Analytics of Business Intelligence

Chapter # 1 - Extract, Transform, and Load

Dr. Wajahat Gilani

Rutgers Business School

October 14, 2019

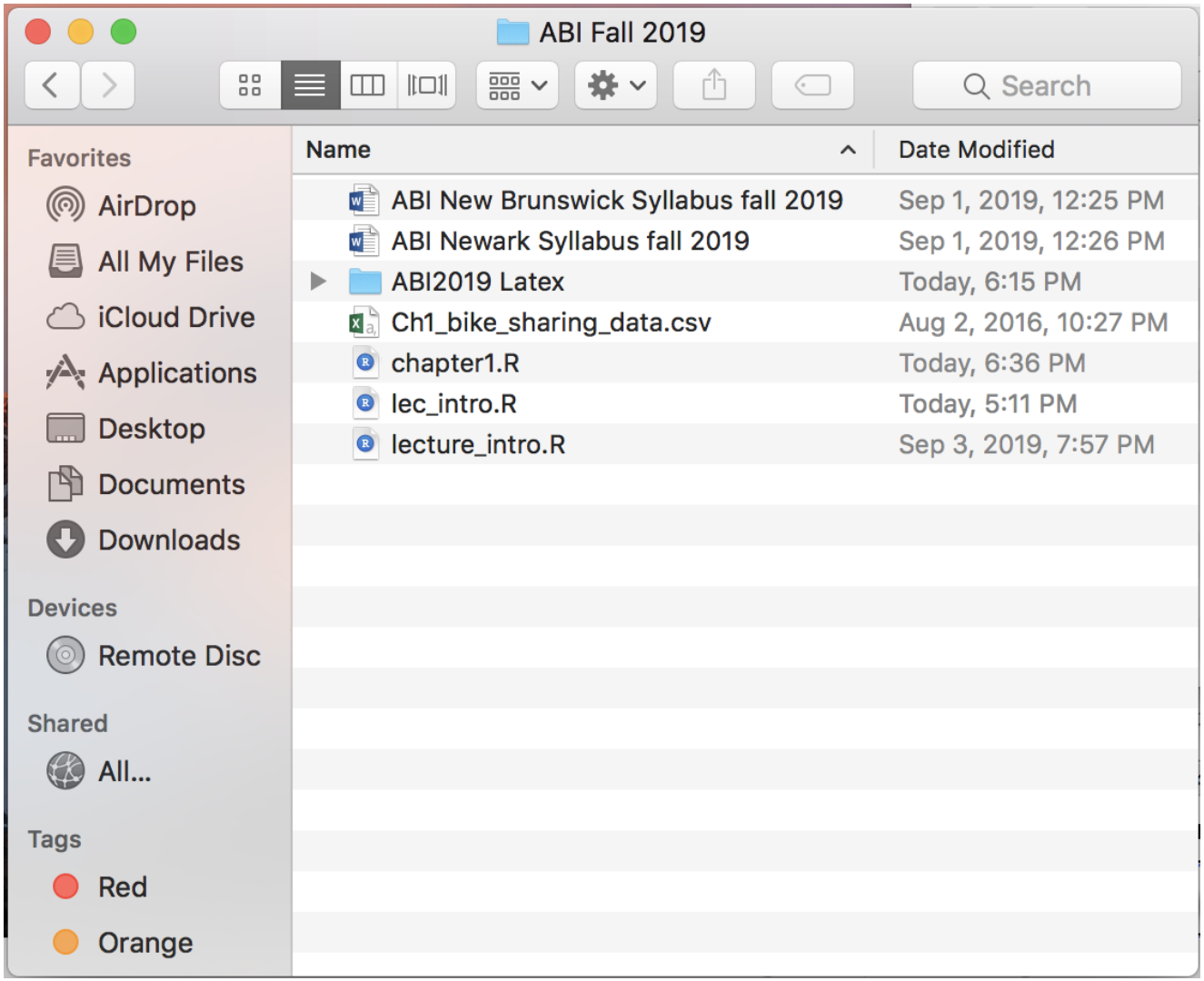
Extracting data from Sources

We will be using the text file Ch1_bike_sharing_data.csv and focus on:

- Importing CSV and other file formats
- Introducing the data container data.table
- Manipulating data in a data.table container

