Holsenbeck_S_5

Stephen Synchronicity

2017-10-09

Homework 5

Note: The last R chunk for section 4 uses 4 mapply functions with control loops that can take up to 10 minutes to compute. The chunk options from this chunk on are set to 'eval=F' to prevent this from running until manually done so to prevent any confusion if all chunks were run and R computed for 10+ mins.

1

(10 points) Add a new column `Season1Days` that contains the number of days a market is opened per week (for the dates it is open).

```
fmdb <- as.tibble(read_csv(file = "C:\\Users\\Stephen\\Documents\\Northeastern\\DA 5020 - Collect Store
```

```
# fmdb <- fmdb %>%
# separate(col='Season1Date',into=c('S1B','S1E'),sep='to',remove=F) m2n<-
# function (x){match(tolower(x), tolower(month.name))} m2n('January') Track
# whether it's open or closed
fmdb$Season1Days <- str_count(fmdb$Season1Time, "[[:alpha:]]{3}")
# separate(col='Season1Time',into=c('S1D','S1T'),sep=':',remove=F,extra='merge')
```

2

(10 points) Add a new column `WeekendOpen` indicating whether a market opens during weekends in `Season1`

```
fmdb$WeekendOpen <- str_replace(fmdb$Season1Time, "(?:.*)((?:Sat:[\\s:0-9AaPpMm-]+;)+)(?:.*)|(?:.*)((?:S
    "\\1\\2")
```

