Analytics of Business Intelligence

Chapter # 2 - Data Cleaning

Dr. Wajahat Gilani

Rutgers Business School

September 16, 2020

Data Cleaning

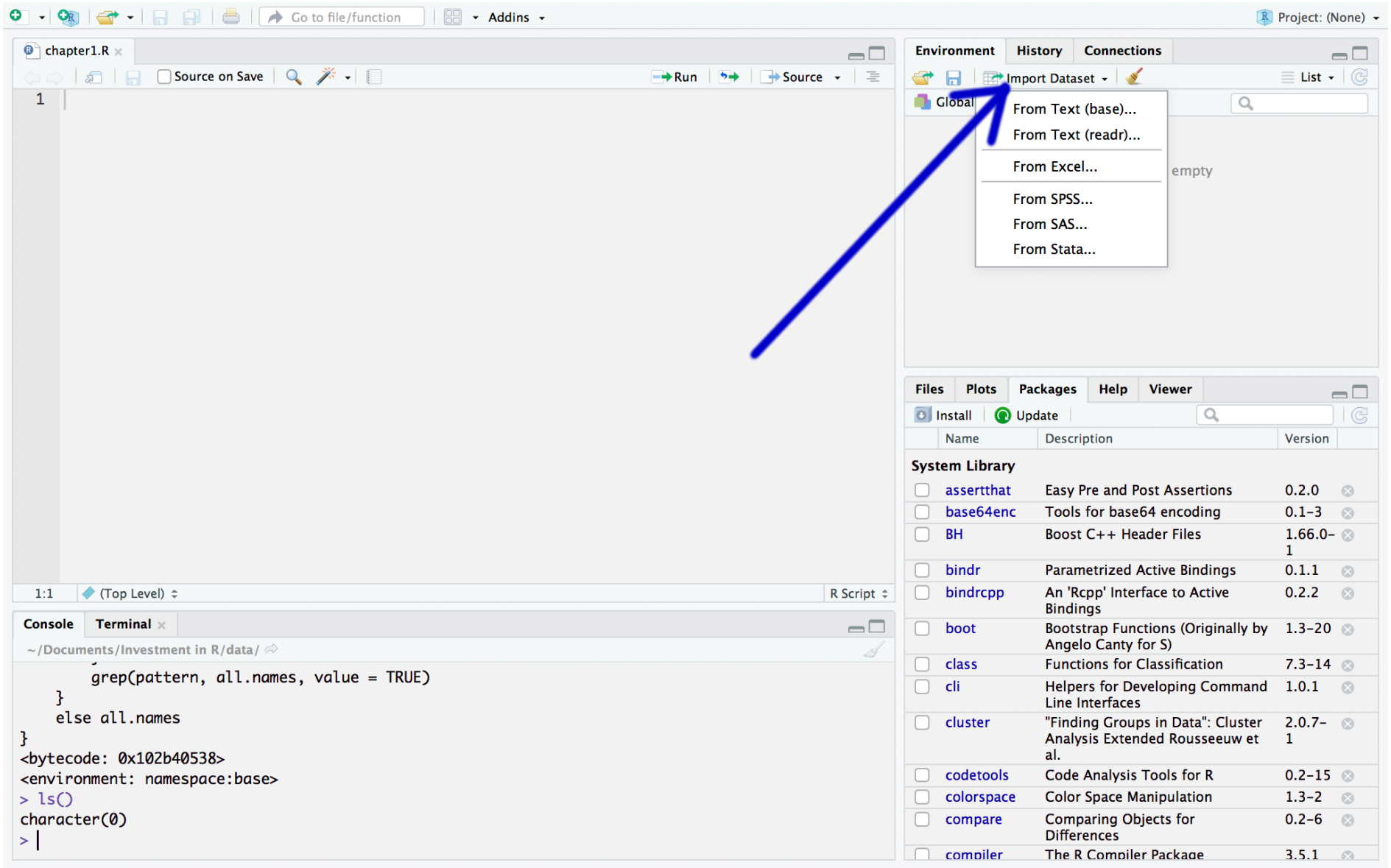
We will be using the text file Ch2_raw_bikeshare_data.csv and focus on:

- Summarizing your data for inspection
- Finding and fixing flawed data
- Converting inputs to data types suitable for analysis
- Adapting string variables to a standard

Anthony Goldbloom, CEO of Kaggle, said: *Eighty percent of data science is cleaning data and the other twenty percent is complaining about cleaning data (personal communications, February 14, 2016).*

