

Calculating the Comorbidity Index with user designed flexibility using SAS and R

Ting Shi, Merck & Co., Inc., Rahway, NJ, USA
Geonhyeok Jeong, University of North Carolina, Chapel Hill, NC

Abstract

Comorbidity refers to the presence of two or more medical conditions or disease in an individual at the same time. These conditions can either occur simultaneously or one can develop after the other. Comorbidity can complicate the diagnosis, treatment, and management of these conditions, as they may interact with each other and affect the overall health and well-being of the individual. Comorbidity indexes are commonly used in medical research and clinical practice to assess the burden of comorbid conditions in individuals. The calculation of comorbidity typically involves using a comorbidity index or scoring system to assign weights or scores to different comorbid conditions. This paper describes a custom SAS macro and R function which gives user the flexibility to customize the different comorbidity indexes and weights for the calculation.

Introduction

Some of the commonly used comorbidity indexes include: Charlson Comorbidity Index (CCI), Elixhauser Comorbidity Index (ECI), NCI Comorbidity Index, Washington University Head and Neck Comorbidity Index (WUHNCI), Simplified Comorbidity Score for a Patient with Non-Small Cell Lung Cancer (NSCLC) and others. All these measures involve using a specified list of medical conditions identified from diagnoses codes and assign the weights for the comorbidities.

The published SAS codes or R packages have the diagnosis codes and weights embedded in the program which doesn't give the user flexibility to define their own sets of codes and weights. This paper illustrates a function design using SAS or R to give the user flexibility when calculating the comorbidity index for the following specifications:

- definition of the medical condition using user-defined medical code lists;
- weight assignment for the different medical condition;
- version of the medical coding system used;
- algorithm for calculating the comorbidity index (DEYO, QUAN or any user defined value);
- string matching type when extracting diagnosis codes;
- inclusion or exclusion of specified diagnosis codes.

The designed function can handle the calculation of multiple user defined Comorbidity indexes.

Overview of CCI calculation

Let's review the steps to acquire the Charlson Comorbidity Index (CCI) score. The CCI is a method of categorizing comorbidities of patients based on prognostic comorbid condition diagnoses. Each comorbid condition has an associated weight (from 1 to 6). The sum of (weight * occurrence of each comorbid condition) results in a single CCI score for a patient. Higher scores are related to higher mortality. A score of zero indicates that no comorbid condition is identified.

The weights and conditions are defined in the original CCI publication (A new method of classifying prognostic comorbidity in longitudinal studies: Development and validation. *Journal of Chronic Diseases*, 40(5), 373-383).

Function Description

Inputs:

- 1) input dataset at the patient level with ICD code list, example below; it needs to have a column for patient_id and one column for ICD diagnosis code

patient_id	diagnosis_code	diagnosis_code_system
F701073416F18	B95.2	ICD-10-CM
F701073416F18	C34.12	ICD-10-CM
F701073416F18	C34.12	ICD-10-CM
F701073416F18	C34.90	ICD-10-CM
F701073416F18	C34.92	ICD-10-CM
F701073416F18	D47.3	ICD-10-CM
F701073416F18	E16.2	ICD-10-CM
F701073416F18	E43	ICD-10-CM
F701073416F18	E43	ICD-10-CM
F701073416F18	E78.5	ICD-10-CM

- 2) input excel file comorb_index.xlsx which includes all the ICD code lists for the contributing medical conditions for comorbidity index calculation. The excel file can be customized by the user for study needs; keep the column names the same as the template below.

Parameters:

Column Name	Description	Default Value	Required (Y/N)	Valid Values	Comments
COM	Abbreviation of comorbidities	NA	Y	Text without space	The value from the column will be used as variable name in the output SAS dataset for the medical condition flag.
COMEDS	Description of the comorbidity from column COM	NA	N	Texts	
WGT	Weight for the specified comorbidity	1	Y	Number	
ICD	ICD code list for the comorbidity	NA	Y	ICD9 or ICD10 code	One code per row
VER	Version of the ICD code from column ICD	NA	Y	9 or 10	
ALG	Algorithm for calculating the comorbidity index	NA	N	DEYO, QUAN or any user defined value	
SEL	matching type for the specified ICD code (Blank: Exact Match, 1: Start with, 2: End with, 3: Included)	NA	N	It has 4 value forms: empty: search for exact match 1: Start with 2: End with 3: Included	
INEX	Flag to indicate if the ICD code needs to be included for calculation	NA	Y	1 OR missing: Included, 0: Excluded	

