```
/*
Alexander Koplenig (2014) -
Stata do-files to reproduce the results given in:
The impact of lacking metadata and data truncation
for the measurement of cultural and linguistic change
using the Google Ngram datasets */

/* Stata version 12.1 */
/* Loop the following do files for each investigated language */

foreach language in ger spa ita fre eng-gb {
do crdata_1gram `language'
do crdata `language'
do crletterfiles
crletterfiles `language'
do cr2gram `language'
do specialcharacters `language'
do cr1gram `language'
do cravailability `language'
do freqspectra `language'
}

/* S1a: program to read in the 1-gram data */

/* name of the do-file: crdata_1gram */
/// 1. preparing the data ///
/// data can be found here:
http://storage.googleapis.com/books/ngrams/books/datasetsv2.html ///
/// the punctation files contain an error: I had to remove the lines with
only a blank manually ///
quietly {
clear
*******************************************************
capture program drop removequotation
program removequotation
quietly {
                              args letter language
                              /// Q*M = quotationmark ///

                       //// clean data: remove quotationmarks, since stata
is known to be choking on quotes ////

                                                 filefilter googlebooks-
`language'-all-1gram-20120701-`letter' `language'_cleanedfile0.txt,
from(\LQ) to("L*M") replace
                                                    local nchanges =
r(occurrences)
                                                         while `nchanges' != 0
{
                                                             filefilter
`language'_cleanedfile0.txt `language'_cleanedfile1.txt, from(\LQ)
to("L*M") replace
                                                             filefilter
`language'_cleanedfile1.txt `language'_cleanedfile0.txt, from(\LQ)
to("L*M") replace
                                                             local nchanges =
r(occurrences)
                                                         }


        filefilter `language'_cleanedfile0.txt `language'_cleanedfile1.txt,
from(`"""') to("Q*M") replace
```

```
                                                    local nchanges =
r(occurrences)
                                                         while `nchanges' != 0
{
                                                             filefilter
`language'_cleanedfile1.txt `language'_cleanedfile2.txt, from(`"""') 
to("Q*M") replace
                                                             filefilter
`language'_cleanedfile2.txt `language'_cleanedfile1.txt, from(`"""') 
to("Q*M") replace
                                                             local nchanges =
r(occurrences)
                                                         }

                       //// replace spaces with tabs, so words can be
insheeted directly  ////

                                                   filefilter
`language'_cleanedfile1.txt `language'_cleanedfile3.txt, from(`" "') to(\t)
replace
                                                    local nchanges =
r(occurrences)
                                                         while `nchanges' != 0
{
                                                             filefilter
`language'_cleanedfile3.txt `language'_cleanedfile2.txt, from(`" "') to(\t)
replace
                                                             filefilter
`language'_cleanedfile2.txt `language'_cleanedfile3.txt, from(`" "') to(\t)
replace
                                                             local nchanges =
r(occurrences)
                                                         }
                                                         }
end

capture program drop get_data
program get_data

args language letter

quietly                     {

/// 2. getting the data & adding wordclass information ///

                       //// insheet the data ////

                                                   insheet word year match vol
using `language'_splitfile_`letter'.txt, clear tab case

                       //// prepare the data ////


                       ////    remove all word that do not have word class
info    ////


                       drop if !regexm(word,"_")



                       //// prepare word class information   ////
```

```
                        replace
word=subinstr(word,"_._._.","_PUNCT_PUNCT_PUNCT",.)
                        replace word=subinstr(word,"_._.","_PUNCT_PUNCT",.)
                        replace word=subinstr(word,"_.","_PUNCT",.)
                        replace
word=subinstr(word,"_PUNCT_PUNCT","_._PUNCT",.)
                        replace
word=subinstr(word,"_PUNCT_PUNCT_PUNCT","_._._PUNCT",.)
                        drop if word=="_PUNCT"

                        foreach string in NOUN VERB ADJ ADV PRON DET ADP NUM
CONJ PRT PUNCT X {
                        drop if word=="_`string'_"
                        }

                        gen wordclass=0

                        replace wordclass=1 if  regexm(word,"_NOUN")
                        replace wordclass=2 if  regexm(word,"_VERB")
                        replace wordclass=3 if  regexm(word,"_ADJ")
                        replace wordclass=4 if  regexm(word,"_ADV")
                        replace wordclass=5 if  regexm(word,"_PRON")
                        replace wordclass=6 if  regexm(word,"_DET")
                        replace wordclass=7 if  regexm(word,"_ADP")
                        replace wordclass=8 if  regexm(word,"_NUM")
                        replace wordclass=9 if  regexm(word,"_CONJ")
                        replace wordclass=10 if regexm(word,"_PRT")
                        replace wordclass=11 if regexm(word,"_PUNCT")
                        replace wordclass=12 if regexm(word,"_X")
                        replace wordclass=13 if regexm(word,"_END_")
                        replace wordclass=14 if regexm(word,"_START_")

                        replace word=subinstr(word,"_NOUN","",.)
                        replace word=subinstr(word,"_VERB","",.)
                        replace word=subinstr(word,"_ADJ","",.)
                        replace word=subinstr(word,"_ADV","",.)
                        replace word=subinstr(word,"_PRON","",.)
                        replace word=subinstr(word,"_DET","",.)
                        replace word=subinstr(word,"_ADP","",.)
                        replace word=subinstr(word,"_NUM","",.)
                        replace word=subinstr(word,"_CONJ","",.)
                        replace word=subinstr(word,"_PRT","",.)
                        replace word=subinstr(word,"_PUNCT","",.)
                        replace word=subinstr(word,"_X","",.)


                        capture label define wc 1 "NOUN" 2 "VERB" 3 "ADJ" 4
"ADV" 5 "PRON" 6 "DET" 7 "ADP" 8 "NUM" 9 "CONJ" 10 "PRT" 11 "PUNCTUATION"
12 "OTHER" 13 "END" 14 "START"

                        label value wordclass wc

                        drop if wordclass==0
                        drop if word==""


                        compress
                        order word wordclass



        //// generate identifier ////
```

```
        egen id=group(word wordclass)
                sort id

                //// save string data seperately to make the files more
manageable ///


                preserve

                keep id word wordclass
                bysort id: gen id2=_n
                drop if id2!=1
                drop id2
                save `language'_`letter'_dict, replace

                restore



                drop word wordclass

                /// sum up potential duplicates ///
                sort id year
                by id year:egen match2=total(match)
                by id year:egen vol2=total(vol)
                by id year: gen id2=_n
                replace match=match2
                replace vol=vol2
                drop if id2>1
                drop id2 match2 vol2

                ////    reshape the data       ////

                reshape wide match vol, i(id) j(year)

                merge 1:1 id using `language'_`letter'_dict
                drop _merge
                capture erase `language'_`letter'_dict.dta

                        order word wordclass

                        compress
                        drop id
                        save `language'_`letter'_wide, replace

                }
end
*********************************************************

timer clear

qui timer on 1

local letters "a b c d e f g h i j k l m n o p q r s t u v w x y z 0 1 2 3
4 5 6 7 8 9 other punctuation"

/// check if all files are available ///

local text="missing files: "

capture program drop isitavailable
program isitavailable
```

```
args letter language
capture confirm file "googlebooks-`language'-all-1gram-20120701-`letter'"
if _rc!=0 {
noisily display "`1' is not available"
}
end

local numberoferrors=0

foreach var of local letters {
```

```
/// 3. inserting special characters & cleaning up ///
local letters "a b c d e f g h i j k l m n o p q r s t u v w x y z 0 1 2 3
4 5 6 7 8 9 other punctuation"
use `1'_a_wide, clear
foreach var of local letters {
                              merge 1:1 word wordclass using `1'_`var'_wide
                              drop _merge
                              }
                              quietly {
                              foreach v in word {

                              forvalues i=124(1)191 {
                              local i2=64+`i'
                              replace `v' = subinstr(`v',
"`=char(195)'`=char(`i')'", "`=char(`i2')'", .)
