This house(ing market) is on fire! Using American Community Survey and Zillow data to explore housing affordability for protective service workers

Background

When it comes to buying a house, there is a lot more to consider than just the sticker price. Wages and salaries vary widely across the country, including within specific occupations, as does the share of household income local residents are accustomed to dedicating to housing costs each month. At one extreme, some places with low housing costs might appear to be very affordable, but incomes might also be much lower than elsewhere; at the other extreme, some places that appear to be extremely expensive when looking at prices, might be more manageable as a result of relatively high wages and salaries.

By combining Zillow (www.zillow.com/research/data) data on home values, rents, and historical housing costs burdens with data on incomes by occuption from the United States Census Bureau's American Community Survey (https://www.census.gov/programs-surveys/acs/) (ACS), we can compare housing affordability for specific types of workers in different communities across the country. In this example, we estimate the share of a household's income that goes to a monthly mortgage payment on the median home across the country's metro areas for two types of workers:

- Fire fighting and prevention, and other protective service workers, including supervisors (who we broadly label *firefighters*),
- Law enforecement workers, including supervisors (who we broadly label *police officers*).

Getting Started

This tutorial will guide you through the steps to load and wrangle Zillow and ACS data to compute the share of monthly income required to buy a typical home for firefighters and police officers across the country using the R programming language.

To get started quickly, the code for this tutorial can be found at the following Github repo ([INSERT%20LINK]).

Part 1: Preliminaries

Start by clearing your environment and setting your options, and calling the following libraries:

- **plyr**: Splitting, applying and combining data
- **dplyr**: Data manipulation
- **readr**: Fast loading of tabular data into R
- **readxl**: Fast loading of Microsoft Excel files into R
- **reshape2**: Transform data between long and wide formats

- **stringi**: Transform strings
- **datasets**: Data set with the names and abbreviations of the 50 states of the United States of America (required only for state-level summary files, see pages 9-10 in the ACS Summary File Technical Documentation (http://www2.census.gov/programs-surveys/acs/summary_file/2014/documentation/tech_docs/2014_SummaryFile_Tech_Doc.pdf))

We call the libraries using a function, *pkgTest*, which checks whether a particular package is loaded in your environment, and if not, installs it.

```
## Preliminaries
rm(list=ls())

## This function will check if a package is installed, and if not, install it
pkgTest <- function(x) {
  if (!require(x, character.only = TRUE))
  {
    install.packages(x, dep = TRUE)
    if(!require(x, character.only = TRUE)) stop("Package not found")
  }
}

## These lines load the required packages
packages <- c("plyr", "dplyr", "readr", "readxl", "reshape2", "stringi", "dataset
s")
lapply(packages, pkgTest)
```

```
## [[1]]
## NULL
##
## [[2]]
## NULL
##
## [[3]]
## NULL
##
## [[4]]
## NULL
##
## [[5]]
## NULL
##
## [[6]]
## NULL
##
## [[7]]
## NULL
```

```
## These lines set several options
options(scipen = 999) # Do not print scientific notation
options(stringsAsFactors = FALSE) ## Do not load strings as factors
```

Part 2: Loading incomes and Census geographies from the American Community Survey

The first step is to load tables from the ACS Summary File FTP (http://www2.census.gov/programs-surveys/acs/summary_file/). There are several decisions to make:

- **Which year of data do you want?** ACS data are available starting in 2005 (at the time of this writing, 2014 is the most recent year).
- **Which Summary File folder do you want?** For the 2014 ACS there are three options for the 1-year ACS data and three options for the 5-year ACS data: an *All-in-one* file, a *State tables* file, and *topic tables*. Read section 2.2 of the technical documentation (http://www.census.gov/programs-surveys/acs/technical-documentation/summary-file-documentation.2014.html), titled *Summary File Organization*, to select the file folder best for your needs. After reading the ACS 2014 Technical Documentation (http://www.census.gov/programs-surveys/acs/technical-documentation/summary-file-documentation.html), we select the *1-year topic tables*.
- **Which sequence, table and line number(s) do you want?** Using the Summary File Lookup files (https://www.census.gov/programs-surveys/acs/technical-documentation/summary-file-documentation.html), identify the sequence, table and line numbers associated with the variable you are interested in (available under the section *Sequence Number/Table Number Lookup File*). For this analysis, we are interested in table B24011, which contains median annual earnings by industry, occupation and class of worker, and is found in sequence 107. Lines 21 and 22 contain the median annual earnings of firefighters and police officers, respectively.
- **Which geographies do you want?** There are two decisions here, both associated with the geographic level you want to analyze. First, you need to select the geographic level required for downloading the appropriate ACS geography, then you need to set the geographic summary level for your analysis.
    - If the geographic summary level you are interested in crosses state lines, the data will be contained in the geographic file for the *United States*. If the geographic summary level you are interested in is fully contained within state lines, the data will be contained in the geographic file for each state. The first table on page 10 (http://www2.census.gov/programs-surveys/acs/summary_file/2014/documentation/tech_docs/2014_SummaryFile_Tech_Doc.pdf) in the ACS Technical Documentation can help you determine which file is right for you. Since our analysis will be at the metropolitan statistical area level, we load the 'United States' file. For an example of how to load data for geographies fully contained within state lines - such as cities, Zip codes, school districts, and Congressional districts - see the Appendix below.
    - Identify the summary code associated with your geographic level of interest by referring to this ACS Geographic Summary Level table (https://www.census.gov/geo/maps-data/data/summary_level.html). The Summary Level Code for metro areas is 310.