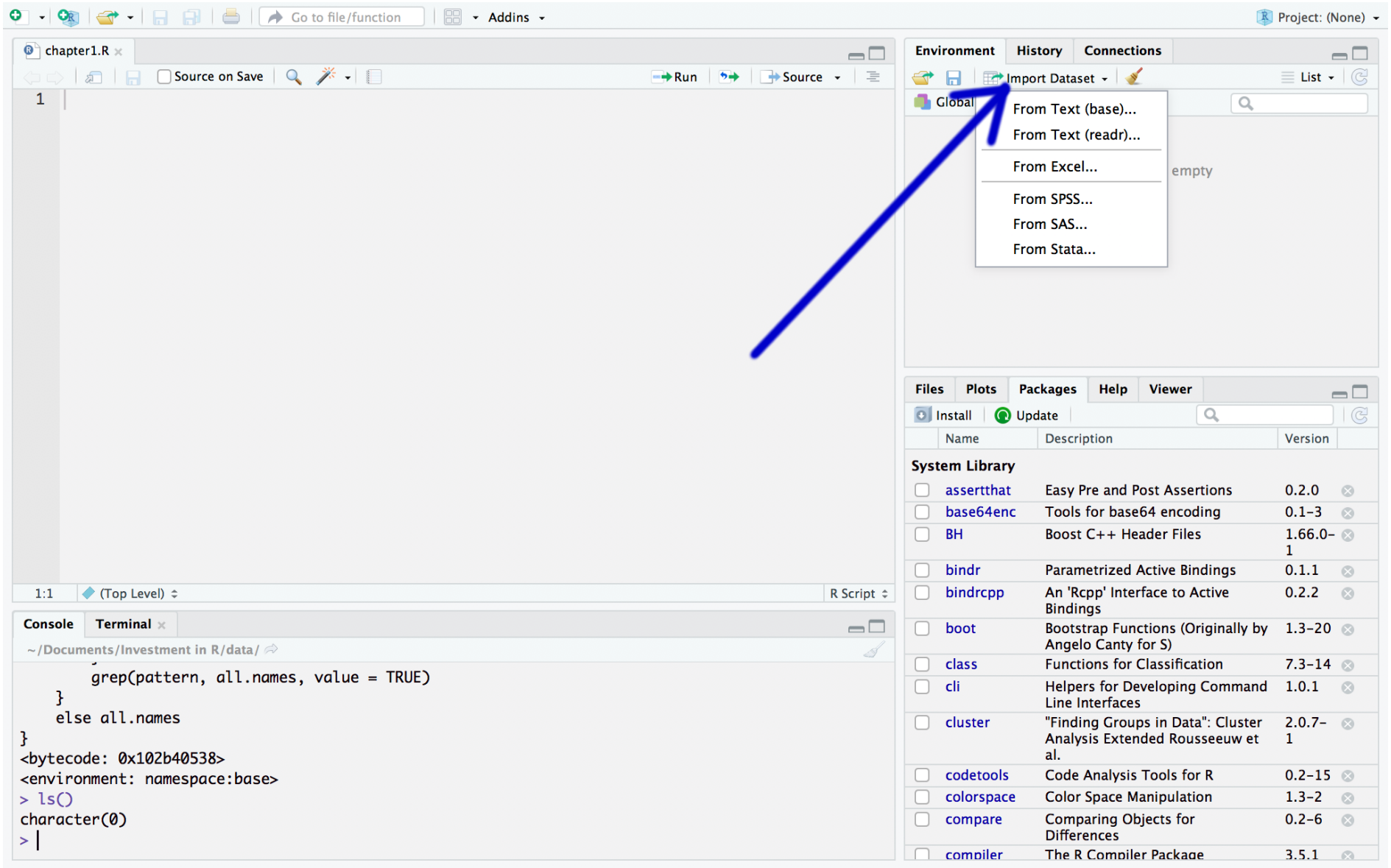
Importing CSV and other file formats

Download the file into your class folder.