                              }
                      forvalues i=160(1)191 {
                              replace `v' = subinstr(`v',
"`=char(194)'`=char(`i')'", "`=char(`i')'", .)
                              }
                              /* ◆   œ  */
```

```
                                replace `v' = subinstr(`v',
"`=char(197)'`=char(147)'", "`=char(156)'", .)
                                /* ?   ž  */
                                replace `v' = subinstr(`v',
"`=char(197)'`=char(190)'", "`=char(158)'", .)
                                /* ?   Ÿ  */
                                replace `v' = subinstr(`v',
"`=char(197)'`=char(184)'", "`=char(159)'", .)
                                /* ?   š  */
                                replace `v' = subinstr(`v',
"`=char(197)'`=char(161)'", "`=char(154)'", .)
                                /* ?   Œ  */
                                replace `v' = subinstr(`v',
"`=char(197)'`=char(146)'", "`=char(140)'", .)
                                /* ?   Ž  */
                                replace `v' = subinstr(`v',
"`=char(197)'`=char(189)'", "`=char(142)'", .)
                                /* ?   •  */
                                replace `v' = subinstr(`v',
"`=char(226)'`=char(128)'`=char(162)'", "`=char(149)'", .)
                                /* ?   ™ */
                                replace `v' = subinstr(`v',
"`=char(226)'`=char(132)'`=char(162)'", "`=char(153)'", .)
                                }

                                        }
save `1'_1gram_based_on_unigram_data, replace

qui timer off 1
qui timer list
local time=r(t1)/(60*60)
local time1: di %9.2f `time'
noisily display "time to finish: `time1' hours"

exit
/* S1b.a: program to read in the 2-gram data */

/* name of the do-file: crdata */

//// Important remark, before starting this program:
//// this will take a while, maybe even a couple of days per language and
you
//// will need lots of available RAM, at least 64GB, more would be better
of course ////

quietly {
clear
*********************************************************
capture program drop removequotation
program removequotation
quietly {
                                args letter language
                                /// Q*M = quotationmark ///

                        //// clean data: remove quotationmarks, since stata
is known to be choking on quotes ////

                                                filefilter googlebooks-
`language'-all-2gram-20120701-`letter' `language'_cleanedfile0.txt,
from(\LQ) to("L*M") replace
                                                   local nchanges =
r(occurrences)
```

```
                                                        while `nchanges' != 0
{
                                                            filefilter
`language'_cleanedfile0.txt `language'_cleanedfile1.txt, from(\LQ)
to("L*M") replace
                                                            filefilter
`language'_cleanedfile1.txt `language'_cleanedfile0.txt, from(\LQ)
to("L*M") replace
                                                            local nchanges =
r(occurrences)
                                                        }


        filefilter `language'_cleanedfile0.txt `language'_cleanedfile1.txt,
from(`"""') to("Q*M") replace
                                                    local nchanges =
r(occurrences)
                                                        while `nchanges' != 0
{
                                                            filefilter
`language'_cleanedfile1.txt `language'_cleanedfile2.txt, from(`"""')
to("Q*M") replace
                                                            filefilter
`language'_cleanedfile2.txt `language'_cleanedfile1.txt, from(`"""')
to("Q*M") replace
                                                            local nchanges =
r(occurrences)
                                                        }

                       //// replace spaces with tabs, so word1 & word2 can
be insheeted directly  ////

                                                  filefilter
`language'_cleanedfile1.txt `language'_cleanedfile3.txt, from(`" "') to(\t)
replace
                                                    local nchanges =
r(occurrences)
                                                        while `nchanges' != 0
{
                                                            filefilter
`language'_cleanedfile3.txt `language'_cleanedfile2.txt, from(`" "') to(\t)
replace
                                                            filefilter
`language'_cleanedfile2.txt `language'_cleanedfile3.txt, from(`" "') to(\t)
replace
                                                            local nchanges =
r(occurrences)
                                                        }
                                                        }
end
*********************************************************
*********************************************************
capture program drop handleit
program handleit
tempname handle
local linenum=1
file open `handle' using `1'_cleanedfile3.txt, read text
file read `handle' line

                while r(eof)==0 {
                        local linenum = `linenum' + 1
                        file read `handle' line
                }
```

```
file close `handle'
global linenumber=`linenum'
end
*******************************************************
*******************************************************
capture program drop splitmyfiles
program splitmyfiles
*written by Nick Winter
*http://www.stata.com/statalist/archive/2004-10/msg00806.html
* splitmyfiles infilename outputstub chunk_size

version 12.0

args input outstub size

tempname in out
file open `in' using `input' , read text
qui file open `out' using `outstub'_1.txt , write text replace

local fnum 1
local i 1
file read `in' line
while !r(eof) {
file write `out' `"`line'"' _n
local ++i
if !mod(`i'-1,`size') {
file close `out'
local ++fnum
qui file open `out' using `outstub'_`fnum'.txt , write text replace
di "." _c
}
file read `in' line
}
file close `in'
file close `out'

global howmany `fnum'

end
*******************************************************
*******************************************************
capture program drop get_data
program get_data

args letter language number

quietly                     {



                        //// insheet the data ////

                                                insheet word1 word2 year
match vol using `language'_splitfile_`letter'_`number'.txt, clear tab case

                        //// prepare the data ////

                        forvalues i=1(1)2 {

                        ////    remove all word that do not have word class
info    ////
```

```
                        drop if !regexm(word`i',"_")



                        //// prepare word class information   ////

                        replace
word`i'=subinstr(word`i',"_._._.","_PUNCT_PUNCT_PUNCT",.)
                        replace
word`i'=subinstr(word`i',"_._.","_PUNCT_PUNCT",.)
                        replace word`i'=subinstr(word`i',"_.","_PUNCT",.)
                        replace
word`i'=subinstr(word`i',"_PUNCT_PUNCT","_._PUNCT",.)
                        replace
word`i'=subinstr(word`i',"_PUNCT_PUNCT_PUNCT","_._._PUNCT",.)
