <dbl> 18.01, 5.04, 21.15, 10.16, 57.96, 6.4…
## $ annual_income_joint        <dbl> NA, NA, NA, NA, 57000, NA, 155000, NA…
## $ verification_income_joint  <fct> , , , , Verified, , Not Verified, , ,…
## $ debt_to_income_joint       <dbl> NA, NA, NA, NA, 37.66, NA, 13.12, NA,…
## $ delinq_2y                  <int> 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0…
## $ months_since_last_delinq   <int> 38, NA, 28, NA, NA, 3, NA, 19, 18, NA…
## $ earliest_credit_line       <dbl> 2001, 1996, 2006, 2007, 2008, 1990, 2…
## $ inquiries_last_12m         <int> 6, 1, 4, 0, 7, 6, 1, 1, 3, 0, 4, 4, 8…
## $ total_credit_lines         <int> 28, 30, 31, 4, 22, 32, 12, 30, 35, 9,…
## $ open_credit_lines          <int> 10, 14, 10, 4, 16, 12, 10, 15, 21, 6,…
## $ total_credit_limit         <int> 70795, 28800, 24193, 25400, 69839, 42…
## $ total_credit_utilized      <int> 38767, 4321, 16000, 4997, 52722, 3898…
```

Lending club: Categoric variables

Let us select some variables of *type* Categoric,

a. pass the data-set through a pipe operator (`%>%`)

b. `select()` variables of interest

- `grade`
- `state`
- `homeownership`
- `disbursement_method`

```
loans <- loans_full_schema %>% # <-- pipe ope
  select(grade,state,homeownership,disburseme
glimpse(loans)
```

```
## Rows: 10,000
## Columns: 4
## $ grade               <fct> C, C, D, A, C, 
## $ state               <fct> NJ, HI, WI, PA,
## $ homeownership       <fct> MORTGAGE, RENT,
## $ disbursement_method <fct> Cash, Cash, Cas
```

Lending club: Categoric variables

Variable	Description
grade	Loan grade, which takes values A through G and represents the quality of the loan and its likelihood of being repaid
state	US state where the borrower resides
homeownership	Indicates whether the person owns, owns but has a mortgage, or rents
disbursement	Mode of loan disbursement

Lending club: classification of Categoric variables

Variable	Type
grade	Ordinal
state	Nominal
homeownership	Nominal
disbursement	Nominal

Thanks!

Slides created via the R packages:

xaringan
gadenbuie/xaringanthemer.

NUS National University of Singapore | Faculty of Arts & Social Sciences