3

(20 points) Find out which markets close before 6PM, and which open only for fewer than 4 hours a day. For simplicity, consider only `Season1Time`. For markets with different open hours across a week, use the average length of open hours for the days they actually open.

```
#Markets that close before 6p
(fmtimes <- fmdb %>% #create dataframe with individual day times
   select(FMID,Season1Time) %>%
   mutate(Mon = str_extract(fmdb$Season1Time,"((?:Mon:[\\s:0-9AaPpMm-]+;)+)"),
          Tue = str_extract(fmdb$Season1Time,"((?:Tue:[\\s:0-9AaPpMm-]+;)+)"),
          Wed = str_extract(fmdb$Season1Time,"((?:Wed:[\\s:0-9AaPpMm-]+;)+)"),
          Thu = str_extract(fmdb$Season1Time,"((?:Thu:[\\s:0-9AaPpMm-]+;)+)"),
          Fri = str_extract(fmdb$Season1Time,"((?:Fri:[\\s:0-9AaPpMm-]+;)+)"),
          Sat = str_extract(fmdb$Season1Time,"((?:Sat:[\\s:0-9AaPpMm-]+;)+)"),
          Sun = str_extract(fmdb$Season1Time,"((?:Sun:[\\s:0-9AaPpMm-]+;)+)"))
)
```

```
## # A tibble: 8,707 x 9
##       FMID                                  Season1Time   Mon
##      <int>                                        <chr> <chr>
##  1 1018261                        Wed: 9:00 AM-1:00 PM;  <NA>
##  2 1018318                        Sat: 9:00 AM-1:00 PM;  <NA>
##  3 1009364                                         <NA>  <NA>
##  4 1010691   Wed: 3:00 PM-6:00 PM;Sat: 8:00 AM-1:00 PM;  <NA>
##  5 1002454 Tue:8:00 am - 5:00 pm;Sat:8:00 am - 8:00 pm;  <NA>
##  6 1011100                        Tue: 3:30 PM-6:30 PM;  <NA>
##  7 1009845                       Tue: 10:00 AM-7:00 PM;  <NA>
##  8 1005586                       Fri: 8:00 AM-11:00 AM;  <NA>
##  9 1008071                        Sat: 9:00 AM-1:00 PM;  <NA>
## 10 1012710                        Sat: 9:00 AM-1:00 PM;  <NA>
## # ... with 8,697 more rows, and 6 more variables: Tue <chr>, Wed <chr>,
## #   Thu <chr>, Fri <chr>, Sat <chr>, Sun <chr>
```

```
fmsix <- fmtimes %>% mutate(After6 = str_extract(Season1Time, pattern = regex("((?:[7-9]).{2,3}\\s[PpMm]
filter(fmsix, is.na(fmsix$After6))  #filter by those without times, ie close before 6 for verification
```

```
## # A tibble: 7,718 x 10
##       FMID                                Season1Time   Mon
##      <int>                                      <chr> <chr>
##  1 1018261                      Wed: 9:00 AM-1:00 PM;  <NA>
##  2 1018318                      Sat: 9:00 AM-1:00 PM;  <NA>
##  3 1009364                                       <NA>  <NA>
##  4 1010691 Wed: 3:00 PM-6:00 PM;Sat: 8:00 AM-1:00 PM;  <NA>
##  5 1011100                      Tue: 3:30 PM-6:30 PM;  <NA>
##  6 1005586                     Fri: 8:00 AM-11:00 AM;  <NA>
##  7 1008071                      Sat: 9:00 AM-1:00 PM;  <NA>
##  8 1012710                      Sat: 9:00 AM-1:00 PM;  <NA>
##  9 1018792                      Wed: 2:30 PM-6:30 PM;  <NA>
## 10 1016782                      Tue: 8:00 AM-6:00 PM;  <NA>
## # ... with 7,708 more rows, and 7 more variables: Tue <chr>, Wed <chr>,
## #   Thu <chr>, Fri <chr>, Sat <chr>, Sun <chr>, After6 <chr>
```

