# Stat 405/705
# Class 8
## Statistical computing with R

Richard P. Waterman

Wharton

# Table of contents I

# Today's module

Topics to be covered in this module:

- Last time
- Case study
- Functions used in today's class
- Next time

# Last time

- Logic: If/else,
- Iteration: For, while, break and next

# Case study

Transactions to behavior:

- Many databases are designed to capture *transactions*
- But many analytic techniques are applied to behavioral patterns
- This requires individual transactions to be aggregated to behaviors via a common ID

# Examples

- Hospital:
  - The patient admissions database shows you who is in the hospital.
  - Combining admissions gives you a patient's history
- Supermarket:
  - Individual items get aggregated to a customer's shopping visit
  - Customer visits get aggregated to a long term behavior
- HR:
  - A cut of the employee database in any given month, shows you who is employed
  - Combining the months, shows you an employee's history
- Outpatient clinic:
  - The visits database tells you about each visit
  - Combining the visits, shows you a patient's history

# The commonality

- All of these examples are of the same essential nature
- Transactions are recorded
- A common ID can be used to link transactions
- Histories can be built across common IDs
- This translates transactions to *customer* behavior/history

# An example transaction database

```
#Read in the datafile (also in the "outpatient.Rdata" file)
outpatient <- read.csv(
"C:\\Users\\richardw\\Dropbox (Penn)\\Teaching\\705s2019\\Data\\Outpatient.
)
#Have a quick look at it
head(outpatient,5)

##      PID  SchedDate   ApptDate             Dept Language Sex
## 1 P10092  7/27/2012  10/5/2012           DERM   ENGLISH   F
## 2 P10151 11/28/2013   1/3/2014       PULMONARY  SPANISH   F
## 3 P10962   2/2/2012  2/10/2012  OTOLARYNGOLOGY  ENGLISH   M
## 4 P10896  11/8/2011  12/6/2011 GENERAL SURGERY  SPANISH   F
## 5 P10320 10/25/2012 12/11/2012      NEPHROLOGY  SPANISH   F
##   Age              Race    Status
## 1 80+ AFRICAN AMERICAN   Arrived
## 2  72          HISPANIC Cancelled
## 3  12 AFRICAN AMERICAN   Arrived
## 4  60          HISPANIC    Bumped
## 5  45          HISPANIC   No Show
```

# Comments on the data fields

- PID is the patient identifier
- Schedule date and Appointment date are both time variables
- Status is the outcome variable of interest
- We only want to use those appointments that had status either Arrived or No Show
- We will focus on the most recent appointment, but add in behavioral history predictors for it

# Summary statistics

```
summary(outpatient)
```

```
##       PID             SchedDate          ApptDate
##  P10848 :  53   1/2/2013 :  12   4/11/2012:  11
##  P10998 :  29   1/22/2013:  11   6/15/2012:  11
##  P10840 :  27   10/7/2012:  11   2/28/2013:  10
##  P10932 :  26   5/10/2012:  11   3/15/2013:  10
##  P10254 :  25   6/21/2012:  11   5/17/2012:  10
##  P10641 :  25   6/27/2012:  11   5/26/2013:  10
##  (Other):3514   (Other)  :3632   (Other)  :3637
##                Dept                Language    Sex
##   NEUROLOGICAL  : 669   ENGLISH   :2992   F:2100
##   CPO           : 658   SPANISH   : 646   M:1599
##  ORTHOPAEDICS   : 382   VIETNAMESE:  20
##   DERM          : 239   Unknown   :  10
##  PLASTIC SURGERY: 221   ARABIC    :   8
##  GENERAL SURGERY: 180   BENGALI   :   4
##  (Other)        :1350   (Other)   :  19
##       Age                              Race
##  80+    : 151   AFRICAN AMERICAN    :1339
```

# Problems

- The date variables have been read in as factors, which will make date manipulations impossible
- A key calculation is the schedule lead time: the difference in days between schedule date and appointment date
- The appointments are not ordered in any way.

```
# The dates are not correctly represented
print(class(outpatient$SchedDate))

## [1] "factor"
```