                        drop if word`i'=="_PUNCT"

                        foreach string in NOUN VERB ADJ ADV PRON DET ADP NUM
CONJ PRT PUNCT X {
                        drop if word`i'=="_`string'_"
                        }

                        gen wordclass`i'=0

                        replace wordclass`i'=1 if  regexm(word`i',"_NOUN")
                        replace wordclass`i'=2 if  regexm(word`i',"_VERB")
                        replace wordclass`i'=3 if  regexm(word`i',"_ADJ")
                        replace wordclass`i'=4 if  regexm(word`i',"_ADV")
                        replace wordclass`i'=5 if  regexm(word`i',"_PRON")
                        replace wordclass`i'=6 if  regexm(word`i',"_DET")
                        replace wordclass`i'=7 if  regexm(word`i',"_ADP")
                        replace wordclass`i'=8 if  regexm(word`i',"_NUM")
                        replace wordclass`i'=9 if  regexm(word`i',"_CONJ")
                        replace wordclass`i'=10 if regexm(word`i',"_PRT")
                        replace wordclass`i'=11 if regexm(word`i',"_PUNCT")
                        replace wordclass`i'=12 if regexm(word`i',"_X")
                        replace wordclass`i'=13 if regexm(word`i',"_END_")
                        replace wordclass`i'=14 if regexm(word`i',"_START_")

                        replace word`i'=subinstr(word`i',"_NOUN","",.)
                        replace word`i'=subinstr(word`i',"_VERB","",.)
                        replace word`i'=subinstr(word`i',"_ADJ","",.)
                        replace word`i'=subinstr(word`i',"_ADV","",.)
                        replace word`i'=subinstr(word`i',"_PRON","",.)
                        replace word`i'=subinstr(word`i',"_DET","",.)
                        replace word`i'=subinstr(word`i',"_ADP","",.)
                        replace word`i'=subinstr(word`i',"_NUM","",.)
                        replace word`i'=subinstr(word`i',"_CONJ","",.)
                        replace word`i'=subinstr(word`i',"_PRT","",.)
                        replace word`i'=subinstr(word`i',"_PUNCT","",.)
                        replace word`i'=subinstr(word`i',"_X","",.)


                        capture label define wc`i' 1 "NOUN" 2 "VERB" 3 "ADJ"
4 "ADV" 5 "PRON" 6 "DET" 7 "ADP" 8 "NUM" 9 "CONJ" 10 "PRT" 11 "PUNCTUATION"
12 "OTHER" 13 "END" 14 "START"

                        label value wordclass`i' wc`i'

                        drop if wordclass`i'==0
                        drop if word`i'==""
                        }
```

```
                        compress
                        order word1 wordclass1 word2 wordclass2



        //// generate identifier ////


        egen id=group(word1 wordclass1 word2 wordclass2)
                sort id

                //// save string data seperately to make the files more
manageable ///


                preserve

                keep id word1 wordclass1 word2 wordclass2
                bysort id: gen id2=_n
                drop if id2!=1
                drop id2
                save `language'_`letter'_dict, replace

                restore



                drop word1 wordclass1 word2 wordclass2

                /// sum up potential duplicates ///
                sort id year
                by id year:egen match2=total(match)
                by id year:egen vol2=total(vol)
                by id year: gen id2=_n
                replace match=match2
                replace vol=vol2
                drop if id2>1
                drop id2 match2 vol2

                ////    reshape the data       ////

                reshape wide match vol, i(id) j(year)

                merge 1:1 id using `language'_`letter'_dict
                drop _merge
                capture erase `language'_`letter'_dict.dta

                        order word1 wordclass1 word2 wordclass2

                        compress
                        save `language'_`letter'_wide_`number', replace

                }
end
*******************************************************

local language "`1'"

timer clear
```

```
qui timer on 1

local letters "a b c d e f g h i j k l m n o p q r s t u v w x y z"

foreach letter of local letters {

                              local `letter' "`letter'_ `letter'a `letter'b
`letter'c `letter'd `letter'e `letter'f `letter'g `letter'h `letter'i
`letter'j `letter'k `letter'l `letter'm `letter'n `letter'o `letter'p
`letter'q `letter'r `letter's `letter't `letter'u `letter'v `letter'w
`letter'x `letter'y `letter'z"

                              local txtlist "punctuation `a' `b' `c' `d'
`e' `f' `g' `h' `i' `j' `k' `l' `m' `n' `o' `p' `q' `r' `s' `t' `u' `v' `w'
`x' `y' `z' 0 1 2 3 4 5 6 7 8 9 other"

/// check if all files are available ///

local text="missing files: "

capture program drop isitavailable
program isitavailable
capture confirm file "googlebooks-`2'-all-2gram-20120701-`1'"
if _rc!=0 {
noisily display "`1' is not available"
}
end

local numberoferrors=0

foreach var of local txtlist {
isitavailable `var' `language'
               if _rc!=0 {
               local numberoferrors=`numberoferrors'+1
               local text="`text'`var', "
               }
               }

/// continue if everything is available ///

if `numberoferrors'==0 {
               noisily display in text "Every file available"
               noisily display in text "Processing files"
                              foreach var of local txtlist {
                              noisily display "File: `language'_`var'"
                              removequotation `var' `language'

                              /// check the size of the file, split it if
it is bigger than one hundred million lines ///
                                      handleit `language'
                                                     if
$linenumber>100000000 {
                                                                                splitmyfile
                                                                                local files
                                                                                }
                                                                                        else
                                                                                        !rer
                                                                                        loca
                                                                                        }


                              noisily di "Number of `language'_splitfiles:
`files'"
```

```
                                forvalues n=1(1)`files' {
                                                get_data `var' `language' `n'
                                                                                                                }
                                                if `files'==1 {
                                                use `language'_`var'_wide_1,
clear
                                                capture drop id
                                                save `language'_`var'_wide,
replace
                                                capture erase
`language'_`var'_wide_1.dta
                                                }

                                                        else {
                                                          use
`language'_`var'_wide_1, clear
                                                                forvalues
n=2(1)`files' {
                                                                merge 1:1
word1 wordclass1 word2 wordclass2 using       `language'_`var'_wide_`n'
                                                                drop _merge
                                                                capture erase
`language'_`var'_wide_`n'.dta
                                                                capture erase
"`language'_splitfile_`var'_`n'.txt"
                                                                }
                                                                capture drop
id
                                                                save
`language'_`var'_wide, replace
                                                                capture erase
`language'_`var'_wide_1.dta
                                                                }
                                                                capture erase
"`language'_splitfile_`var'_1.txt"
                                                                capture erase
"`language'_cleanedfile0.txt"
                                                                capture erase
"`language'_cleanedfile1.txt"
                                                                capture erase
"`language'_cleanedfile2.txt"
                                                                capture erase
"`language'_cleanedfile3.txt"
                                                                }

                                                                                                                }
                                else {
                                noisily display as error "`numberoferrors'
files seem to be missing (`text')"
                                }

qui timer off 1
qui timer list
local time=r(t1)/(60*60)
local time1: di %9.2f `time'
noisily display "time to finish: `time1' hours"
}
exit

/* S1b.b: program to read in the 2-gram data */

/* name of the do-file: crletterfiles.do */
```

```
capture program drop crletterfiles
program crletterfiles
args language
quietly {
foreach firstletter in a b c d e f g h i j k l m n o p q r s t u v w x y z
{
                              use `language'_`firstletter'__wide, clear
                                     foreach secondletter in a b c d e f g
h i j k l m n o p q r s t u v w x y z {









}
end
exit

/* S1b.c: program to read in the 2-gram data */

/* name of the do-file: cr2gram.do */

capture program drop cr2gramperyear
program cr2gramperyear
quietly {
args language year
/* create empty data set */
clear
set obs 1
gen str word1=""
gen wordclass1=.
gen str word2=""
gen wordclass2=.
gen match`year'=.
order word1 wordclass1 word2 wordclass2 match`year'
save `language'_y`year', replace

/* append files */
foreach var in 0 1 2 3 4 5 6 7 8 9 other punctuation ///
                                                        a b c d e f g h i j k
l m n o p q r s t u v w x y z {
                      use `language'_`var'_wide, clear
                      noisily display "Append: `var' / `year' /`language'"
                      capture keep word1 wordclass1 word2 wordclass2
match`year'
                                                if _rc==0 {
                                                drop if match`year'==.