```
wd <- data.frame(c("Mon","Tue","Wed","Thu","Fri","Sat","Sun"),"sta","end",stringsAsFactors = F)
wd$Start  <- do.call(paste, c(wd[1],wd[2],sep = "."))
wd$End  <- do.call(paste, c(wd[1],wd[3],sep = "."))#Vector for naming

fmsix <- fmsix %>% #Sepearate the times into cols
  separate(col=Mon,into=c("Mon","Times"),sep=":",extra="merge") %>%
  separate(col=Times,into=c(wd$Start[1],wd$End[1]),sep="-",extra="merge") %>%
   separate(col=Tue,into=c("Tue","Times"),sep=":",extra="merge") %>%
  separate(col=Times,into=c(wd$Start[2],wd$End[2]),sep="-",extra="merge") %>%
   separate(col=Wed,into=c("Wed","Times"),sep=":",extra="merge") %>%
  separate(col=Times,into=c(wd$Start[3],wd$End[3]),sep="-",extra="merge") %>%
   separate(col=Thu,into=c("Thu","Times"),sep=":",extra="merge") %>%
  separate(col=Times,into=c(wd$Start[4],wd$End[4]),sep="-",extra="merge") %>%
   separate(col=Fri,into=c("Fri","Times"),sep=":",extra="merge") %>%
  separate(col=Times,into=c(wd$Start[5],wd$End[5]),sep="-",extra="merge") %>%
   separate(col=Sat,into=c("Sat","Times"),sep=":",extra="merge") %>%
  separate(col=Times,into=c(wd$Start[6],wd$End[6]),sep="-",extra="merge") %>%
   separate(col=Sun,into=c("Sun","Times"),sep=":",extra="merge") %>%
  separate(col=Times,into=c(wd$Start[7],wd$End[7]),sep="-",extra="merge")
tCols <- grep("(.*.sta)|(.*.end)",names(fmsix)) #Vector of cols needing time transforms
```

```
#strptime(fmsix$Tue.sta[5],format="%H:%M") Test Function
fmsix <- cbind(fmsix,as.data.frame(apply(fmsix[,tCols], 2, function(x) strptime(x, "%I:%M %p")))) #Conv
fmsix <- subset(fmsix, select=-seq(from=(min(tCols)-1),to=max(tCols))) #remove previous time string col
fmsix <- fmsix %>% #Add cols w/ numeric # of hrs difference
  mutate(Mon.hrs = as.numeric(abs(difftime(Mon.sta,Mon.end,units="hours"))),
         Tue.hrs = as.numeric(abs(difftime(Tue.sta,Tue.end,units="hours"))),
         Wed.hrs = as.numeric(abs(difftime(Wed.sta,Wed.end,units="hours"))),
         Thu.hrs = as.numeric(abs(difftime(Thu.sta,Thu.end,units="hours"))),
         Fri.hrs = as.numeric(abs(difftime(Fri.sta,Fri.end,units="hours"))),
         Sat.hrs = as.numeric(abs(difftime(Sat.sta,Sat.end,units="hours"))),
         Sun.hrs = as.numeric(abs(difftime(Sun.sta,Sun.end,units="hours"))))
tCols <- grep("(.*.hrs)",names(fmsix)) #refresh the cols vector
fmsix <- transform(fmsix, mHrs = rowMeans(fmsix[,tCols], na.rm = TRUE)) #add a col with means
fmsix %>%
  filter(mHrs<4) #filter for those less than 4

#tCols <- grep("(.*.sta)|(.*.end)",names(fmsix)) #refresh the cols vector
#eCols <- seq(from=min(tCols),to=max(tCols),by=2)
# test <- for(i in eCols){
#   print(i)
#   as.data.frame(apply(fmsix[,eCols],2,function(i) difftime(fmsix[,i],fmsix[,(i+1)],units="hours")))
# }
```