# Problems with the data

- We want to reorder the database, by PID and then ApptDate within ID.
- Start by looking at a single patient's record (it is not sorted):

```
outpatient[outpatient$PID == "P10141",] #Patient P10141's history

##          PID  SchedDate   ApptDate  Dept Language Sex Age
## 926   P10141  12/7/2011 12/15/2011   CPO   ENGLISH   F 80+
## 1183 P10141 12/23/2012  1/21/2013   CPO   ENGLISH   F 80+
## 1535 P10141 12/15/2011  4/15/2012   CPO   ENGLISH   F 80+
## 1956 P10141 12/15/2011  4/15/2012   CPO   ENGLISH   F 80+
## 2499 P10141  5/27/2012  6/21/2012   CPO   ENGLISH   F 80+
## 2682 P10141  7/11/2013  8/14/2013   CPO   ENGLISH   F 80+
## 3186 P10141 12/23/2012  1/21/2013   CPO   ENGLISH   F 80+
## 3579 P10141  5/16/2012   6/6/2012   CPO   ENGLISH   F 80+
##           Race    Status
## 926   FILIPINO   Arrived
## 1183 FILIPINO Cancelled
## 1535 FILIPINO Cancelled
## 1956 FILIPINO   No Show
```

# Problems

- The way to do complicated sorts in R is by using the `order` command
- It returns the permutation vector that if applied to the original vector, would sort it
- Example:

```
my.data <- data.frame( #Some example data
        X = c(3,6,2,2,5,6,1,2,6,6,2,3),
        Y = c("B","A","B","D","A","C","A","B","D","A","C","A")
)
order(my.data$X) # Take a look at the "order" command applied to X

##  [1]  7  3  4  8 11  1 12  5  2  6  9 10

# Make number 7 in the original vector the first element
# Make number 3 in the original vector the second element
# Make number 10 in the original vector the last element
```

# Sorting

Sorting by the X column:

```
# You sort by applying the "order"
# permutation to the rows of the original object:
my.data[order(my.data$X),]

##    X Y
## 7  1 A
## 3  2 B
## 4  2 D
## 8  2 B
## 11 2 C
## 1  3 B
## 12 3 A
## 5  5 A
## 2  6 A
## 6  6 C
## 9  6 D
## 10 6 A
```

# Sorting on more than one variable

To sort on more than one columns, simply add the additional sorting arguments to `order`. The following code will sort on X, then Y within X:

```
my.data[order(my.data$X,my.data$Y),] # Sort on X, then Y
```

```
##    X Y
## 7  1 A
## 3  2 B
## 8  2 B
## 11 2 C
## 4  2 D
## 12 3 A
## 1  3 B
## 5  5 A
## 2  6 A
## 10 6 A
## 6  6 C
## 9  6 D
```

It worked – the data frame is sorted by X!

# Sorting the patient list

- We want to sort by `PID`, then `ApptDate` within PID.
- The problem is, that `ApptDate` has been read as a factor, not as a date, so won't sort correctly.
- We need to turn it into a date variable
- The commands to do this date coercion are `as.POSIXlt` and `as.POSIXct`. `POSIXlt` keeps the time as we think of it, seconds, hours, day, month year etc. `as.POSIXct` keeps the time as an integer, the number of seconds since Jan 1, 1970 (A Unix idea).
- Unfortunately if we just try `as.POSIXlt` on our dates we are in trouble (Excel's csv files do not save a legitimate date format!):

```
as.POSIXlt(outpatient$ApptDate)
```

```
## Error in as.POSIXlt.character(as.character(x), ...):  character
string is not in a standard unambiguous format
```

# Dealing with dates

- The fix up is to use the command `strptime` which can be used to convert arbitrary character strings to dates
- We need to inform R, how the dates are formatted

```
# This can read the dates because we are specifying
# the format of the date string
strptime(outpatient$SchedDate, format="%m/%d/%Y")
```

```
##    [1] "2012-07-27 EDT" "2013-11-28 EST" "2012-02-02 EST"
##    [4] "2011-11-08 EST" "2012-10-25 EDT" "2012-06-26 EDT"
##    [7] "2013-10-31 EDT" "2013-04-20 EDT" "2011-07-14 EDT"
##   [10] "2012-07-20 EDT" "2012-12-24 EST" "2011-07-16 EDT"
##   [13] "2011-11-14 EST" "2011-04-17 EDT" "2013-10-07 EDT"
##   [16] "2012-10-02 EDT" "2014-07-21 EDT" "2011-08-19 EDT"
##   [19] "2013-01-08 EST" "2013-07-17 EDT" "2011-07-31 EDT"
##   [22] "2012-11-03 EDT" "2013-01-25 EST" "2012-08-13 EDT"
##   [25] "2012-06-27 EDT" "2013-11-21 EST" "2011-09-24 EDT"
##   [28] "2013-12-17 EST" "2011-02-27 EST" "2013-09-25 EDT"
##   [31] "2012-04-13 EDT" "2014-08-29 EDT" "2012-05-22 EDT"
##   [34] "2012-01-13 EST" "2012-05-21 EDT" "2012-05-06 EDT"
```