Macro / Function Parameters:

Parameter Name	Description	Default Value	Required (Y/N)	Valid Values	Comments
INPUT	Input dataset which includes the patientid and ICD code	NA	Y	Dataset name	
WHERE	where clause used to subset the input dataset	NA	N	For example, where diag_date>='01Jan2015'd	
ICDVAR	Column name from INPUT for the ICD code	NA	Y	Column name from INPUT, for example diagnosis_code	
PATID	Column name from INPUT for patient id	NA	Y	Column name from INPUT, for example patient_id	
INEXFILE	Path including file name for the input excel file comorb_index.xlsx	NA	Y	For example: /efs/analysis/oncology/study01/code/study_files/codelists/comorb_index.xlsx	
SHEET	Sheet from the INEXFILE to be imported	NA	Y	Sheet name from comorb_index.xlsx to be imported	
SUBSET	Subset condition of the imported sheet	NA	N	For example, where ALG='QUAN'	
OUTPUT	Output dataset name (required)	NA	Y	Dataset name	Output dataset with one patient per row with calculated comorbidity index and flags for the conditions

Programming code flow

In this section, we provide detailed code examples for calculating the Charlson Comorbidity Index (CCI) using both SAS and R. These examples illustrate the steps involved in importing datasets, matching patient IDs and ICD codes, and calculating CCI scores.

1) Input data and excel data manipulation

The first step is to check if the input data contains multiple columns of ICD codes. If so, these columns are consolidated into one using an array function in SAS or loop syntax in R. Additionally, any dots (.) in the ICD codes in the Excel file are removed using the **'compress'** function in SAS and **'str_remove'** in R.

SAS code	R code
<pre>%* Check if ICD columns are single or multiple; %if %sysfunc(scan(&ICDVAR,2,' ')) ne %then %do; data revised_input; length dx \$100.; set &input; pat_key = &patid; %if %length(&adj_age) > 0 %then %do; age = &adj_age; %end; array dxicd &ICDVAR; do over dxicd; if dxicd ne " then icd_full_like = compress(dxicd ,'.'); output; end;</pre>	<pre># Check if ICD_code is single or multiple if (str_length(adj_age) != 0) { INPUT\$age <- INPUT[,adj_age] } else { INPUT\$age <- 0 } colname_input <- c(ICDVAR, PATID, 'age') input_col <- INPUT[,colname_input] if (length(ICDVAR) != 1) { input__ <- INPUT[,colname_input] input_col <- input__[c(PATID, 'age', ICDVAR[1])] colnames(input_col) <- c(PATID, 'age', 'icd_code') for (i in 2:length(ICDVAR)){ aa <- input__[c(PATID, 'age', ICDVAR[i])] colnames(aa) <- c(PATID, 'age', 'icd_code')</pre>

```

        keep pat_key icd_full_like
        %if %length(&adj_age) > 0 %then %do; age %end;
        ;
    run;
%end;
%else %do;

    data revised_input;
        set &input ;

        ICD_full_like = strip(compress(&ICDVAR,','));

        pat_key = &patid;
        %if %length(&adj_age) > 0
        %then %do; age = &adj_age; %end;
        %if %length(&where) ^= 0 %then %do;
            &where;
        %end;
        keep pat_key ICD_full_like
        %if %length(&adj_age) > 0 %then %do; age %end;
        ;
    run;

%end;

%* excel file import and manipulation;
proc import out= ref_data (where = (inex eq 1))
    datafile="&INEXFILE"
    dbms=xlsx
    replace;
    getnames=YES;
    sheet = "&SHEET";
run;

```

```

    input_col <- bind_rows(input_col,aa)
}
input_col$icd_code <- str_remove(input_col$icd_code,['.'])
} else {
    input_col$icd_code <- str_remove(input_col[,ICDVAR],['.'])
}

if (str_length(WHERE) != 0){
    input_col <- input_col %>% filter(!
rlang::parse_expr(noquote(WHERE)))
}

# excel file import and manipulation

excel_data <- read_excel(INEXFILE, sheet = SHEET) %>%
filter(INEX
== 1) %>% as.data.frame()
excel_data$ICD <- str_remove(excel_data$ICD,['.'])
excel_data$SEL <-
ifelse(is.na(excel_data$SEL),0,excel_data$SEL)

for (i in 1:dim(excel_data)[1]){
    if (excel_data$SEL[i] == 1) {
        excel_data$ICD_like[i] <- paste0(excel_data$ICD[i], '%')
    } else if (excel_data$SEL[i] == 2) {
        excel_data$ICD_like[i] <- paste0('%', excel_data$ICD[i])
    } else if (excel_data$SEL[i] == 3) {
        excel_data$ICD_like[i] <- paste0('%', excel_data$ICD[i], '%')
    } else {
        excel_data$ICD_like[i] <- excel_data$ICD[i]
    }
}

if (str_length(SUBSET) != 0 ) {