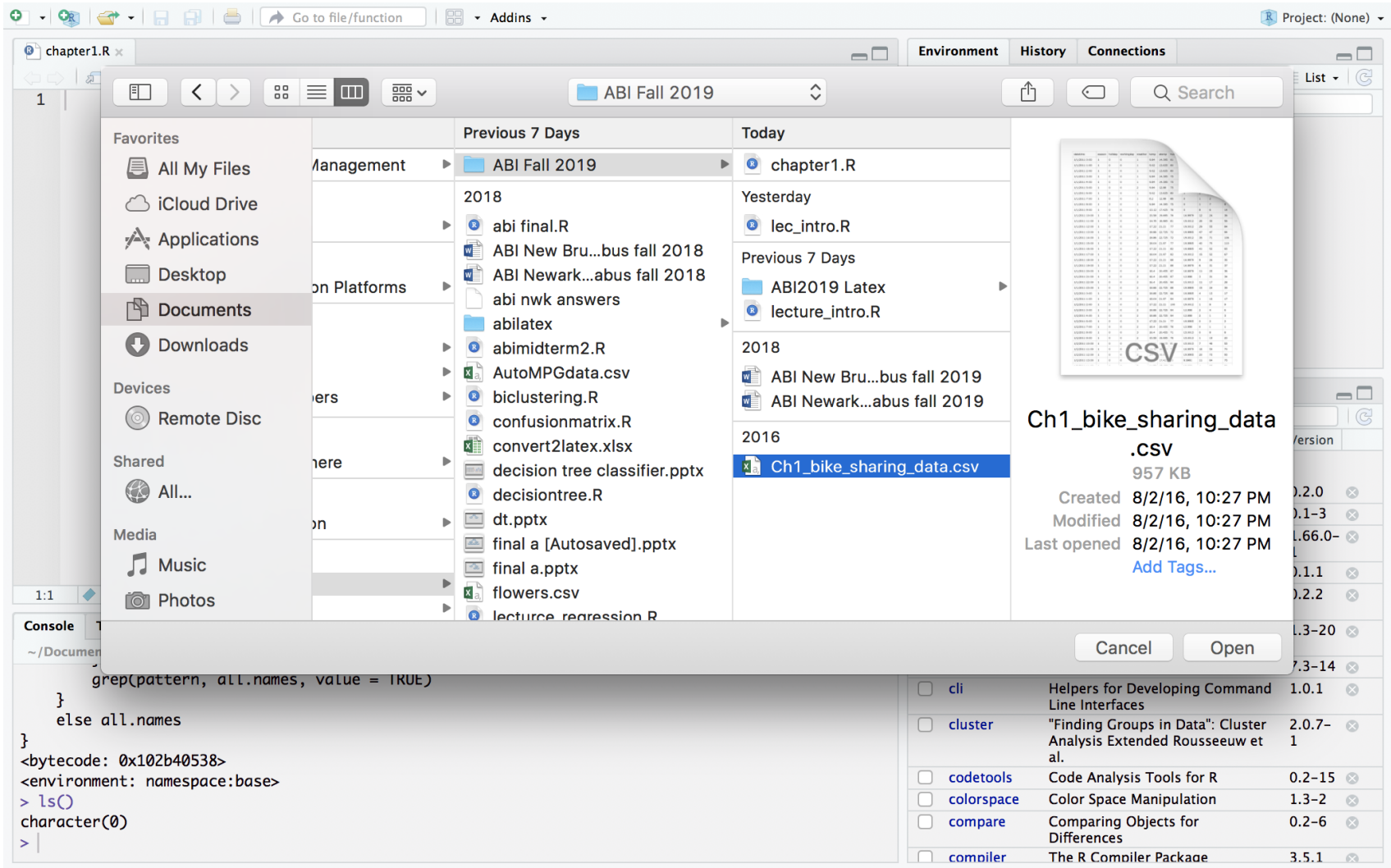
Importing CSV and other file formats

Importing data using "Import Dataset" from the top right window



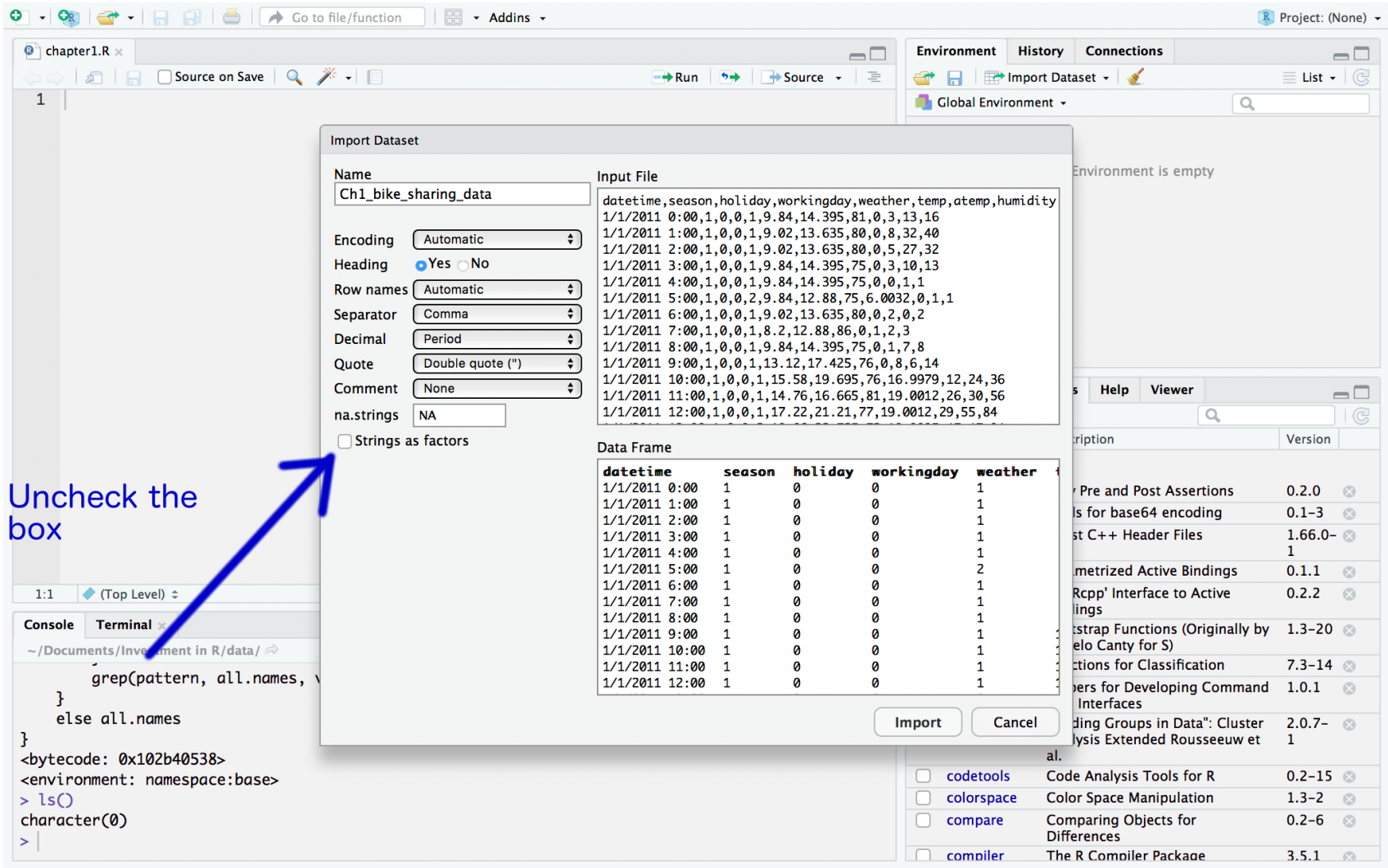
Importing CSV and other file formats

Go to the folder where you downloaded Ch2_raw_bikeshare_data.csv and click on "Open".



Importing CSV and other file formats

Uncheck the box that says: "Strings as factors", then click "Import"



Create a new data.frame from an existing one

```
1 bikeraw = copy(Ch2_raw_bikeshare_data)
```

The tables bikeraw and Ch2_raw_bikeshare_data have the same exact data and structure but are completely different and independent from each other.

```
1 str(bikeraw)
```

```
> str(bikeraw)
'data.frame':   17379 obs. of  13 variables:
 $ datetime  : chr  "1/1/2011 0:00" "1/1/2011 1:00" "1/1/2011 2:00" "1/1/2011 3:00" ...
 $ season    : int  1 1 1 1 1 1 1 1 1 1 ...
 $ holiday   : int  0 0 0 0 0 0 0 0 0 0 ...
 $ workingday: int  0 0 0 0 0 0 0 0 0 0 ...
 $ weather   : int  1 1 1 1 1 2 1 1 1 1 ...
 $ temp      : num  9.84 9.02 9.02 9.84 9.84 ...
 $ atemp     : num  14.4 13.6 13.6 14.4 14.4 ...
 $ humidity  : chr  "81" "80" "80" "75" ...
 $ windspeed : num  0 0 0 0 0 ...
 $ casual    : int  3 8 5 3 0 0 2 1 1 8 ...
 $ registered: int  13 32 27 10 1 1 0 2 7 6 ...
 $ count     : int  16 40 32 13 1 1 2 3 8 14 ...
 $ sources   : chr  "ad campaign" "www.yahoo.com" "www.google.fi" "AD campaign" ...
```