4

(40 Points) The seasons are not standardized and would make analysis difficult. Create four new columns for four seasons (Spring, Summer, Fall, Winter), indicating whether a market is available in that season. Also, create two additional columns `HalfYear` and `YearRound` to identify those who open across seasons. Define "half year" and "year round" on your own terms, but explain them before you write the code (or as comments in your code). (Hint: you may want to create even more auxiliary columns, `Season1BeginDate` and `Season1EndDate` for example.)

```
fmSeasons <- fmdb %>% select(FMID, Season1Date, Season2Date, Season3Date, Season4Date) %>%
    separate(Season1Date, into = c("S1.beg", "S1.end"), sep = " to ", extra = "merge") %>%
    separate(Season2Date, into = c("S2.beg", "S2.end"), sep = " to ", extra = "merge") %>%
    separate(Season3Date, into = c("S3.beg", "S3.end"), sep = " to ", extra = "merge") %>%
    separate(Season4Date, into = c("S4.beg", "S4.end"), sep = " to ", extra = "merge")
# month.num<- seq(as.Date('2017/1/1'), by = 'month', length.out = 12)
```

```
tCols <- grep("(S\\d.beg)|(S\\d.end)", names(fmSeasons))  #refresh the cols vector
# month.namestonums <- function(x) { str_replace_all(x,'January','1')
# str_replace_all(x,'February','2') str_replace_all(x,'March','3')
# str_replace_all(x,'April','4') str_replace_all(x,'May','5')
# str_replace_all(x,'June','6') str_replace_all(x,'July','7')
# str_replace_all(x,'August','8') str_replace_all(x,'September','9')
# str_replace_all(x,'October','10') str_replace_all(x,'November','11')
# str_replace_all(x,'December','12') } fmSeasons <-
# apply(fmSeasons,2,function(x){month.namestonums(x)})

fmSeasons <- apply(fmSeasons, 2, function(x) {
    str_replace_all(x, "January", "1")
})
fmSeasons <- apply(fmSeasons, 2, function(x) {
```

```
    str_replace_all(x, "February", "2")
})
fmSeasons <- apply(fmSeasons, 2, function(x) {
    str_replace_all(x, "March", "3")
})
fmSeasons <- apply(fmSeasons, 2, function(x) {
    str_replace_all(x, "April", "4")
})
fmSeasons <- apply(fmSeasons, 2, function(x) {
    str_replace_all(x, "May", "5")
})
fmSeasons <- apply(fmSeasons, 2, function(x) {
    str_replace_all(x, "June", "6")
})
fmSeasons <- apply(fmSeasons, 2, function(x) {
    str_replace_all(x, "July", "7")
})
fmSeasons <- apply(fmSeasons, 2, function(x) {
    str_replace_all(x, "August", "8")
})
fmSeasons <- apply(fmSeasons, 2, function(x) {
    str_replace_all(x, "September", "9")
})
fmSeasons <- apply(fmSeasons, 2, function(x) {
    str_replace_all(x, "October", "10")
})
fmSeasons <- apply(fmSeasons, 2, function(x) {
    str_replace_all(x, "November", "11")
})
fmSeasons <- apply(fmSeasons, 2, function(x) {
    str_replace_all(x, "December", "12")
})
fmSeasons <- apply(fmSeasons, 2, function(x) {
    gsub("(^\\d{1,2}\\b(?!\\/))", "\\1\\/1\\/2017", perl = T, x)
})  # Make months into date string
fmSeasons <- apply(fmSeasons, 2, function(x) {
    gsub("(\\d{1,2}\\/\\d{1,2}\\/\\d{2,4}).*", "\\1", perl = T, x)
})  #Make sure all are mdy dates
fmSeasons <- as.tibble(fmSeasons)
fmSeasons <- fmSeasons %>% mutate_at(vars(matches("(beg)|(end)")), funs(mdy)) %>%
    mutate(s1.int = S1.beg %--% S1.end, s2.int = S2.beg %--% S2.end, s3.int = S3.beg %--%
        S3.end, s4.int = S4.beg %--% S4.end)
# Table with dates and intervals
fmInts <- fmSeasons %>% mutate_at(vars(s1.int:s4.int), funs(as.period(.)))  #
# With intervals as periods
```

```
# all markets longer than 11mo
tCols <- grep("s\\d.int", names(fmSeasons))  #refresh the cols vector
fmYR <- fmSeasons %>% mutate_at(vars(s1.int:s4.int), funs(as.period(.))) %>% mutate(yearRound = ifelse(
    as.period(months(11), unit = "seconds"), T, F))
filter(fmYR, yearRound == T)
```

```
## # A tibble: 724 x 14
##       FMID     S1.beg     S1.end S2.beg S2.end S3.beg S3.end S4.beg S4.end
```

```
