Agenda

- R Data Structures : Arrays
- R Data Structures : Lists
- R Data Structures : Data Frames
- R Data Structures : Factors
- **Data Manipulation in R**
  - Data Acquisition (Import & Export)
  - Subsetting Variables
  - Creating new variables

R Data Structures : Arrays

Arrays

- Arrays are numeric objects with dimension attributes.

```
array<-1:20
is.matrix(array)
[1]  FALSE
dim(array)
NULL
```

- The vector is not a matrix and it has no (NULL) dimensional attributes. We give the object dimensions like this (say, with five rows and four columns):

```
dim(array)<-c(5,4)
```

Now it does have dimensions and it is a matrix:

```
dim(array)
[1] 5 4
is.matrix(array)
[1] TRUE
```

```
myarray <- array(vector, dimensions, dimnames)
```

Arrays(contd.)

- When we look at array it is presented as a two-dimensional table (but note that it is not a table object):

```
array
      [,1] [,2] [,3] [,4]
[1,]    1    6   11   16
[2,]    2    7   12   17
[3,]    3    8   13   18
[4,]    4    9   14   19
[5,]    5   10   15   20
is.table(array)
[1] FALSE
```

Note that the values have been entered into array in columnwise sequence: this is the default in R.

- Thus a vector is a one-dimensional array that lacks any dim attributes.
- A matrix is a two-dimensional array.
- Arrays of three or more dimensions do not have any special names in R; they are simply referred to as three-dimensional or five-dimensional arrays.

Arrays(contd.)

- Here is a three-dimensional array of the first 16 lower-case letters with three matrices each of four rows and two columns:

```
A<-letters[1:16]
dim(A)<-c(2,2,4)
A
, , 1
     [,1] [,2]
[1,] "a"  "c"
[2,] "b"  "d"
, , 2
     [,1] [,2]
[1,] "e"  "g"
[2,] "f"  "h"
, , 3
     [,1] [,2]
[1,] "i"  "k"
[2,] "j"  "l"
, , 4
     [,1] [,2]
[1,] "m"  "o"
[2,] "n"  "p"
```

Arrays(contd.)

We want to select all the letters a to l. These are all the rows and all the columns of tables 1 and 2, so the appropriate subscripts are [,,1:2]

```
A[,,1:2]
,,1
     [,1] [,2]
[1,]  "a"  "c"
[2,]  "b"  "d"

, , 2
     [,1] [,2]
[1,]  "e"  "g"
[2,]  "f"  "h"
```

- Next, we want only the letters m to r. These are all the rows and all the columns from the third table, so the appropriate subscripts are [,,3]:

```
A[,,3]
     [,1] [,2]
[1,]  "i"  "k"
[2,]  "j"  "l"
```

Arrays(contd.)

- Here, we want only d, g, j, m, p and s. These are the third rows of all three tables, so the appropriate subscripts are [,,3]:

```
A[,,3]
   [,1] [,2]
[1,] "i"  "k"
[2,] "j"  "l"
```

Note that when we drop the whole first dimension (there is just one row in A[3,,]) the shape of the resulting matrix is altered (two rows and three columns in this example). This is a feature of R, but you can override it by saying drop = F to retain all three dimensions:

R Data Structures : Lists

Lists

- Like an R vector, an R list can contains items of different data types. List elements are accessed using two-part names, It is indicated with the dollar sign $ in R.

- A list allows you to gather a variety of (possibly unrelated) objects under one name. For example, a list may contain a combination of vectors, matrices, data frames, and even other lists.
- You create a list using the list() function:
  mylist <- list(object1, object2, ...)

```
> x <- list(u=2, v="abc")
> x
          $u
          [1] 2
          $v
          [1] "abc"
> x$u
     [1]
```

List Indexing

Consider a list :
j <- list(name="hue", sal=6000, union=F)

The list components can be accessed in several ways:

```
> j$sal
 [1] 6000
> j[["sal"]]
[1] 6000
> j[[2]]
[1] 6000
```