Missing Values

```
1 library(data.table)
2 setDT(bikeraw)
3 is.na(bikeraw)
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual
[1,]	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
[2,]	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
[3,]	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
[4,]	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
[5,]	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
[6,]	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
[7,]	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
[8,]	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
[9,]	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE

We can get a unique count of all the True and False using the table() function.

```
1 table(is.na(bikeraw))
```

FALSE TRUE
225373 554

Function table() Example

```
1 c1=c('a','b','c')
2 c2=c('b','a','b')
3 c3=c('d','c','a')
4 dt=data.table(c1,c2,c3)
```

	c1	c2	c3
1	a	b	d
2	b	a	c
3	c	b	a

The table() function can count how many unique values there are in a column.

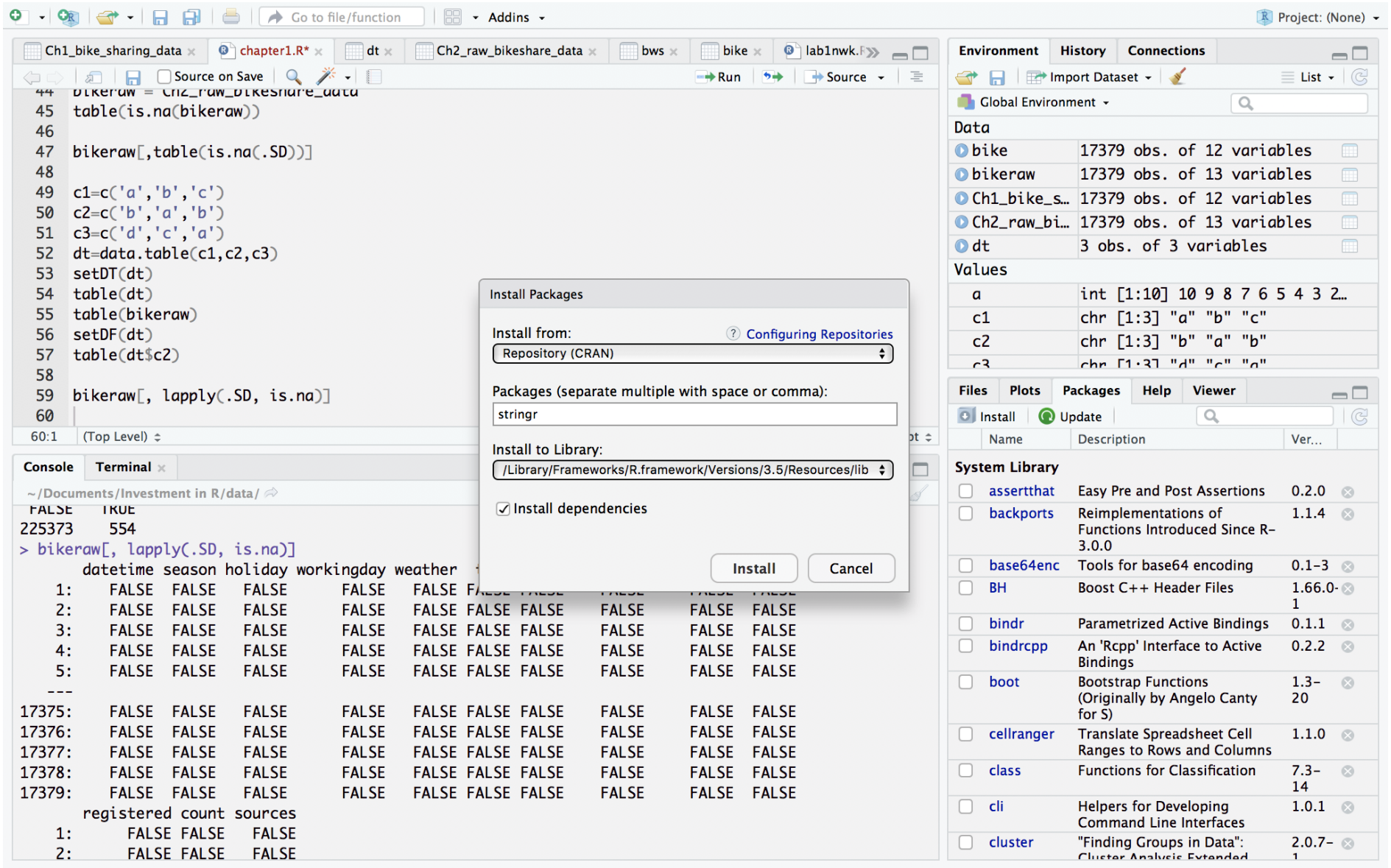
```
1 table(dt$c2)
```

a b
1 2

But what if we have a data.table with many columns, how do we focus on which columns have NA's?

Stringr Package

Download a new package called stringr.



Stringr Package (str_detect)

The str_detect() function is a function from the stringr library. It allows you to search for any particular string within the data.table. Using the dt table example:

	c1	c2	c3
1	a	b	d
2	b	a	c
3	c	b	a