##      <chr>     <date>     <date> <date> <date> <date> <date> <date> <date>
##  1 1012158 2016-01-01 2016-12-31     NA     NA     NA     NA     NA     NA
##  2 1009959 2016-01-01 2016-12-31     NA     NA     NA     NA     NA     NA
##  3 1010775 2015-01-18 2020-01-05     NA     NA     NA     NA     NA     NA
##  4 1005636 2013-01-01 2013-12-31     NA     NA     NA     NA     NA     NA
##  5 1000061 2013-01-01 2013-12-31     NA     NA     NA     NA     NA     NA
##  6 1000062 2013-01-01 2013-12-31     NA     NA     NA     NA     NA     NA
##  7 1000064 2013-01-01 2013-12-31     NA     NA     NA     NA     NA     NA
##  8 1009004 2014-01-01 2014-12-31     NA     NA     NA     NA     NA     NA
##  9 1000065 2013-01-01 2013-12-31     NA     NA     NA     NA     NA     NA
## 10 1011313 2016-01-01 2016-12-01     NA     NA     NA     NA     NA     NA
## # ... with 714 more rows, and 5 more variables: s1.int <S4: Period>,
## #   s2.int <S4: Period>, s3.int <S4: Period>, s4.int <S4: Period>,
## #   yearRound <lgl>
```

```
# All Markets longer than 6mo and less than 11 mo
fmHY <- fmInts %>% mutate(halfYear = ifelse(!is.na(fmInts$s1.int) & fmInts$s1.int <
    as.period(months(11)) & fmInts$s1.int > as.period(months(6)), T, NA))
filter(fmHY, halfYear == T)
```

```
## # A tibble: 661 x 14
##       FMID     S1.beg     S1.end     S2.beg     S2.end     S3.beg     S3.end S4.beg S4.end
##      <chr>     <date>     <date>     <date>     <date>     <date>     <date> <date> <date>
##  1 1010691 2014-04-02 2014-11-30         NA         NA         NA         NA     NA     NA
##  2 1008071 2014-05-03 2014-11-22         NA         NA         NA         NA     NA     NA
##  3 1012710 2016-04-09 2016-11-19         NA         NA         NA         NA     NA     NA
##  4 1012790 2016-09-03 2017-06-24 2017-09-02 2018-06-30         NA         NA     NA     NA
##  5 1000060 2013-04-20 2013-12-21         NA         NA         NA         NA     NA     NA
##  6 1018426 2017-05-13 2017-12-16         NA         NA         NA         NA     NA     NA
##  7 1010487 2016-04-16 2016-11-22 2016-11-26 2016-12-24 2016-01-02 2016-04-09     NA     NA
##  8 1004086 2014-10-04 2015-07-25         NA         NA         NA         NA     NA     NA
##  9 1004425 2017-05-01 2017-12-01         NA         NA         NA         NA     NA     NA
## 10 1002175 2016-04-16 2016-11-19         NA         NA         NA         NA     NA     NA
## # ... with 651 more rows, and 5 more variables: s1.int <S4: Period>,
## #   s2.int <S4: Period>, s3.int <S4: Period>, s4.int <S4: Period>,
## #   halfYear <lgl>
```

```
rm(fmInts)
```

```
fmSeasonsDetail <- left_join(fmSeasons, fmYR[, c(1, 14)], by = "FMID")
fmSeasonsDetail <- left_join(fmSeasonsDetail, fmHY[, c(1, 14)], by = "FMID")
```

```
# Testing Int_overlaps, returns logical
# int_overlaps(fmSeasonsDetail$s1.int[[1]],interval(ymd('2017-2-1'),ymd('2017-7-1')))
seasons <- tibble::tribble(~Date, "March 20", "June 21", "September 22", "December 21")  # Seasons Tabl
seasons <- mdy(paste(seasons$Date, year(today())))  # makes dates
spring <- interval(seasons[1], seasons[2])
summer <- interval(seasons[2], seasons[3])
fall <- interval(seasons[3], seasons[4])
winter <- interval(seasons[4] - years(1), seasons[1])  # define each as intervals
# Season overlap functions Input: Market interval Function: defines seasonal
# intervals based on yr of input interval Output: Returns overlap of market
# interval and season interval
sOlp.sp <- function(x) {
    if (!is.na(x)) {
        y <- year(int_start(x))
```

```
        year(seasons) <- y
        sn <- seasons
        sp <- interval(sn[1], sn[2])
        return(intersect(x, sp))
    } else {
        return(NA)
    }
}
sOlp.su <- function(x) {
    if (!is.na(x)) {
        y <- year(int_start(x))
        year(seasons) <- y
        sn <- seasons
        su <- interval(sn[2], sn[3])
        return(intersect(x, su))
    } else {
        return(NA)
    }
}
sOlp.fa <- function(x) {
    if (!is.na(x)) {
        y <- year(int_start(x))
        year(seasons) <- y
        sn <- seasons
        fa <- interval(sn[3], sn[4])
        return(intersect(x, fa))
    } else {
        return(NA)
    }
}
sOlp.wi <- function(x) {
    if (!is.na(x)) {
        y <- year(int_start(x))
        year(seasons) <- y
        sn <- seasons
        year(sn[1]) <- y + 1
        wi <- interval(sn[4], sn[1])
        return(intersect(x, wi))  #return overlap
    } else {
        return(NA)
    }
}
# test <- test %>% group_by(s1.int) %>% filter(!is.na(S1.end))%>% does not work
# to filter table, fn(x)s updated to # handle Na mutate(Spring=sOlp.sp(s1.int))
# is.na(test$s1.int[[3]]) sOlp.sp(test$s1.int[[1]]) #Testing the Functions
```

