PharmaSUG 2018 - Paper BB-22

Expand Your Skills from SAS® to R with No Complications

Andrii Korchak, Experis Clinical Solutions, Kyiv, Ukraine

ABSTRACT

Learning a new language is rarely a quick and simple process. Learning word by word from the very beginning requires a great deal of energy and time. Is there a way not to start all over again? Is it possible to use the knowledge of your native language, and just pull up a little to begin to communicate?

In this article, we will outline the main similarities and differences between SAS and R, and show how to start programming in R relying on your SAS skills. In particular, we will program DATA steps and procedure calls to perform data analysis in both languages, and present results by creating a table and graphs. We will learn how to recreate some popular SAS operations in R and make use of a few R-specific ones.

Use your SAS dictionary to break through a language barrier!

INTRODUCTION

The usage of R in pharmaceutical industry is growing at a rapid rate. R made a wide selection of statistical procedures and data visualization instruments available for everyone. However, such a huge variety of packages can be confusing for beginners.

Many SAS programmers hesitate to start programming in R because they cannot decide what to learn first. In this paper, we will show how to do the usual SAS tasks by the three most essential packages: R BASE, DPLYR, and GGPLOT2.

BASE package is the key element of R environment.

The instruments of BASE are universal and can do a significant part of work, but if you want to solve a more sophisticated problem, you will need to use additional equipment.

DPLYR is the next tool to learn for beginners. It provides a universal mechanism for data manipulation that makes it possible to do with data almost all that you can in the SAS DATA step. This package provides comprehensive tools for data manipulation.

GGPLOT2 package based on Leland Wilkinson’s famous book "The Grammar of Graphics." It is a universal tool for data visualization that allows doing a publication-ready graphics in a fast and efficient way. Moreover, you can create custom plots by adding graphic elements one on top of another. This approach makes GGPLOT2 quite easy and straightforward to use.

BASE SAS AND R

The materials in the following chapters require the reader to be familiar with basic syntax of SAS and R languages.

For SAS you can find interactive courses on Learn / Training section of SAS Support Cite.[5]
For R you can learn basics from the book “R for Data Science” by Garrett Grolemund and Hadley Wickham[6] or take an interactive course “SWIRL, Learn R, in R”[7].

SAS examples are done in SAS STUDIO UNIVERSITY EDITION 2.5 9.4M4.
R scripts run in R Studio Version 1.0.153, R version 3.4.2 (2017-09-28).

DATA MANIPULATION IN SAS AND R

When you are working with data, you are repeating three main processes:

1. Reading the data.
2. Data Analysis and Transformation.
3. Data Visualization and Reporting.

In this article, we will have a look at how these procedures are carried out in both SAS and R.

1. READING THE DATA

You can get data from a number of sources. Those can be external files (XLS, CSV, and TXT), databases, or/and data from other statistical environments (SPSS, Stata, etc.)

1.1 Reading Delimited data

Sometimes it might be necessary to read Delimited data. In SAS, you can use INPUT STATEMENT.
In R, you can do it by `read.table()`. This function has many arguments to accommodate for your input. There is a list of functions with pre-specified default parameters, which allow you to solve typical problems:

- *`read.csv()`*
- *`read.csv2()`*
- *`read.delim()`*
- *`read.delim2()`*

Function names speaks for themselves. The difference between `read.csv()` and `read.csv2()` in default delimiter. It is "." dot for `read.csv()` function and "," comma for `read.csv2()`. The same approach applies to `read.delim()` and `read.delim2()` accordingly.

In the example below, we are reading text data into the Data Set / Data Frame.
Example 1.1 shows how to read data from external file in SAS and R.

SAS Language

```
data dm;
   input SUBJID AGE POFL $;
   datalines;

1 23 Y
2 45 Y
3 61 N
;
run;
proc print data=dm;
run;
```

Result of Execution:

Data from WORK.DM

Obs	SUBJID	AGE	POFL
1	1	23	Y
2	2	45	Y
3	3	61	N

R Language

```
dm<-read.table(header=TRUE,
    stringsAsFactors = FALSE,
         text ="
 SUBJID AGE POFL
 1 23 Y
 2 45 Y
 3 61 N"
)
dm
str(dm)
```

Result of Execution:

```
> dm
 SUBJID AGE POFL
1     1    23    Y
2     2    45    Y
3     3    61    N
 str(dm)
'data.frame':3 obs. of 3 variables:
 $ SUBJID: int  1 2 3
 $ AGE   : int  23 45 61
 $ POFL  : chr  "Y" "Y" "N"
```

Example 1.1 Reading a text input into the Data Set / Data Frame.