```
1 library(stringr)
2 str_detect(dt,'d')
```

FALSE FALSE TRUE

Now lets run it on the data.table bikeraw:

```
1 str_detect(bikeraw,'NA')
```

What do you see?

Stringr Package (str_detect)

The results show:

FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE

```
> str(bikeraw)
'data.frame':   17379 obs. of  13 variables:
 $ datetime  : chr  "1/1/2011 0:00" "1/1/2011 1:00" "1/1/2011 2:00" "1/1/2011 3:00" ...
 $ season    : int  1 1 1 1 1 1 1 1 1 1 ...
 $ holiday   : int  0 0 0 0 0 0 0 0 0 0 ...
 $ workingday: int  0 0 0 0 0 0 0 0 0 0 ...
 $ weather   : int  1 1 1 1 1 2 1 1 1 1 ...
 $ temp      : num  9.84 9.02 9.02 9.84 9.84 ...
 $ atemp     : num  14.4 13.6 13.6 14.4 14.4 ...
 $ humidity  : chr  "81" "80" "80" "75" ...
 $ windspeed : num  0 0 0 0 0 ...
 $ casual    : int  3 8 5 3 0 0 2 1 1 8 ...
 $ registered: int  13 32 27 10 1 1 0 2 7 6 ...
 $ count     : int  16 40 32 13 1 1 2 3 8 14 ...
 $ sources   : chr  "ad campaign" "www.yahoo.com" "www.google.fi" "AD campaign" ...
```

So all of our original 554 NA that we found are all in the sources column.

```
1 bikeraw[is.na(sources),NROW(sources)]
```

df[Row Section,Column Section]

Erroneous Values - Humidity

```
> str(bikeraw)
'data.frame':   17379 obs. of  13 variables:
 $ datetime  : chr  "1/1/2011 0:00" "1/1/2011 1:00" "1/1/2011 2:00" "1/1/2011 3:00" ...
 $ season    : int  1 1 1 1 1 1 1 1 1 1 ...
 $ holiday   : int  0 0 0 0 0 0 0 0 0 0 ...
 $ workingday: int  0 0 0 0 0 0 0 0 0 0 ...
 $ weather   : int  1 1 1 1 1 2 1 1 1 1 ...
 $ temp      : num  9.84 9.02 9.02 9.84 9.84 ...
 $ atemp     : num  14.4 13.6 13.6 14.4 14.4 ...
 $ humidity  : chr  "81" "80" "80" "75" ...
 $ windspeed : num  0 0 0 0 0 ...
 $ casual    : int  3 8 5 3 0 0 2 1 1 8 ...
 $ registered: int  13 32 27 10 1 1 0 2 7 6 ...
 $ count     : int  16 40 32 13 1 1 2 3 8 14 ...
 $ sources   : chr  "ad campaign" "www.yahoo.com" "www.google.fi" "AD campaign" ...
```

We see that the column humidity has numbers as values, but the column is of type "chr" which is strings (alphanumeric). We want to investigate why this is the case. We will use function grep() that allows us to use "regular expressions". Regular expressions are a way to find certain patterns in strings.

```
1 bikeraw[grep('[a-z A-Z]',humidity)]
```

Erroneous Values - Humidity

This is the row that will be returned:

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count	sources
1	8/18/2012 21:00	3	0	0	1	27.06	31.06	x61	0	90	248	338	www.bing.com

As we can see that one row has caused the whole column to become a "chr" data type instead of a number. We will manually correct it.