It's worth noting here that the winter season presents quite a bit of complexity when attempting to find the overlap between the winter season and the market intervals. If I attempted to continue with the same inquiry of determining the overlap interval with each season, there would need to an iterative loop for winter, that accounted for five different cases, which are: 1. The beginning of the market inverval overlaps with winter. 2. The end of the market inverval overlaps with winter. 3. The beginning and end of the market inverval overlaps with winter, with no gap (year round). 4. The beginning and end of the market inverval overlaps with winter, with a gap (during December/the holidays). 5. The market interval does not overlap with winter. Due to time constraints, rather than get mired in writing this loop and then having to mapply

it (which will likely take forever) I'm going to stick with the same method used for the previous seasons. While imperfect, it will have to do in this instance.

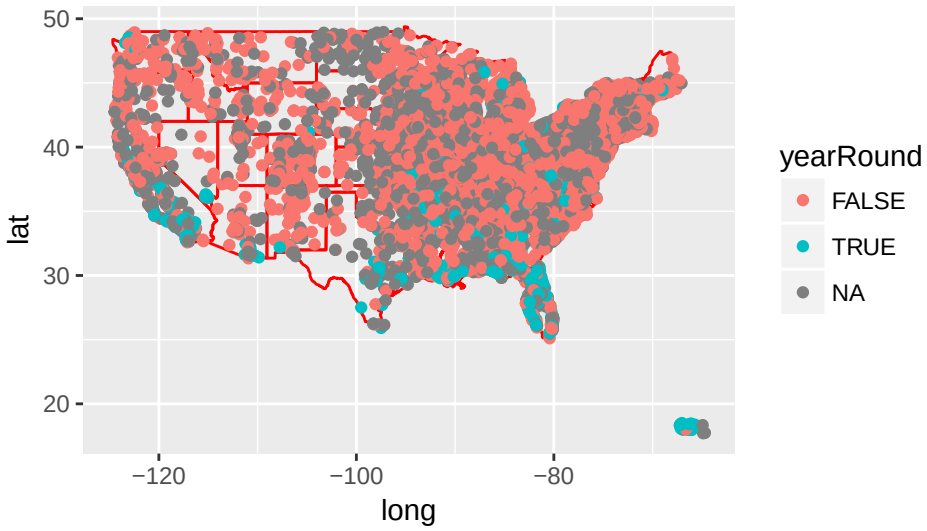
```
# Create seasonal interval dataframes
test <- fmSeasonsDetail
Spring <- as.data.frame(do.call(c, mapply(sOlp.sp, test$s1.int, SIMPLIFY = F)))
colnames(Spring) <- "Spring"
Summer <- as.data.frame(do.call(c, mapply(sOlp.su, test$s1.int, SIMPLIFY = F)))
colnames(Summer) <- "Summer"
Fall <- as.data.frame(do.call(c, mapply(sOlp.fa, test$s1.int, SIMPLIFY = F)))
colnames(Fall) <- "Fall"
Winter <- as.data.frame(do.call(c, mapply(sOlp.wi, test$s1.int, SIMPLIFY = F)))
colnames(Winter) <- "Winter"
```

```
fmSeasonsDetail <- cbind(fmSeasonsDetail, Spring, Summer, Fall, Winter)  #View all in single data frame
rm(Spring, Summer, Fall, Winter, test, fmSeasons, fmHY, fmYR)  #Free up Space
fmSeasonsDetail$FMID <- as.numeric(fmSeasonsDetail$FMID)
fmdb <- left_join(fmdb, fmSeasonsDetail[, c(1, 14:19)], by = "FMID")
```

5

(20 points) *Open question*: explore the new variables you just created. Aggregate them at different geographic levels, or some other categorical variable. What can you discover?

```
# prereqs install.packages(c('maps', 'mapdata'))
# devtools::install_github('dkahle/ggmap')
library(maps)
library(mapdata)
library(ggplot2)
library(ggmap)
states <- map_data("state")
usmap <- ggplot(data = states) + geom_polygon(aes(x = long, y = lat, fill = F, group = group),
    fill = NA, color = "red") + coord_fixed(1.3) + guides(fill = FALSE)
cont.usa <- fmdb %>% filter(State != "Alaska" & State != "Hawaii")
usmap + geom_point(data = cont.usa, mapping = aes(x = x, y = y, color = yearRound))
```




```
# Source:http://eriqande.github.io/rep-res-web/lectures/making-maps-with-R.html
```

A)With few exceptions, the year round farmer's markets are located primarily in the southern territories including Puerto Rico and the US Virgin Islands. Note: Alaska and Hawaii not included due to the distortion of scale.

Below is a test module that was used when lubridate was malfunctioning. A restart ameliorated the issue. I kept the code in case a bug report needed to be made.

```
seasons <- tibble::tribble(~Date, "March 20", "June 21", "September 22", "December 21")  # Seasons Tabl
seasons <- mdy(paste(seasons$Date, year(today())))  # makes dates
spring <- interval(seasons[1], seasons[2])
summer <- interval(seasons[2], seasons[3])
fall <- interval(seasons[3], seasons[4])
winter <- interval(seasons[4] - years(1), seasons[1])  # define each as intervals

sOlp.wi <- function(x) {
    if (!is.na(x)) {
        y <- year(int_start(x))
        year(seasons) <- y
        sn <- seasons
        year(sn[1]) <- y + 1
        wi <- interval(sn[4], sn[1])
        return(intersect(x, wi))  #return overlap
    } else {
        return(NA)
    }
}
fmSeasons$s1.int[[2870]]
(test <- interval("2017-01-01", "2017-12-31"))
sOlp.wi(fmSeasons$s1.int[[2870]])
```