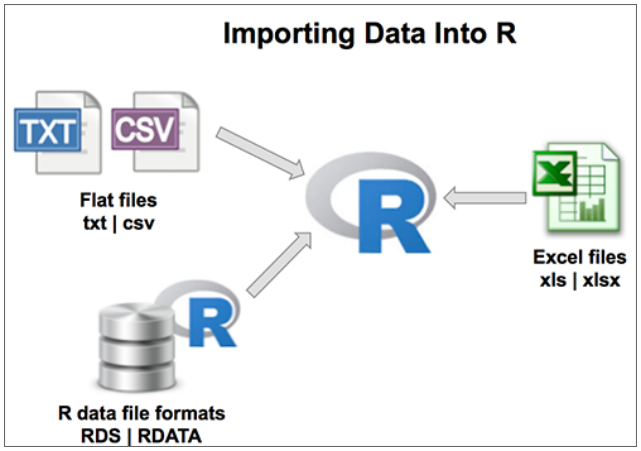
Open in Colab

WEEK 2: IMPORTING DATA

This week we'll cover importing data from common formats and 3 methods to programmatically listing and reading files from subfolders.



Courtesy of **STHDA**

```
Raw Data/
├── sub1/
│   ├── sub1_rest.xlsx
│   ├── sub1_trial1.xlsx
│   ├── sub1_trial2.xlsx
│   └── ...
├── sub2/
│   ├── sub2_rest.xlsx
│   ├── sub2_trial1.xlsx
│   ├── sub2_trial2.xlsx
│   └── ...
├── sub3/
│   ├── sub3_rest.xlsx
│   ├── sub3_trial1.xlsx
│   ├── sub3_trial2.xlsx
│   └── ...
└── ...
```

CLEAR THE ENTIRE WORKSPACE

In [83]:

```
rm(list=ls())
```

LOAD REQUIRED LIBRARIES

In [85]:

```
ReqdLibs = c("here", "purrr", "readxl", "chron",
                        "ggplot2", "ggthemes", "dplyr")
invisible(lapply(ReqdLibs, library, character.only = TRUE))
```

THEME DEFAULTS

This is only for the Jupyter Notebook so you the figures axes are larger

In [86]:

```
thm <- theme(strip.text.x = element_text(size = 20, face = "bold"),
             strip.text.y = element_text(size = 20, face = "bold"),
             legend.text = element_text(size = 16, face = "bold"),
             legend.position = "top",
             legend.title = element_text(size = 16, face = "bold"),
             title = element_text(size = 14, face = "bold"),
             text = element_text(colour = "black", size = 18),
             plot.title = element_text(colour = "black", size = 22, face = "bold"), #
nolint
             axis.ticks.length = unit(0.3, "cm"),
             axis.line = element_line(colour = "black", linewidth = 0.85),
             axis.ticks = element_line(colour = "black", linewidth = 0.85),
             axis.text = element_text(colour = "black", size = 24),
             axis.title = element_text(size = 25))
```

RETRIEVE THE FOLDER PATH TO THE DATA FILES, A.K.A, THE *ROOT FOLDER*

Note that this section is slightly different from the R project folder because the Jupyter notebook is outside teh R project folder and is read within the root folder `ReproRehab_Bootcamp` so we are going to need to direct it to the `Materials/Week 2/R project` folder.

In [87]:

```
folder_path = here("Materials/Week 2/R project", "Raw Data")
# output the folder name
print(folder_path)
# make sure it exists
dir.exists(folder_path)
```

```
[1] "/Users/rinivarghese/Documents/My Documents/JHU/Lab/Professional Develo
pment/ReproRehab/02_Bootcamp/ReproRehab_Bootcamp/Materials/Week 2/R projec
t/Raw Data"
```

TRUE

WHAT SUBFOLDERS AND FILES ARE WITHIN THE ROOT FOLDER? LET'S CHECK.