Part 2.1: Set parameters

Based on the above decisions, set your parameters appropriately.

```
## Set parameters to load ACS data
year <- 2014
summaryFileFolder <- '1_year_seq_by_state'
geoLevelFile <- 'United States'
geoSummaryLevel <- 310
sequence <- 107
tableNumber <- 'B24011'
Lines <- c(21, 22)
acsDrive <- file.path('http://www2.census.gov',
                      'programs-surveys',
                      'acs',
                      'summary_file',
                      as.character(year),
                      'data')
```

Part 2.2: Load data tables

Next we load the ACS and geography tables.

```
## Load geographies
geos <- read_csv(file.path(acsDrive,
                           summaryFileFolder,
                           gsub(" ", "", geoLevelFile, fixed = TRUE),
                           paste0('g', as.character(year), 1, 'us', '.csv')),
                 col_names = FALSE)

## Load ACS tables
temp <- tempfile()
path <- file.path(acsDrive,
                  summaryFileFolder,
                  gsub(" ", "", geoLevelFile, fixed = TRUE),
                  paste0(as.character(year), 1, 'us',
                         stri_pad_left(sequence, 4, pad = '0'),
                         '000', '.zip'))
download.file(path, destfile = temp)
acs <- read_csv(unz(temp,
                    paste0('e', as.character(year), '1', 'us',
                           stri_pad_left(sequence, 4, pad = '0'),
                           '000', '.txt')),
                col_names = FALSE)
unlink(temp)
```

The following command, which appears several times when loading the data, removes spaces from the *geoLevelFile* name.

```
gsub(" ", "", geoLevelFile, fixed = TRUE)
```

Similarly, the function *stri_pad_left* (https://cran.r-project.org/web/packages/stringi/stringi.pdf) takes the sequence number and converts it to a character string of a specified length, padding the string with leading characters (zeros in this case).

These two functions are particularly important when loading data for sub-state geographies, as described in the Appendix.

Part 2.3: Load variable names

Then we load the associated variable names.

```
## Download variable and geography names
temp <- tempfile()
path <- file.path(acsDrive,
                  paste0(as.character(year),
                         '_1yr_Summary_FileTemplates',
                         '.zip'))
download.file(path, temp)

## Load variable and geography names
acsNames <- read_excel(unzip(temp,
                             file.path(
                               paste0(as.character(year),
                               '_1yr_Summary_FileTemplates'),
                               paste0('Seq',
                                      as.character(sequence),
                                      '.xls'))),
                       sheet = 'E')

geoNames <- read_excel(unzip(temp,
                             paste0(
                               as.character(year),
                               '_',
                               'SFGeoFileTemplate',
                               '.xls')),
                       sheet = 1)

## Close download link
unlink(temp)
```

Part 2.4: Apply variable names to data tables

Next we reformat the column names and apply them to the data tables.

```
## Reformat variable and geography names, then apply them to the data tables
acsNames <- data.frame(varName = colnames(acsNames), varDesc = t(acsNames[1, ]))
geoNames <- data.frame(geoName = colnames(geoNames), geoDesc = t(geoNames[1, ]))
colnames(acs) <- acsNames$varName
colnames(geos) <- geoNames$geoName
```

Part 2.5: Get rid of data that we don't need

Because of the table structure in the ACS FTP, we loaded a lot of ACS data that we don't actually need for this particular analysis, so let's keep only the variables that we're planning on using: The geographic identifier(s) and the data variables.

```
## Keep only variables of interest in ACS data
acsVars <- paste(tableNumber, as.character(stri_pad_left(Lines, 3, '0')), sep='_')
acs <- acs[, c('STUSAB', 'LOGRECNO', acsVars)]
acs$STUSAB <- toupper(acs$STUSAB)
```

Similarly, we loaded a lot of geographies that we do not plan on using for this analysis. Let's get rid of that data.