Adding & Deleting List Elements

New components can be added *after* a list is created.

```
> z <- list(a="jklm",b=200)
> z
$a
[1] "jklm"
$b
[1] 200
```

#Adding to List

```
> z$c <- " r-project" # add a c
component
> z
$a
[1] "jklm"
$b
[1] 200
$c
[1] " r-project "
```

#delete a list component by setting it to NULL.

```
>z$b <- NULL
> z
$a
[1] "jklm"
$c
```

Applying Functions To Lists

Using the lapply() and sapply()

Functions

```
> lapply(list(3:9,22:24),median)
[[1]]
[1] 6
[[2]]
[1] 23
```

Recursive Lists

Lists can be recursive, meaning that you can have lists within

lists.

```
> c(list(a=3,b=4,c=list(d=6,e=8)))
$a
[1] 3
$b
[1] 4
$c
$c$d
[1] 6
$c$e
[1] 8
```

R Data Structures : Data Frames

Creating A Data Frame

A data frame is a list, which contains the components of that list being equal-length

vectors.

```
> kids <- c("Sam", "Nick", "Rits")
> ages <- c(11,10,8)
> d <- data.frame(kids,ages,stringsAsFactors=FALSE)
```

Accessing Data Frames

Data frame can be accessed in multiple ways:

```
> d[[1]]
[1] "Sam" "Nick" "Rits”
> d$kids
[1] "Sam" "Nick" "Rits "
> d[,1]
[1] "Sam" "Nick“ “Rits”
```

Other Functions Structure of data frame

```
> str(d)
'data.frame': 3 obs. of 3 variables:
$ kids: chr "Sam" "Nick“ “Rits”
$ ages: num 11 10 8
```

Names of variables :

```
>names(d)
[1] " kids" "ages"
```

Identifying class of data :

```
class(d)
[1] “data.frame“
```

Other Functions

Structure of data frame
> str(d)
'data.frame': 3 obs. of 3 variables:
$ kids: chr "Sam" "Nick“ “Rits”
$ ages: num 11 10 8
Names of variables
>names(d)
[1] " kids" "ages"
Identifying class of data
class(d)
[1] “data.frame“
First few(2) observations
>head(d,2)
Last few(2) observations
>tail(d,2)

Subset Of Data Frames

- A data frame can be viewed in terms of row-and-column. In particular, we can extract sub data frames by rows or columns.

```
> d[2:3,]
  kids ages
2 Nick 10
3 Rits 8
```

```
> d[2:3, 2]
  [1] 10 8
> class(d[2:3, 2])
[1] “numeric”
```

```
> d[d$ages >= 10,]
  kids ages
1 Sam 12
2 Nick 10
```

rbind() and cbind() Functions

- In using rbind() to add a row, the added row is typically in the form of another data frame or list.

```
>rbind(d,list("Lara",15))
kids ages
1 Sam 11
2 Nick 10
3 Rits 8
4 Lara 15
```

R Data Structures : Factors

Factors

- Categorical (nominal) and ordered categorical (ordinal) variables in R are called factors. Factors are important in R as they determine how to analyze the data and present visually.
- An R *factor* might be viewed simply as a vector with extra information i.e. a record of the distinct values in that vector, called *levels*.

```
> x <- c(1:3)
> xfac <- factor(x)
> xf
[1] 1 2 3
Levels: 1 2 3
> str(xfac)
Factor w/ 3 levels “1","2","3": 1 2 3
```

Functions With Factors

Using tapply() Function

```
> numbers <- c(24,28,45,36,34,26)
> subject <- c("maths","english","english","maths","english","maths")
> tapply(numbers,subject,mean)
english   maths
  35.67   28.67
```

- R will treat factors as nominal variables and ordered factors as ordinal variables in statistical procedures and graphical analyses.
- You can use options in the factor( ) and ordered( ) functions to control the mapping of integers to strings (overriding the alphabetical ordering)