OK, once we have the *'folder path'* to the root folder, i.e., `Raw Data` , it is important to know exactly what the structure within is.
In our case, the structure looks something like this:

```
Raw Data/
├── sub1/
│   ├── sub1_rest.xlsx
│   ├── sub1_trial1.xlsx
│   ├── sub1_trial2.xlsx
│   └── ...
│
├── sub2/
```

```
| ├── sub2_rest.xlsx
| ├── sub2_trial1.xlsx
| ├── sub2_trial2.xlsx
| └── ...
├── sub3/
| ├── sub3_rest.xlsx
| ├── sub3_trial1.xlsx
| ├── sub3_trial2.xlsx
| └── ...
└── ...
```

In [88]:

```
# options(warn=0)
subfolder_path = here(folder_path,'Sub1')

files.test=list.files(subfolder_path)
files.test

#Let's read in one file to see how ugly the data are
temp0 = read_excel(here(subfolder_path,files.test[1]),)
head(temp0,20)

dim(temp0)
```

'Sub1_rest.xlsx' · 'Sub1_trial1.xlsx' · 'Sub1_trial2.xlsx' · 'Sub1_trial3.xlsx' · 'Sub1_trial4.xlsx' · 'Sub1_trial5.xlsx' · 'Sub1_trial6.xlsx'

```
New names:
• `` -> `...3`
• `` -> `...6`
• `` -> `...9`
• `` -> `...11`
```

ID code:	11	...3	Test number:	88	...6	Ba press.
<chr>	<chr>	<lgl>	<chr>	<chr>	<lgl>	
Last name:	SUBJECT	NA	Test date:	6/21/2017	NA	Temp (degr
First name:	NO1	NA	Test time:	10:16	NA	Hum
Sex:	M	NA	N. of steps:	53	NA	(degr
Age:	26	NA	Duration (hh:mm:ss):	00:02:59	NA	H flo
Height (in):	72.834645669291334	NA	BSA (m^2):	2.1000404912839743	NA	
Weight (lb):	189.59770013580487	NA	BMI (Kg/m^2):	25.127830533235937	NA	BTP
Notes:	NA	NA	HR max (bpm):	194	NA	BT
NA	NA	NA	NA	NA	NA	UN (
NA	NA	NA	NA	NA	NA	(Ko
NA	NA	NA	NA	NA	NA	V
NA	NA	NA	NA	NA	NA	
NA	NA	NA	NA	NA	NA	
NA	NA	NA	NA	NA	NA	F
NA	NA	NA	NA	NA	NA	F
NA	NA	NA	NA	NA	NA	
NA	NA	NA	NA	NA	NA	
NA	NA	NA	NA	NA	NA	MVV
NA	NA	NA	NA	NA	NA	MVV
NA	NA	NA	NA	NA	NA	
NA	NA	NA	NA	NA	NA	

55 · 129

SO, AS YOU SEE ABOVE, THE DATA ARE QUITE MESSY

we will modify the code so we only import a certain range Let's just do the first 5 rows where the data is in long format

rows where the data is in long format

In [89]:

```
temp=suppressMessages(read_excel(here(subfolder_path,files.test[1]),range =
cell_cols("J:O")))
head(temp,10)
# the first two rows are header-like information so remove it
temp=temp[-c(1,2),-2]
head(temp)
dim(temp)
```

A tibble: 10 × 6

t	...2	Rf	VT	
<chr>	<dbl>	<chr>	<chr>	
hh:mm:ss	NA	b/min	l	l
0	NA	NA	NA	
00:00:02	2	18.518518518518519	0.75485305361151733	13.978760252065
00:00:05	3	20.979020979020977	0.55389892987980249	11.620257270205
00:00:08	3	18.237082066869302	0.70588961229617564	12.873366789595
00:00:10	2	25.531914893617021	0.78239498935139695	19.97604228131
00:00:13	3	21.978021978021978	0.49575484331783432	10.895710842150
00:00:16	3	18.927444794952681	0.78749534782174502	14.90527472217
00:00:19	3	20.477815699658706	0.64264516726385934	13.159969295505
00:00:22	3	20.066889632107021	0.56817993359677721	11.401604018664