                                                order word1 wordclass1 word2
wordclass2 match`year'
                                                append using
`language'_y`year'
                                                keep word1 wordclass1 word2
wordclass2 match`year'
```

```
                                                save `language'_y`year',
replace
                                                }
                                }
                        }

end

/* delete empty datasets */
forvalues i=1500(1)2009 {
        cr2gramperyear `1' `i'
        use `1'_y`i', replace
        capture sum match`i'
         if r(N)==0 {
                erase `1'_y`i'.dta
                }
                else {
                drop if match`i'==.
                compress
                save `1'_y`i', replace
                }
        }

exit

/* S1b.d: program to clean 2-gram data */

/* name of the do-file: specialcharacters.do */

capture program drop specialcharacters
program specialcharacters
args language year
use `language'_y`year', clear
di in text "Language: `language' / Year: `year'"
quietly {
foreach v in word1 word2 {

                                forvalues i=124(1)191 {
                                local i2=64+`i'
                                replace `v' = subinstr(`v',
"`=char(195)'`=char(`i')'", "`=char(`i2')'", .)
                                }
                        forvalues i=160(1)191 {
                                replace `v' = subinstr(`v',
"`=char(194)'`=char(`i')'", "`=char(`i')'", .)
                                }
                                /* ?   œ  */
                                replace `v' = subinstr(`v',
"`=char(197)'`=char(147)'", "`=char(156)'", .)
                                /* ?   ž  */
                                replace `v' = subinstr(`v',
"`=char(197)'`=char(190)'", "`=char(158)'", .)
                                /* ?   Ÿ  */
                                replace `v' = subinstr(`v',
"`=char(197)'`=char(184)'", "`=char(159)'", .)
                                /* ?   š  */
                                replace `v' = subinstr(`v',
"`=char(197)'`=char(161)'", "`=char(154)'", .)
                                /* ?   Œ  */
                                replace `v' = subinstr(`v',
"`=char(197)'`=char(146)'", "`=char(140)'", .)
                                /* ?   Ž  */
```

```
                                replace `v' = subinstr(`v',
"`=char(197)'`=char(189)'", "`=char(142)'", .)
                                /* ?   •  */
                                replace `v' = subinstr(`v',
"`=char(226)'`=char(128)'`=char(162)'", "`=char(149)'", .)
                                /* ?   ™ */
                                replace `v' = subinstr(`v',
"`=char(226)'`=char(132)'`=char(162)'", "`=char(153)'", .)
                                }

                                compress
                                save `language'_y`year', replace
                                        }
end

forvalues i=1500(1)2009 {
                        capture confirm file "`1'_y`i'.dta"
                        if _rc==0 {
                        specialcharacters `1' `i'
                        }
                        }

exit

/* S2.a: program to get the censorship names */

/* name of the do-file: prog_get2words.do */

capture program drop getrelatives
program getrelatives
tokenize `0', parse(_)
local word1="`1'"
local word2="`3'"
clear
set obs 101
gen year=1899+_n
gen `word1'`word2'_rel=0
gen `word1'`word2'_abs=0
save `word1'_`word2', replace

quietly {
forvalues i=1900(1)2000 {
use ger_y`i'.dta, clear
drop if wordclass1==14|wordclass2==13
qui sum match`i'
gen rel`i'=match`i'*1000000/r(sum)
keep if word1=="`word1'" & word2=="`word2'"
 if _N>0 {
                        collapse (sum) match (sum) rel, by(word1 word2)
                        local match`i'=match`i'[1]
                        local rel`i'=rel`i'[1]
                        use `word1'_`word2', clear
                        replace `word1'`word2'_abs=`match`i'' if year==`i'
                        replace `word1'`word2'_rel=`rel`i'' if year==`i'
                        noisily di "`word1' `word2' `i' match: `match`i'"
                        save `word1'_`word2', replace
                                }
                                else {
                        noisily di "`word1' `word2' `i' match: 0"
                }
                        }
}
```

```
end

local names `"Konrad_Adenauer Hannah_Arendt Bill_Haywood Theodor_Blank
Friedrich_Paschen Marie_Bonaparte Pierre_Curie Douglas_Hyde"'

foreach name of local names {
        getrelatives `name'
}

use Konrad_Adenauer, clear
foreach name of local names {
        merge 1:1 year using `name', nogenerate
        capture erase `name'.dta
}
save censorship, replace

/* S2.b: program to reproduce Figure 1  */

/* name of the do-file: plotcensorship.do */

capture program drop suppressed
program suppressed
use censorship, clear
tsset year, y
tssmooth ma `1'`2'_ma=`1'`2'_rel, window(5 1 5)
twoway (tsline `1'`2'_ma, lcolor(`5') lwidth(medthick) lpattern(solid)) ///
                       (tsline `1'`2'_rel, lcolor(gs5)) ///
                               , ytitle("") ytitle(, size(large)) ///
                                xscale(nofextend) ///
                                ylabel(, labsize(vlarge) nogrid
angle(vertical)) ttitle("") ttitle(, size(large)) ///
                                title("`3': `1' `2'", box bexpand
fcolor(white)) legend(off) graphregion(color(white)) ///
                                tlabel(1900 1929 1939 1960 2000,
labsize(large) angle(45) labcolor(`4')) xline(1929 1939 1960, lcolor(gs5)
lpattern(dash)) ///
                                 nodraw scheme(s2mono)
 graph save `1'_`2'.gph, replace
 end

suppressed Konrad Adenauer A black cranberry
suppressed Hannah Arendt B black cranberry
suppressed Theodor Blank C black cranberry
suppressed Bill Haywood  D black cranberry

suppressed Pierre Curie F black emerald
suppressed Marie Bonaparte E black emerald
suppressed Friedrich Paschen H black emerald
suppressed Douglas Hyde G black emerald

graph combine  ///
               Konrad_Adenauer.gph Hannah_Arendt.gph Theodor_Blank.gph
Bill_Haywood.gph ///
               Marie_Bonaparte.gph Pierre_Curie.gph Friedrich_Paschen.gph
Douglas_Hyde.gph  , ///
               scheme(s2mono) graphregion(color(white)) cols(2) xsize(1)
ysize(2) imargin(0 0 0 0)

graph export Figure1.tif, replace wid(980)
window manage close graph
exit

/* S3: program to reproduce Figure 2  */
```

```
/* name of the do-file: booklength.do */

/* the total_count files are available here
http://storage.googleapis.com/books/ngrams/books/datasetsv2.html*/