```

<pre> data revised_ref_data; length icd3 icd4 icd_full \$10; set ref_data; icd3 = compress(icd, '.'); icd4 = compress(icd, '.'); icd_full = compress(icd, '.'); if sel eq . then icd_like = compress(icd, '.'); else if sel eq 1 then icd_like = cats(compress(icd, '.'), '%'); else if sel eq 2 then icd_like = cats('%', compress(icd, '.')); else if sel eq 3 then icd_like = cats('%', compress(icd, '.'), '%'); %if %length(&subset) ^= 0 %then %do; &subset; %end; run; </pre>	<pre> excel_data <- excel_data %>% filter(! rlang::parse_expr(noquote(SUBSET))) } uni <- unique(excel_data\$COM) uni_unquote<- paste0(uni, collapse = ',') %>% noquote() </pre>
---	---

2) Merge Input dataset and excel file

After manipulating the input dataset and Excel file, merge the two datasets based on the ICD codes. SQL syntax is used in both SAS and R to perform this operation. To avoid redundant data, use **'if last.'** in SAS and **'unique'** in R to remove duplicate entries.

SAS code	R code
<pre> %* merge input and excel file; proc sql noprint; select distinct com into :com_name separated by ' ' from revised_ref_data; </pre>	<pre> # merge input and excel file merge_like <- paste0('select a.',PATID,' as pat_key, a.icd_code,a.age, b.COM, b.WGT, b.INEX from input_col as a left join excel_data as b on a.icd_code like b.ICD_like') %>% sqldf() </pre>

```

create table merge_like as
select a.*, d.com, d.wgt as cci, d.inex, d.icd_like
from revised_input as a
left join revised_ref_data as d
on a.ICD_full_like like d.icd_like;
quit;

%***** Remove duplicate observations;
proc sort data = merge_like;
  by pat_key com cci ;
run;

data merge34full_a;
  set merge_like;
  by pat_key com cci;
  if last.cci;
run;

proc sort data = merge_like out = appen nodupkey;
  by pat_key %if %length(&adj_age) > 0
%then %do; age %end; com;
  where com ne ' ';
run;

proc transpose data = appen out = appen_wide
  (drop = _: rename=(pat_key = pat_key1));
  by pat_key %if %length(&adj_age) > 0
%then %do; age %end; ;
  id com;
  var inex;
run;

```

```

merge_like <- merge_like[!is.na(merge_like$COM),]

nodup_merge_full <- unique(merge_like[,c('pat_key', 'age',
'COM', 'WGT', 'INEX' )])

form <-
paste0(colnames(nodup_merge_full)[colnames(nodup_merge_f
ull) == 'pat_key'] %>% noquote(), '~ COM')

indi_com <- dcast(nodup_merge_full, form, value.var = 'INEX')

nodup_merge_full$WGT <-
ifelse(is.na(nodup_merge_full$WGT), 0,
nodup_merge_full$WGT)

sql_cci <- sqldf('select pat_key, age, com, sum(WGT) as cci
  from nodup_merge_full
  group by pat_key')

sel_uni_noquote <- paste0('pat_key' ,',', 'age',',', 'cci',',',
uni_unquote) %>% noquote()

```

3) Calculate CCI score and age adjusted CCI score

Finally, calculate the CCI score based on the weighted sum of comorbid conditions. If the input data includes patients' ages, calculate the age-adjusted CCI score using conditional logic.