# Sorting the patient list

- We will now turn the original "factor" dates into real dates
- And add a column to the data frame that calculates the difference in days between schedule time and appointment time

```
outpatient$SchedDate <- strptime(outpatient$SchedDate, format="%m/%d/%Y")
outpatient$ApptDate <- strptime(outpatient$ApptDate, format="%m/%d/%Y")

# The difference between the two dates (measured in seconds)
# This will be a new column in the data frame
outpatient$SchedLag <- (outpatient$ApptDate - outpatient$SchedDate)
# Turn it into days
outpatient$SchedLag <- as.numeric(outpatient$SchedLag)/(60 * 60 * 24)
```

# Sorting the patient list

- Now we have the date in the right format we can sort the data frame
- We will also only pull out the Status levels we care about (No Show and Arrived).

```
# A new sorted data frame
new.outpatient <- outpatient[order(outpatient$PID,
                                   outpatient$ApptDate),]

#Now subset by the outcomes we care about
new.outpatient <- new.outpatient[
 (new.outpatient$Status == "Arrived" | # Note the logical OR
  new.outpatient$Status == "No Show"),
                                  ]
```

# Sorting the patient list

- Just for fun, how does Schedule Lag predict Status? We need a logistic regression.
- Highly significant: the longer the lag, the higher the probability of a no show.

```
summary(glm(Status  ~ SchedLag, data = new.outpatient, family="binomial"))

##
## Call:
## glm(formula = Status ~ SchedLag, family = "binomial", data = new.outpati
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.0402  -0.6530  -0.5964  -0.5670   1.9574
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.756369   0.068657 -25.582  < 2e-16 ***
## SchedLag     0.009985   0.001297   7.697  1.4e-14 ***
## ---
```

# Building a patient history

- There are lots of approaches to doing this
- I will pull out each patient in turn as a mini-data frame and calculate summaries of interest
- I will calculate two: the number of prior visits and the proportion of prior no shows
- We will use a `for` loop to iterate over patients, populating the behavioural history data frame as we go
- We start by figuring out how many unique patients there are, and how many columns we want in the behavioral data frame.
- It can be much more efficient to do it this way than using commands like `rbind`

# Building a patient history

- We will have columns: PID, Sex, Age, Prior Visits, No Show Rate, Current Visit Lag, Current Visit Status
- We can find the number of patients with the `unique` command

```
#The number of unique patient IDs
num.pats <- length(unique(new.outpatient$PID))
num.pats

## [1] 885
```

# Building a patient history

- We will have 7 columns and 885 rows
- Start by filling the data frame in with NAs (they will be overwritten later)
- One issue is going to be with factor variables. Unless we keep them as characters, they get coerced later on to integers

```
behavior.df <- data.frame(
  PID = as.character(rep(NA,num.pats)),
  Sex = rep(NA,num.pats),
  Age = rep(NA,num.pats),
  Prior.Visits = rep(NA,num.pats),
  No.Show.Rate = rep(NA,num.pats),
  Current.Visit.Lag = rep(NA,num.pats),
  Current.Visit.Status = as.character(rep(NA,num.pats)),
  stringsAsFactors = FALSE #Stop the use of factors, keep as characters
)
```

# Building a patient history

- Now, extract a patient and summarize

```
#An example patient
test.patient <- new.outpatient[new.outpatient$PID =="P11000",]
print(test.patient)

##         PID  SchedDate   ApptDate          Dept Language Sex
## 483  P11000 2013-02-03 2013-02-12 ORTHOPAEDICS  ENGLISH   F
## 1026 P11000 2013-02-23 2013-03-11 ORTHOPAEDICS  ENGLISH   F
## 1016 P11000 2013-04-08 2013-04-27 ORTHOPAEDICS  ENGLISH   F
##      Age                  Race  Status SchedLag
## 483   19 WHITE (NON-HISPANIC) Arrived  9.00000
## 1026  19 WHITE (NON-HISPANIC) Arrived 15.95833
## 1016  19 WHITE (NON-HISPANIC) Arrived 19.00000
```

# Building a patient history

- Prior visits is 1 - the number of rows
- Prior no show rate is the number of prior no shows divided by the number of prior visits

```
# nrow is a function to count the number of rows
prior.visits <- nrow(test.patient) - 1

#Find the status of the prior visits, ignoring the current one
prop.no.show <- (# The proportion of no shows
sum( #This counts the number of No Shows
 test.patient[-nrow(test.patient),"Status"] #This pulls out the status
                                            #of prior visits
                == "No Show")  #This finds the "No Shows"
                             / prior.visits #Normalize by the number of prior
                             )
```