capture program drop booklength
program booklength
version 12.1
/* make the data stata readable */
filefilter googlebooks-`1'-all-totalcounts-20120701.txt cleanedfile0.txt,
from(\t) to(\n) replace
                                                        local nchanges =
r(occurrences)
                                                             while `nchanges' != 0
{
                                                                 filefilter
cleanedfile0.txt cleanedfile1.txt, from(\t) to(\n) replace
                                                                 filefilter
cleanedfile1.txt cleanedfile0.txt, from(\t) to(\n) replace
                                                                 local nchanges =
r(occurrences)
                                                             }

/* importing the data */
insheet year match pages volumes using cleanedfile0.txt, comma clear names
capture erase cleanedfile0.txt
capture erase cleanedfile1.txt

/*time series analysis */
tsset year, yearly

gen book_length=match/volume
tssmooth ma book_length_ma=book_length, w(5 1 5)

/* save the data */
keep year book_length book_length_ma
if "`1'"!="eng-gb" {
rename book_length `1'_book_length
rename book_length_ma `1'_book_length_ma
}
else {
rename book_length enggb_book_length
rename book_length_ma enggb_book_length_ma
}

/* keep data from 1900 to 2000 */
keep if tin(1900,2000)
save `1'_book_length, replace

/* plot the data*/
if "`1'"!="eng-gb" {
        use `1'_book_length, clear
        qui sum `1'_book_length
}
else {
        use eng-gb_book_length, clear
        qui sum enggb_book_length

}
gen max=r(max)
twoway (area max year if tin(1914,1918), color(bluishgray)) ///
```

```
                              (area max year if tin(1939,1945),
color(bluishgray)) ///
                        (tsline *_book_length, lcolor(gs0) lpattern(solid))
///
                        (tsline *_book_length_ma, lcolor(cranberry)
lwidth(medium) lpattern(solid)) ///
                              , ytitle("", size(large))  yscale(nofextend)
///
                               ylabel(minmax, format(%10.1e) labsize(large)
nogrid) ttitle("") ttitle(, size(large)) ///
                               title("`2' data", box bexpand fcolor(white))
legend(off) graphregion(color(white)) ///
                               tlabel(1900 1920 1940 1960 1980 2000,
labsize(large) angle(90)) nodraw scheme(s2mono)
 graph save book_length_`1'.gph, replace



end

booklength ger `"German"'
booklength fre `"French"'
booklength spa `"Spanish"'
booklength ita `"Italian"'
booklength eng-gb `"British English"'

graph combine  book_length_eng-gb.gph ///
                                                   book_length_fre.gph ///
                                                   book_length_ita.gph ///
                                                   book_length_spa.gph ///
                                                   , scheme(s2mono)
graphregion(color(white)) ///
                                                   cols(2)
                                                   graph save
book_lengthes.gph, replace

use ger_book_length, clear
sum ger_book_length
gen max=r(max)
twoway (area max year if tin(1914,1918), color(bluishgray)) ///
                              (area max year if tin(1939,1945),
color(bluishgray)) ///
                              (tsline ger_book_length, lcolor(gs0)
lpattern(solid)) ///
                              (tsline ger_book_length_ma, lcolor(cranberry)
lwidth(medium) lpattern(solid)) ///
                              , ytitle("", size(large))  yscale(nofextend)
///
                               ylabel(#4, format(%9.0f) labsize(large)
nogrid) ttitle("") ttitle(, size(large)) ///
                               title("German data", box bexpand
fcolor(white)) legend(off) graphregion(color(white)) ///
                               tlabel(1900 1920 1960 1940 1980 2000,
labsize(large) angle(90)) scheme(s2mono)
 graph save book_length_ger.gph, replace


/* combine the plots */
graph combine  book_length_ger.gph book_lengthes.gph, scheme(s2mono)
graphregion(color(white)) ///
                                                   cols(2) xsize(4) ysize(2)
  graph export Figure2.tif, replace  wid(980)
window manage close graph
```

```
/* combine the data */
use eng-gb_book_length, clear
foreach var in fre spa ita ger {
        merge 1:1 year using `var'_book_length
        drop _merge
}

tsset year,y
keep if tin(1900,2000)

save booklength, replace

exit

/* S4: program that calculates 1-gram freqs
using the 2-gram data  */

/* name of the do-file: cr1gram.do */

clear
set obs 1
gen str word1=""
gen wordclass1=.
save `1'_1gram_based_on_bigram_data, replace
quietly {
forvalues i=1500(1)2009 {
                capture use `1'_y`i', clear
                if _rc==0 {
                noisily di "year: `i'"
                collapse (sum) match, by(word1 wordclass1) fast
                merge 1:1 word1 wordclass1 using
`1'_1gram_based_on_bigram_data
                drop _merge
                compress match`i'
                save `1'_1gram_based_on_bigram_data, replace
                }
                }

}
/* remove START unigrams */
drop if wordclass==14

/* cleaning up */
rename word1 word
rename wordclass1 wordclass
order match*, sequential
order word wordclass
drop if wordclass==.
compress
save `1'_1gram_based_on_bigram_data, replace

use `1'_1gram_based_on_bigram_data.dta, clear

/* generate a variable that contains the summed token frequency
for each word type */

egen bigram_match=rowtotal(match*)

keep word wordclass bigram_match
save `1'_frequencylist_2gram, replace

/* repeat for unigram_data */
```

```
use `1'_1gram_based_on_unigram_data.dta, clear
keep word wordclass match*
egen unigram_match=rowtotal(match*)

keep word wordclass unigram_match

save `1'_frequencylist_1gram, replace

/* merge both frequencylists */

use `1'_frequencylist_1gram, clear
merge 1:1 word wordclass using `1'_frequencylist_2gram

tab _merge

/* calculate the frequencies presented in the text */

sum unigram_match if _merge==1|_merge==3
local denom=r(N)

sum unigram_match if _merge==3
local bigram_types=r(N)

qui sum unigram_match if _merge==1, d
local loss=r(N)*100/`denom'
local types=r(N)+`bigram_types'
di "1-gram types: `types'"
di "2-gram types: `bigram_types'"
di "relative loss: `loss'"
di r(p25)
di r(p50)
di r(p75)
di r(p90)
di r(p95)
di r(p99)
di "`1'"

exit

/* S5: program that reproduces Figure 4  */

/* name of the do-file: freqspectra.do */

capture program drop cr_hapax
program cr_hapax
/* create a set to store the data */

use `1'_totalcounts, clear
keep if year>1749

/* generate variables to store the information */
gen hapax_unigram=.

