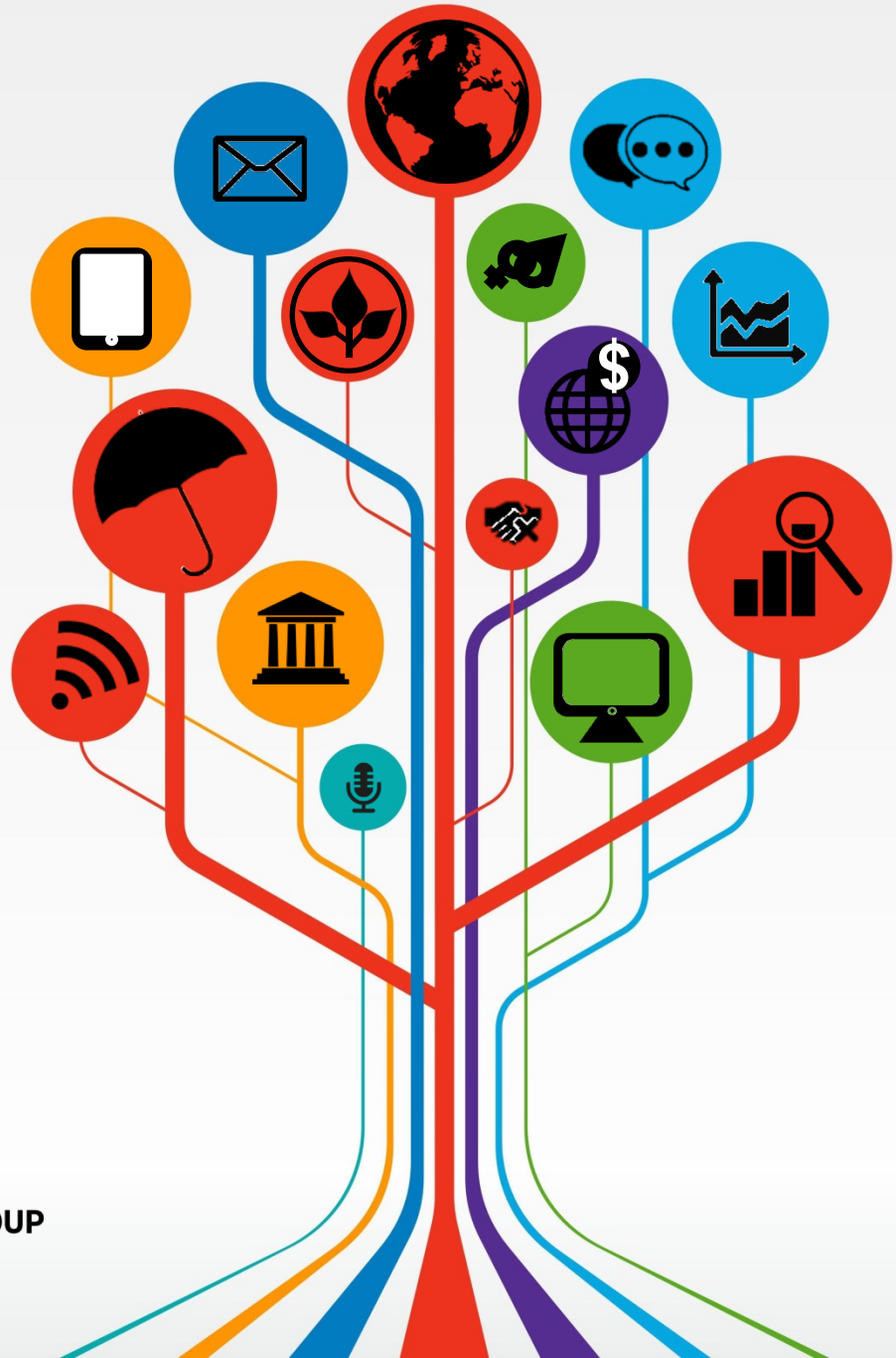
RA Technical Onboarding

DIME Best Practices in quantitate data work and documentation

Kristoffer Bjarkefur
Luiza Andrade
Benjamin Daniels

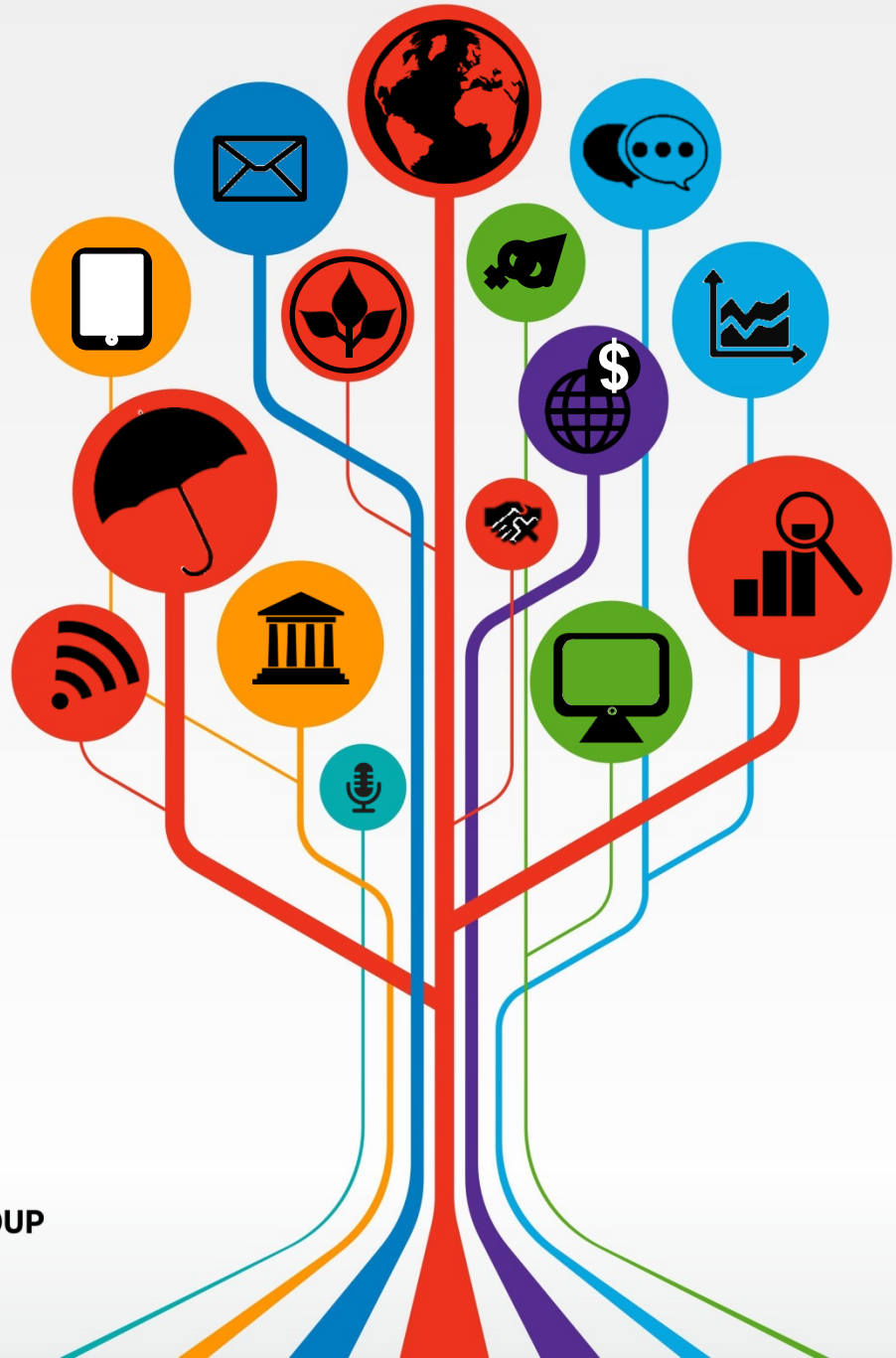
November 2017



i2i DIME
TRANSFORM DEVELOPMENT

WORLD BANK GROUP

How your code is a tool for the whole project?



i2i DIME
TRANSFORM DEVELOPMENT

WORLD BANK GROUP

Think reproducible

- All of DIME's data work should be reproducible. That we should be able to replicate every step of it
- This makes collaboration within teams and project turn over smoother, but is also extremely important as part of the research process
- The research output is not just a paper or report, but whole process. That means codes, data, documentation are just as important as the final text

Excel vs. Stata

Can I use Excel?

