

Data frames

In this lecture

- Dataframe
- Create
- Access rows and columns
- Edit
- Add new rows and columns

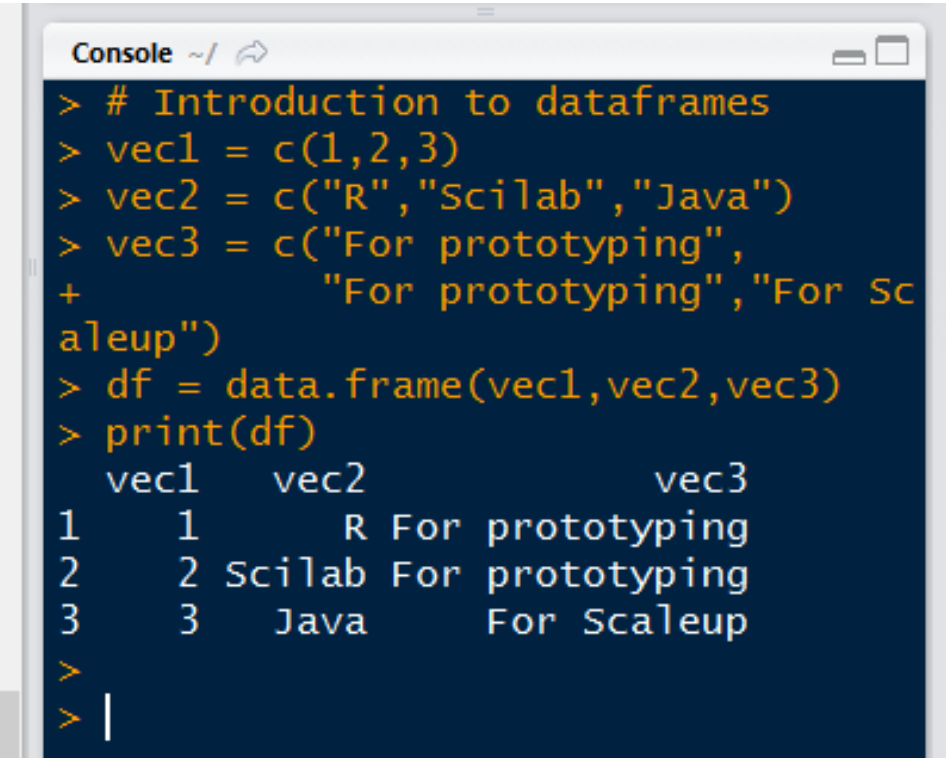
Dataframes: Create dataframe





Data frames are generic data objects of R, used to store tabular data

Code

```
# Introduction to data frames  
vec1 = c(1,2,3)  
vec2 = c("R","Scilab","Java")  
vec3 = c("For prototyping",  
        "For prototyping","For Scaleup")  
df = data.frame(vec1,vec2,vec3)  
print(df)
```

Console Output



```
Console ~/      
> # Introduction to dataframes  
> vec1 = c(1,2,3)  
> vec2 = c("R","Scilab","Java")  
> vec3 = c("For prototyping",  
+         "For prototyping","For Scaleup")  
> df = data.frame(vec1,vec2,vec3)  
> print(df)  
  vec1 vec2      vec3  
1    1   R For prototyping  
2    2 Scilab For prototyping  
3    3   Java   For Scaleup  
>  
> |
```

Create a dataframe using data from a file

- A dataframe can also be created by reading data from a file using the following command

➤ *newDF = read.table(path="Path of the file")*

- In the path, please use `'` instead of `\`

➤ Example: *"C:/Users/hii/Documents/R/R-Workspace/"*

- A separator can also be used to distinguish between entries. Default separator is space, `' '`

➤ *newDF = read.table(file="path of the file", sep)*

Accessing rows and columns

- `df[val1,val2]` refers to row “val1”, column “val2”. Can be number or string
- “val1” or “val2” can also be array of values like “1:2” or “c(1,3)”
- `df[val2]` (no commas) - just refers to column “val2” only

Code

```
# accessing first & second row:
print(df[1:2,])

# accessing first & second column:
print(df[,1:2])

# accessing 1st & 2nd column -
# alternate:
print(df[1:2])
```