A tibble: 6 × 5

t	Rf	VT	VE
<chr>	<chr>	<chr>	<chr>
00:00:02	18.518518518518519	0.75485305361151733	13.978760252065134
00:00:05	20.979020979020977	0.55389892987980249	11.620257270205645
00:00:08	18.237082066869302	0.70588961229617564	12.873366789595908
00:00:10	25.531914893617021	0.78239498935139695	19.97604228131226
00:00:13	21.978021978021978	0.49575484331783432	10.895710842150205
00:00:16	18.927444794952681	0.78749534782174502	14.90527472217814

53 · 5

NOW WE DO THIS ITERATIVELY.

We go to our root folder `Raw Data` then we loop through all the subjects' folders within it to repeat the steps described above.

CREATE A LIST OF ALL THE FOLDER NAMES WITHIN THE ROOT FOLDER

CREATE A LIST OF ALL THE FOLDER NAMES WITHIN THE ROOT FOLDER

In [90]:

```
dir.list = dir(folder_path)
dir.list
```

'Sub1' · 'Sub10' · 'Sub11' · 'Sub12' · 'Sub13' · 'Sub2' · 'Sub3' · 'Sub4' · 'Sub5' · 'Sub6' · 'Sub7' · 'Sub8' · 'Sub9'

CREATE AN EMPTY DATA.FRAME THAT CAN ACCOMMODATE ANY VARIABLE TYPE

`data.frame(list())` : Converts the empty list into a data frame. Lists can accommodate any data type, i.e., numeric, string, characters, booleans etc.
Since the list is empty, the resulting data frame will have no columns and no rows. In other words, this command initializes an empty data frame, which can later be filled with data, but starts with no content (i.e., no columns and no rows).

In [91]:

```
data.all = data.frame(list())
```

METHOD 1: USING A `for` LOOP TO COMPILE THE MASTER DATASET

We use a for loop in R to iteratively "stack" individual participant data tables. For each participant, the steps are the samea as we did above.

In [ ]:

```
for(i in 1:length(dir.list)){
  files.import=list.files(here(folder_path,dir.list[i]))
  for(j in 1:length(files.import)){
    #Give me only the columns I need
    temp=suppressMessages(read_excel(here(folder_path,dir.list[i],files.import[j]),
                                     range = cell_cols("J:O")))
    #Remove the rows I don't need
    temp=temp[-c(1,2),-2]
    #Convert to numeric
    temp[,c(2:5)]=apply(temp[,c(2:5)],2,as.numeric)
    # Convert chron::times value (fraction of a day) to seconds:
    # times(...) returns a numeric representing days; multiply by 24 hours/day and
3600 seconds/hour to get seconds.
    temp$t <- as.numeric(times(temp$t)) * 24 * 3600
    #Assign Sub id
    temp$Sub=dir.list[i]
    #Assign trial id
    if(nchar(files.import[j])<16){
    temp$trial="rest"
    }else{


temp$trial=paste("trial",as.numeric(substr(files.import[j],nchar(files.import[j])-5,n
char(files.import[j])-5)))
    }
    # this final step is where the 'stacking' happens
```

```
    data.all=rbind(data.all,temp)

  }
}
```

LAST STEP! CHECK THE COMPILED DATASET DIMENSIONS

In [96]:

```
tail(data.all,10)
dim(data.all)
```

A tibble: 10 × 7

t	Rf	VT	VE	VO2	Sub	trial
<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<chr>	<chr>
416	24.89627	0.6294024	15.669769	532.97544	Sub9	trial 6
418	29.70297	0.7375331	21.906923	797.05276	Sub9	trial 6
420	26.90583	0.6038998	16.248426	548.15285	Sub9	trial 6
424	17.39130	0.8844276	15.381350	561.38412	Sub9	trial 6
426	22.55639	0.6416436	14.473163	509.72379	Sub9	trial 6
429	20.76125	0.5324928	11.055213	361.50629	Sub9	trial 6
431	26.78571	0.8946286	23.963267	873.85555	Sub9	trial 6
434	27.64977	0.5273923	14.582274	494.51991	Sub9	trial 6
435	34.09091	0.4304827	14.675545	394.51961	Sub9	trial 6
437	32.60870	0.1407739	4.590455	35.72638	Sub9	trial 6