The main reason we use Stata or R

- In Excel you make changes directly to the data and save new versions of the data set

- In Stata and R your code is the instruction to get from the raw data to the final analysis. Instead of saving versions of the data, we save those instructions in a do-file or an R-script

Your code is an output

Many RAs come to DIME thinking that the code is a *mean to an end*, but my main point of this session is that your code is equally as much an *end* as a table or a written report is!

Coding difference
Academia vs. DECIE

- In academia:
  - Being correct is the only thing that matters
- At DIME
  - Correct is equally important as in academia
  - Past, current and future team members contribute to the same code, and therefore we need to standardize how we code, and focus on skills for coding as a team

Inductive vs. Deductive

- Learning how to code inductively:
  - I saw someone using this code for this – therefore I am using the same code
  - Copying undocumented code without understanding it
- Learning how to code deductively:
  - What are the reasons someone coded a task a certain way? Does those reasons apply to you as well?
  - Read documentation to understand what commands do and how they are intended to be used

File naming conventions

- Never, ever save any data files or do-files in the folder using any variation of these formats:
  - endline_data_new.csv, endline_data_old.csv
  - baseline_clean_v1.dta, baseline_clean_v2.dta etc.
  - Data_cleaning_June8.do
- Exception: Outputs (tables, graphs, documentation) can be good practice to date
- Syncing software (dropbox, box) have version control

Explicit and dynamic file paths

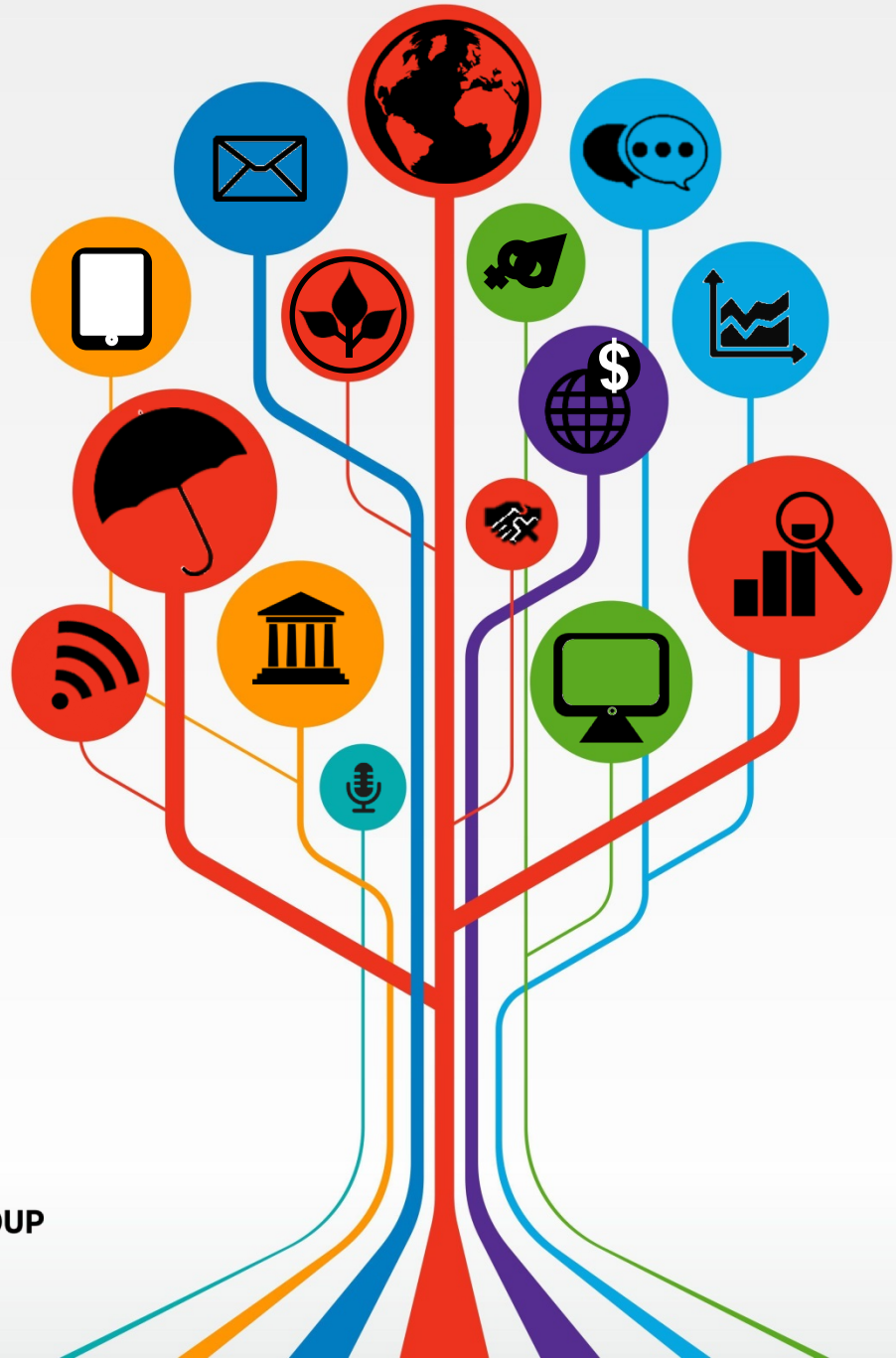
- Explicit vs. implicit file paths

```
*Explicit
use "C:\Users\wb462869\Box Sync\GAFSP Bangladesh\data\midline round 2\data\iapp_mlr2_panel.dta"

*Implicit
cd "C:\Users\wb462869\Box Sync\GAFSP Bangladesh\data\midline round 2\data\"
use iapp_mlr2_panel.dta
```