Console Output

```
> print(df[1:2,])
vec1  vec2          vec3
1    1    R For prototyping
2    2 Scilab For prototyping
> # accessing first & second column:
> print(df[,1:2])
vec1  vec2
1    1    R
2    2 Scilab
3    3  Java
> # accessing 1st & 2nd column - alternate:
> print(df[1:2])
vec1  vec2
1    1    R
2    2 Scilab
3    3  Java
```

Subset

subset() which extracts subset of data based on conditions

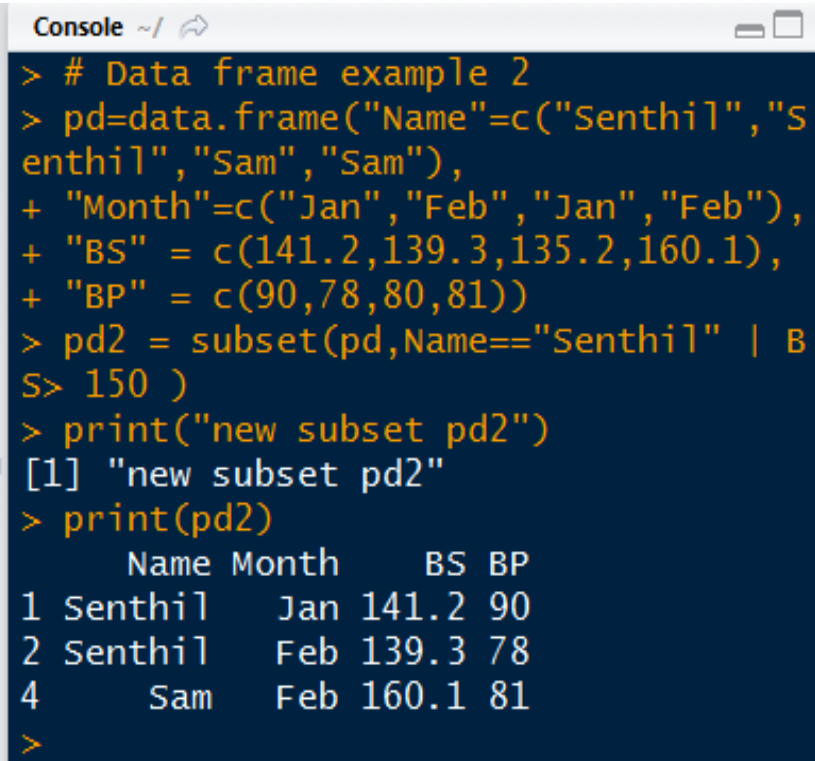
Code

```
# Data frame example 2
pd=data.frame("Name"=c("Senthil","Senthil","Sam","Sam"),
"Month"=c("Jan","Feb","Jan","Feb"),
"BS" = c(141.2,139.3,135.2,160.1),
"BP" = c(90,78,80,81))

pd2 = subset(pd,Name=="Senthil" | BS> 150 )

print("new subset pd2")
print(pd2)
```

Console Output



```
Console ~/
> # Data frame example 2
> pd=data.frame("Name"=c("Senthil","Senthil","Sam","Sam"),
+ "Month"=c("Jan","Feb","Jan","Feb"),
+ "BS" = c(141.2,139.3,135.2,160.1),
+ "BP" = c(90,78,80,81))
> pd2 = subset(pd,Name=="Senthil" | BS> 150 )
S> 150 )
> print("new subset pd2")
[1] "new subset pd2"
> print(pd2)
      Name Month    BS BP
1 Senthil   Jan 141.2  90
2 Senthil   Feb 139.3  78
4      Sam   Feb 160.1  81
>
```