```
1 bikeraw[grep('[a-z A-Z]',humidity),humidity:='61']
```

Notice we put it in quotes. It's because the data type of the column is still "chr". We have to convert the datatype for the column.

Data type	Explanation	Example
Numeric	A number having a decimal value	9.84
Integer	A number without decimals	3
Character	A string variable	"www.google.com"
Factor	A categorical variable that has a character and integer representation	"ad campaign", "blog": 1,2
Date	A date or time in various formats	2016-02-16 18:56:57 EST

Converting Data Type- Humidity

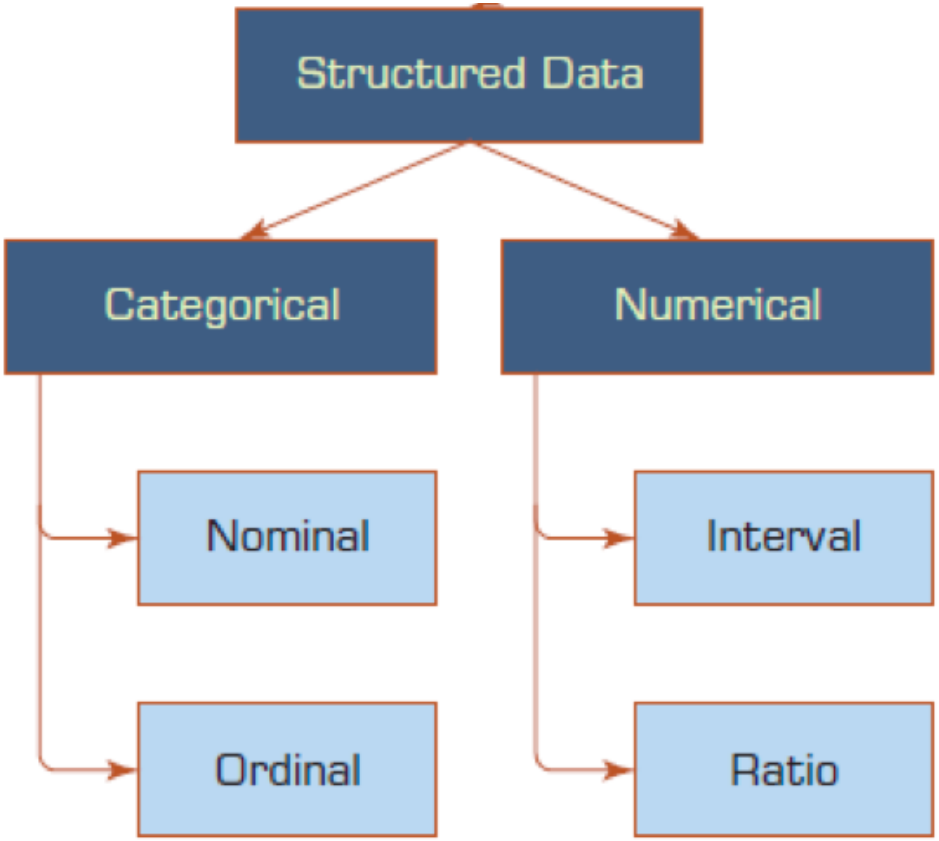
R has a series of functions called as.{*type*}() that allows you to change data from one type to another.

```
1 bikeraw[,humidity:=as.numeric(humidity)]
```

```
Classes 'data.table' and 'data.frame':  17379 obs. of  12 variables:
 $ datetime  : chr  "1/1/2011 0:00" "1/1/2011 1:00" "1/1/2011 2:00" "1/1/2011 3:00" ...
 $ season    : int  1 1 1 1 1 1 1 1 1 1 ...
 $ holiday   : int  0 0 0 0 0 0 0 0 0 0 ...
 $ workingday: int  0 0 0 0 0 0 0 0 0 0 ...
 $ weather   : int  1 1 1 1 1 2 1 1 1 1 ...
 $ temp      : num  9.84 9.02 9.02 9.84 9.84 ...
 $ atemp     : num  14.4 13.6 13.6 14.4 14.4 ...
 $ humidity  : int  81 80 80 75 75 75 80 86 75 76 ...
 $ windspeed : num  0 0 0 0 0 ...
 $ casual    : int  3 8 5 3 0 0 2 1 1 8 ...
 $ registered: int  13 32 27 10 1 1 0 2 7 6 ...
 $ count     : int  16 40 32 13 1 1 2 3 8 14 ...
 - attr(*, ".internal.selfref")=<externalptr>
```

Factors

Theoretically data tends to be of 2 main types, numerical and categorical. Categorical can be subdivided into nominal and ordinal data, ordinal meaing that it has some sort of order. If the categorical data can only have a certain number of values AND ONLY those values, then those values in a categorical column will be considered factors, and we should make sure R understands that.



Factors Example