Note, that if you specify `header=TRUE` option in `read.table()`, the column names will be read from the text file. Column types assigned automatically.

1.2 Reading data from External Files

In this article, we will be using data from OSMI Mental Health in Tech Survey.
The data is available as a CSV file. You can download it from the OSMI site[1] or GitHub page with additional materials[3].

Firstly, we need to upload the data into our system.

Example 1.2 shows how to read data from external file in SAS and R:

SAS Language

```
  filename csv_srs "/folders/ mental-heath-in-tech-2016_20161114.csv";
  proc import datafile = csv_srs
                         out = raw_ds
                        dbms = csv replace ;
            guessingrows = MAX ;
            getnames = no  ;
            datarow=2;
  run;
```

Result of Execution:

```
NOTE: 1433 records were read from the infile CSV_SRS.
      The minimum record length was 219.
      The maximum record length was 3504.
 NOTE: The data set WORK.RAW_DS has 1433 observations and 63 variables.
 NOTE: DATA statement used (Total process time):
      real time          0.02 seconds
      cpu time            0.02 seconds
```

R Language

```
   setwd("E:/R/folder with data");
   path <-  "mental-heath-in-tech- 2016_20161114.csv"
   csv_file<-read.csv(path, stringsAsFactors = FALSE)
  # Checking number of Rows and Columns
   dim(csv_file)
   # Reading names of First 3 columns
   names(csv_file)[1:3]
```

Result of Execution:

```
> dim(csv_file)
[1] 1433   63
> names(csv_file)[1:3]
[1] "Are.you.self.employed."
[2] "How.many.employees.does.your.company.or.organization.have."
[3] "Is.your.employer.primarily.a.tech.company.organization."
```

Example 1.2 Reading data from external files in SAS and R.

PROC IMPORT allows to read data from various data souses. In R the same action can be done by `read.csv()` function. In the SAS part, we specified `getnames = no` option because in CSV file column`s names are longer than 32 symbols. It leads to column names VAR1 to VAR63. To obtain at least variable labels we developed a work around. Further information about it can be found in the Additional materials [3].

R doesn’t have such problems and we read meaningful names. Option `stringsAsFactors = FALSE`

allows to get DATA FRAME with strings instead of factors.
So now we know how to get data into the environment and can move on to the second chapter.

2. TRANSFORMING DATA

SAS in one of the most convenient languages for working with data. Combinations of the DATA steps and Procedures Calls easily allow to manipulate data.

Typical data flow looks as follows:

```
 *** Do not run ***;
proc sort data = row_data;
   by BY_VAR1 descending BY_VAR2;
   where STAT ~="NOT DONE";
run;

data output_ds ( drop = ( VAR1 ) );
   set row_data (  keep = ( BY_VAR1  BY_VAR2 VAR1 )
                  rename = ( LOG_VAR1 = AVAL )
                  );
   by BY_VAR1 descending BY_VAR2;
   if first.BY_VAR1 then BLFL = "Y" ;
   LOG_VAR1 = log( VAR1 );
run;
 *** End Do not run ***;
```

However, R also has something to offer. It is a DPLYR package.

It provides a universal mechanism for data manipulation that enables you to operate data almost like in the SAS DATA step. This package provides comprehensive tools for data manipulation, and can work as a substitute for the following SAS procedures:

- PROC SORT – allows sort data with `arrange()` function.
- SET / MERGE statements – allows combine data vertically and horizontally by functions such `bind_rows(), left_join(),inner_join()`, etc.
- BY statement – `group_by()` function allow to process data group wise.
- IF / WHERE – `filter()` function can be used to pick only necessary records.
- KEEP, DROP, RENAME – `select() and rename()` allow to work with columns of data frame.
- New Variables – `mutate() and summarise()` making possible adding new columns.

To install a package you can use R function `install.packages()`.
The first argument is the name of the package.
The next command below installs the package DPLYR from CRAN repository:

```
     install.packages("dplyr")
```

To start using the package it has to be loaded with `library()` function.

```
     library("dplyr")
```

Now all is ready to work.
Before starting, it is always a good idea to check documentation. It can be done by `help()` function.

```
     help("dplyr")
```

2.1 Pipe operator

Let`s start with one thing that doesn't do any data manipulation, but makes the programmer`s life easier – it is Pipe Operator %>%.
We are already familiar with assignments operators = and <- . The Pipe is different.
It places the left argument onto the first position of the right function e.g. x %>% f(,y) is the same as f(x,y).