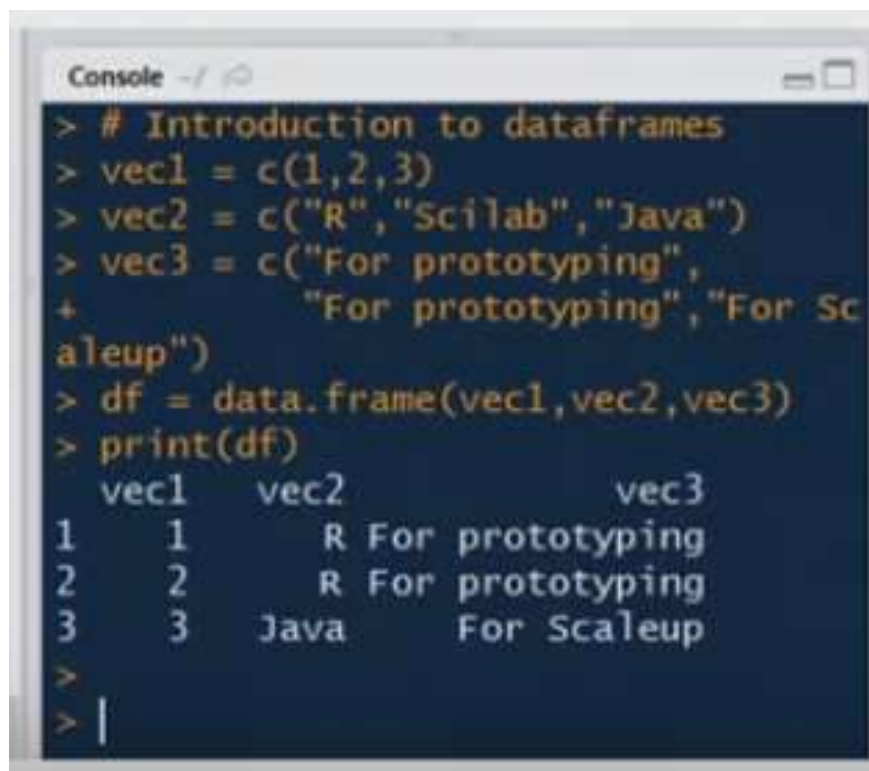
Editing dataframes

Dataframes can be edited by direct assignment

Code

```
# Introduction to dataframes
vec1 = c(1,2,3)
vec2 = c("R","Scilab","Java")
vec3 = c("For prototyping", "For
prototyping", "For Scaleup")
df = data.frame(vec1,vec2,vec3)
print(df)
df[[2]][2] = "R"
```

Console Output



```
Console - /
> # Introduction to dataframes
> vec1 = c(1,2,3)
> vec2 = c("R","Scilab","Java")
> vec3 = c("For prototyping",
+         "For prototyping", "For Sc
aleup")
> df = data.frame(vec1,vec2,vec3)
> print(df)
  vec1  vec2      vec3
1    1    R For prototyping
2    2    R For prototyping
3    3  Java   For Scaleup
>
> |
```

Editing dataframes

- A dataframe can also be edited using the `edit()` command
- Create an instance of data frame and use `edit` command to open a table editor, changes can be manually made

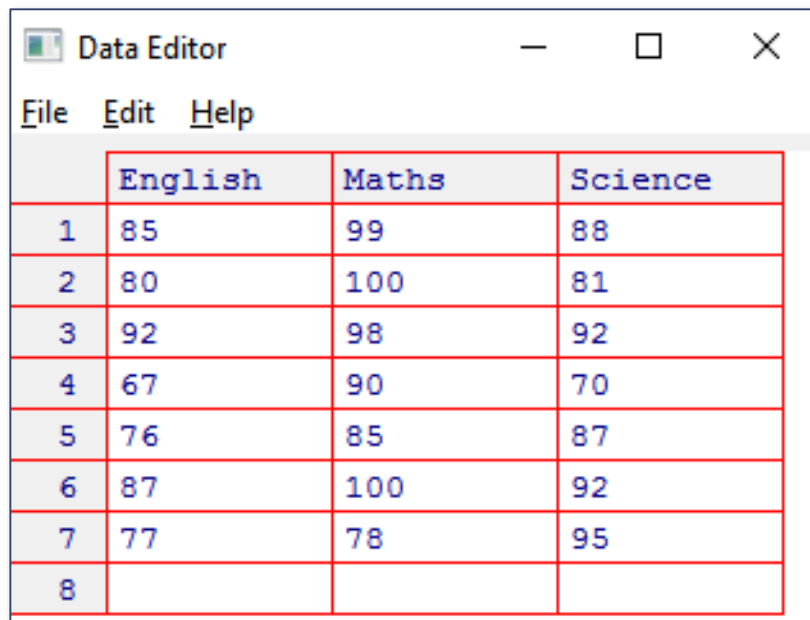
Code

```
# Editing a data frame  
myTable = data.frame()  
myTable = edit(myTable)
```