save `1'_freq_spectrum, replace

/* calculate the fraction of types with match==1 for the unigram data */

use `1'_1gram_based_on_unigram_data, clear
        drop vol*
        keep match1750-match2009
forvalues i=1750(1)2009 {
        qui sum match`i'
```

```
        local denom=r(N)
        qui sum match`i' if match`i'==1
        local fractionuni`i'=r(N)/`denom'
        di "lang: `1' / unigram / year: `i' / fraction: `fractionuni`i''"
        drop match`i'
}

/* store the results */

use `1'_freq_spectrum, clear
forvalues i=1750(1)2009 {
        replace hapax_unigram=`fractionuni`i'' if year==`i'
}
save `1'_freq_spectrum, replace
end

capture program drop plot_hapax
program plot_hapax
/* plot the information */
use `1'_freq_spectrum, clear

qui tsset year, y

/* restrict for time span 1800 - 2000 */
keep if tin(1800,2000)

/* test for stationary */

foreach var in hapax match {
qui pperron `var', trend
if r(pval)<.01 {
        display as error "`var' time series is stationary"
        exit 198
}
else {
        display in text "`var' time series is non stationary"
        qui pperron D.`var', trend
        if r(pval)>.01 {
        display as error "D.`var' time series is non stationary"
        exit 198
}
else {
        display in text "D.`var' time series is stationary"
}
}
}
twoway (tsline match, lcolor(gs0) color(emerald) lcolor(emerald)) ///
                        (tsline hapax_unigram, yaxis(2) lcolor(cranberry)
lwidth(medium) lpattern(solid)) ///
                              , yscale(reverse axis(2))  ytitle("Corpus
size", size(large) color(emerald) ) ///
                             ytitle("Hapax legomena (%)", size(large)
axis(2) color(cranberry)) tlabel(, labsize(large) nogrid) ///
                              ylabel(#3, labsize(medium) nogrid
labcolor(emerald)tlcolor(emerald)) ///
                              ylabel(#3, labsize(large) nogrid axis(2)
labcolor(cranberry)tlcolor(cranberry)) ttitle("") ttitle(, size(large)) ///
                              title("") legend(off)
graphregion(color(white)) ///
                              yscale(nofextend lcolor(emerald))
yscale(nofextend lcolor(cranberry) axis(2)) xscale(nofextend) ///
                              scheme(s2mono) nodraw
 graph save `1'_hapax.gph, replace
```

```
xcorr D.match D.hapax, lag(20) ///
                       xlabel(, labsize(large) nogrid) mcolor(cranberry)
lwidth(medium) msize(medium) ///
                       ytitle("Cross-correlation", size(large) ) ///
                       ytitle("", size(large) axis(2)) ///
                       ylabel(-1 0 1, labsize(large) nogrid) ///
                       ylabel(minmax, labsize(large) nogrid axis(2)
angle(horizontal)) ///
                       yscale(noextend axis(2)) ///
                       title("") fxsize(35) xlabel(#3,labsize(large))
xtitle(,size(large)) ///
                       scheme(s2mono) graphregion(color(white)) ///
               nodraw
 graph save `1'_cross.gph, replace

graph combine `1'_hapax.gph `1'_cross.gph, scheme(s2mono) xsize(4) ysize(2)
///
               title("`2' data", box bexpand fcolor(white))  imargin(b=1
t=10 r=0) nodraw
graph save `1'_cross.gph, replace
end

cr_hapax eng-gb
cr_hapax ita
cr_hapax spa
cr_hapax ger
cr_hapax fre
plot_hapax eng-gb `"British English"'
plot_hapax fre French
plot_hapax ger German
plot_hapax ita Italian
plot_hapax spa Spanish

graph combine eng-gb_cross.gph fre_cross.gph ger_cross.gph ita_cross.gph
spa_cross.gph, ///
        scheme(s2mono) cols(1) xsize(1) ysize(2) imargin(0 0 0 0) ///
        graphregion(color(white))
 graph export Figure3.tif, replace wid(980)
 window manage close graph
/* time span restriction 1900-2000 as reported in text */
quietly {
foreach language in eng-gb fre ger ita spa {
use `language'_freq_spectrum, clear
tsset year, y

corr D.match D.hapax if tin(1900,2000)
local cor: di %3.2f r(rho)
noisily di "`language' cor: `cor' "
}
}
exit

/* S6: program that implements the GNgC design rules
and reproduces Figure 4 and the ANOVA  */

/* name of the do-file: BNCzipf.do */

/* Michel et al SOM II.3A
 separate words
! (exclamation-mark)
@ (at)
% (percent)
```

```
^ (caret)
* (star)
( (open-round-bracket)
) (close-round-bracket)
[ (open-square-bracket)
] (close-square-bracket)
- (hyphen)
= (equals)
{ (open-curly-bracket)
} (close-curly-bracket)
| (pipe)
\ (backslash)
: (colon)
: (semi-colon)
< (less-than)
, (comma)
> (greater-than)
? (question-mark)
/ (forward-slash)
~ (tilde)
` (back-tick)
" (double quote)
# (hash)
+ (plus)
$ (dollar)
*/

/* the first local macro identifies the corpus (e.g. bncwritten)
in the second local macro, you can decide in how many subparts you want to
split the data, of course this has to do with the amount of RAM
you have available on your machine (e.g. for the bncwritten data 10 slices
are
okay)*/
use `1', clear
collapse (sum) match, by(word) fast
set seed 18042014
gen random=100*runiform()

egen group = cut(random), group(`2') icodes

drop random

sum group
local max=r(max)
local min=r(min)

quietly {
forvalues i=`min'(1)`max' {
        noisily di "now processing: `i'"
        global id=`i'
preserve
keep if group==`i'

local separates "36 43 35 33 64 37 94 42 40 41 91 93 45 61 123 125 124 92
58 59 60 44 62 63 47 126 96 34 147 148"

foreach separateword of local separates {
        if `separateword'==34 {
        replace word = subinstr(word, `"""', `" " "', .)

        }
        else {
```

```
        replace word = subinstr(word, "`=char(`separateword')'", "
`=char(`separateword')' ", .)
        }
}

*Apostrophe, but only when it not precedes the letter s (Alice's)
replace word = subinstr(word, `"`=char(39)'"', `" `=char(39)' "', .)
replace word = subinstr(word, `" `=char(39)' s"', `"`=char(39)'s"', .)

replace word=trim(word)
replace word=itrim(word)
split word
egen toexpand=noccur(word), string(`" "')
replace toexpand=1+toexpand
expand toexpand
bysort word:gen id=_n

sum id
local max=r(max)
forvalues i=1(1)`max' {
        replace word=word`i' if id==`i'
}

collapse (sum) match, by(word) fast
gsort -match
compress
save $id, replace
restore
}
}

sum group
local max=r(max)
local min=r(min)
use `min', clear
capture erase `min'.dta
forvalues i=1(1)`max' {
        append using `i'
        capture erase `i'.dta
}
collapse (sum) match, by(word) fast
compress
gsort -match word
save `1'_gncstyle, replace

        /* set seed to make the results reproducible */
        set seed 227270

/* program to calculate zipf exponent */
capture program drop myzipf
   program myzipf
            version 12.1
            args lnf alpha
            tempname H
            quietly {
                                    generate double `H'=1/(rank^`alpha')
                                    sum `H'
                                    global H_val=log(r(sum))
                                    replace `lnf'=(-
1)*`alpha'*$match_logr - $T*$H_val
                                    }
    end
```

```
capture program drop calc_trunc
program calc_trunc
/* create an empty data set to store the information */
clear
set obs 1
gen str whichcorpus=""
gen str6 size=""
/* this variable indicates if the data is untruncated (=0) or truncated */
gen norvstrunc=.