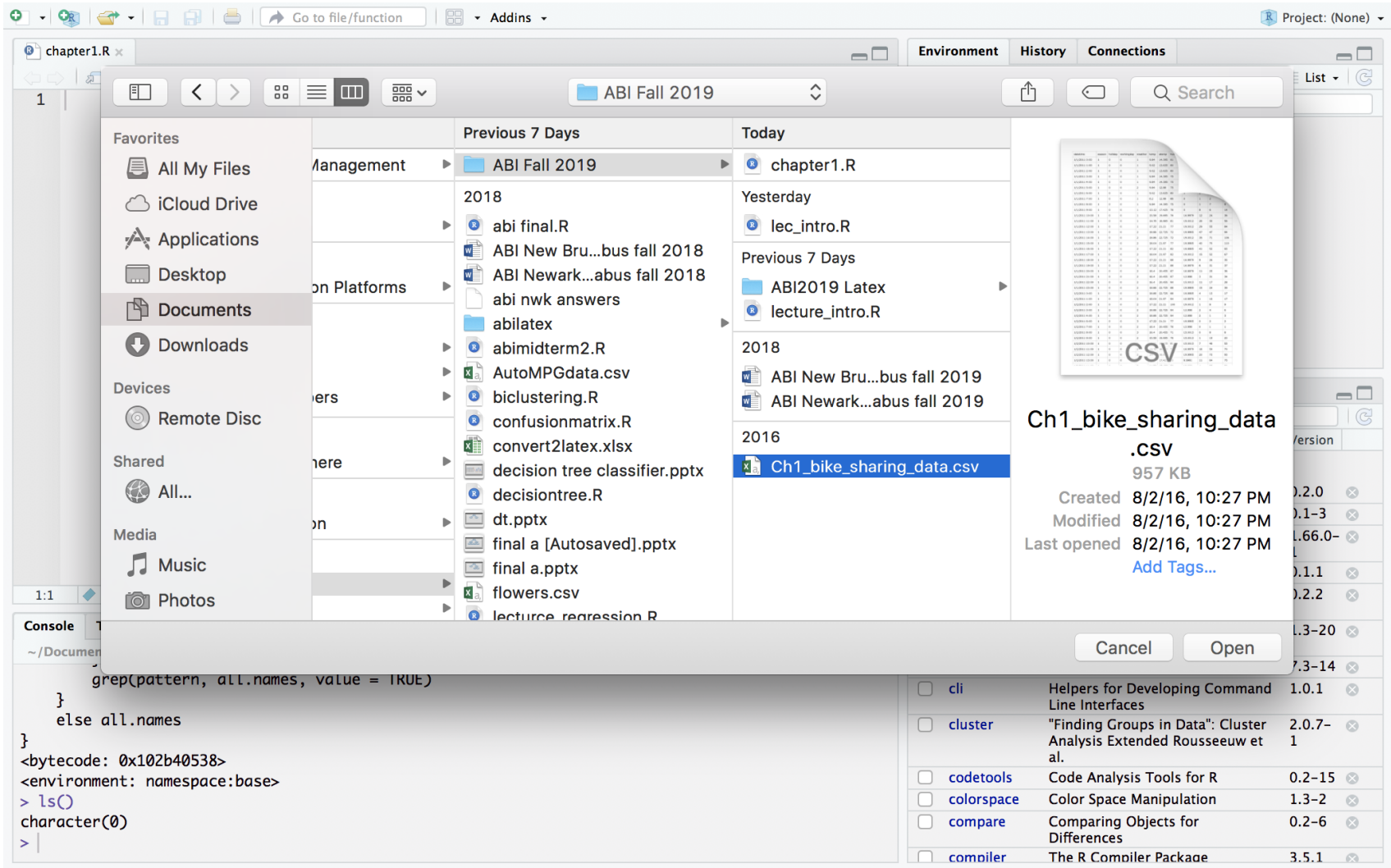
Importing CSV and other file formats

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



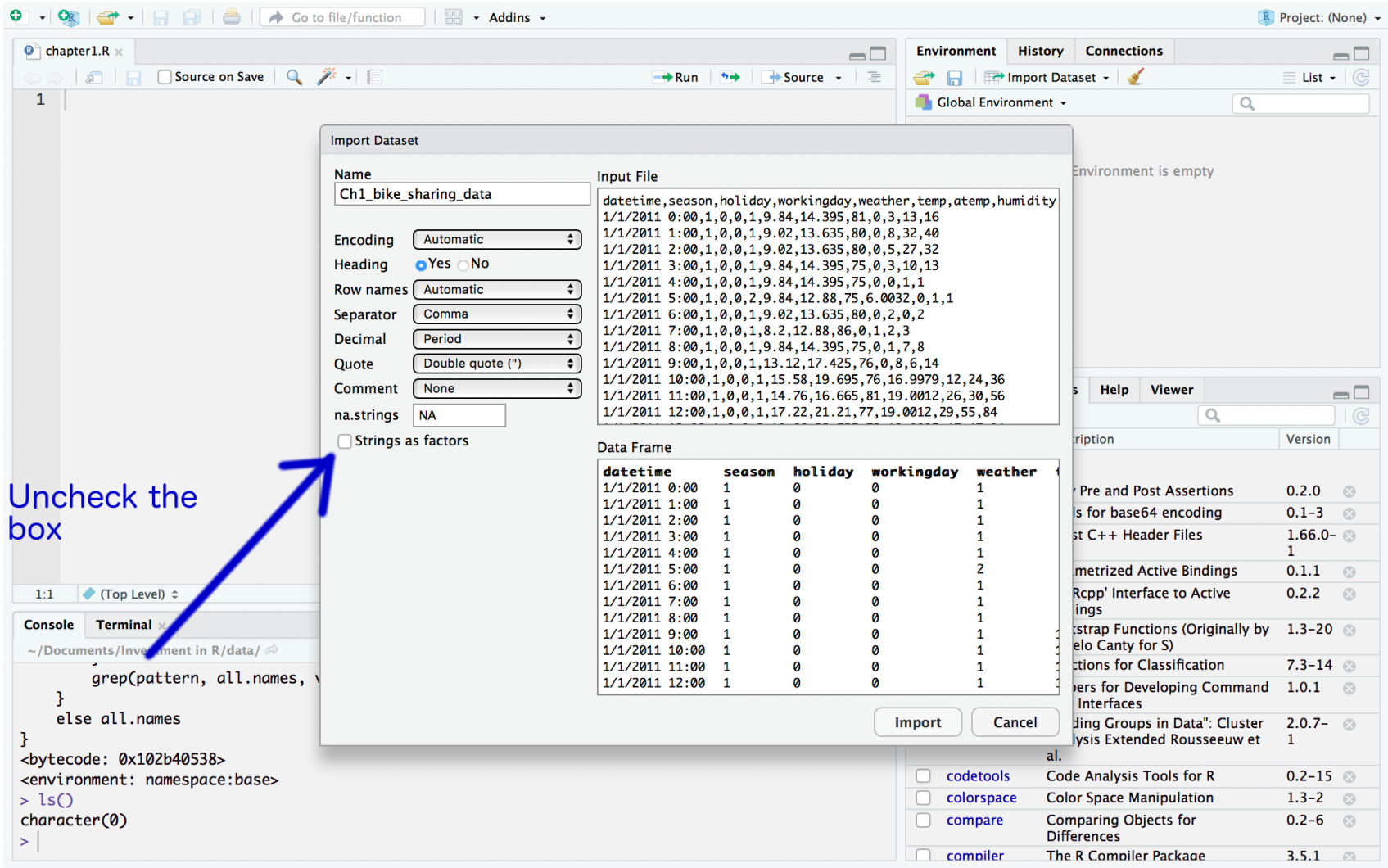
Importing CSV and other file formats

Go to the folder where you downloaded Ch1_bike_sharing_data.csv and click on "Open".



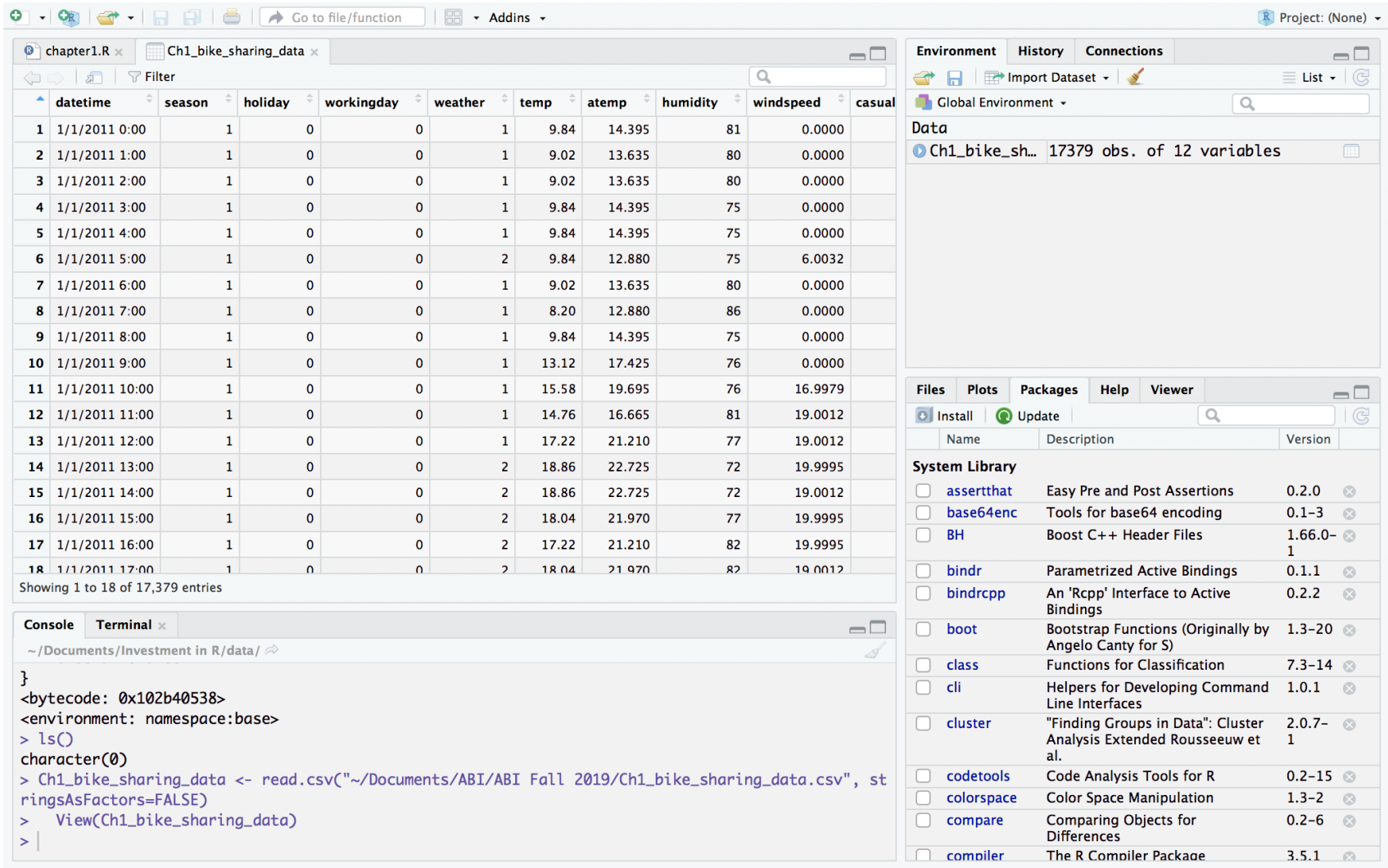
Importing CSV and other file formats

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



Importing CSV and other file formats

Table is loaded in the top right window and it's opened to view.