You shouldn't worry about the operator`s length - there is a short cut in R Studio: Ctrl+Shift+M, Cmd+Shift+M.

Let`s do a quick check on our OSMI dataset for wrong AGE values. It can be done by combination of `unique()` and `sort()` R functions:

Example 2.1.1 Getting a vector of unique values:

```
# variable # 56 is Age
names(csv_file)[56]
sort(unique(csv_file[,56]))
```

```
> names(csv_file)[56]
[1] "What.is.your.age."
> sort(unique(csv_file[,56]))
 [1]   3 15  17  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33
[19]  34 35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51
[37]  52 53  54  55  56  57  58  59  61  62  63  65  66  70  74  99 323
```

Example 2.1.1 Output from sort(unique()) execution.

The same task can be accomplished by the Pipe Operator:

Example 2.1.1 Getting vector of unique values with Pipe:

```
csv_file[,56] %>% unique %>% sort
```

```
> csv_file[,56] %>% unique %>% sort
 [1]   3 15  17  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33
[19]  34 35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51
[37]  52 53  54  55  56  57  58  59  61  62  63  65  66  70  74  99 323
```

Example 2.1.2 Output from sort(unique()) execution with Pipe operator.

What if we want to use left expression not on the first position?
If it is required to use the left expression differently from the first position, you can take into consideration the following:
put a dot '.' on position, where you want the Pipe operator to put a value.

For example R function `rep(x,y)` repeats its first argument(x) y times.
We can call this function two times to figure out how '.' works with the Pipe.
Example 2.1.3 changing the position of the Pipe Argument:

```
1 %>% rep(2)
1 %>% rep(2,.)
```

```
> 1 %>% rep(2)
[1] 1 1
> 1 %>% rep(2,.)
[1] 2
```

Output 2.1.3 Output from the Pipe operator with a dot.

As you can see, the first time '1' is used as first argument of `rep()` function. This value is repeated two times. The second call shows, that '2' used as the first argument. Its value is repeated once.

Now we can go ahead and re-create SAS DATA step by DPLYR functions.

2.2 Drop, Keep, and Rename your variables

In SAS selecting variables can be done by keep=, drop=, and rename= DATA step options or corresponding statements.

In R `select()` and `rename()` functions can be used to do the same thing.
The first argument always is a data frame; all other arguments are unquoted columns.

To select variables (keep in SAS terms) from the data frame specify a name or list of names separated by comma`s. To drop a column use minus operator '-' before the column`s name.
In addition, we can use the following list of functions to select records more efficiently:

- *`starts_with()`* – equivalent of SAS ':' operator;
- *`ends_with()`* - ends with a prefix;
- *`contains()`* – contains a string;
- *`matches()`* – name matches regular expression;
- *`num_range()`* – equivalent of SAS '-' operator.

Example 2.2.1 is based on the dm data set. This dataset was created in Example 1.1.

Example 2.2.1.1, usage of Drop, Keep, and Rename options:

Keep option

SAS Language	R Language

```
data dm_k;
   set dm( keep = SUBJID AGE);
run;
title "Data from &syslast.";
proc print;
run;
```
	```
	# keep only SUBJID, AGE
	df_k <- dm %>%
	 select( c("SUBJID","AGE"))
	df_k
	```

Result of Execution:

Data from WORK.DM_K

Obs	SUBJID	AGE
1	1	23
2	2	45
3	3	61

```
> df_k <- dm %>%
    select(c("SUBJID","AGE"))
> df_k
  SUBJID AGE
1   1   23
2   2   45
3   3   61
```

Example 2.2.1.2 Using `rename()` function to change column`s name.

Renaming variables

SAS Language	R Language

```
*** Renaming variables ***;
data dm_rename ;
  set dm ;
  rename SUBJID = PATNUM;
run;
title "Data from &syslast.";
proc print data = dm_rename; run;
```
	```
	dm_rename <- dm %>%
	     rename("PATNUM" = SUBJID )
	dm_rename
	```

Result of Execution:

Data from WORK.DM_RENAME

Obs	PATNUM	AGE	POFL
1	1	23	Y
2	2	45	Y
3	3	61	N

```
> dm_rename <- dm %>%
     rename("Column1" = SUBJID )
> dm_rename
 PATNUM AGE POFL
1    1      23   Y
2    2      45   Y
3    3      61   N
```

From Output 2.2.1.2 we can understand that `rename()` function in R different from SAS version.
In R you should specify a new name first, and only after it an old one. Also, `rename()` function does not drop columns, that were not specified.(The same as in SAS).

