Introduction to R for Stata Users

02: Data Manipulation

Purushottam Mohanty
04 June 2021

overview

Data manipulation or data cleaning is an important aspect of any project and it is important to be well acquainted with all the tools provided by R to facilitate an easy transition from Stata to R.

While Stata has specific commands for specific data cleaning operations, R is more versatile and often the same operation can be performed in multiple ways through different packages.

In this slide deck I use `dplyr` and `tidyr` packages to perform data manipulation operations. The same tasks can also be performed using `baseR` functions but they are often more complicated and cumbersome.

I also provide equivalent commands in Stata where possible to help easily understand R functions.

tidyverse

The `tidyverse` package in R is a composite of many different data manipulation, functional programming and data visualization packages. Check documentation about tidyverse, dplyr and tidyr for more information.

```
# only loads the primary tidyverse packages
library(tidyverse)
tidyverse_packages()
```

```
##  [1] "broom"         "cli"           "crayon"        "dbplyr"
##  [5] "dplyr"         "dtplyr"        "forcats"       "googledrive"
##  [9] "googlesheets4" "ggplot2"       "haven"         "hms"
## [13] "httr"          "jsonlite"      "lubridate"     "magrittr"
## [17] "modelr"        "pillar"        "purrr"         "readr"
## [21] "readxl"        "reprex"        "rlang"         "rstudioapi"
## [25] "rvest"         "stringr"       "tibble"        "tidyr"
## [29] "xml2"          "tidyverse"
```

Packages like `ggplot2` for data visualization, `forcats` for handling categorical data and `stringr` for text manipulation enable so many possibilities. Packages within tidyverse can be separately loaded as well.

pipe %>% operator

Many different packages in R, including `tidyr` and `dplyr` follow the pipe `%>%` operator syntax which makes way for clean looking code and saves considerable time by not having to specify the dataframe name everytime.

```
# both lines are equivalent
df %>% filter(!continent == 'Europe') %>% group_by(continent, year) %>%
  summarize(mean_gdppc = mean(gdpPercap))
summarise(group_by(filter(df, !continent == 'Europe'), continent, year), mean_gdppc =
```

The first line can be read as, specifying the dataframe, filtering the rows and then grouping based on column names and the summarizing the gdpPercap variable. As you can see the first line is easier to read and logical in nature. With complicated and lengthy code, `%>%` operator becomes extremely handy.

dplyr | tidyr

create new columns (1/n)

In R, new columns can be created using `dplyr::mutate()` function.

```
df %>%
  mutate(pop_mn = pop / 1000000)
```

Note that `mutate()` creates a new column if dataframe `df` doesn't have a column with the specified namespace (i.e. `pop_mn`) or overwrites the existing column. So, `mutate()` is a substitute for Stata commands `generate` and `replace` depending on the namespace provided.

```
# Stata equivalent
generate pop_mn = pop / 1000000
replace pop_mn = pop / 1000000
```

create new column (2/n)

The original dataframe imported from the `gapminder` package.

```
df
```

```
## # A tibble: 1,704 x 6
##    country     continent  year lifeExp      pop gdpPercap
##    <fct>       <fct>     <int>   <dbl>    <int>     <dbl>
##  1 Afghanistan Asia       1952    28.8  8425333      779.
##  2 Afghanistan Asia       1957    30.3  9240934      821.
##  3 Afghanistan Asia       1962    32.0 10267083      853.
##  4 Afghanistan Asia       1967    34.0 11537966      836.
##  5 Afghanistan Asia       1972    36.1 13079460      740.
##  6 Afghanistan Asia       1977    38.4 14880372      786.
##  7 Afghanistan Asia       1982    39.9 12881816      978.
##  8 Afghanistan Asia       1987    40.8 13867957      852.
##  9 Afghanistan Asia       1992    41.7 16317921      649.
## 10 Afghanistan Asia       1997    41.8 22227415      635.
## # … with 1,694 more rows
```

create new column (3/n)

New column `pop_mn` has been created.

```
df %>%
  mutate(pop_mn = pop / 1000000)
```

```
## # A tibble: 1,704 x 7
##    country     continent  year lifeExp      pop gdpPercap pop_mn
##    <fct>       <fct>     <int>   <dbl>    <int>     <dbl>  <dbl>
##  1 Afghanistan Asia       1952    28.8  8425333      779.   8.43
##  2 Afghanistan Asia       1957    30.3  9240934      821.   9.24
##  3 Afghanistan Asia       1962    32.0 10267083      853.  10.3
##  4 Afghanistan Asia       1967    34.0 11537966      836.  11.5
##  5 Afghanistan Asia       1972    36.1 13079460      740.  13.1
##  6 Afghanistan Asia       1977    38.4 14880372      786.  14.9
##  7 Afghanistan Asia       1982    39.9 12881816      978.  12.9
##  8 Afghanistan Asia       1987    40.8 13867957      852.  13.9
##  9 Afghanistan Asia       1992    41.7 16317921      649.  16.3
## 10 Afghanistan Asia       1997    41.8 22227415      635.  22.2
## # … with 1,694 more rows
```

create new column (4/n)