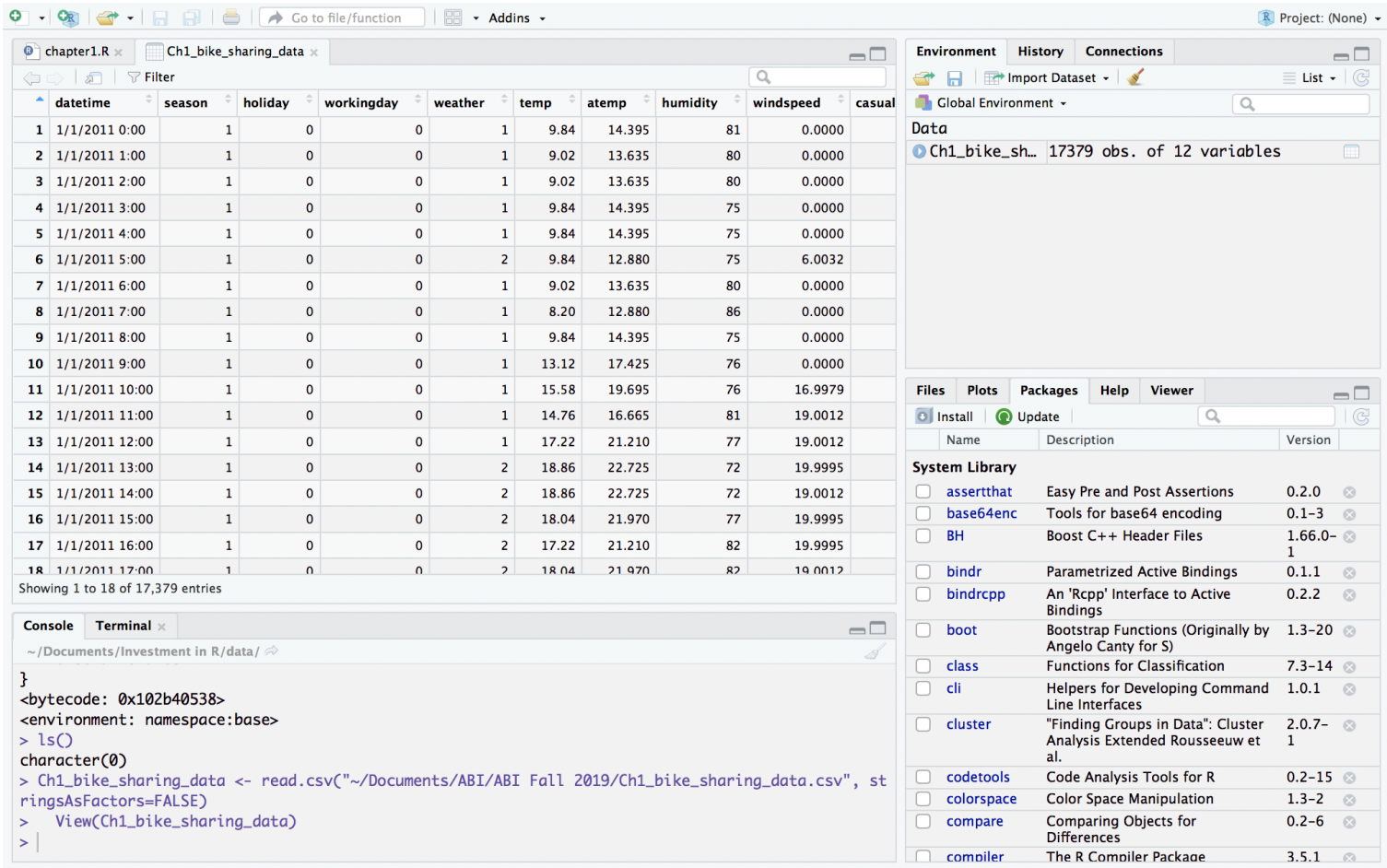


Importing CSV and other file formats

Notice the functions:

read.csv()

View()



chapter1.R | Ch1_bike_sharing_data

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual
1	1/1/2011 0:00	1	0	0	1	9.84	14.395	81	0.0000	
2	1/1/2011 1:00	1	0	0	1	9.02	13.635	80	0.0000	
3	1/1/2011 2:00	1	0	0	1	9.02	13.635	80	0.0000	
4	1/1/2011 3:00	1	0	0	1	9.84	14.395	75	0.0000	
5	1/1/2011 4:00	1	0	0	1	9.84	14.395	75	0.0000	
6	1/1/2011 5:00	1	0	0	2	9.84	12.880	75	6.0032	
7	1/1/2011 6:00	1	0	0	1	9.02	13.635	80	0.0000	
8	1/1/2011 7:00	1	0	0	1	8.20	12.880	86	0.0000	
9	1/1/2011 8:00	1	0	0	1	9.84	14.395	75	0.0000	
10	1/1/2011 9:00	1	0	0	1	13.12	17.425	76	0.0000	
11	1/1/2011 10:00	1	0	0	1	15.58	19.695	76	16.9979	
12	1/1/2011 11:00	1	0	0	1	14.76	16.665	81	19.0012	
13	1/1/2011 12:00	1	0	0	1	17.22	21.210	77	19.0012	
14	1/1/2011 13:00	1	0	0	2	18.86	22.725	72	19.9995	
15	1/1/2011 14:00	1	0	0	2	18.86	22.725	72	19.0012	
16	1/1/2011 15:00	1	0	0	2	18.04	21.970	77	19.9995	
17	1/1/2011 16:00	1	0	0	2	17.22	21.210	82	19.9995	
18	1/1/2011 17:00	1	0	0	2	18.04	21.970	82	19.0012	

Showing 1 to 18 of 17,379 entries

```
}
<bytecode: 0x102b40538>
<environment: namespace:base>
> ls()
character(0)
> Ch1_bike_sharing_data <- read.csv("~/Documents/ABI/ABI Fall 2019/Ch1_bike_sharing_data.csv", st
ringsAsFactors=FALSE)
>   View(Ch1_bike_sharing_data)
>
```