2.3 Filtering Data

Selecting records from data set is a common task. In SAS you can do it by **where** and **if** statements. DPLYR package allows to do the same with `filter()` function.
`filter()` – first argument is a data frame; all other arguments are conditions. Note that, If you list more than one condition all of them will be treated as one condition separated by AND operator(&).

Example 2.3.1 Filtering Observations

SAS Language

```
* Where statement ;
data dm_i;
  set dm;
  where POFL = "Y" and AGE > 21;
run;
```

R Language

```
dm_i <- dm %>%
  filter(POFL=="Y", AGE > 21)
dm_i
```

Result of Execution:

Data from WORK.DM_I

Obs	SUBJID	AGE	POFL
1	1	23	Y
2	2	45	Y

Result of Execution:

```
> dm_i <- dm %>%
   filter(POFL=="Y", AGE > 21)
> dm_i
  SUBJID AGE POFL
1      1 23   Y
2      2 45   Y
```

Example 2.3.1 shows the same results. SAS programmer must remember that in R check for equality is done by == operator.

2.4 Creating new variables

To create a new variable in SAS you just need to initialize it.

In R it can be done by `mutate()` function. The first argument is data frame; all others are names of new variables with assigned expressions.
`mutate()` considers functions as vectoring(e.g. input and output vectors have the same dimensions).

Example 2.4.1 Creating new variables:

SAS Language

```
*** Creating new Variables ***;

data df_new_var;

  set dm nobs=n_obs;

  N_ROWS = n_obs;
  col4 = ifc(_N_ >= n_obs/2,
       ">=n/2","<n/2","missing");

  USUBJID ="A00-"||put(SUBJID,3.-l);
  drop SUBJID ;

run;

title "Data from &syslast.";
proc print;
run;
```

R Language

```
df_new_var <- dm %>%
 mutate(
  N_ROWS = n(),
 col4 = if_else(
    row_number()>= n()/2,
    ">=/2",
    "<n/2",
    "missing"
 ),
  USUBJID = paste0("A00-",SUBJID)
) %>% select(-SUBJID)
df_new_var
```

Result of Execution:

Result of Execution:

SAS Language	R Language
Data from WORK.DF_NEW_VAR	`> df_new_var`

Obs	AGE	POFL	N_ROWS	col4	USUBJID
1	23	Y	3	<n/2	A00-1
2	45	Y	3	>=n/2	A00-2
3	61	N	3	>=n/2	A00-3

```
> df_new_var
  AGE POFL N_ROWS   col4 USUBJID
1  23    Y      3    <n/2   A00-1
2  45    Y      3  >= n/2   A00-2
3  61    N      3  >= n/2   A00-3
```

In Example 2.4.1, we are creating two new variables: n_2 and col4, and modifying an existing one: SUBJID.
 As you can see, R allows to refer to the aggregate functions such as n() and refer to windowing functions such as `row_number()`(the current row number).

2.5 Sort the Data

Sorting Data sets based on values of one or more columns is also a common task.
In SAS it can be done by the SORT procedure. In DPLYR package we have similar functionality by `arrange()`.

`arrange()` function takes a data frame as first argument, after which you can specify a list of variable names separated by a comma. If you want to sort out in DESCENDING order you should put the variable name inside of `desc()` function.

Example 2.5.1 Sorting data

SAS Language	R Language

`*** Sorting data ;`
`*** save result into new data set;`
`proc sort data=dm out= dm_sort;`
`    by POFL;`
`run;`
`title "Data from &syslast.";`
`proc print;`
`run;`
	`## Sorting data with arrange`
	`## saving into new data frame`
	`dm_sort <- dm %>%`
	`  arrange( POFL )`
	`dm_sort`

Result of Execution:

Data from WORK.DM_SORT

Obs	SUBJID	AGE	POFL
1	3	61	N
2	1	23	Y
3	2	45	Y

```
> dm_sort <- dm %>% arrange(POFL )
> dm_sort
  SUBJID AGE POFL
1      3  61    N
2      1  23    Y
3      2  45    Y
```

2.6 By Groups

Often, while working with data you need to separate records by groups.
In SAS, the BY statement allows you to do so. This statement can be used in SAS DATA step and SAS Procedures.

DRLYR function *group_by()* serves the same purpose.
The first argument is Data Frame, the others are variables used for grouping.