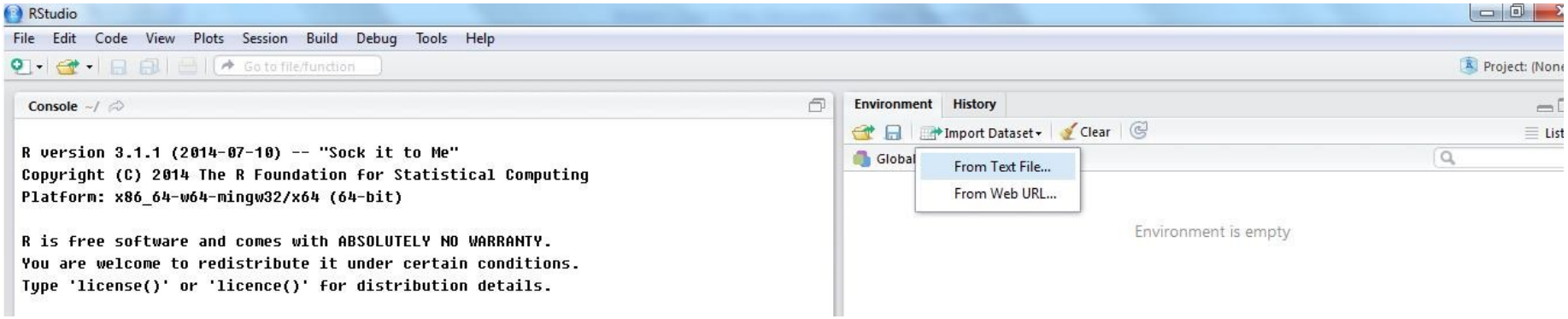
Data Manipulation In R

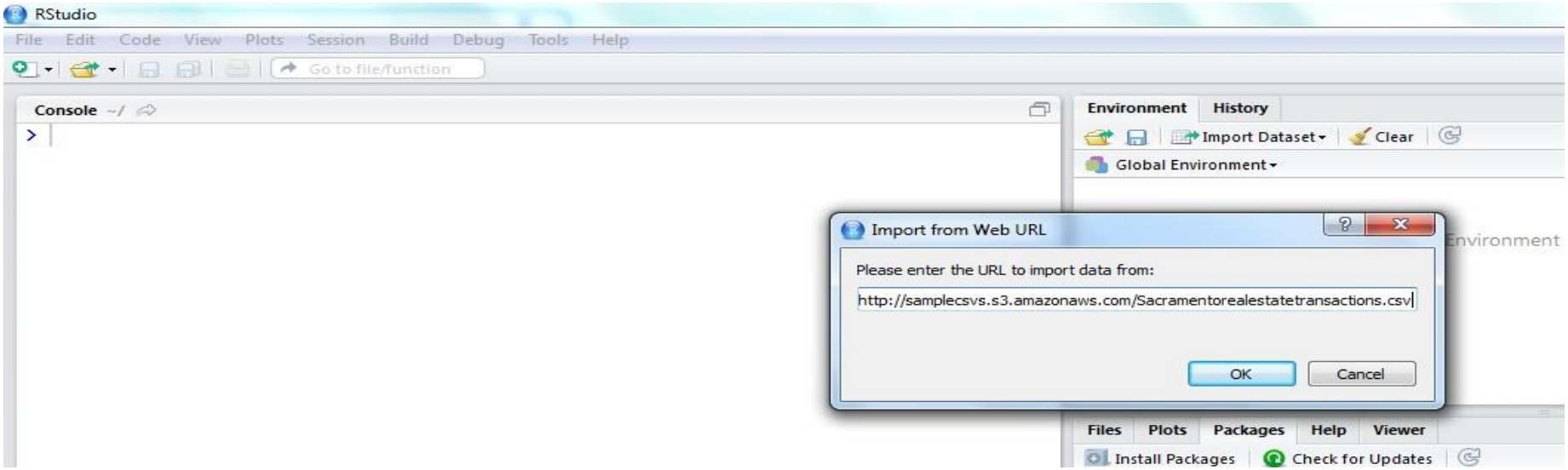
Data Acquisition (Import & Export)