	English	Maths	Science
1	85	99	88
2	80	100	81
3	92	98	92
4	67	90	78
5	76	85	87
6	87	100	92
7	77	78	95

Enter these
values in the table

And close the
editor



	English	Maths	Science
1	85	99	88
2	80	100	81
3	92	98	92
4	67	90	78
5	76	85	87
6	87	100	92
7	77	78	95
8			

Adding extra rows and columns

Extra row can be added with “rbind” function and extra column with “cbind”

Code

```
# continuing from previous example
# adding extra row and column:
df = rbind(df,data.frame(vec1=4,
vec2="C", vec3="For Scaleup"))
print("adding extra row")
print(df)
df = cbind(df,vec4=c(10,20,30,40))
print("adding extra col")
print(df)
```

Console Output

```
> # continuing from previous example
> # adding extra row and column:
> df = rbind(df,data.frame(vec1=4,
+                          vec2="C",
+                          vec3="For Scaleup"))
> print("adding extra row")
[1] "adding extra row"
> print(df)
  vec1  vec2      vec3
1    1    R For prototyping
2    2 Scilab For prototyping
3    3   Java   For Scaleup
4    4    C   For Scaleup
> df = cbind(df,vec4=c(10,20,30,40))
> print("adding extra col")
[1] "adding extra col"
> print(df)
  vec1  vec2      vec3 vec4
1    1    R For prototyping 10
2    2 Scilab For prototyping 20
3    3   Java   For Scaleup 30
4    4    C   For Scaleup 40
```

Deleting rows and columns

There are several ways to delete a row/column, some cases are shown below

Code

continuing from previous example

Deleting rows and columns:

```
df2 = df[-3,-1]
```

```
print(df2)
```

conditional deletion:

```
df3 = df[!names(df) %in% c("vec3")]
```

```
print(df3)
```

```
df4 = df[!df$vec1==3,]
```

```
print(df4)
```

A '-' sign before value and before ',' for rows & after ',' for columns

'!' means no to those rows /columns which satisfy the condition

```
> print(df2)
  vec2      vec3 vec4
1    R For prototyping 10
2 Scilab For prototyping 20
4    C    For Scaleup 40

> # conditional deletion:
> df3 = df[!names(df) %in% c("vec3")]
> print(df3)
  vec1  vec2 vec4
1     1    R  10
2     2 Scilab 20
3     3   Java 30
4     4    C  40

> df4 = df[!df$vec1==3,]
> print(df4)
  vec1  vec2      vec3 vec4
1     1    R For prototyping 10
2     2 Scilab For prototyping 20
4     4    C    For Scaleup 40
>
```

Manipulating rows – the factor issue


- When character columns are created in a data.frame, they become factors
- Factor variables are those where the character column is split into categories or factor levels

Code

```
# Manipulating rows in data frame
# continued from previous page
df[3,1]= 3.1
df[3,3]= "Others"
print(df)
```

Console Output

```
> # Manipulating rows in dataframe
> # continued from previous page
> df[3,1]= 3.1
> df[3,3]= "Others"
Warning message:
In `[<-.factor`(`*tmp*`, iseq, value = "Others") :
  invalid factor level, NA generated
> print(df)
  vec1  vec2      vec3
1  1.0    R For prototyping
2  2.0 Scilab For prototyping
3  3.1  Java      <NA>
```



Notice the NA values displayed instead of the string “Others”.
Also see the use of the word “factor” in the warning above

Resolving factor issue

New entries need to be consistent with factor levels which are fixed when the dataframe is first created

Code

```
vec1 = c(1,2,3)
vec2 = c("R","Scilab","Java")
vec3 = c("For prototyping",
        "For prototyping","For Scaleup")
df = data.frame(vec1,vec2,vec3,
stringsAsFactors = F)
# Now trying the same manipulation
df[3,3]= "Others"
print(df)
```

Console Output

```
> vec1 = c(1,2,3)
> vec2 = c("R","Scilab","Java")
> vec3 = c("For prototyping",
+         "For prototyping","For Scaleup")
> df = data.frame(vec1,vec2,vec3,
+ stringsAsFactors = F)
> # Now trying the same manipulation
> df[3,3]= "Others"
> print(df)
  vec1  vec2      vec3
1    1    R For prototyping
2    2 Scilab For prototyping
3    3   Java      Others
>
> |
```