We should also extract the CBSA Code (an integer code associated with each metro area) from the GEOID field in the geography data. You can learn more about understanding the different geographic levels embedded in GEOIDs by reading this page (https://www.census.gov/geo/reference/geoidentifiers.html) and selecting "GEOID Structure for Geographic Areas".

```
## Keep only geographies of interest
geos <- subset(geos, SUMLEVEL == geoSummaryLevel)
geos$CBSACode <- as.integer(substr(geos$GEOID, nchar(geos$GEOID) - 4, nchar(geos$G
EOID)))
geos <- geos[, c('STUSAB', 'LOGRECNO', 'CBSACode', 'NAME')]
```

Part 2.6: Merge the ACS data with geographic identifiers

Finally, let's merge the ACS data and the geographic codes.

```
## Merge geos and acs data
incomes <- merge(geos, acs[, c('STUSAB', 'LOGRECNO', acsVars)], by = 'LOGRECNO')
```

Now we have a dataframe, called "incomes," that contains the median annual earnings for firefighters and police officers for each metro area in 2014. We have successfully loaded and formatted incomes from the ACS and are ready to start loading Zillow data on home values and mortgage rates.

Part 3: Loading median home values and mortgage rates from Zillow

Zillow publishes monthly median home values for a wide range of geographic levels ranging from the country and U.S. states down to the neighborhood and ZIP code. These and other data are regularly updated on the Zillow Data (http://www.zillow.com/research/data/) page. The median home value (ZHVI) for each geographic level reflects the value of the typical home in each geography, regardless if the home is currently for-sale. It is based on Zillow 'Zestimates' of nearly 120 million homes nationwide (as of November 2015). You can read more about how Zillow calculates the median home value and Zestimates here (http://www.zillow.com/research/zhvi-methodology-6032/), and more about Zillow's coverage and accuracy here (http://www.zillow.com/howto/DataCoverageRentZestimateAccuracy.htm).

Since the types of homes that transact each month changes, median sale prices tend to be highly volatile due to compositional shifts: For instance, if more condos sell in a given month, and in the next month more mansions sell, then the median sale price will appear to have increased more dramatically over the month than the value of any single home. The ZHVI controls for these compositional changes since it reflects the value of the entire housing stock. However, some researchers prefer median sale or list prices, which are also available on the Zillow Data page.

Part 3.1: Load Zillow data on median home values and mortgage rates

First, we load Zillow data on median home values at the metro level. We then reshape the data to allow us to merge them with the ACS data on incomes. Since the ACS data correspond to average incomes in 2014, and the Zillow data are a monthly series, we keep only median home values for months in 2014 and take the average across the 12 months of the year.

```
## Load Zillow metro median home values
zillow <- read_csv('http://files.zillowstatic.com/research/public/Metro/Metro_Zhvi
_AllHomes.csv',
                   col_names = TRUE)
zillow <- melt(zillow, id=c('RegionID', 'RegionName', 'SizeRank'))
zillow <- subset(zillow, substr(variable, 1, 4) == '2014')
zillow <- ddply(zillow, .(RegionID, RegionName, SizeRank), summarise,
                MedianHomeValue = mean(value))
```

We then load data on mortgage rates. Zillow publishes the average interest rate quoted for a standard 30-year fixed rate mortgage to a borrower with a good credit score (a FICO score of 720 or higher) on a loan with a loan-to-value ratio of 0.8 or less. This is the most common loan product Americans use to buy homes. The data are reported in 15-minute increments throughout the business day (6 am to 5 pm Pacific Time, excluding federal holidays). Interest rates are reported already multiplied by 100 – a mortgage rate of 4.2% is reported as 4.2 rather than 0.042 in the data – so we divide by 100 to facilitate computation.

```
mortgageRate <- read_csv('http://files.zillowstatic.com/research/public/MortgageRa
teConventionalFixed.csv',
                         col_names = TRUE)
mortgageRate <- subset(mortgageRate, substr(Date, 1, 4) == '2014')
mortgageRate <- ddply(mortgageRate, .(substr(Date, 1, 4)), summarise,
                      avgRate2014 = round(mean(MortgageRateConventionalFixed, na.r
m = TRUE)/100, 3))
```

Part 3.2: Load a crosswalk between Zillow metro areas and Census metro areas

The Zillow data include metros with slightly different names than the Census data. A county-level crosswalk between the two naming conventions is available here (http://files.zillowstatic.com/research/public/CountyCrossWalk_Zillow.csv) (metro areas are composed of groups of counties).