Importing Data Using R Studio

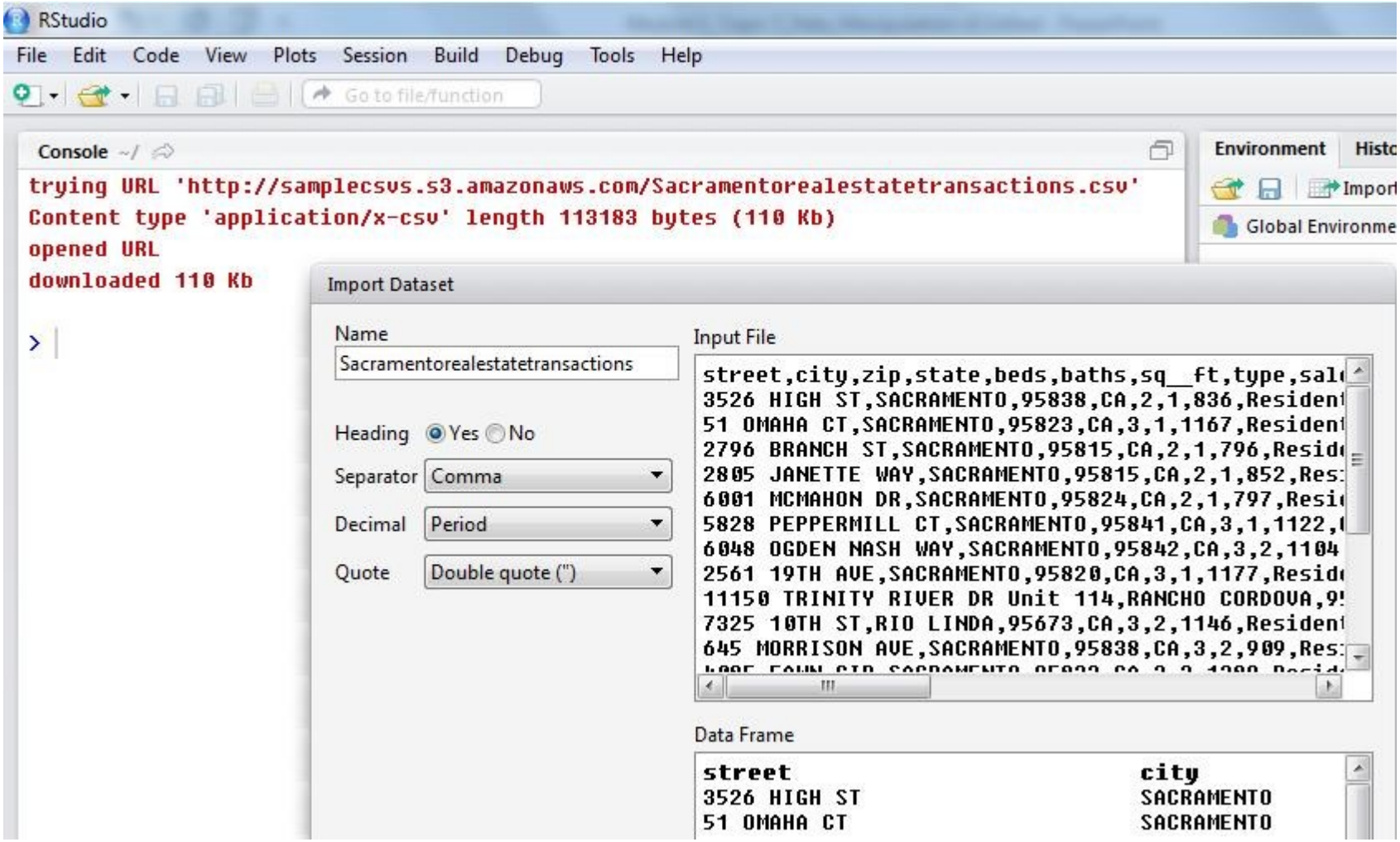
R Studio can be used to read a CSV , Tab delimited as well as a CSV in a Web URL

url : *http://samplecsvs.s3.amazonaws.com/Sacramentorealestatetransactions.csv*





Importing Data Using R Studio(contd.)