There are two types of functions:

- Summary functions - taking a vector and returning one value
  e.g. *max(), first(), mean(),* etc.
- Windowing functions - taking a vector and returning a vector of the same length
  e.g. *lead(), lag(), row_number(),* etc.

After grouping you can use both Summary and Windowing functions within groups.

In DRLYR `mutate()` works with functions in the same way as with windowing.
Functions called within `summarise()` produce outputs as Summary functions.
After computing you should remove grouping from data by operating ungroup() function.

In Example 2.6.1 we are working with the new data VS.
It consists of four variables: Subject Identifier, Test1 result, Test2 result and Period.

We are going to create a mean test score within each period, after which we will calculate average value for each Subject.

Example 2.6.1 Computing by Group Statistics:

SAS Language

```
*** Grouping Variables ***;
data vs;
  input  SUBJID HT WT PERIOD ;
  datalines;
001 180 .  1
001 181 77 2
002 173 85 1
002 173 83 2
;
run;
title "Data from &syslast.";
proc print;
run;
```

R Language

```
# Grouping variables and
# statistics in R;
vs <- read.table(header = TRUE,
       stringsAsFactors = FALSE,
           text = "
                SUBJID HT WT PERIOD
                001 180 NA 1
                001 181 77 2
                002 173 85 1
                002 173 83 2");
  print( vs );
```

Output From SAS code:

Data from WORK.VS

Obs	SUBJID	HT	WT	PERIOD
1	1	180	.	1
2	1	181	77	2
3	2	173	85	1
4	2	173	83	2

Output From R code:

```
> print( vs )

 SUBJID  HT WT PERIOD
1     1 180 NA      1
2     1 181 77      2
3     2 173 85      1
4     2 173 83      2
```

SAS Language

```
proc means data = vs;
  var  HT WT ;
  by SUBJID;
  output out = vs_s;
run;
```

R Language

```
vs_s <- vs %>%
  group_by( SUBJID ) %>%
  summarise_at(
       vars(HT,WT),
        funs( sum(!is.na(.)),
             mean(., na.rm=TRUE),
               sd(., na.rm=TRUE),
              min(. ,na.rm=TRUE),
              max(. ,na.rm=TRUE))
            )%>% ungroup()
vs_s
```

Output From SAS code:

Data from WORK.VS
The MEANS Procedure
SUBJID=1

Variable	N	Mean	Std Dev	Minimum	Maximum
HT	2	180.5000000	0.7071068	180.0000000	181.0000000
WT	1	77.0000000	.	77.0000000	77.0000000

SUBJID=2

Variable	N	Mean	Std Dev	Minimum	Maximum
HT	2	173.0000000	0	173.0000000	173.0000000
WT	2	84.0000000	1.4142136	83.0000000	85.0000000

Output From R code:

```
> vs_s
# A tibble: 2 x 11
  SUBJID HT_sum WT_sum HT_mean WT_mean   HT_sd       WT_sd     HT_min WT_min HT_max WT_max
   <int>  <int>   <int>   <dbl>   <dbl>   <dbl>       <dbl>     <dbl>  <dbl>   <dbl>  <dbl>
1     1     2       1     180.5     77    0.7071068   NaN        180     77     181     77
2     2     2       2     173.0     84    0.0000000   1.414214   173     83     173     85
```

Example 2.6.1 Computing by Group Statistics
`group_by()` R function allows to use statistics group wise(as SAS BY statement).
After the `ungrpoup()` call you can use summary functions across all the values.
`summarise()` produces one row for each group.

2.7 LAG in SAS and LAG in R

The following example shows how to work with windowing functions in SAS and R.
Example 2.7.1 Windowing functions:

SAS Language

```
  *** LAG function ***;
  data vs_lag;
      set vs ( keep = SUBJID HT);
      by SUBJID;
      LAG_HT = lag( HT );
    if first.SUBJID then LAG_HT = .;
  run;
  title "Lag in SAS,Data from syslast.";
  proc print;
  run;
```

R Language

```
  # lag and lead functions;
  vs_lag_lead <- vs %>%
    select(SUBJID, TEST1) %>%
    group_by( SUBJID) %>%
    mutate(
      LAG  = lag( HT ),
      LEAD = lead( HT )
      ) %>% ungroup()
  vs_lag_lead
```

Output From SAS code:

Obs	SUBJID	HT	LAG_HT
1	1	180	.
2	1	181	180
3	2	173	.
4	2	173	173

Output From R code:

```
# A tibble: 4 x 4
  SUBJID   HT  LAG  LEAD
   <int> <int> <int> <int>
1    1   180   NA   181
2    1   181  180    NA
3    2   173   NA   173
4    2   173  173    NA
```

Note that *lag()* and *lead()* in R are working within groups by default;
And there is no lead() function in SAS.