Environment | History | Connections

Global Environment

Data

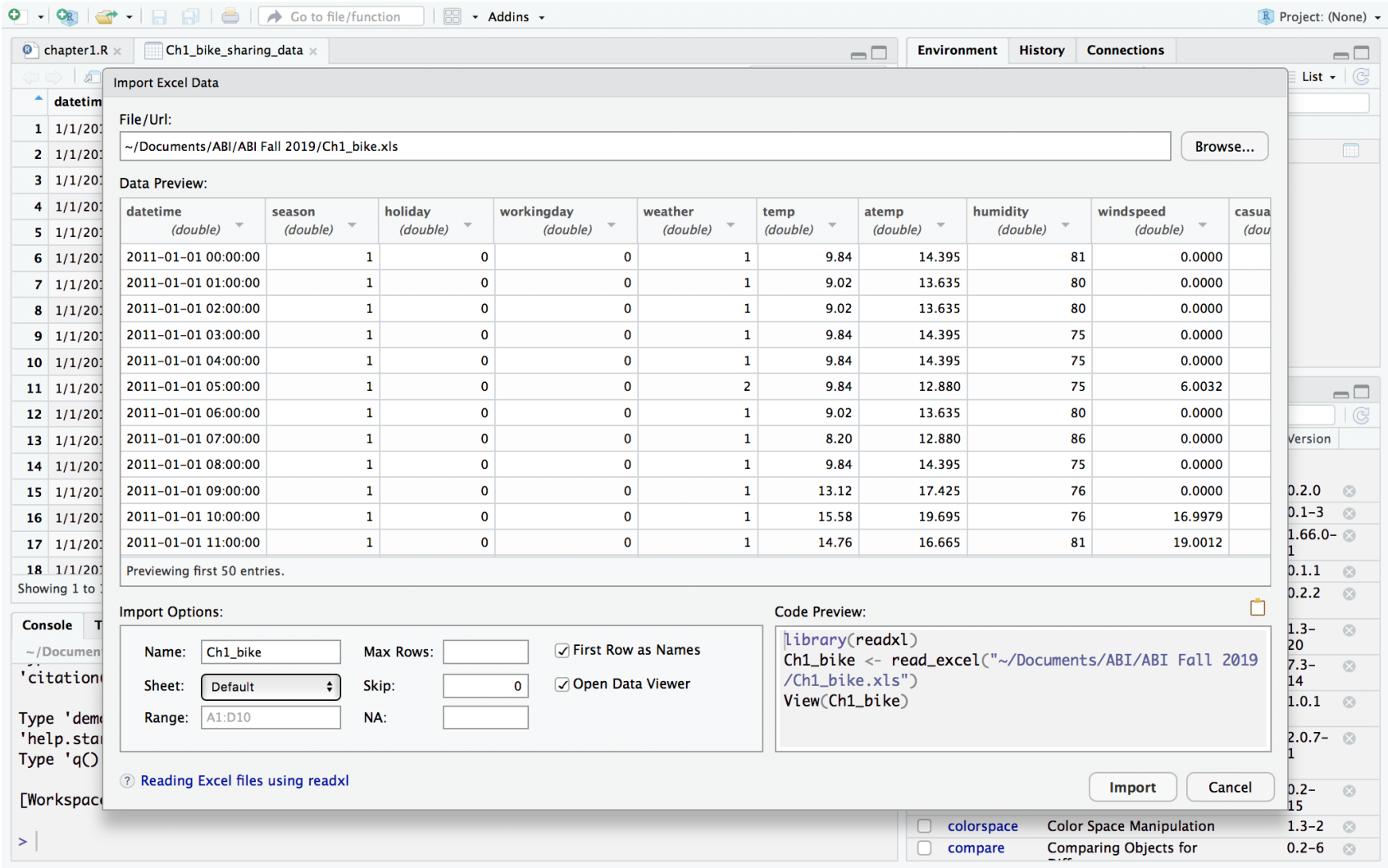
Ch1_bike_sh… 17379 obs. of 12 variables

Files | Plots | Packages | Help | Viewer

Install | Update

Name	Description	Version
System Library		
assertthat	Easy Pre and Post Assertions	0.2.0
base64enc	Tools for base64 encoding	0.1-3
BH	Boost C++ Header Files	1.66.0-1
bindr	Parametrized Active Bindings	0.1.1
bindrcpp	An 'Rcpp' Interface to Active Bindings	0.2.2
boot	Bootstrap Functions (Originally by Angelo Canty for S)	1.3-20
class	Functions for Classification	7.3-14
cli	Helpers for Developing Command Line Interfaces	1.0.1
cluster	"Finding Groups in Data": Cluster Analysis Extended Rousseeuw et al.	2.0.7-1
codetools	Code Analysis Tools for R	0.2-15
colorspace	Color Space Manipulation	1.3-2
compare	Comparing Objects for Differences	0.2-6
compiler	The R Compiler Package	3.5.1

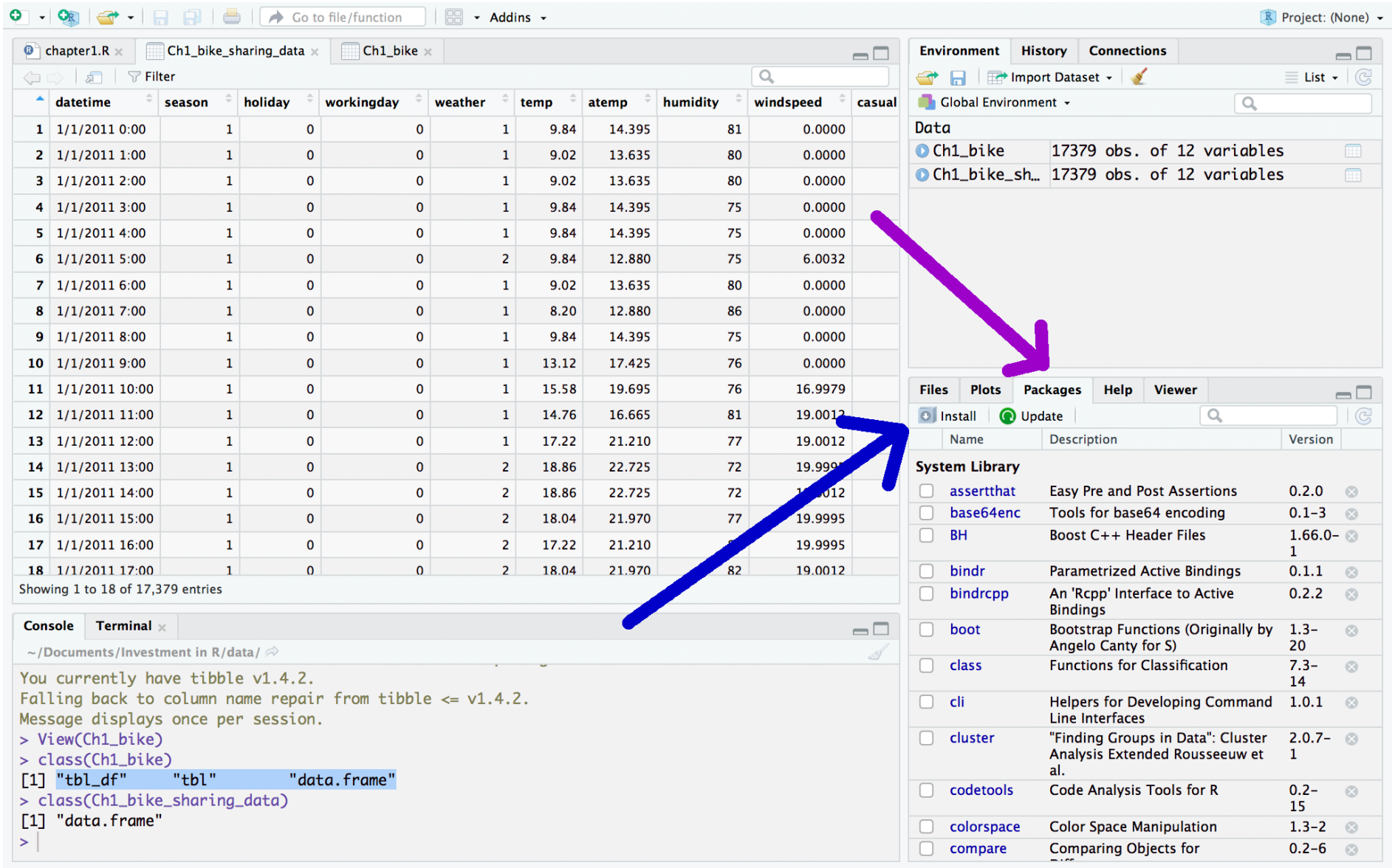
Reading from Excel



library(readxl)

read_excel()