Importing Data

From A Comma Delimited Text File

```
# read csv file mydata.csv

mydata = read.csv("c:\\mydata.csv")
```

From Excel

```
# read in the first worksheet from the workbook myexcel.xlsx
library(xlsx)
mydata <- read.xlsx("c:\\myexcel.xlsx", 1) or
mydata <- read.xlsx("c:\\myexcel.xls", 1)

# read in the worksheet named mysheet
mydata <- read.xlsx("c:\\myexcel.xlsx", sheetName = "mysheet")
```

Importing Data(contd.)

From Tab delimited text File

```
#input Tab delimited text File
mydata = read.table("mydata.txt")
```

From SAS

```
#input SAS file
library(sas7bdat)
mydata=read.sas7bdat("C:\\mydata.sas7bdat")
```

From SPSS

```
#input SPSS file
library(foreign)
mydata = read.spss("C:\\mydata.sav", to.data.frame=TRUE)
```

Importing Data(contd.)

From Stata

```
# input Stata file
library(foreign)
mydata <- read.dta("c:/mydata.dta")
```

From systat

```
# input Systat file
library(foreign)
mydata <- read.systat("c:/mydata.dta")
```

From minitab

```
# input Systat file
library(foreign)
mydata <- read.systat("c:/mydata.mtp")
```

Reading A PDF file

```
# Install pdftotxt software from
# http://www.foolabs.com/xpdf/download.html
# Unzip and copy the contents under program files
# Assign path to the .pdf (~ is the name of the pdf)
d <- "C:\\Users\\Imarticus\\Desktop\\ ~.pdf"
# set path to pdftotxt.exe and convert pdf to text
exe <- "C:\\Program Files (x86)\\xpdfbin-win-3.04\\xpdfbin-win-
            3.04\\bin64\\pdftotext.exe”
system(paste("\"", exe, "\" \"", d, "\"", sep = "")
# Convert pdf into txt
txtfile <- sub(".pdf", ".txt", d)
shell.exec(txtfile);
shell.exec(txtfile)
```

Exporting Data

For SPSS, SAS and Stata load the **foreign** packages.
For Excel, you will need the **xlsx** package.

To a Tab Delimited Text File

```
write.table(mydata, "c:/mydata.txt", sep="\t")
```

To an Excel Spreadsheet

```
library(xlsx)
write.xlsx(mydata, "c:/mydata.xlsx")
```

To SPSS

```
# write out text datafile and
# an SPSS program to read it
library(foreign)
write.foreign(mydata, "c:/mydata.txt", "c:/mydata.sps", package="SPSS")
```

Exporting Data(contd.)

To SAS

```
# write out text datafile and
# an SAS program to read it
library(foreign)
write.foreign(mydata, datafile = "c:/mydata.txt", codefile
="c:/mydata.sas", package="SAS")
```

To Stata

```
# export data frame to Stata binary format
library(foreign)
write.dta(mydata, "c:/mydata.dta")
```

Creating New Variables

Creating New Variables

```
#Create Data Frame
mydata<-data.frame (u = c(4,  8,  6,  4),v = c(9,  27,  7,  8), z = c(1,  7,  17,  3))

 # Create New variables as sum and mean of column values
mydata$sumx   <-   mydata$u +  mydata$v + mydata$z

mydata$meanx  <- (mydata$u   +   mydata$v + mydata$v)/3

Mydata$mean <- average(mydata$u   +   mydata$v + mydata$v)

#Using attach and detach functions
attach(mydata)

mydata$sumx    <-   u +  v + z
mydata$meanx   <- (u +  v + z)/3
detach (mydata)

#Using transform function
mydata    <-    transform(mydata,
   sumx = u + v + z,
   meanx = (u + v + z)/3)
```

Why Use Attach/Detach In R?

- R objects that reside in other R objects can require a lot of typing to access. For example, to refer to a variable x in a data frame df, one could type df$x. This is no problem when the data frame and variable names are short, but can become burdensome when longer names or repeated references are required, or objects in complicated structures must be accessed.
- The attach() function in R can be used to make objects within data frames accessible in R with fewer keystrokes. So the same can be accessed as following

```
> attach(df)
>x
```

Why Use Attach/Detach In R?(contd.)