30124 · 7

METHOD 2: USING `map_df` IN THE *PURRR* PACKAGE

`map_df` is used in two layers:

- **Outer Layer**: Iterates over each directory in ***dir.list***, listing files within each directory.
- **Inner Layer**: Processes each file in the current directory by: 1) reading the Excel file, 2) cleaning the data, 3) converting time to seconds, and 4) assigning identifiers.

This nesting efficiently combines results from all files into a single data frame, allowing for streamlined data aggregation from multiple directories.

In [ ]:

```
data.all <- map_df(dir.list, function(dir_name) {
  # List all files in the current directory
```

```
  files.import <- list.files(here(folder_path, dir_name))

  map_df(files.import, function(file_name) {
    # Read the Excel file
    temp <- read_excel(here(folder_path, dir_name, file_name), range =
cell_cols("J:O"))

    # Clean the data
    temp <- temp[-c(1, 2), -2]
    temp[, 2:5] <- apply(temp[, 2:5], 2, as.numeric)

    # Convert chron::times (fraction of a day) to seconds:
    # gives days; multiply by 24*3600 to convert to seconds (24 hours/day * 3600
sec/hour).
    temp$t <- as.numeric(times(temp$t)) * 24 * 3600

    # Assign Sub id
    temp$Sub <- dir_name

    # Assign trial id
    temp$trial <- ifelse(nchar(file_name) < 16, "rest", paste("trial",
as.numeric(substr(file_name, nchar(file_name) - 5, nchar(file_name) - 5))))

    return(temp)
  })
})

# Now data.all contains all the processed data
head(data.all)
dim(data.all)
```

A tibble: 6 × 7

t	Rf	VT	VE	VO2	Sub	trial
<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<chr>	<chr>
2	18.51852	0.7548531	13.97876	408.6987	Sub1	rest
5	20.97902	0.5538989	11.62026	304.7868	Sub1	rest
8	18.23708	0.7058896	12.87337	347.7043	Sub1	rest
10	25.53191	0.7823950	19.97604	614.8135	Sub1	rest
13	21.97802	0.4957548	10.89571	253.6431	Sub1	rest
16	18.92744	0.7874953	14.90527	437.0967	Sub1	rest

15062 · 7

METHOD 3: COMBINED POWER OF R LIBRARIES

We now use `map_df` with 3 other packages: `lubridate`, `dplyr`, and `tidyr` to do all the same things as above, but less hard-coded (more modular).

In [ ]:

```
library(lubridate)
library(tidyr)
library(dplyr)
```

In [19]:

```
data.all <- map_df(dir.list, function(dir_name) {
  # List all files in the current directory
  files.import <- list.files(here(folder_path, dir_name))

  map_df(files.import, function(file_name) {

    # Read the Excel file
    temp <- suppressMessages(read_excel(here(folder_path, dir_name, file_name), range
= cell_cols("J:O"))) %>%

      # Clean and transform the data
      slice(-c(1, 2)) %>%                    # Remove the first two rows
      select(-2) %>%                         # Remove the second column
      mutate(across(2:5, as.numeric),        # Convert columns 2 to 5 to numeric
             t = as.numeric(hms(t)),         # Convert time to seconds
             id = file_name) %>%             # Assign file name as identifier
      separate(id,into = c("Sub","trial","extn"),sep = "[_\\.]") %>%  # Now separate
the identifier into sub & trial
      select(!c("extn"))                     # Don't need file extensions in the
table!

    return(temp)
  })
})

# Now data.all contains all the processed data
head(data.all)
dim(data.all)
```