- Dynamic vs. Static file path

```
*Dynamic (and explicit)
global BOX_folder    "C:\Users\wb462869\Box Sync"
global MLR2_folder   "$BOX_folder\GAFSP Bangladesh\data\midline round 2"
use "$MLR2_folder\data\iapp_mlr2_panel.dta"

*Static (and explicit)
use "C:\Users\wb462869\Box Sync\GAFSP Bangladesh\data\midline round 2\data\iapp_mlr2_panel.dta"
```

Coding Styles



i2i DIME
TRANSFORM DEVELOPMENT

WORLD BANK GROUP

White Space. Stata does not distinguish between one empty space and many empty spaces, or one line break or many line breaks. It makes a big difference to the human eye and we would never share a Word document, an Excel sheet or a PowerPoint presentation without thinking about white space - although we call it formatting

Is this slide easy to read?

White Space

- Stata does not distinguish between one empty space and many empty spaces, or one line break or many line breaks

- It makes a big difference to the human eye and we would never share a Word document, an Excel sheet or a PowerPoint presentation without thinking about white space – although we call it formatting

Vertical lines

```
gen NoPlotDataBL = 0
replace NoPlotDataBL = 1 if c_plots_total_area >= .

gen NoHarvValueDataBL = 0
replace NoHarvValueDataBL = 1 if c_harv_value >= .

rename c_gross_yield c1_gross_yield
rename c_net_yield  c1_net_yield
rename c_harv_value c1_harv_value
rename c_total_earnings c1_total_earnings
rename c_input_spend c2_inp_total_spending
rename c_IAAP_harv_value c1_IAAP_harv_value
rename c_plots_total_area  c1_total_plotsize
rename  c1_cropPlotShare_??? c1_cropPlotShare_all_???

tempfile BL_append
save  `BL_append'
```

```
gen     NoPlotDataBL = 0
replace NoPlotDataBL = 1        if c_plots_total_area >= .

gen     NoHarvValueDataBL = 0
replace NoHarvValueDataBL = 1   if c_harv_value >= .

rename  c_gross_yield           c1_gross_yield
rename  c_net_yield             c1_net_yield
rename  c_harv_value            c1_harv_value
rename  c_total_earnings        c1_total_earnings
rename  c_input_spend           c2_inp_total_spending
rename  c_IAAP_harv_value       c1_IAAP_harv_value
rename  c_plots_total_area      c1_total_plotsize

rename  c1_cropPlotShare_???    c1_cropPlotShare_all_???

tempfile BL_append
save    `BL_append'
```