- Now what if your session already has another variable x stored locally?
- Then following are the various ways to access
  - Reference variables directly (e.g. lm(ds$x ~ ds$y))
  - Specify the data frame for commands which support this (e.g. lm(y ~ x, data=ds))
  - Use the with() function, which returns the value of whatever expression is evaluated (e.g. with(ds,lm(y ~x)))
  - These next three are equivalent
- > mean(ds$cesd[ds$female==1])
- [1] 36.88785
- > with(ds, mean(cesd[female==1]))
- [1] 36.88785
- > with(subset(ds, female==1), mean(cesd))
- [1] 36.88785

Subsetting Variables

Sample Dataframe : company

```
#Variables or Lists
executive <- c(1, 2, 3, 4, 5)
date <- c("10/24/08", "10/28/08", "10/1/08", "10/12/08", "5/1/09")
country <- c("US", "US", "UK", "UK", "UK")
gender <- c("M", "F", "F", "M", "F")
age <- c(32, 45, 25, 39, 99)
u1 <- c(5, 3, 3, 3, 2)
u2 <- c(4, 5, 5, 3, 2)
u3 <- c(5, 2, 5, 4, 1)
u4 <- c(5, 5, 5, NA, 2)
u5 <- c(5, 5, 2, NA, 1)
#Dataframe
company <- data.frame(executive, date, country, gender, age, u1, u2, u3, u4, u5,
stringsAsFactors=FALSE)
```

Subsetting Variables

Selecting (keeping) variables

For example the below statement selects variables : country, age , u1, u2, u3, u4, and u5 from the company data frame and saves them to the data frame newdata.

```
newdata  <-   company[, c(3, 6 : 10)]
```

Another option :

```
Myvars   <-   c("country", "u1", "u2", "u3", "u4", "u5")
      Newdata  <-   company [myvars]
```

Using paste function for variables with prefix "u" :

```
Myvars    <-   paste("u", 1:5, sep="")
Newdata   <-   company [myvars]
```

Subsetting Variables(contd.)

Excluding (dropping) variables

Exclude variables u3 and u5 with the statements

```
Myvars    <-   names(company) %in% c("u3",  "u5")
Newdata   <-   company [!myvars]
```

Trimming :

```
Newdata   <-   company[c(-8, -10)]
```

Set columns u3 and u4 to undefined (NULL) :

```
company$u3   <- company$u5  <- NULL
```

Selecting Observations

Selecting or excluding observations (rows) is a part of ETL (Data preparation job) :

```
newdata     <- company[1:3,]

newdata     <- company[which(company$gender=="M" & company$age > 25) ,]

attach(company)
newdata     <-    company[which(gender=='M' & age > 25) ,]
detach(company)
```

Selecting Observations(contd.)

To limit analyses to observations in between January 1, 2009 and December 31, 2009.

```
company$date  <- as.Date(company$date, "%m/%d/%y")
startdate <- as.Date("2009-01-01")
enddate   <- as.Date("2009-12-31")
newdata <- company[which(company$date >= startdate & company$date <=
```

The Subset( ) Function

The subset function is probably the easiest way to select variables and observations.

```
 # select all rows that have a value of age greater than or equal to 35 or age less than 24.
newdata <- subset(company, age >= 35 | age < 24, select=c(u1 , u2,  u3,  u4))

# select all men over the age of 25 and keep variables gender through q4
newdata <- subset(company, gender=="M" & age > 25, select=gender:u4)
```

Random Samples

The sample ( ) function enables you to take a random sample (with or without replacement) of size *n* from a dataset.

```
#Draw a random sample of 5
#from all the data points
#without replacement

mysample <- company[sample(1:nrow(company), 5, replace=FALSE),]
```

Next class - Data Manipulation in R(Contd..)

- Renaming and Recoding Variables• Reshaping Data
- Merging & Concatenating Datasets
- Using SQL Statements To Manipulate Data Frames
- Data Type Conversion
- Data Values

Questions