A tibble: 6 × 7

t	Rf	VT	VE	VO2	Sub	trial
<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<chr>	<chr>
2	18.51852	0.7548531	13.97876	408.6987	Sub1	rest
5	20.97902	0.5538989	11.62026	304.7868	Sub1	rest
8	18.23708	0.7058896	12.87337	347.7043	Sub1	rest
10	25.53191	0.7823950	19.97604	614.8135	Sub1	rest
13	21.97802	0.4957548	10.89571	253.6431	Sub1	rest
16	18.92744	0.7874953	14.90527	437.0967	Sub1	rest

15062 · 7

VISUALIZATION

Now that we have our compiled dataset, let's visualize it at different levels

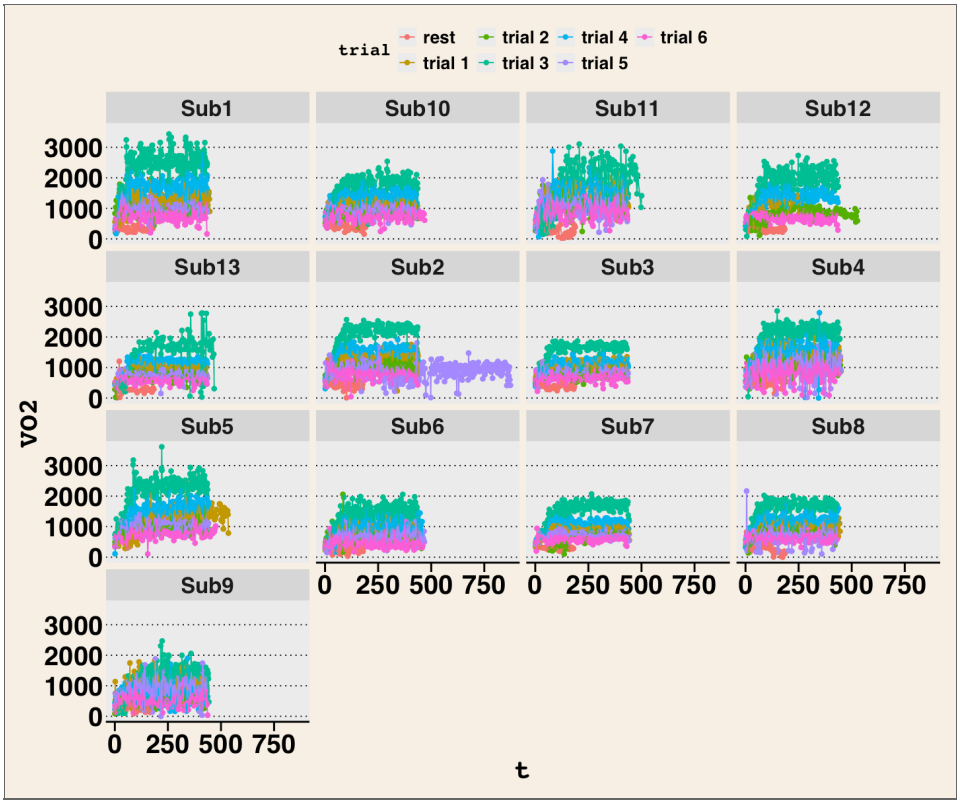
EACH TIMEPOINT FOR EACH TRIAL FOR PER PARTICIPANT

In [97]:

```
options(repr.plot.width = 12, repr.plot.height = 10)

#Visualize raw data by subject
```

```
ggplot(data.all,aes(x=t,y=VO2,color=trial))+
  geom_point()+
  geom_line()+
  facet_wrap(~Sub) + theme_wsj() + thm
```