2.8 Combining data horizontally

Joins another important part of data manipulation. Both SAS and R languages allow combining data horizontally.DPLYR package has a few functions for different merges:
`inner_join(), left_join(), right_join(),`

`full_join(), semi_join(), anti_join().`

The function names speak for themselves. The first two arguments are data sets, the third one is the joining key. If the third argument is missing – natural join is performed.
Example 2.8.1 Merge Data sets

SAS Language	**R Language**

```
data advs ;
    merge vs( in = in_x )
           dm( in = in_y );
    by SUBJID ;
    if in_x and in_y ;
     * inner join;
    keep HT WT SUBJID POFL ;
run;
title  "Combined Data Set,";
title2 "Data from &syslast.";
proc print;
run;
```
	```
	advs <- vs %>%
	 inner_join(y = dm,
	            by =
	    c("SUBJID" = "SUBJID")) %>%
	 select(HT, WT, SUBJID, POFL)

	advs
	```

Output From SAS code:	Output From R code:

Data from WORK.ADVS
 	
Obs | SUBJID | HT | WT | POFL
1 | 1 | 180 | . | Y
2 | 1 | 181 | 77 | Y
3 | 2 | 173 | 85 | Y
4 | 2 | 173 | 83 | Y
	```
	> advs
	  SUBJID POFL  HT WT
	1      1    Y 180 NA
	2      1    Y 181 77
	3      2    Y 173 85
	4      2    Y 173 83
	```

It is evident that, DPLYR can merge data sets vertically as SAS does.
If you need to combine two data sets without creating new rows (e.g. using set operators) you can use the following functions:

- *intersect(x1, x2),*
- *union(x1, x2)*
- *setdiff(x1, x2).*

`intersect()` keeps rows that only appear in both data sets, `union()`keeps all unique rows, and `setdiff()` keeps only rows from the first data set, which are missing from the second one.

ADDITIONAL MATERIALS

Presenting results of your analysis may be the most important part of the process.
You can carry this out in the form of Table or Graph. In the Additional Materials, we have analyzed some demographic information from OSMI Survey. We have used information about Gender, Age, Tech/IT Role and attitude to Working Remotely.

The example of Graph with Age distribution done by SAS SGPLOT and R GGPLOT2 is also included in the Additional materials.

CONCLUSION

SAS DATA step is a core of the SAS Language. It provides a wide range of methods for data transformation and manipulation.

R has many packages. Even checking the whole list will take a lot of time, but to do the main part of tasks you do not need to be an expert in all of them.

DPLYR has simple syntax, and that`s why it is suitable for new R users. On the other hand, it can be a substitute to SAS tools for data manipulation

REFERENCES

1. Open Sourcing Mental Illness - Changing how we talk about mental health in the tech community - Stronger Than Fear. Available at https://osmihelp.org/research
2. Creative Commons Attribution-ShareAlike 4.0 International. Available at https://creativecommons.org/licenses/by-sa/4.0/legalcode
3. Additional materials: GitHub with codes and raw data https://github.com/korchakandrey/Expand-Your-Skills-From-SAS-To-R
4. Package 'dplyr' documentation. https://cran.r-project.org/web/packages/dplyr/dplyr.pdf
5. SAS e-Learning materials available at https://support.sas.com/edu/elearning.html
6. R for Data Science by Garrett Grolemund and Hadley Wickham http://r4ds.had.co.nz/
7. SWIRL, Learn R, in R free course http://swirlstats.com/

RECOMMENDED READING

- R packages for data science available at https://www.tidyverse.org/
- SAS 9.4 Product Documentation® available at http://support.sas.com/documentation/94/
- Introduction to dplyr available at https://cran.r-project.org/web/packages/dplyr/vignettes/dplyr.html
- GGPLOT2 documentation available at https://cran.r-project.org/web/packages/ggplot2/ggplot2.pdf
- RStudio Cheat Sheets available at https://www.rstudio.com/resources/cheatsheets/

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Andrii Korchak

Experis Clinical Solutions / Intego Group LLC

+1 (407) 512 1006 (Ext. 2435)
Andrii.Korchak@intego-group.com
www.intego-group.com