/*this variable indicates the fraction of hapax legomena  */
gen hapax=.

/* zipf exponent plus confidenceinterval */
gen zipfexponent=.
gen zipflowerbound=.
gen zipfupperbound=.

/* this variable indicates the number of the repetetion */
gen round=.

/* the next two variables show what is in the data that is truncated
        tokens & types */
gen types=.
gen tokens=.

save zipf`1', replace

quietly {

        /* this step is repeated 20 times to rule out random noise */
forvalues r=1(1)20 {
foreach i in 0 `2' {
                                        if "`1'"=="bnc" {
                                        use bncwritten_gncstyle, clear
                                        }
                                        else {
                                        use
treet_wortformen_dereko2011_gncstyle, clear
                                        }
                                        drop if match<=`i'
                                        gen rest=match
                                        foreach size in 050 025 0125 00625
003125 {



                                        /* calculate zipf exponent */
                                        foreach size in 050 025 0125 00625
003125 {

        preserve
                                                                        keep
if s`size'>0
                                                                        sum
s`size' if s`size'>0
                                                                        local
denom=r(N)

                                                                        /*
types & tokens */
                                                                        local
types=r(N)
```

```
local
tokens=r(sum)
sum
s`size' if s`size'==1
local
hapax=r(N)/`denom'
sum
s`size'
local
sample=1000000/r(sum)
local
csize=r(sum)
gen
sample=rbinomial(s`size', `sample')
drop
if sample<1|sample==.

sum
sample if sample>0

local
samplesize=r(sum)

        noisily di "round: `r' size: `size' samplesize: `samplesize'
corpussize: `csize' hapaxe: `hapax'"


drop
match
rename
sample match
keep
match
gsort
-match
gen
rank=_n
gen
match_logr=match*log(rank)
sum
match_logr
global
match_logr=r(sum)
sum
match
global
T=r(sum)

ml
model lf myzipf ()

ml
maximize
mat
pred=r(table)
local
zipfexponent=pred[1,1]
local
zipflowerbound=pred[5,1]
local
zipfupperbound=pred[6,1]
use
zipf`1', clear
```

```
                                                                        local
new = _N + 1
                                                set obs `new'
                                                replace size="`size'" in
`new'

        replace zipfexponent=`zipfexponent' in `new'

        replace zipflowerbound=`zipflowerbound' in `new'

        replace zipfupperbound=`zipfupperbound' in `new'

        replace norvstrunc=`i' in `new'

        replace hapax=`hapax' in `new'

        replace whichcorpus="`1'" in `new'

        replace round=`r' in `new'

        replace types=`types' in `new'

        replace tokens=`tokens' in `new'
                                                                        save
zipf`1', replace

        restore



end

calc_trunc dereko 20
calc_trunc bnc 4

/* program to plot the data as presented in the text */
capture program drop plot_trunc
program plot_trunc
use zipf`1', clear
*encode size, generate(sizenr)
generate sizenr=.
replace sizenr=1 if size=="050"
replace sizenr=2 if size=="025"
replace sizenr=3 if size=="0125"
replace sizenr=4 if size=="00625"
replace sizenr=5 if size=="003125"

label define sizer 1"50%" 2"25%" 3"12.5%" 4"6.25%" 5"3.125%"
label value sizenr sizer

label define trunc 0"untruncated" 20"truncated (<20)"  4"truncated (<4)"
label value norvstrunc trunc

anova zipfexp i.sizen##i.norvs
margins sizen#norvs
marginsplot,x(norvstrunc) graphregion(color(white)) ///
        title ("") ///
        xtitle("") xlabel(,nolab ) ///
        ylabel(, angle(horizontal)) ytitle("`5'", size(large)) ///
        yscale(nofextend) xscale(off) ///
        legend(`4') nodraw fysize(35)
        graph save `1'_margins.gph, replace
```

```
qui sum zipfexponent if norvstrunc==0,d
local mean_untrunc: di %12.4f r(mean)

qui sum zipfexponent if norvstrunc!=0,d
local mean_trunc: di %12.4f r(mean)
di `mean_untrunc'
di `mean_trunc'
graph box zipfexponent, over(sizenr, label(angle(90))) over(norvstrunc,
label(labsize(large)))  ///
        scheme(s2color) graphregion(color(white)) asyvars ///
        title ("`2' data",size(large)  box bexpand fcolor(white)) ///
        ylabel(`mean_untrunc' `mean_trunc',  noticks angle(horizontal) ///
        labsize(medium) nogrid) ///
        yscale(noextend) ytitle("`3'") nodraw ///
        ytitle(, size(large)) ///
        yline(`mean_untrunc' `mean_trunc', lcolor(black)) legend(off) ///
        box(1, fintensity(inten50) lwidth(thin)) marker(1, msize(small))
        graph save `1'_zipf.gph, replace

graph combine `1'_zipf.gph `1'_margins.gph, ///
        scheme(s2mono) cols(1) xsize(1) ysize(2) ///
        graphregion(color(white)) nodraw

        graph save `1'_prediction.gph, replace
end

plot_trunc bnc BNC `"Zipf exponent"'`"subtitle("Corpus
size:",justification(left)) ring(0) cols(1) bplace(neast)
region(color(none))"' `"Linear prediction"'
plot_trunc dereko DeReKo `" "' off `" "'

graph combine bnc_prediction.gph dereko_prediction.gph, ///
        scheme(s2mono) cols(2) xsize(1) ysize(1) imargin(0 0 0 0) ///
        graphregion(color(white)) b1title("Adjusted predictions with 95%
CIs", size(small))
        graph export Figure4.tif, replace wid(980) hei(490)
window manage close graph
exit

/* S7: program to draw random corpus samples, ML estimate the Zipf exponent
and reproduce Figure 5   */

/* name of the do-file: zip_1_2gram.do */

                replace `ngram'zipfexponent=`zipfexponent' if year==`i'
                replace `ngram'zipflowerbound=`zipflowerbound' if year==`i'
                replace `ngram'zipfupperbound=`zipfupperbound' if year==`i'
                save `1'_zipfexponent, replace
                restore
                drop match`i'
                local param: di %12.4f `zipfexponent'
                noisily di "`ngram' / zipf parameter for `i' = `param'"
        }
}
}

capture program drop plot_zipf
program plot_zipf
/* plot the information */
use `1'_zipfexponent, clear

qui tsset year, y
```

```
/* smooth the data */
qui tssmooth ma uni_ma=unizipfe, window(5 1 5)
qui tssmooth ma bi_ma=bizipfe, window(5 1 5)

qui corr D.bizipfexponent D.unizipfexponent
local r: di %4.2f r(rho)