For this example, we are only interested in the metro-to-metro crosswalk, not the county-to-metro crosswalk, so we remove county identifiers and remove duplicate rows.

```
## Load Zillow Metro-CBSA crosswalk
metroXW <- read_csv('http://files.zillowstatic.com/research/public/CountyCrossWalk
_Zillow.csv',
                    col_names = TRUE)
metroXW <- metroXW[, c('MetroRegionID_Zillow', 'CBSACode')]
metroXW <- metroXW[!duplicated(metroXW), ]
```

Finally, we merge the crosswalk onto the Zillow data on median home values

```
## Merge Zillow data on median home values with crosswalk
zillow <- merge(zillow, metroXW, by.x = 'RegionID', by.y = 'MetroRegionID_Zillow')
```

Part 4: Computing mortgage affordability

Merging the Zillow and ACS data is now fairly straightforward:

```
## Merge Zillow and ACS data
zillow <- merge(zillow, incomes, by = 'CBSACode')
```

We can then use a standard formula (http://www.wikihow.com/Calculate-Mortgage-Payments) for computing a monthly loan payment based on the loan amount, the interest rate, and the loan amortization (how long before a borrower will pay off the loan). The formula we use is:

$$M = P\frac{r(1+r)^n}{(1+r)^n - 1}$$

where:

- **M** = the monthly payment, in dollars
- **P** = the loan principal, or 80% of the home value in this example (the home value minus the down payment)
- **r** = the monthly interest rate (the annual interest rate divided by 12), and
- **n** = the number of monthly payments (360 for a 30-year loan)

To make our code easier to read, we implement this using several parameters

```
## Calculate monthly mortgage payment
nperiods <- 360 # mortgage loan term, in months
compoundingFactor <- (1 + (mortgageRate$avgRate2014 / 12)) ^ nperiods
zillow$monthlyPymt <- (0.8 * zillow$MedianHomeValue) *
  ((mortgageRate$avgRate2014 / 12) * compoundingFactor) / (compoundingFactor - 1)
```

The share of a household's income that would go toward a monthly mortgage payment on the median home in a given metro area is then simply the ratio of the monthly mortgage payment and the monthly income (annual income divided by 12).

```
## Compute monthly payment relative to income
zillow$MortgagePymtToInc_Firefighters <- zillow$monthlyPymt / (zillow$B24011_021 /
12)
zillow$MortgagePymtToInc_Police <- zillow$monthlyPymt / (zillow$B24011_022 / 12)
```

Of course, in these calculations we made several implicit assumptions. For example, we assume that these households are buying a home with only one income. Many households have two incomes (or more in some rare instances). We also assume the household is buying the median home in the metro area. Some buyers may buy a more or less expensive home. There are many different flavors of mortgage affordability that one could compute, but the basic calculation outlined above provides a reasonable basis for comparison.

Exploring the Results

A quick examination of the results shows that Kokomo (IN), Lumberton (NC) and Manitowoc (WI) are the most affordable metro areas for firefighters. Under our assumptions, firefighters in these communities would need to spend less than 7% of their monthly income on his or her mortgage to buy the median home. By contrast, places like Ithaca (NY), Columbia (MO) and Beaver Dam (WI) – as well as larger places like San Jose (CA), San Francisco (CA) and Kahului (HI) – appear to be far less affordable for firefighters with mortgage-payment-to-income ratios in excess of 1, largely due to very low incomes. This means that even if a firefighter spent their entire income on their mortgage, they could not afford to buy the median home (again, assuming a single-income household).

```
head(zillow[order(zillow$MortgagePymtToInc_Firefighters),
            c('RegionName', 'B24011_021', 'MedianHomeValue', 'MortgagePymtToInc_Fi
refighters')], 10)
head(zillow[order(-zillow$MortgagePymtToInc_Firefighters),
            c('RegionName', 'B24011_021', 'MedianHomeValue', 'MortgagePymtToInc_Fi
refighters')], 10)
```