Let's overwite `pop_mn` where `pop_mn` is log of population. Note how the old `pop_mn` column is overwritten with log population values. Also, see how the `%>%` function allows us to perform multiple operations on the same dataframe.

```
df %>%
  mutate(pop_mn = pop / 1000000) %>%
  mutate(pop_mn = log(pop))
```

```
## # A tibble: 1,704 x 7
##    country     continent  year lifeExp      pop gdpPercap pop_mn
##    <fct>       <fct>     <int>   <dbl>    <int>     <dbl>  <dbl>
##  1 Afghanistan Asia       1952    28.8  8425333      779.   15.9
##  2 Afghanistan Asia       1957    30.3  9240934      821.   16.0
##  3 Afghanistan Asia       1962    32.0 10267083      853.   16.1
##  4 Afghanistan Asia       1967    34.0 11537966      836.   16.3
##  5 Afghanistan Asia       1972    36.1 13079460      740.   16.4
##  6 Afghanistan Asia       1977    38.4 14880372      786.   16.5
##  7 Afghanistan Asia       1982    39.9 12881816      978.   16.4
##  8 Afghanistan Asia       1987    40.8 13867957      852.   16.4
##  9 Afghanistan Asia       1992    41.7 16317921      649.   16.6
## 10 Afghanistan Asia       1997    41.8 22227415      635.   16.9
## # … with 1,694 more rows
```

filter rows (1/n)

In R rows can be filtered using `dplyr::filter()` function.

```
df %>%
  filter(continent == "Europe")
```

This keeps only those rows for which continent is Europe. Similarly, for keeping all rows outside continent Europe one can use the "!=" logical operation. Multiple conditions can also be specified using,

```
df %>%
  filter(continent %in% c("Europe", "Africa"))
df %>%
  filter(continent == "Europe" | continent == "Africa")
```

Both lines above are equivalent. Note how the `%in%` syntax makes code much more concise and readable.

```
# Stata equivalent
keep if continent == "Europe"
keep if continent == "Europe" | continent == "Africa"
```

filter rows (2/n)

Only keeps the rows where continent is Europe.

```
df %>%
  filter(continent == "Europe")
```

```
## # A tibble: 360 x 6
##    country continent  year lifeExp     pop gdpPercap
##    <fct>   <fct>     <int>   <dbl>   <int>     <dbl>
##  1 Albania Europe     1952    55.2 1282697     1601.
##  2 Albania Europe     1957    59.3 1476505     1942.
##  3 Albania Europe     1962    64.8 1728137     2313.
##  4 Albania Europe     1967    66.2 1984060     2760.
##  5 Albania Europe     1972    67.7 2263554     3313.
##  6 Albania Europe     1977    68.9 2509048     3533.
##  7 Albania Europe     1982    70.4 2780097     3631.
##  8 Albania Europe     1987    72   3075321     3739.
##  9 Albania Europe     1992    71.6 3326498     2497.
## 10 Albania Europe     1997    73.0 3428038     3193.
## # … with 350 more rows
```

filter rows (3/n)

Keeps all rows where continent is either Europe or Africa. (check no. of rows)

```
df %>%
  filter(continent %in% c("Africa", "Europe"))
```

```
## # A tibble: 984 x 6
##    country continent  year lifeExp     pop gdpPercap
##    <fct>   <fct>     <int>   <dbl>   <int>     <dbl>
##  1 Albania Europe     1952    55.2 1282697     1601.
##  2 Albania Europe     1957    59.3 1476505     1942.
##  3 Albania Europe     1962    64.8 1728137     2313.
##  4 Albania Europe     1967    66.2 1984060     2760.
##  5 Albania Europe     1972    67.7 2263554     3313.
##  6 Albania Europe     1977    68.9 2509048     3533.
##  7 Albania Europe     1982    70.4 2780097     3631.
##  8 Albania Europe     1987    72   3075321     3739.
##  9 Albania Europe     1992    71.6 3326498     2497.
## 10 Albania Europe     1997    73.0 3428038     3193.
## # … with 974 more rows
```

filter columns (1/n)