Vertical lines

```
*-create dummy for employed
gen employed = 1
replace employed = 0 if _merge == 2
label var employed "Person exists in employment data"
label define yesno 1 "yes" 0 "no"
label val employed yesno
```

```
*-create dummy for being employed
gen         employed = 1
replace     employed = 0 if _merge == 2
label var   employed "Person exists in employment data"
label def            yesno 1 "yes" 0 "no"
label val   employed yesno
```

Indentations

Makes code much more readable!

Use for preserve/restore, loops and all other commands with curly brackets

```
*Example 1
foreach jobCode of local jobCodes {
    gen hired_`jobCode' if (type_employment == `jobCode')
}


*Example 2
preserve
    keep     if treatment = 0
    tempfile controlsOnly
    save     `controlsOnly'
restore
```

```
* Renaming pond variables
    forvalues pondNo = 1/3 {
        forvalues fishNo = 1/6 {
            foreach varname in ish_code ish_name ish _free _free_source1_ _free_source_other        ///
                _buy _buy_tk _harvest _stage f_h_n f_h_u f_consume f_c_n f_c_u f_sold f_s_n         ///
                f_s_u f_s_tk f_current f_ct_n f_ct_u f_ct_tk m_avg_n m_avg_u m_h_n m_h_u           ///
                m_consume m_c_n m_c_u m_sold m_s_n m_s_u m_s_tk m_current m_ct_n m_ct_u m_ct_tk {

                rename pd`pondNo'_f`varname'`fishNo' d2_pd`pondNo'_f`fishNo'_`varname'

                if "`varname'" == "ish_code" {
                    rename d2_pd`pondNo'_f`fishNo'_ish_code d2_pd`pondNo'_f`fishNo'_code
                    label var d2_pd`pondNo'_f`fishNo'_code "Fish code for fish `fishNo' in pond `pondNo'"
                    label val d2_pd`pondNo'_f`fishNo'_code fish
                }

                *Removing trailing underscores
                if substr("`varname'",length("`varname'"),1) == "_" {
                    local varname = substr("`varname'",1,length("`varname'")-1)
                    rename d2_pd`pondNo'_f`fishNo'_`varname'_   d2_pd`pondNo'_f`fishNo'_`varname'
                }

                *Removing trailing underscores
                if substr("`varname'",1,1) == "_" {
                    local varname = substr("`varname'",2,.)
                    rename d2_pd`pondNo'_f`fishNo'__`varname'   d2_pd`pondNo'_f`fishNo'_`varname'
                }
            }
        *Droping variables that were used for the survey
        drop d2_pd`pondNo'_fish_select`fishNo' d2_pd`pondNo'_f`fishNo'_ish_name d2_pd`pondNo'_f`fishNo'_ish
        }
    }

*Dropping variables created when enumerator
*incorrectly added a 7th fish to a pond
    drop d2_pd?_fish*7
```

Break up long rows of code

One should never have to scroll horizontally to be able to read code

Two recommended ways to break up lines:

1. ///
   - Everything on the same row will be interpreted as a comment and the following row will be interpreted as if it was the same row
   - Good for breaking a long line of code into a few rows
2. # delimit ; - # delimit cr
   - Everything between *# delimit ;* and *# delimit cr* is executed as one line unless it is manually specified using a semicolon
   - Good for breaking a very long line of code into many rows