3rd-Party Library Packages



Reading from Excel

```
1 class(Ch1_bike_sharing_data)
```

"data.frame"

```
1 class(Ch1_bike)
```

"tbl_df" "tbl" "data.frame"

These are two different types of data containers. A data.frame is native to R and is the simplest most basic type of table structure. A tbl_df comes from the readxl library, and it is more advanced and has more features. The industry standard has been to use data.table and that is the one we will use.

The pandas package ripped-off R, data.table than ripped-off pandas...and the circle of IT continues.

Create a new data.frame from an existing one

```
1 bike = Ch1_bike_sharing_data
```

The tables bike and Ch1_bike_sharing_data have the same exact data and structure but are completely different and independent from each other.

```
1 str(bike)
```

```
> str(bike)
'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 ...
```

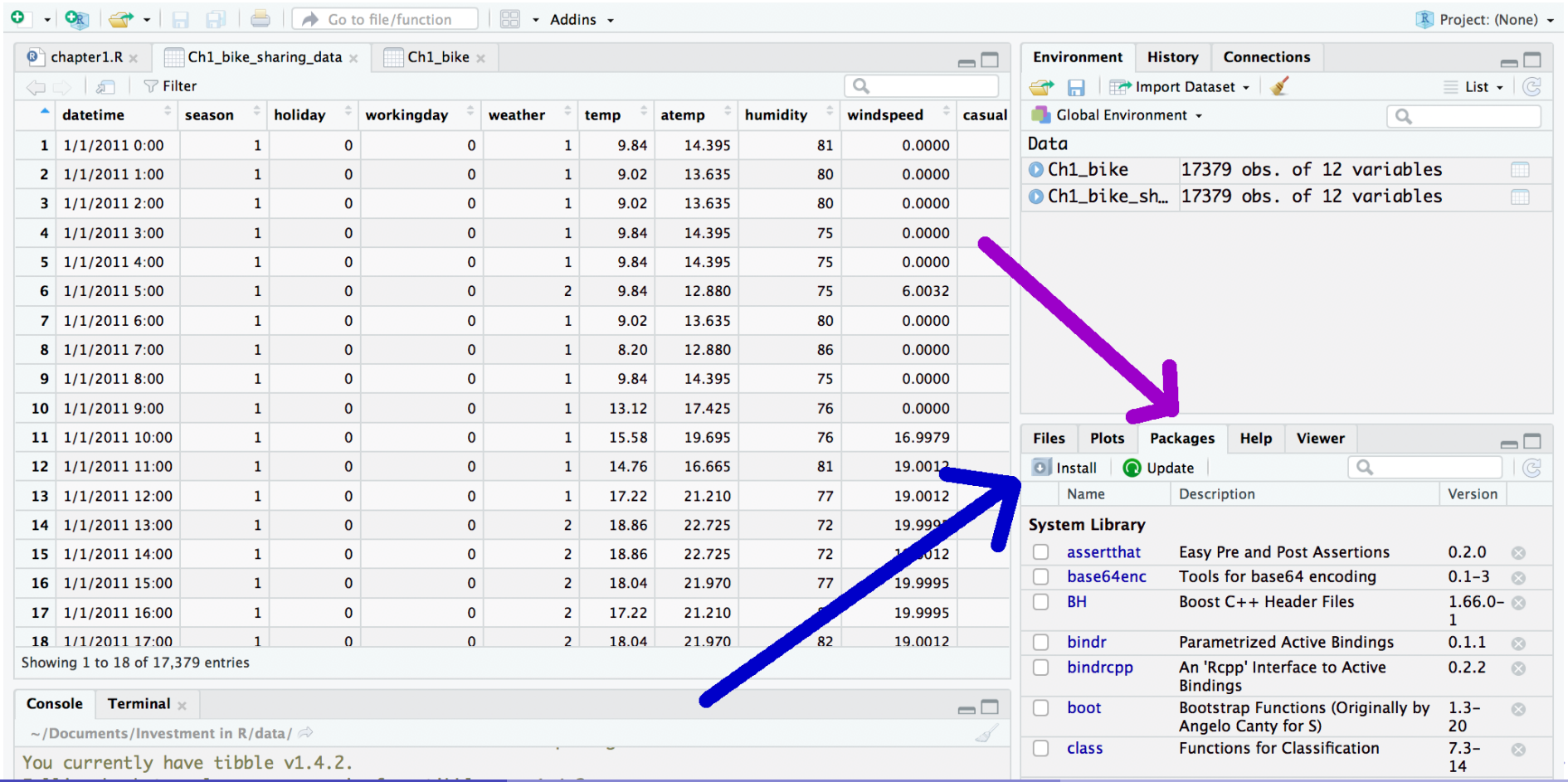
Dictionary for Data

- datetime: hourly date + timestamp
- season: 1=spring, 2=summer, 3=fall, 4=winter
- holiday: Whether the day is considered a holiday
- workingday: Whether the day is neither a weekend nor holiday
- weather:
  - 1: clear, few clouds, partly cloudy
  - 2: mist+cloudy, mist+broken clouds, mist+few clouds, mist
  - 3: light snow, light rain+thunderstorm+scattered clouds, light rain+scattered clouds
  - 4: heavy rain + ice pellets + thunderstorm + mist, snow + fog
- temp: Temperature in celsius
- atemp: feels like temperature in celsius
- humidity: relative humidity
- windspeed: wind speed
- casual: number of non-registered user rentals initiated
- registered: number of registered user rentals initiated
- count: number of total rentals