In R, columns can be filtered using the `dplyr::select()` function.

```
df %>%
  select(continent)
df %>%
  select(country, continent)
```

Similarly, columns can be dropped using a `-` sign before the column name

```
df %>%
  select(-country, -continent)
```

Unlike Stata, for R both keeping and dropping columns is done using the same function. Also, one can use `select()` to order columns with or without dropping columns using the `select(columnA, columnB, everything())` syntax. The `everything()` function selects all columns not specified in `select()`.

```
# Stata equivalent
keep country continent
drop country continent
```

filter columns (2/n)

```
df %>%
  dplyr::select(continent, country)
```

```
## # A tibble: 1,704 x 2
##    continent country
##    <fct>     <fct>
##  1 Asia      Afghanistan
##  2 Asia      Afghanistan
##  3 Asia      Afghanistan
##  4 Asia      Afghanistan
##  5 Asia      Afghanistan
##  6 Asia      Afghanistan
##  7 Asia      Afghanistan
##  8 Asia      Afghanistan
##  9 Asia      Afghanistan
## 10 Asia      Afghanistan
## # … with 1,694 more rows
```

filter columns (3/n)

```
df %>%
  dplyr::select(-continent, -country)
```

```
## # A tibble: 1,704 x 4
##     year lifeExp      pop gdpPercap
##    <int>   <dbl>    <int>     <dbl>
##  1  1952    28.8  8425333      779.
##  2  1957    30.3  9240934      821.
##  3  1962    32.0 10267083      853.
##  4  1967    34.0 11537966      836.
##  5  1972    36.1 13079460      740.
##  6  1977    38.4 14880372      786.
##  7  1982    39.9 12881816      978.
##  8  1987    40.8 13867957      852.
##  9  1992    41.7 16317921      649.
## 10  1997    41.8 22227415      635.
## # … with 1,694 more rows
```

order rows (1/n)

In R, rows can be ordered is ascending or descending order using the `dplyr::arrange()` function.

```
df %>%
  arrange(year, gdpPercap) # ascending order
df %>%
  arrange(desc(year), desc(gdpPercap)) # descending order
```

```
# Stata equivalent
sort year gdpPercap
gsort -year -gdpPercap # using gtools
```

order rows (2/n)

```
df %>%
  arrange(year, gdpPercap) # ascending order
```

```
## # A tibble: 1,704 x 6
##    country           continent  year lifeExp       pop gdpPercap
##    <fct>             <fct>     <int>   <dbl>     <int>     <dbl>
##  1 Lesotho           Africa     1952    42.1    748747      299.
##  2 Guinea-Bissau     Africa     1952    32.5    580653      300.
##  3 Eritrea           Africa     1952    35.9   1438760      329.
##  4 Myanmar           Asia       1952    36.3  20092996      331
##  5 Burundi           Africa     1952    39.0   2445618      339.
##  6 Ethiopia          Africa     1952    34.1  20860941      362.
##  7 Cambodia          Asia       1952    39.4   4693836      368.
##  8 Malawi            Africa     1952    36.3   2917802      369.
##  9 Equatorial Guinea Africa     1952    34.5    216964      376.
## 10 China             Asia       1952    44   556263527      400.
## # … with 1,694 more rows
```

order rows (3/n)

```
df %>%
  arrange(desc(year), desc(gdpPercap)) # descending order
```

```
## # A tibble: 1,704 x 6
##    country          continent  year lifeExp       pop gdpPercap
##    <fct>            <fct>     <int>   <dbl>     <int>     <dbl>
##  1 Norway           Europe     2007    80.2   4627926    49357.
##  2 Kuwait           Asia       2007    77.6   2505559    47307.
##  3 Singapore        Asia       2007    80.0   4553009    47143.
##  4 United States    Americas   2007    78.2 301139947    42952.
##  5 Ireland          Europe     2007    78.9   4109086    40676.
##  6 Hong Kong, China Asia       2007    82.2   6980412    39725.
##  7 Switzerland      Europe     2007    81.7   7554661    37506.
##  8 Netherlands      Europe     2007    79.8  16570613    36798.
##  9 Canada           Americas   2007    80.7  33390141    36319.
## 10 Iceland          Europe     2007    81.8    301931    36181.
## # … with 1,694 more rows
```

distinct (1/n)

In R, duplicates can be removed from a dataframe using `dplyr::distinct()` function. The `.keep_all = T` option ensures that all columns are kept after removal of the duplicates.

```
df %>%
  distinct(.keep_all = T)
```