```
1 data = c(1,2,2,3,1,2,3,3,1,2,3,3,1)
2 fdata = factor(data)
3 fdata
```

[1] 1 2 2 3 1 2 3 3 1 2 3 3 1
Levels: 1 2 3

```
1 rdata = factor(data,labels=c("I","II","III"))
2 rdata
```

[1] I II II III I II III III I II III III I
Levels: I II III

```
1 levels(fdata) = c('I','II','III')
2 fdata
```

[1] I II II III I II III III I II III III I
Levels: I II III

Factors - Holiday/Workingday

```
1 unique(bikeraw$holiday)
2 bikeraw[,holiday:=factor(holiday, levels = c(0,1),
    labels = c('no','yes'))]
3 unique(bikeraw$holiday)
```

[1] no yes
Levels: no yes

```
1 unique(bikeraw$workingday)
2 bikeraw[,workingday:=factor(workingday, levels = c
    (0,1), labels = c('no','yes'))]
3 unique(bikeraw$workingday)
```

[1] no yes
Levels: no yes

Ordered Factors - Season/Weather

```
1 unique(bikeraw$season)
2 bikeraw[,season:=factor(season, levels = c(1,2,3,4),
    labels = c('spring','summer','fall','winter'),
    ordered = T)]
3 unique(bikeraw$season)
```

[1] spring summer fall winter
Levels: spring < summer < fall < winter

```
1 bikeraw[,weather:=factor(weather, levels = c(1,2,3,4),
     labels = c('clr_part_cloud','mist_cloudy','lt_rain
    _snow','hvy_rain_snow'),ordered = T)]
2 unique(bikeraw$weather)
```

[1] clr_part_cloud mist_cloudy lt_rain_snow hvy_rain_snow
Levels: clr_part_cloud < mist_cloudy < lt_rain_snow < hvy_rain_snow

Date and Time Conversions

```
1 bikeraw[,.(datetime)]
```

```
                datetime
    1:    1/1/2011 0:00
    2:    1/1/2011 1:00
    3:    1/1/2011 2:00
    4:    1/1/2011 3:00
    5:    1/1/2011 4:00
   ---
17375: 12/31/2012 19:00
17376: 12/31/2012 20:00
17377: 12/31/2012 21:00
17378: 12/31/2012 22:00
17379: 12/31/2012 23:00
```

We need to convert the "characters" to actual "dates"

```
Classes 'data.table' and 'data.frame':  17379 obs. of  12 variables:
 $ datetime  : chr  "1/1/2011 0:00" "1/1/2011 1:00" "1/1/2011 2:00" "1/1/2011 3:00" ...
 $ season    : int  1 1 1 1 1 1 1 1 1 1 ...
 $ holiday   : int  0 0 0 0 0 0 0 0 0 0 ...
 $ workingday: int  0 0 0 0 0 0 0 0 0 0 ...
 $ weather   : int  1 1 1 1 1 2 1 1 1 1 ...
 $ temp      : num  9.84 9.02 9.02 9.84 9.84 ...
 $ atemp     : num  14.4 13.6 13.6 14.4 14.4 ...
 $ humidity  : int  81 80 80 75 75 75 80 86 75 76 ...
 $ windspeed : num  0 0 0 0 0 ...
```

Date and Time Conversions - Without Time

The as.Date() function creates dates

```
1 dt=c(as.Date('2018-01-01'),as.Date('2018-03-03'),as.
      Date('2018-05-01'),as.Date('2018-05-31'))
2 price=c(4,8,10,12)
3 sampdt = data.table(dt,price)
```

```
Classes 'data.table' and 'data.frame':  4 obs. of  2 variables:
 $ dt   : Date, format: "2018-01-01" "2018-03-03" "2018-05-01" "2018-05-31"
 $ price: num  4 8 10 12
 - attr(*, ".internal.selfref")=<externalptr>
```

```
1 bikeraw[,datetime:=as.Date(datetime, '%m/%d/%Y %H:%M')
      ]
2 str(bikeraw)
```