SAS code	R code
<pre> %* Calculate Comorbidity index score; proc sql; create table pre_final as select a.cci,a.pat_key, b.* from (select pat_key, sum(cci) as CCI from merge34full_a group by pat_key) as a left join appen_wide as b on a.pat_key = b.pat_key1; quit; data &output; set pre_final; retain pat_key cci &com_name; array abc[*] cci &com_name; do i = 1 to dim(abc); if abc[i] eq . then abc[i] = 0; end; if cci eq . then cci = 0; %**Calculate the Age adjusted CCI score *; %if %length(&adj_age) > 0 %then %do; if age<50 then acci=cci; else if 50<= age<60 then acci=cci+1; else if 60<= age<70 then acci=cci+2; else if 70<= age<80 then acci=cci+3; </pre>	<pre> # Caculate Cormobidity index score result <- sqldf('select a.cci,a.age ,b.* from sql_cci as a inner join indi_com as b on a.pat_key = b.pat_key') nrow_result <- dim(result)[1] leftover_id <- uni[!uni %in% colnames(result)] empty_frame <-matrix(0,ncol= length(leftover_id), nrow= nrow_result) %>% as.data.frame() colnames(empty_frame) <- leftover_id result <- cbind(result, empty_frame) result_real <- paste0('select ', sel_uni_noquote, ' from result') %>% sqldf() result_real[is.na(result_real)] <- 0 #Calculate the Age adjusted CCI score for (i in 1:dim(result_real)[1]){ if (result_real\$age[i] <50) { result_real\$acci[i] = result_real\$cci[i] </pre>

```
else if age>=80 then acci=cci+4;  
else acci = cci;  
%end;
```

```
format pat_key 11. cci 2.;  
informat pat_key 11.;  
drop i pat_key1;  
run;
```

```
} else if (result_real$age[i] <60 ){  
  result_real$acci[i] = result_real$cci[i]+1  
} else if (result_real$age[i] <70 ){  
  result_real$acci[i] = result_real$cci[i]+2  
} else if (result_real$age[i] <80 ){  
  result_real$acci[i] = result_real$cci[i]+3  
} else if (result_real$age[i] >= 80 ){  
  result_real$acci[i] = result_real$cci[i]+4  
}  
}
```

```
if (str_length(adj_age) != 0) {  
  result_real <-result_real  
} else {  
  result_real <-result_real %>% select(-c(age,acci))  
}
```

```
return(result_real)
```

Sample call of c0a0cis SAS macro/R function

Example of c0a0cis SAS macro / R function inputs

c0a0cis SAS macro	c0a0cis R function
<pre>c0a0cis(INPUT = ClaimsDatabase ,WHERE = diag_date>='01Jan2015'd ,ICDVAR = diagnosis_code ,PATID = patient_id ,ADJ_AGE = age ,INEXFILE = comorb_index.xlsx ,SHEET = cci ,SUBSET = if ver eq 10 and alg = 'QUAN'; ,OUTPUT = cci_score_ClaimsDatabase);</pre>	<pre>c0a0cis (INPUT = ClaimsDatabase ,WHERE = diag_date >= '01-01-2015' ,ICDVAR = 'diagnosis_code' ,PATID = 'patient_id' ,ADJ_AGE = 'age' ,INEXFILE = 'comorb_index.xlsx' ,SHEET = 'cci' ,SUBSET = "ver == 10 & alg == 'QUAN'");</pre>

Example process of SAS Macro / R Function with Charlson Comorbidity Index

1) Import Dataset about each patient and Charlson Comorbidity Index code

Dataset about patient		Charlson Comorbidity Index			
patient_id	ICDVAR	comorbidity	description	weight	icd
F000504167881	J449	RHEUM	Rheumatological disease	1	M36.0
F000504167881	R0600	CVD	Cerebrovascular disease	1	436
F000504167881	Z5111	VASCUL	Peripheral vascular disease	1	I73.9
F000504167881	D508	LIVER1	Mild liver disease	1	K70.3
F000504167881	C3411	PULMON	Chronic pulmonary disease	1	J60
		CHF	Congestive heart disease	1	I42.9
		RHEUM	Rheumatological disease	1	710.0
		MI	Myocardial infarction	1	I22
		ULCER	Peptic ulcer disease	1	K25

2) Match Patient_id ICD code and Comorbidity Index ICD code

Result of matching					
patient_id	icd	COM	COMDES	WGT	icd_like
F000504167881	J449	PULMON	Chronic pulmonary disease	1	J44%
F000504167881	C3411	MALIGNANCY	Cancer	2	C34%
F000504167881	R0600				
F000504167881	Z5111				
F000504167881	D508				

3) Calculate CCI score per patient and show what disease they have

Result of Charlson Comorbidity Index score						
pat_key	CCI	MALIGNANCY	PULMON	METASTATIC	RENAL	VASCUL
FB44113639D	8	1	0	1	1	0
F9F526ADBE1	2	1	0	0	0	0
F5893217714	2	1	0	0	0	0
FD0C44C783F	8	1	0	1	1	0
F87126D722B	1	1	0	0	0	0
F48F26D36E6	2	1	0	0	0	0
FA4DF3E96AB	8	1	0	0	0	0
F2AD4BEDFBF	9	1	0	1	1	0
FC332D6D9C6	6	1	0	0	0	0
F67A2278E48	8	1	0	0	0	0
FDEECA8ADA9	2	1	0	0	0	0

Advantages of the new function compared to an existing open source comorbidity function.