Duplicates can also be removed from a particular column using,

```
df %>%
  distinct(country, .keep_all = T)
```

The `distinct()` can also be used to view unique values for a column(s).

```
df %>%
  distinct(country) # all countries present in df
```

```
# Stata equivalent
duplicates drop # for all rows
duplicates drop country, force # for a column
duplicates report country
```

distinct (2/n)

```
df %>%
  distinct(.keep_all = T)
```

```
## # A tibble: 1,704 x 6
##    country     continent  year lifeExp      pop gdpPercap
##    <fct>       <fct>     <int>   <dbl>    <int>     <dbl>
##  1 Afghanistan Asia       1952    28.8  8425333      779.
##  2 Afghanistan Asia       1957    30.3  9240934      821.
##  3 Afghanistan Asia       1962    32.0 10267083      853.
##  4 Afghanistan Asia       1967    34.0 11537966      836.
##  5 Afghanistan Asia       1972    36.1 13079460      740.
##  6 Afghanistan Asia       1977    38.4 14880372      786.
##  7 Afghanistan Asia       1982    39.9 12881816      978.
##  8 Afghanistan Asia       1987    40.8 13867957      852.
##  9 Afghanistan Asia       1992    41.7 16317921      649.
## 10 Afghanistan Asia       1997    41.8 22227415      635.
## # … with 1,694 more rows
```

distinct (3/n)

```
df %>%
  distinct(country, .keep_all = T)
```

```
## # A tibble: 142 x 6
##    country     continent  year lifeExp      pop gdpPercap
##    <fct>       <fct>     <int>   <dbl>    <int>     <dbl>
##  1 Afghanistan Asia       1952    28.8  8425333      779.
##  2 Albania     Europe     1952    55.2  1282697     1601.
##  3 Algeria     Africa     1952    43.1  9279525     2449.
##  4 Angola      Africa     1952    30.0  4232095     3521.
##  5 Argentina   Americas   1952    62.5 17876956     5911.
##  6 Australia   Oceania    1952    69.1  8691212    10040.
##  7 Austria     Europe     1952    66.8  6927772     6137.
##  8 Bahrain     Asia       1952    50.9   120447     9867.
##  9 Bangladesh  Asia       1952    37.5 46886859      684.
## 10 Belgium     Europe     1952    68    8730405     8343.
## # … with 132 more rows
```

distinct (4/n)

```
df %>%
  distinct(country)
```

```
## # A tibble: 142 x 1
##    country
##    <fct>
##  1 Afghanistan
##  2 Albania
##  3 Algeria
##  4 Angola
##  5 Argentina
##  6 Australia
##  7 Austria
##  8 Bahrain
##  9 Bangladesh
## 10 Belgium
## # … with 132 more rows
```

summarize (1/n)

Summarize operates with `mutate()` at the backend and creates a new dataframe with specified columns based on the statistics specified. Note that `summarize()` and `summarise()` are equivalent to each other and can be interchanged.

```
df %>%
  summarize(mean_pop = mean(pop), median_gdppc = median(gdpPercap))
```

Summarise can be performed across multiple columns in combination with the `across()` function.

```
df %>%
  summarize(across(c("pop","gdpPercap"), mean))
```

```
# Stata equivalent
collapse (mean) pop (median) gdpPercap
```

summarize (2/n)

```
df %>%
  summarize(mean_pop = mean(pop), median_gdppc = median(gdpPercap))
```

```
## # A tibble: 1 x 2
##    mean_pop median_gdppc
##       <dbl>        <dbl>
## 1 29601212.        3532.
```

```
df %>%
  summarize(across(c("pop","gdpPercap"), mean))
```

```
## # A tibble: 1 x 2
##         pop gdpPercap
##       <dbl>     <dbl>
## 1 29601212.     7215.
```

```
# Stata equivalent
bysort continent year: egen mean_pop = mean(pop)
bysort continent year: egen median_gdppc = median(gdpPercap)
collapse (mean) mean_pop median_gdppc, by(continent year)
```

group operations (1/n)

Grouped row operations can be performed using the `dplyr::group_by()` function.

```
df %>%
  group_by(continent, year) %>% # grouping variables
  mutate(mean_pop = mean(pop)) # group wise operation to perform
```