Convert data.frame to data.table

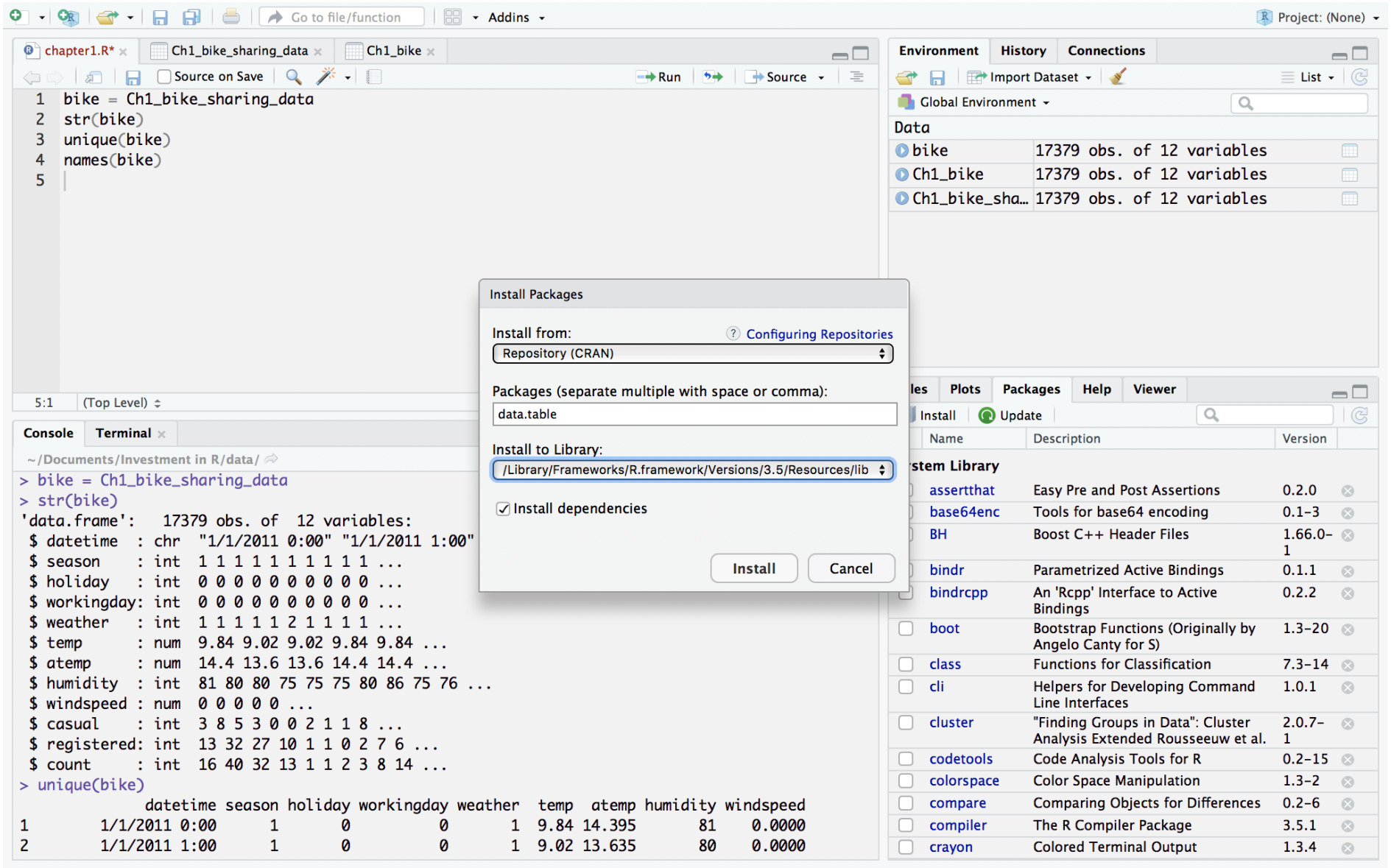
The table bike is a data.frame, which is the native basic table structure in R, we want to convert it into a data.table, which is a more powerful table structure in R.

We have to first install the data.table package.



Install data.table

Type data.table into the Packages space.



Load the data.table Package

Once you install a new package, you don't have to do it again, but you have to load it into memory when you use it in your code.

```
1 library(data.table)
```

Now we will convert the bike data.frame into a data.table.

```
1 setDT(bike)
```

If you run the class command:

```
1 class(bike)
```

This will be your output:
"data.table" "data.frame"
To convert it back into a data.frame (basic table), just use the command:

```
1 setDF(bike)
```

Filtering Data

Usual filtering capabilities:

== > < >= <=

& | ! %in%

is.na() *is.finite*()

df[Row Section]

df[Row Section,Column Section]

df[RowSection,ColumnSection,Grouping Section]

Filter for all the data in winter:

```
1 bike[season==4]
```

Filter for all the data in winter and summer:

```
1 bike[season==4 | season==1]
```

Notice the logic, even though we want bot winter AND summer, from a filtering point of view you can't have a row of data that has both the 4 AND 1 value, but you can filter for all the rows that have values 4 OR filter for rows that have values 1.

Filtering Data

Filter for all the data in winter and summer, using %in%:

```
1 bike[season %in% c(1,4)]
2 f=c(1,4)
3 bike[season %in% f]
```

Filter for the rows that are in winter AND the weather was clear (value 1)

```
1 bike[season==4 & weather==1]
```

Filter for the rows that are NOT in winter AND the weather was clear (value 1)

```
1 bike[season != 4 & weather==1]
```

Notice how when you run these lines, the results show up in the console. BUT if you could also place the results in another table.

```
1 nowinter = bike[season != 4 & weather==1]
```

Selecting Columns

df[Row Section,Column Section]

Select the datetime and windspeed columns:

```
1 bike[,.(datetime,windspeed)]
```

The results come back as a data.table, but if you don't use .(), and have just one column, then it comes back as a vector.

```
1 bike[,windspeed]
```

```
> bike[,windspeed]
   [1]  0.0000  0.0000  0.0000  0.0000  0.0000  6.0032  0.0000  0.0000  0.0000
  [10]  0.0000 16.9979 19.0012 19.0012 19.9995 19.0012 19.9995 19.9995 19.0012
  [19] 16.9979 16.9979 16.9979 12.9980 15.0013 19.9995 19.9995 16.9979 19.0012
  [28] 12.9980 12.9980 19.9995 12.9980 15.0013 15.0013 15.0013 16.9979 19.9995
  [37]  8.9981 12.9980 11.0014 11.0014 12.9980 22.0028 30.0026 23.9994 22.0028
  [46] 19.9995 11.0014 23.9994 27.9993 26.0027 19.0012 26.0027 12.9980 19.0012
  [55] 26.0027 16.9979 22.0028 19.9995 19.0012 19.0012 16.9979 16.9979 15.0013
  [64]  7.0015  0.0000  7.0015  8.9981  8.9981  7.0015  7.0015  7.0015  8.9981
  [73]  6.0032  7.0015  7.0015  8.9981 11.0014 15.0013 22.0028 19.9995 11.0014
  [82] 12.9980 15.0013 16.9979 16.9979 15.0013 12.9980  7.0015  7.0015  0.0000
```