The existing open source function for calculating comorbidity indexes in R is called `comorbidity` from the comorbidity R package. For more detailed information, you can refer to the package reference: Comorbidity R package. (n.d.). Retrieved from <https://cran.r-project.org/web/packages/comorbidity/comorbidity.pdf>. Our new function provides several advantages over this open source comorbidity function.

1) Various inputs

Existing Open Source R function	c0a0cis R function
<pre>comorbidity(x ,id ,code ,map ,assign0 ,labelled = TRUE ,tidy.codes = TRUE)</pre>	<pre>c0a0cis (INPUT ,WHERE ,ICDVAR ,PATID ,INEXFILE ,SHEET ,SUBSET)</pre>

Except labelled and tidy.codes inputs, which is not related to handling data, the number of inputs of c0a0cis is bigger than the open source function.

2) The existing open source R function needs an additional function, score, to calculate the CCI

Existing R function code								c0a0cis R function code							
patient_id		mi	chf	pvd	cevd	dementia	cpd	pat_key		cci	MI	CHF	VASCUL	CVD	DEMENTIA
F000504167881		0	0	0	0	0	1	F000504167881		3	0	0	0	0	0

The existing open source R function requires an additional function, score, to calculate the CCI. However, the new code does not require any additional functions.

3) Able to process multiple columns

Multiple ICD column dataset								c0a0cis R function
enrolid	index_date	svcdte	pdx	dx1	dx2	dx3	dx4	<pre> c0a0cis (INPUT ,WHERE ,ICDVAR = c('pdx','dx1','dx2','dx3','dx4') ,PATID ,INEXFILE ,SHEET ,SUBSET) </pre>
704277703	2016-12-19	2016-12-19	B20	B20	B258	B370	B399	
704277703	2016-12-19	2016-12-19	B20	B20	B258	B370	B399	
704277703	2016-12-19	2016-12-19	B20	B20	B258	B370	B399	
704277703	2016-12-19	2016-12-19	B20	B20	B258	B370	B399	
704277703	2016-12-19	2016-12-19	B20	B20	B258	B370	B399	

Like the picture above, when the dataset for patient information has multiple columns of icd codes such as pdx, dx1, dx2, dx3, and dx4, with the existing open source R function, we cannot handle it at all at once. However, with the new function, we just need to designate the columns of icd codes like above.

Conclusion

Our new function, **c0a0cis**, is a big improvement over current methods such as the open source comorbidity R package. It offers solutions to challenges such as handling multiple columns and providing greater flexibility. Users can easily calculate CCI scores and work more efficiently with their colleagues. With its flexibility and speed, handling complex data becomes effortless. This helps speed up analyzing patient health issues, making it easier to make smart decisions in healthcare.

Reference

- Charlson, M. E., Pompei, P., Ales, K. L., & MacKenzie, C. R. (1987). A new method of classifying prognostic comorbidity in longitudinal studies: Development and validation. *Journal of Chronic Diseases*, 40(5), 373-383.
- Deyo, R. A., Cherkin, D. C., & Ciol, M. A. (1992). Adapting a clinical comorbidity index for use with ICD-9-CM administrative databases. *Journal of Clinical Epidemiology*, 45(6), 613-619.
- Quan, H., Sundararajan, V., Halfon, P., Fong, A., Burnand, B., Luthi, J. C., ... & Ghali, W. A. (2005). Coding algorithms for defining comorbidities in ICD-9-CM and ICD-10 administrative data. *Medical Care*, 1130-1139.
- Elixhauser Comorbidity Index. (n.d.). Retrieved from <http://mchp-appserv.cpe.umanitoba.ca/viewConcept.php?printer=Y&conceptID=1436>
- van Walraven, C., Austin, P. C., Jennings, A., Quan, H., & Forster, A. J. (2015). A modification of the Elixhauser comorbidity measures into a point system for hospital death using administrative data. *Medical Care*, 53(6), 634-640. <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4812819/>
- Comorbidity R package. (n.d.). Retrieved from <https://cran.r-project.org/web/packages/comorbidity/comorbidity.pdf>

Contact

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

Geonhyeok Jeong : statsergeant@gmail.com

Ting Shi : ting.shi@merck.com