The `dplyr` pipe operation implies that dataset is grouped as long as a separate `ungroup()` function is provided. It's a healthy practice to provide `ungroup()` function after the end of the grouped operation to avoid confusion.

```
df %>%
  group_by(continent, year) %>% # grouping variables
  mutate(mean_pop = mean(pop)) %>% # group wise operation to perform
  ungroup() %>% # dataframe is now ungrouped
  mutate(mean_gdppc = mean(gdpPercap)) # ungrouped operation
```

```
# Stata equivalent
bysort continent year: egen mean_pop = mean(pop)
egen mean_gdppc = mean(gdpPercap)
```

group operations (2/n)

```
df %>%
  group_by(continent, year) %>% # grouping variables
  summarize(mean_pop = mean(pop), mean_gdppc = mean(gdpPercap)) # group wise operatio
```

```
## # A tibble: 60 x 4
## # Groups:   continent [5]
##    continent  year  mean_pop mean_gdppc
##    <fct>     <int>     <dbl>      <dbl>
##  1 Africa     1952  4570010.      1253.
##  2 Africa     1957  5093033.      1385.
##  3 Africa     1962  5702247.      1598.
##  4 Africa     1967  6447875.      2050.
##  5 Africa     1972  7305376.      2340.
##  6 Africa     1977  8328097.      2586.
##  7 Africa     1982  9602857.      2482.
##  8 Africa     1987 11054502.      2283.
##  9 Africa     1992 12674645.      2282.
## 10 Africa     1997 14304480.      2379.
## # … with 50 more rows
```

reshape - long to wide (1/n)

Dataframes can be transformed from long to wide using the `tidyr::pivot_wider()` function. Here's how the original dataframe looks.

```
df
```

```
## # A tibble: 1,704 x 6
##    country     continent  year lifeExp      pop gdpPercap
##    <fct>       <fct>     <int>   <dbl>    <int>     <dbl>
##  1 Afghanistan Asia       1952    28.8  8425333      779.
##  2 Afghanistan Asia       1957    30.3  9240934      821.
##  3 Afghanistan Asia       1962    32.0 10267083      853.
##  4 Afghanistan Asia       1967    34.0 11537966      836.
##  5 Afghanistan Asia       1972    36.1 13079460      740.
##  6 Afghanistan Asia       1977    38.4 14880372      786.
##  7 Afghanistan Asia       1982    39.9 12881816      978.
##  8 Afghanistan Asia       1987    40.8 13867957      852.
##  9 Afghanistan Asia       1992    41.7 16317921      649.
## 10 Afghanistan Asia       1997    41.8 22227415      635.
## # … with 1,694 more rows
```

reshape - long to wide (1/n)

Transforming from long to wide form.

```
df %>%
  pivot_wider(names_from = year, values_from = c("lifeExp", "pop", "gdpPercap"))
```

country	continent	lifeExp_1952	lifeExp_1957	lifeExp_1962	lifeExp_1967	lifeExp_1972	lifeExp_1977	lifeExp_1982	lifeExp_1987	lifeExp_1992	lifeExp_1997	life
Afghanistan	Asia	28.801	30.33200	31.99700	34.02000	36.08800	38.43800	39.854	40.822	41.674	41.763	
Albania	Europe	55.230	59.28000	64.82000	66.22000	67.69000	68.93000	70.420	72.000	71.581	72.950	
Algeria	Africa	43.077	45.68500	48.30300	51.40700	54.51800	58.01400	61.368	65.799	67.744	69.152	
Angola	Africa	30.015	31.99900	34.00000	35.98500	37.92800	39.48300	39.942	39.906	40.647	40.963	
Argentina	Americas	62.485	64.39900	65.14200	65.63400	67.06500	68.48100	69.942	70.774	71.868	73.275	
Australia	Oceania	69.120	70.33000	70.93000	71.10000	71.93000	73.49000	74.740	76.320	77.560	78.830	
Austria	Europe	66.800	67.48000	69.54000	70.14000	70.63000	72.17000	73.180	74.940	76.040	77.510	
Bahrain	Asia	50.939	53.83200	56.92300	59.92300	63.30000	65.59300	69.052	70.750	72.601	73.925	
Bangladesh	Asia	37.484	39.34800	41.21600	43.45300	45.25200	46.92300	50.009	52.819	56.018	59.412	
Belgium	Europe	68.000	69.24000	70.25000	70.94000	71.44000	72.80000	73.930	75.350	76.460	77.530	
Benin	Africa	38.223	40.35800	42.61800	44.88500	47.01400	49.19000	50.904	52.337	53.919	54.777	
Bolivia	Americas	40.414	41.89000	43.42800	45.03200	46.71400	50.02300	53.859	57.251	59.957	62.050	

reshape (cont.) - wide to long

merging columns

merging rows

data.table