```
Classes 'data.table' and 'data.frame':  17379 obs. of  13 variables:
 $ datetime  : Date, format: "2011-01-01" "2011-01-01" "2011-01-01" "2011-01-01" ...
 $ season    : Ord.factor w/ 4 levels "spring"<"summer"<..: 1 1 1 1 1 1 1 1 1 1 ...
 $ holiday   : Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1 1 1 1 ...
 $ workingday: Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1 1 1 1 ...
 $ weather   : Ord.factor w/ 4 levels "clr_part_cloud"<..: 1 1 1 1 1 2 1 1 1 1 ...
 $ temp      : num  9.84 9.02 9.02 9.84 9.84 ...
 $ atemp     : num  14.4 13.6 13.6 14.4 14.4 ...
```

Date and Time Conversions - With Time

The strptime() function creates dates with times, not recommended because of large data storage.

```
1 bikeraw[,datetime:=strptime(datetime, '%m/%d/%Y %H:%M')]
2 str(bikeraw)
```

```
Classes 'data.table' and 'data.frame':  17379 obs. of  13 variables:
 $ datetime  : POSIXct, format: "2011-01-01 00:00:00" "2011-01-01 01:00:00" "2011-01-01 02:00:00" "2011-01-01 03:00:00" ...
 $ season    : Ord.factor w/ 4 levels "spring"<"summer"<..: 1 1 1 1 1 1 1 1 1 1 ...
 $ holiday   : Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1 1 1 1 ...
 $ workingday: Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1 1 1 1 ...
 $ weather   : Ord.factor w/ 4 levels "clr_part_cloud"<..: 1 1 1 1 1 2 1 1 1 1 ...
 $ temp      : num  9.84 9.02 9.02 9.84 9.84 ...
 $ atemp     : num  14.4 13.6 13.6 14.4 14.4 ...
 $ humidity  : num  81 80 80 75 75 75 80 86 75 76 ...
 $ windspeed : num  0 0 0 0 0 ...
 $ casual    : int  3 8 5 3 0 0 2 1 1 8 ...
 $ registered: int  13 32 27 10 1 1 0 2 7 6 ...
 $ count     : int  16 40 32 13 1 1 2 3 8 14 ...
 $ sources   : chr  "ad campaign" "www.yahoo.com" "www.google.fi" "AD campaign" ...
 - attr(*, ".internal.selfref")=<externalptr>
```

Adapting String Variables To Standards

If you check the sources column, it is still a chr datatype, infact it is the only chr column datatype left in our table. The problem is that R **CANNOT group character items to summarize them in analysis**. This implies that maybe sources should be categorical data too, but before we convert we have to ask ourselves if the variation in data is helpful. To be specific:

- How many unique kids of advertising sources are in sources?
- How many categories would you like to have in your analysis dataset?

```
1 unique(bikeraw$sources)
```

```
 [1] "ad campaign"      "www.yahoo.com"    "www.google.fi"    "AD campaign"      "Twitter"          "www.bing.com"
 [7] "www.google.co.uk" "facebook page"    "Ad Campaign"      "Twitter    "      NA                 "www.google.com"
[13] "direct"           "blog"
```

Observations:

- 2 Twitter values?
- Multiple ad campaigns
- The NA

Adapting String Variables To Standards

- 2 Twitter values?
- Multiple ad campaigns
- The NA

```
1 bikeraw[,sources:=tolower(sources)]
2 bikeraw[,sources:=trimws(sources)]
3 bikeraw[is.na(sources),sources:='unknown']
4 unique(bikeraw$sources)
```

```
[1] "ad campaign"      "www.yahoo.com"    "www.google.fi"    "twitter"          "www.bing.com"     "www.google.co.uk"
[7] "facebook page"    "unknown"          "www.google.com"   "direct"           "blog"
```

11 values left, is that better?

Miller's Law

https://lawsofux.com/millers-law

George Miller, Princeton Professor and psychologist, stated that the number of objects an average person can hold in working memory is about seven, also known as The Magical Number Seven, Plus or Minus Two.

After discussing it with advertisers, you might come to the realization that its not that important to know which search engine the user used, just that they came from the web (matters if you are paying for google AdWords). Therefore we want to replace anything that has a website to just "web".

```
1 bikeraw[grep('www.[a-z]*.[a-z]*',sources),sources:='
    web']
2 unique(bikeraw$sources)
```

[1] "ad campaign" "web" "twitter" "facebook page" "unknown" "direct" "blog"

We are now left with 7 and we are ready to rock.