Example of row breaks

```
local cropcodes     101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116     ///
                    117 118 119 120 121 122 123 124 125 126 127 128 129 130 133 138     ///
                    139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154     ///
                    155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170     ///
                    171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186     ///
                    187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202     ///
                    203 204 205 207 208 209 210 211 212 213 214 215 216 217 218 219     ///
                    220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235     ///
                    236 237 238 239
```

```
* Code to export Graph
#delimit ;

histogram cons_bread, percent normal
  start(0) bin(10)
  bfcolor  ("178 0 80") blcolor("76 0 32")
  ytitle   ("Frequency")
  title    ("Food Security")
  xtitle   ("Number of days")
  subtitle ("Bread Consumption (All Sample)")
  note     ("Includes anyone in the HH who consumed bread last week");
  * Saving graph
  graph save "$outputs/Graph1_bread_consm.gph", replace;

#delimit cr;
```

Comments

- Two purposes:
  1. Tell the reader where you do what
  2. Tell the reader why you do what you do

Purpose 2 makes the difference between good and ok usage of comments, since purpose 1 can often be read from the code

Different types of comments

1. */* comment */*

   Used for long comments or to explain many lines of code in the following section

2. ** comment*

   Used to explain what happens on the following few rows

3. *// comment*

   Used to explain the same line of code

Example of comments

```
******************
***Preparation before randomization
******************

*Install commands needed:
ssc install tuples

*Specify settings that ensure stability across users
set more off
version  12.1                // Set version. Very important for randomization
set      seed 469302642      // randomly generated 9 digit number fom random.org

*Load data with villages to randomize
use "$subfolder/villages.dta", clear

*Local for number of treatments
local    num_treatments 6    // Change number of treatment groups here if needed
```

Macros: Local and Global

- You all know them – but why do both exist?
- Difference in scope – Use them appropriately according to scope
- Only define globals in the master do-file. Use locals everywhere else

Macros: naming convention

- Always give a local or a global a name where the reader can tell what it represents

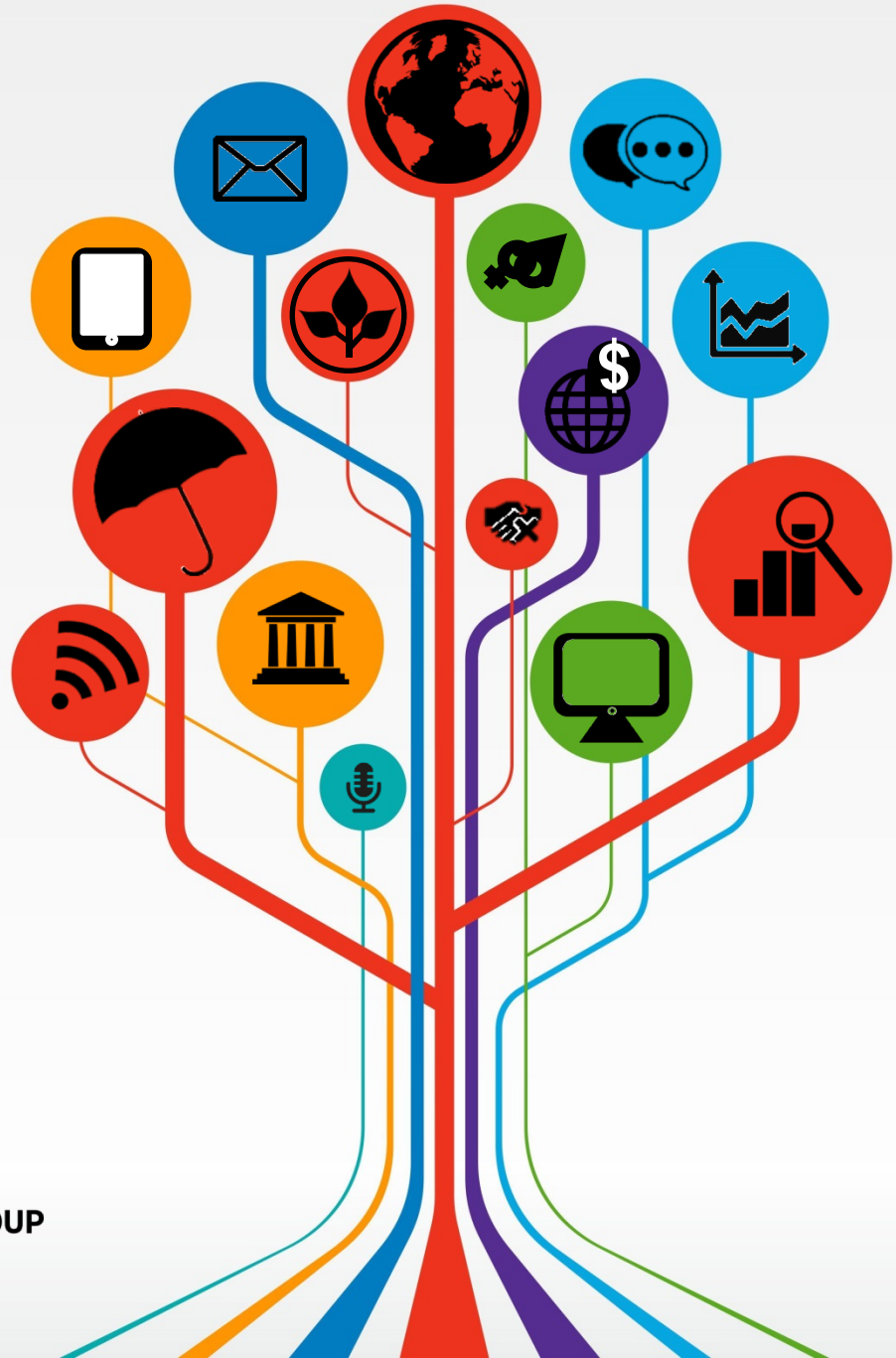
```
*Before
forvalues x = 1/3 {
    forvalues y = 1/10 {
        forvalues z = 1/6 {

            sum variable c1_harv_c`z'_s`x'_p`y'
        }
    }
}

*After
forvalues seasnum = 1/3 {
    forvalues plotnum = 1/10 {
        forvalues cropnum = 1/6 {

            sum variable c1_harv_c`cropnum'_s`seasnum'_p`plotnum'
        }
    }
}
```