Selecting Columns - Applying Functions

We can apply functions to column data, for example, what is the average temperature:

```
1 bike[,mean(temp)]
```

20.37647

Now what if we want the average, standard deviation, minimum, and maximum temperatures:

```
1 bike[,.(mean(temp),sd(temp),min(temp),max(temp))]
```

```
> bike[,.(mean(temp),sd(temp),min(temp),max(temp))]
          V1       V2   V3 V4
1: 20.37647 7.894801 0.82 41
```

We can store the results in another table and rename the columns:

```
1 bb=bike[,.(mean(temp),sd(temp),min(temp),max(temp))]
2 names(bb)=c('Mean','SD','Min','Max')
```

Selecting Columns - Applying Functions

```
> bb
       Mean       SD  Min Max
1: 20.37647 7.894801 0.82  41
```

What if we want to see the average, standard deviation, min and max, but now according to the seasons. we must now use the 3rd section of the data.table, to do aggregations.

df[RowSection,ColumnSection,Grouping Section]

The grouping section allows us to apply functions to subsections of the data. For example:

```
1 bike[,.(mean(temp),sd(temp),min(temp),max(temp)),by=
    season]
```

Selecting Columns - Applying Functions

```
> bike[,.(mean(temp),sd(temp),min(temp),max(temp)),by=season]
   season       V1       V2    V3    V4
1:      1 12.26501 4.867764  0.82 29.52
2:      2 22.33119 5.708559  6.56 38.54
3:      3 28.96282 3.850012 15.58 41.00
4:      4 17.34866 5.008863  5.74 31.16
```

3 Steps of Aggregation:

1. Create 4 subtables, 1 table for each season
2. Apply the functions to each subtable
3. Combine the results for each of the subtables into 1 table (that's why we will have 4 rows)

Now what if we wanted to find the mean, max, etc. for each season's own weather, i.e. for each particular weather type in each season.

```
1 bike[,.(mean(temp),sd(temp),min(temp),max(temp)),by=.(
      season,weather)]
```

Selecting Columns - Applying Functions

```
> bike[,.(mean(temp),sd(temp),min(temp),max(temp)),by=.(season,weather)]
    season weather        V1       V2    V3    V4
 1:      1       1 12.274462 5.113609  0.82 29.52
 2:      1       2 12.272100 4.506817  0.82 29.52
 3:      1       3 12.211111 4.138957  3.28 22.96
 4:      1       4  7.653333 1.706966  5.74  9.02
 5:      2       3 19.552759 5.450124  6.56 33.62
 6:      2       2 21.561556 5.324698  9.02 34.44
 7:      2       1 23.033711 5.734585  7.38 38.54
 8:      3       1 29.405250 3.950161 15.58 41.00
 9:      3       3 26.825279 2.909918 19.68 37.72
10:      3       2 28.037592 3.333418 18.86 39.36
11:      4       2 17.936843 4.832713  6.56 31.16
12:      4       3 18.424853 4.383840  9.02 27.06
13:      4       1 16.912618 5.124104  5.74 30.34
```

Let's put the results in a new table bws, and put the season and weather in order:

Selecting Columns - Applying Functions

```
1 bws = df[,.(mean(oi),sd(oi),min(oi),max(oi)),by=.(
      ticker,Date)]
2 names(bws)=c('ticker','Date','Mean','SD','Min','Max')
3 bws=bws[order(ticker,Date)]
```

```
> bws[order(season,weather)]
    season weather      Mean       SD   Min   Max
 1:      1       1 12.274462 5.113609  0.82 29.52
 2:      1       2 12.272100 4.506817  0.82 29.52
 3:      1       3 12.211111 4.138957  3.28 22.96
 4:      1       4  7.653333 1.706966  5.74  9.02
 5:      2       1 23.033711 5.734585  7.38 38.54
 6:      2       2 21.561556 5.324698  9.02 34.44
 7:      2       3 19.552759 5.450124  6.56 33.62
 8:      3       1 29.405250 3.950161 15.58 41.00
 9:      3       2 28.037592 3.333418 18.86 39.36
10:      3       3 26.825279 2.909918 19.68 37.72
11:      4       1 16.912618 5.124104  5.74 30.34
12:      4       2 17.936843 4.832713  6.56 31.16
13:      4       3 18.424853 4.383840  9.02 27.06
```

Creating New Variables

What if we wanted to filter the original bike data, by only the days that had a temperature that was less than the average temperature for its particular season? The simplest way to do this is to simply create a new column, that has the average temperature for each particular season. Again, we will use aggregation and apply grouping by season, but this time we will create new columns within the table with those results:

```
1 bike[,seasonAvgTemp:=mean(temp),by=season]
```

Lets view the top 10 rows:

```
1 head(bike,10)
```

Lets view the last 10 rows:

```
1 tail(bike,10)
```

Creating New Variables

Now we will simply filter and store the results in a new table named bike2:

```
1 bike2=bike[temp<seasonAvgTemp]
```

If we wanted to create multiple variables, from multiple columns, the syntax changes slightly:

```
1 bike[,c('seasonAvgTemp','seasonMinTemp'):=.(mean(temp)
    ,min(temp)),by=season]
```

Data.table - Shift Example (dfexp)

ticker	oi
a	2
a	4
a	6
a	8
a	10
b	5
b	10
b	15
b	20
b	25

The shift() function is a powerful tool that allows you to do analysis across different time-periods.

```
1  ticker=c(rep('a',5),rep('b',5))
2  oi = c(seq(2,10,2),seq(5,25,5))
3  dfexp=data.table(ticker,oi)
4
5  dfexp[, c('oi2', 'oi3', 'oi4'):=
       shift(oi, n = 1:3)]
```

Data.table - Shift Example (dfexp)

ticker	oi	oi2	oi3	oi4
a	2	NA	NA	NA
a	4	2	NA	NA
a	6	4	2	NA
a	8	6	4	2
a	10	8	6	4
b	5	10	8	6
b	10	5	10	8
b	15	10	5	10
b	20	15	10	5
b	25	20	15	10

Three new columns have now been created. Notice when there isn't enough previous rows, a NA is placed in the column.

What is wrong with the shift results?

```
1 dfexp[, c('oi2', 'oi3', 'oi4'):= shift(oi, n = 1:3)]
```

Data.table - Shift Example (dfexp)

ticker	oi	oi2	oi3	oi4
a	2	NA	NA	NA
a	4	2	NA	NA
a	6	4	2	NA
a	8	6	4	2
a	10	8	6	4
b	5	NA	NA	NA
b	10	5	NA	NA
b	15	10	5	NA
b	20	15	10	5
b	25	20	15	10

Since the shift only happens in sub-groups, b doesn't look at any of a's previous rows.

```
1 dfexp[, c('oi2', 'oi3', 'oi4'):= shift(oi, n = 1:3),by
    =ticker]
```