**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 Cormobidity 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

A screenshot of a computer

Description automatically generated

1. 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** |
| %\* 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;    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 maniupulation;  proc import out= ref\_data (where = (inex eq 1))  datafile="&INEXFILE"  dbms=xlsx  replace;  getnames=YES;  sheet = "&SHEET";  run;    data revised\_ref\_data;  length icd3 icd4 icd\_full $10;  set ref\_data;    icd3 = compress(icd, '.');  icd4 = compress(icd, '.');  icd\_full = compress(icd, '.');    if sel eq . then icd\_like = compress(icd,'.');  else if sel eq 1 then icd\_like =  cats(compress(icd,'.'),'%');  else if sel eq 2 then icd\_like =  cats('%',compress(icd,'.'));  else if sel eq 3 then icd\_like =  cats('%',compress(icd,'.'),'%');    %if %length(&subset) ^= 0 %then %do;  &subset;  %end;  run; | # Check if ICD\_code is single or multiple    if (str\_length(adj\_age) != 0) {  INPUT$age <- INPUT[,adj\_age]  } else {  INPUT$age <- 0  }    colname\_input <- c(ICDVAR, PATID, 'age')  input\_col <- INPUT[,colname\_input]    if (length(ICDVAR) != 1) {  input\_\_ <- INPUT[,colname\_input]  input\_col <- input\_\_[,c(PATID, 'age', ICDVAR[1])]  colnames(input\_col) <- c(PATID, 'age', 'icd\_code')    for (i in 2:length(ICDVAR)){  aa <- input\_\_[,c(PATID, 'age', ICDVAR[i])]  colnames(aa) <- c(PATID, 'age', 'icd\_code')  input\_col <- bind\_rows(input\_col,aa)  }  input\_col$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 maniupulation    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 ) {  excel\_data <- excel\_data %>% filter(!! rlang::parse\_expr(noquote(SUBSET)))  }    uni <- unique(excel\_data$COM)  uni\_unquote<- paste0(uni,collapse = ',') %>% noquote() |

1. **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** |
| %\* merge input and excel file;  proc sql noprint;  select distinct com  into :com\_name separated by ' '  from revised\_ref\_data;    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 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()  merge\_like <-merge\_like[!is.na(merge\_like$COM),]  nodup\_merge\_full <- unique(merge\_like[,c('pat\_key', 'age', 'COM', 'WGT','INEX' )])  form <-paste0(colnames(nodup\_merge\_full)[colnames(nodup\_merge\_full) == 'pat\_key'] %>% noquote(),'~ COM')  indi\_com <- dcast(nodup\_merge\_full, form, value.var = 'INEX')  nodup\_merge\_full$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() |

1. **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** |
| %\* 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;  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; | # 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$cci[i]  } else if (result\_real$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** |
| **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**  **);** | **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'”**  **)** |

**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** |
|  |  |

1. **Match Patient\_id ICD code and Comorbidity Index ICD code**

|  |
| --- |
| **Result of matching** |
|  |

1. **Calculate CCI score per patient and show what disease they have**

|  |
| --- |
| **Result of Charlson Comorbidity Index score** |
|  |

**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** |
| **comorbidity**(**x**  **,id**  **,code**  **,map**  **,assign0**  **, labelled = TRUE**  **, tidy.codes = TRUE**  ) | **c0a0cis ( INPUT**  **,WHERE**  **,ICDVAR**  **,PATID**  **,INEXFILE**  **,SHEET**  **,SUBSET**  **)** |

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.

1. **The existing open source R function needs an additional function, score, to calculate the CCI**

|  |  |
| --- | --- |
| **Existing R function code** | **c0a0cis R function code** |
|  |  |

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.

1. **Able to process multiple columns**

|  |  |
| --- | --- |
| **Multiple ICD column dataset** | **c0a0cis R function** |
| **A table with numbers and letters  Description automatically generated** | **c0a0cis ( INPUT**  **,WHERE**  **,ICDVAR = c(‘pdx’,’dx1’,’dx2’,’dx3’,’dx4’)**  **,PATID**  **,INEXFILE**  **,SHEET**  **,SUBSET**  **)** |

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](mailto:statsergeant@gmail.com)

Ting Shi : [ting.shi@merck.com](mailto:ting.shi@merck.com)