noisily di `r'

format uni_ma bi_ma unizipfe bizipfe %4.3f

twoway (tsline unizipfexponent, lcolor(emerald) lwidth(medthick)
lpattern(solid)) ///
                       (tsline bizipfexponent, lcolor(cranberry)
lwidth(medthick) lpattern(solid)) ///
                       (tsline uni_ma, lcolor(gs9) lwidth(medium)
lpattern(solid)) ///
                       (tsline bi_ma, lcolor(gs9) lwidth(medium)
lpattern(solid)) ///
                             ,  ytitle("{&alpha}", size(large) ) ///
                              ylabel(minmax, labsize(small) nogrid
angle(v) ) ///
                              ttitle("") ttitle(, size(large)) ///
                              title("") graphregion(color(white)) tlabel(,
labcolor(`4') labsize(large) grid) ///
                              yscale(nofextend lcolor(emerald))
xscale(nofextend) ///
                              scheme(s2mono) title("`2' ({it:r} = `r')",
box bexpand fcolor(white)) ///
                              legend(label(1 "Unigram data") ///
                              label(2 "Bigram data") size(medium)  order(1
2) ring(0) pos(7) cols(1) region(color(none))) `3'


                              *nodraw
 graph save `1'_biuni.gph, replace


end

plot_zipf fre French legend(off) white
plot_zipf eng-gb `"British English"' "" white
plot_zipf ger German legend(off) white
plot_zipf spa Spanish legend(off) black
plot_zipf ita Italian legend(off) black

graph combine eng-gb_biuni.gph fre_biuni.gph ger_biuni.gph ita_biuni.gph
spa_biuni.gph, ///
        scheme(s2mono) cols(2) ///
        graphregion(color(white))
                             graph export Figure5.tif, replace wid(980)

/* correlation between corpus size and zipf exponent */
capture program drop compare_corpussize_zipf
program compare_corpussize_zipf
quietly {
use `1'_totalcounts, clear
tsset year,y
keep if tin(1800,2000)
merge 1:1 year using `1'_zipfexponent

*cor differences match zipf
```

```
qui cor D.unizipfexponent D.match
local r: di %4.2f r(rho)
noisily di "`1' correlation: `r'"

}
end

foreach language in eng-gb fre ger ita spa {
compare_corpussize_zipf `language'
}

/* inter-languages correlations */
use zipf_total, clear

tsset year, y
qui {
foreach var1 in enggb fre spa ita ger {
foreach var2 in enggb fre spa ita ger {
        qui corr D.zipf_`var1' D.zipf_`var2'
        local rho: di %4.2f r(rho)
        noisily di "`var1' `var2' = `rho'"
}
}

/* SI2: program to reproduce the time series
in the supplementary information  */

clear
set obs 1
gen str word1=""
gen str word2=""
save completenames, replace

local names1 "Robert_Froriep August_Dorner Jakob_Frohschammer Carl_Gerhardt
Robert_Applegarth"
local names2 "Serafino_Vannutelli Henry_Head Charles_Ricketts Bill_Haywood
Stephen_Crane"
local names3 "Piet_Mondrian Louis_Ginzberg Ita_Wegman Hermann_Maas
Amedeo_Modigliani Max_Taut"
local names4 "Karl_Freund Fritz_Heider Theodor_Haubach Francis_Poulenc
Lucie_Mannheim"
local names5 "Curt_Bois Erik_Erikson Manes_Sperber Theodor_Blank
Richard_Krebs"
local names6 "Rudolf_Peierls Brigitte_Helm Konrad_Adenauer Hannah_Arendt
Kurt_Schumacher"
local names7 "Theodor_Heuss Egon_Schiele Walter_Ulbricht Wilhelm_Peters
August_Macke"
local names8 "Andreas_Hermes Erich_Ollenhauer Max_Beckmann Herwarth_Walden
Alfred_Kurella"
local names9 "Clara_Zetkin Otto_Suhr Elly_Ney Ernst_Reuter Buster_Keaton"
local names10 "Max_Reimann Oskar_Schlemmer El_Lissitzky Lyonel_Feininger
Gottlieb_Daimler "
local names11 "Basil_Zaharoff Italo_Balbo Walther_Funk Horst_Wessel
Georges_Bonnet "
local names12 "Hugo_Junkers Adolf_Hitler Robert_Heger Mark_Hanna
Ernst_Heinkel "
local names13 "Gustav_Weigand Ernst_Eckstein Dietrich_Eckart Douglas_Hyde
Richard_Mayr "
local names14 "Louis_Botha Guglielmo_Marconi Friedrich_Paschen
Alfred_Ploetz Adele_Sandrock "
local names15 "Marie_Bonaparte Robert_Ley Edouard_Rod Saavedra_Lamas
Berthe_Morisot "
```

```
local names16 "Leo_Schlageter Giuseppe_Bottai Hans_Buchner Gustaf_Kossinna
Dwight_Morrow "
local names17 "Rudolf_Diesel Katharina_Schratt Alfred_Rosenberg
Largo_Caballero Theodor_Fritsch "
local names18 "Arthur_Griffith Albert_Steffen Pierre_Curie Max_Bodenstein
Oliveira_Salazar "
local names19 "Orla_Lehmann Giuseppe_Motta Stanley_Baldwin
Lombardo_Toledano Wilhelm_Hauer "
local names20 "Max_Fleischer Gustav_Nachtigal Philipp_Lenard Boris_III"

forvalues i=1/20 {
foreach name of local names`i' {
        tokenize `name', parse(_)
        use completenames, clear
        local new=_N+1
        set obs `new'
        replace word1="`1'" in `new'
        replace word2="`3'" in `new'
        drop if word1==""
        save completenames, replace
}
}
/* merge the names */
quietly {
forvalues i=1900(1)2000 {
use ger_y`i'.dta, clear
drop if wordclass1==14|wordclass2==13
qui sum match`i'
gen rel`i'=match`i'*1000000/r(sum)
merge m:1 word1 word2 using completenames
keep if _merge!=1
collapse (sum) match* (sum) rel*, by(word1 word2)fast
                       noisily di "merging: `i'"
                       save completenames, replace

                       }
}

capture program drop suppressed
program suppressed
twoway (tsline `1'_`2'_ma, lcolor(`5') lwidth(medthick) lpattern(solid))
///
                       (tsline `1'_`2'_rel, lcolor(gs5)) ///
                              , ytitle("") ytitle(, size(large)) ///
                               xscale(nofextend) ///
                               ylabel(#3, labsize(large) nogrid
angle(vertical)) ttitle("") ttitle(, size(large)) ///
                               title("`3': `1' `2'", box bexpand
fcolor(white)) legend(off) graphregion(color(white)) ///
                               tlabel(1900 1929 1939 1960 2000,
labsize(large) labcolor(`4')) xline(1929 1939 1960, ///
                               lcolor(gs11) lpattern(dash)) scheme(s2mono)
 graph export `3'_`1'_`2'.tif, replace wid(600)
 end

/* plot all censored names */

use completenames.dta , clear
drop match*
order _all, sequential
order word1 word2

forvalues i=1/100 {
```

```
local name`i'=word1[`i']+"_"+word2[`i']
}

xpose, clear
drop in 1/2

gen year=1899+_n
order year
tsset year, y

forvalues i=1/100 {
tssmooth ma v`i'_ma=v`i', w(5 1 5)
ren v`i' `name`i''_rel
ren v`i'_ma `name`i''_ma
}

local i=1

forvalues n=1/10 {
foreach name of local names`n' {
        tokenize `name', parse(_)
        suppressed `1' `3' `i' black cranberry
        capture `++i'
}
}

forvalues n=11/20 {
foreach name of local names`n' {
        tokenize `name', parse(_)
        suppressed `1' `3' `i' black emerald
        capture `++i'
}
}

exit
```