# Building a patient history

- We will put these two calculations into a single function that takes a patient transaction data frame and returns a single row data frame with the variables we care about
- Later we will iterate over the patients and drop this into the behavior.df data frame
- This function does not fit on this slide, we'll review it in the R code file

```
# The extraction function takes a patient history data frame
extract.hist <- function(df){
# Special case: there is a single record, no history
if(nrow(df) == 1){
return(data.frame(
  PID = as.character(df[1,"PID"]),
  Sex = df[1,"Sex"],
  Age = df[1,"Age"],
  Prior.Visits = 0, #No prior visits
  No.Show.Rate = NA, #Undefined prior no show rate
  Current.Visit.Lag = df[1,"SchedLag"],
```

# Testing the code

```
#Someone with 1 visit
new.outpatient[new.outpatient$PID == "P10001",]

##         PID  SchedDate   ApptDate           Dept Language
## 2599 P10001 2012-11-23 2012-11-26  NEUROLOGICAL   ENGLISH
##      Sex Age                 Race  Status SchedLag
## 2599   M  45 WHITE (NON-HISPANIC) Arrived        3

extract.hist(new.outpatient[new.outpatient$PID == "P10001",])

##      PID Sex Age Prior.Visits No.Show.Rate
## 1 P10001   M  45            0           NA
##   Current.Visit.Lag Current.Visit.Status
## 1                 3              Arrived
```

# Testing the code

```
#Someone with multiple visits
new.outpatient[new.outpatient$PID == "P10127",]

##         PID  SchedDate   ApptDate  Dept Language Sex Age
## 1395 P10127 2014-07-05 2014-07-27  CPO   ENGLISH   M  33
## 1764 P10127 2014-08-03 2014-08-18  CPO   ENGLISH   M  33
## 2390 P10127 2014-08-17 2014-08-20  CPO   ENGLISH   M  33
## 534  P10127 2014-08-20 2014-09-06  CPO   ENGLISH   M  33
##                  Race  Status SchedLag
## 1395 AFRICAN AMERICAN Arrived       22
## 1764 AFRICAN AMERICAN No Show       15
## 2390 AFRICAN AMERICAN Arrived        3
## 534  AFRICAN AMERICAN Arrived       17

extract.hist(new.outpatient[new.outpatient$PID == "P10127",])

##      PID Sex Age Prior.Visits No.Show.Rate
## 1 P10127   M  33            3    0.3333333
##   Current.Visit.Lag Current.Visit.Status
## 1                17               Arrived
```

# Finishing it off: applying the `for` loop

```
row.counter <- 1 #start by filling the first row
for(pid in unique(new.outpatient$PID)){ #iterate over id's
 behavior.df[row.counter, ] <- extract.hist(df = # Pull out history
                     new.outpatient[new.outpatient$PID == pid,])
row.counter <- row.counter + 1 # Increment the row counter by 1
 }
```

# Finishing it off: applying the `for` loop

Ready for analysis, with behavioral history variables

```
# Have a look at the new behavioural data frame
head(behavior.df,5)

##      PID Sex Age Prior.Visits No.Show.Rate
## 1 P10001   2  41            0           NA
## 2 P10002   2   4            0           NA
## 3 P10003   2   4            0           NA
## 4 P10004   2  36            2            0
## 5 P10005   1  18            0           NA
##   Current.Visit.Lag Current.Visit.Status
## 1            3.0000               Arrived
## 2           19.0000               Arrived
## 3          121.0417               Arrived
## 4            7.0000               Arrived
## 5           52.0000               No Show
```

# Big picture: roll-up to patient level

- Sort by patient ID and Appointment date
- Extract each patient's complete transaction history
- Create summaries of each patient to populate a new patient level data frame

# Predefine the dimensions of the patient level data frame

- Though R has natural commands to build up vectors, matrices and data frames (c, rbind and cbind) repeated application of these constructs within a loop can be very inefficient.
- I recommend that you pre-calculate the dimensions of the data frame, create it as a "dummy" object, then fill it via the indexing operator.

# Module summary

Topics covered today include:

- A case study using the functions and idioms learnt to date
- Date functions
- Sorting

# Next time

- We have all the building blocks in place. Time to write some useful code.
- Monte Carlo simulations.

# Today's function list

Do you know what each of these functions does?

```
as.POSIXlt
nrow
order
strptime
unique
```