For police, Battle Creek and Saginaw, Michigan, along with several communities in upstate New York, appear to be the most affordable communities. Under our assumptions, police officers in these communities would need to spend less than 6% of their monthly income on thier mortgage to buy the median home. At the other extreme, Oak Harbor (WA), Honolulu (HI), San Jose (CA) and Santa Fe (NM) appear to be the least affordable places for police officers.

```
head(zillow[order(zillow$MortgagePymtToInc_Police),
            c('RegionName', 'B24011_022', 'MedianHomeValue', 'MortgagePymtToInc_Po
lice')], 10)
head(zillow[order(-zillow$MortgagePymtToInc_Police),
            c('RegionName', 'B24011_022', 'MedianHomeValue', 'MortgagePymtToInc_Po
lice')], 10)
```

There is obviously a lot more one could explore in these data. However, the code described above should allow you to do it yourself!

Appendix: Loading ACS Data for Sub-state Geographies

Loading data for geographies that are fully contained within state lines requires several modifications to the code example above. Since the ACS FTP file structure uses both full state names (e.g., California) and state abreviations (e.g., CA), we write a loop that loads each state file. Of course, if your analysis requires only geographies in a single state, you can bypass the loop.

Here we provide an example of loading city-level median incomes for firefighters and police officers.

The first step is to load U.S. state names and abreviations, which are conveniently available in the package, *datasets*. However, we manually bind a state name and abbreviation for the District of Columbia, which is not included in the package data.

```
## Load state names and abreviations
states <- rbind(data.frame(stateName = state.name,
                           stateAbrev = state.abb),
                c('District of Columbia', 'DC'))
```

We then set our parameters as before. We set the *geoLevelFile* parameter to *NULL* as it will be replaced by state names. We also set the *geoSummaryLevel* to 160, which corresponds to *State-Place*. *State-place* (https://www.census.gov/geo/reference/gtc/gtc_place.html) includes cities, boroughs, towns and villages.

```
## Set parameters to load ACS data, at the city level
year <- 2014
summaryFileFolder <- '1_year_seq_by_state'
geoLevelFile <- NULL
geoSummaryLevel <- 160
sequence <- 107
tableNumber <- 'B24011'
Lines <- c(21, 22)
acsDrive <- file.path('http://www2.census.gov',
                      'programs-surveys',
                      'acs',
                      'summary_file',
                      as.character(year),
                      'data')
```

Next, we write a loop to load the ACS and geography tables for each state.

```
## Load geographies, for sub-state geographies
geos <- data.frame()
for (i in 1:nrow(states)){
  geos_temp <- read_csv(file.path(acsDrive,
                           summaryFileFolder,
                           gsub(" ", "", states$stateName[i], fixed = TRUE),
                           paste0('g', as.character(year), 1,
                                  tolower(states$stateAbrev[i]),
                                  '.csv')),
                 col_names = FALSE)
  geos <- rbind(geos, geos_temp)
  rm(geos_temp)
}
rm(i)

## Load ACS tables, for sub-state geographies
acs <- data.frame()
for (i in 1:nrow(states)){
  temp <- tempfile()
  path <- file.path(acsDrive,
                    summaryFileFolder,
                    gsub(" ", "", states$stateName[i], fixed = TRUE),
                    paste0(as.character(year), 1,
                           tolower(states$stateAbrev[i]),
                           stri_pad_left(sequence, 4, pad = '0'),
                           '000', '.zip'))
  download.file(path, destfile = temp)
  acs_temp <- read_csv(unz(temp,
                    paste0('e', as.character(year), '1',
                           tolower(states$stateAbrev[i]),
                           stri_pad_left(sequence, 4, pad = '0'),
                           '000', '.txt')),
                col_names = FALSE)
  acs <- rbind(acs, acs_temp)
  rm(acs_temp)
  unlink(temp)
}
rm(i)
```

The rest of the code should run without any changes except for the following: When merging the geography names and the ACS data for sub-state geographies, merge the data frames on both the *STUSAB* (state abreviation) and *LOGRECNO* (logical record number) fields rather than just the *LOGRECNO* field.

```
## Merge geos and acs data, sub-state geographies
incomes <- merge(geos, acs[, c('STUSAB', 'LOGRECNO', acsVars)], by = c('STUSAB',
'LOGRECNO'))
```