Tips for usages of locals

Use locals tot shorten variable names and make them more explanatory

```
*Before
sum     c_harv_value                                if c_harv_q14_p`plotNum'_s`seasonNum' == 1
reg     c_harv_value fertilizer seedtype labor      if c_harv_q14_p`plotNum'_s`seasonNum' == 1



*After
local   grewRice     c_harv_q14_p`plotNum'_s`seasonNum'
sum     c_harv_value                                if `grewRice' == 1
reg     c_harv_value fertilizer seedtype labor      if `grewRice' == 1
```

Become more exact in your coding



i2i DIME
TRANSFORM DEVELOPMENT

WORLD BANK GROUP

Help files

- Get in the habit of using the help file as often as possible!
  - Even with familiar commands, always more to learn
- Help files are only summaries of the reference manual
  - coding practices, common mistakes, alternative approaches
- Access the reference manual by clicking:

```
[R] regress
```

Example of help file

Winsorizing a variable

winsor *varname* [**if** *exp*] [**in** *range*] , **generate**(*newvar*)
{ **p**(#) | **h**(#) } [{ **highonly** | **lowonly** }]

Options

generate(*newvar*) specifies the name of the new variable. It is a required option.

p(#) specifies the fraction of the observations to be modified in each tail. *p* should be greater than 0 and less than 0.5 and imply a value of *h* as just below.

h(#) specifies the number of the observations to be modified in each tail. *h* should be at least 1 and less than half the number of non-missing observations.

Just one of **p**() and **h**() should be specified.

highonly and **lowonly** specify that Winsorizing should be one-sided, referring only to the tail with the highest values or only to the tail with the lowest values, respectively. These options should not be specified together.

Syntax used in help files

- Varlist/varname
- Options
- […] – Optional
- { … | … } – Use either or
- <u>reg</u>ress – Shortest accepted abbreviation underlined
- Sometimes customized names are used in the syntax, those names are always explained below

Structure of help files

- All help-files follow the same structure. The most useful are:
  - Syntax: How to enter the command
  - Menu: Where to find it in drop down menu
  - Options: Syntax and explanation of options
  - Examples: Examples on how to specify the command
  - Saved results: First step to code Stata dynamically
- Many commands are related and share help file:
  - Drop/keep, decode/encode

Help file usage and coding knowledge