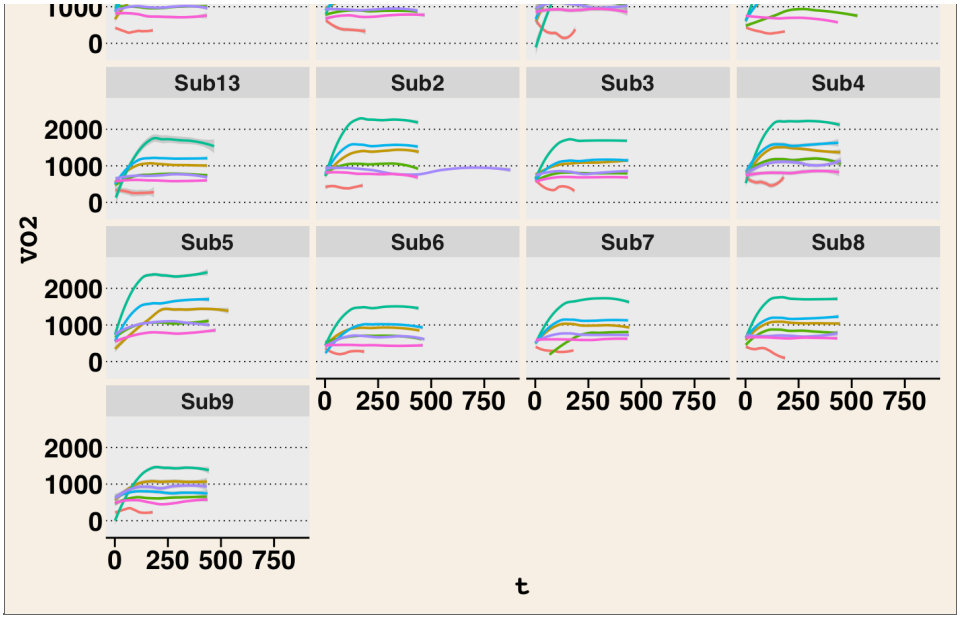


SMOOTHED TRACES ACROSS TIME FOR EACH TRIAL PER PARTICIPANT

In smoothing, we use a 'loess' procedure. This function is in-built within the geom_smooth function of ggplot. The 'loess' or 'lowess' procedure follows a kind of windowed smoothing procedure. It assigns weights to each point within the window, with closer points given higher weights, to calculate the smoothed value for each data point.

In [98]:

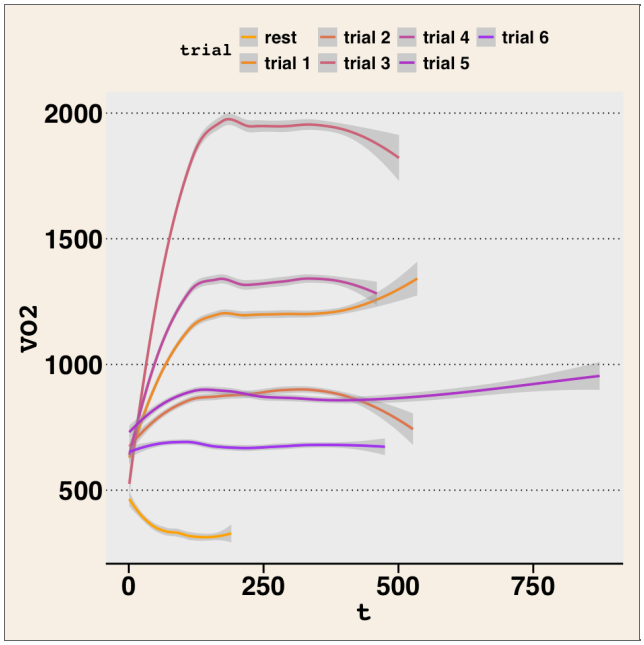
```
#Visualize data using loess by subject
ggplot(data.all,aes(x=t,y=VO2,color=trial))+
  geom_smooth(method = 'loess', formula = 'y~x')+
  facet_wrap(~Sub) + theme_wsj() + thm
```



EACH TRIAL AVERAGED ACROSS PARTICPANTS

In [99]:

```
options(repr.plot.width = 8, repr.plot.height = 8)
#create a color gradient
colfunc <- colorRampPalette(c("orange", "purple"))
#Visualize each trial across all participants
ggplot(data.all,aes(x=t,y=VO2,color=trial))+
  geom_smooth(method = 'loess', formula = 'y~x')+
  scale_colour_manual(values = c(colfunc(7))) + theme_wsj() + thm
```



THE END