# New York Real Estate Writeup

*Group M: Alex Chaffers, Robbie Zielinski*

## Introduction:

Economic news is ever-present in the media. With every new jobs report or stock market fluctuation, experts debate how these developments will affect individuals. However, despite these debates how people experience individual trends remains obsucre. Our analsyis hopes to uncover part of this link by observing how economic trends influence the real estate markets. Theoreticaly, an economic dowturn should coincide with a decline in home prices and number of transactions. Investigating the truth behind this statement would yield insight into economic trends affect communities and individuals.

In 2008 the American economy entered a tailspin leading to a severe recessions. As a result home prices fell dramatically. The economy has largley recovered from this crisis. However, diffrent areas of the economy have rebounded much faster than others. New York State provides a good expample of this phenomenon. The state has expereinced an uneven recovery since 2008. New York City and its surrounding counites have driven the states recovery while upstate region has had more difficulty recouping the losses of the recession. This discrepency allows us to look at intra-county housing data to compare the real estate market in a economically succesfull county versus a struggling one. Doing so allows us to see if the divergent economic trends in these counties are reflected in thier repsective real estate markets. To visualize these trends we created a shiny app using `leaflet` to map housing transactions throughout Upstate New York.

## The ACS and `acs` Package:

The American Community Survey, or ACS collects annual data on various demographic and economic indicators. Administered by the Census Bureau, the ACS provides a wealth of state county, and census tract level data. The ACS package allows easy access to this data. We were able to use the `geo.make` function to specefiy an area of interest of 50 New York counties.

```
library(acs)
library(RMySQL)
library(ggmap)
library(operators)
library(tidyverse)
library(pbapply)
library(tcltk)
library(lubridate)
library(doParallel)
```

```
cs_key <- "3f58748f63ce427e4133d32fd5b094e102b4ec31"
big_geo <- geo.make(state = 36, county = c("Albany", "Allegany", "Broome",
                                           "Cattaraugus", "Cayuga",
                                           "Chautauqua", "Chemung", "Chenango",
                                           "Clinton", "Columbia", "Cortland",
                                           "Delaware", "Dutchess", "Erie",
                                           "Essex", "Franklin", "Fulton",
                                           "Genesee", "Greene", "Hamilton",
                                           "Herkimer", "Jefferson", "Lewis",
                                           "Livingston", "Madison", "Monroe",
                                           "Montgomery", "Nassau", "Niagara",
```

```
                                                "Oneida", "Onondaga", "Ontario",
                                                "Orange", "Orleans", "Oswego",
                                                "Otsego", "Putnam", "Rensselaer",
                                                "Rockland", "St. Lawrence", "Saratoga",
                                                "Schenectady", "Schoharie", "Schuyler",
                                                "Seneca", "Steuben", "Suffolk",
                                                "Sullivan", "Tioga", "Tompkins",
                                                "Ulster", "Warren", "Washington",
                                                "Wayne", "Westchester", "Wyoming",
                                                "Yates"))
```

Using the `acs.fetch` function, we gather population and income data for our specfied area using an API provided by the Census Bureau. The code below will provide the total population, and median income for the counites we are interested in.

```
bigPop_2015 <- acs.fetch(geo = big_geo, endyear = 2015,
                         table.number ="B01003", key = acs_key)
bigPop_2014 <- acs.fetch(geo = big_geo, endyear = 2014,
                         table.number ="B01003", key = acs_key)
bigPop_2013 <- acs.fetch(geo = big_geo, endyear = 2013,
                         table.number ="B01003", key = acs_key)
bigPop_2012 <- acs.fetch(geo = big_geo, endyear = 2012,
                         table.number ="B01003", key = acs_key)
bigPop_2011 <- acs.fetch(geo = big_geo, endyear = 2011,
                         table.number ="B01003", key = acs_key)
bigPop_2010 <- acs.fetch(geo = big_geo, endyear = 2010,
                         table.number ="B01003", key = acs_key)
bigPop_2009 <- acs.fetch(geo = big_geo, endyear = 2009,
                         table.number ="B01003", key = acs_key)


bigIncome_2015 <- acs.fetch(geo = big_geo, endyear = 2015,
                            table.number =  "B19013", key = acs_key)
bigIncome_2014 <- acs.fetch(geo = big_geo, endyear = 2014,
                            table.number =  "B19013", key = acs_key)
bigIncome_2013 <- acs.fetch(geo = big_geo, endyear = 2013,
                            table.number =  "B19013", key = acs_key)
bigIncome_2012 <- acs.fetch(geo = big_geo, endyear = 2012,
                            table.number =  "B19013", key = acs_key)
bigIncome_2011 <- acs.fetch(geo = big_geo, endyear = 2011,
                            table.number =  "B19013", key = acs_key)
bigIncome_2010 <- acs.fetch(geo = big_geo, endyear = 2010,
                            table.number =  "B19013", key = acs_key)
bigIncome_2009 <- acs.fetch(geo = big_geo, endyear = 2009,
                            table.number =  "B19013", key = acs_key)
```

The `acs` package will load this data as a distinct acs object. To make our data easier to work with we seperated the estimates and created a dataframe containing the population and median income acs data. The following chunks show this proccess. To put our data in a workable format we converted each acs object into a data frame, and then merged these frames to create one data frame containg all of our acs data.

```
pop_2015 <- as.data.frame(bigPop_2015@estimate) %>%
  mutate(county_full = rownames(.),
         county = str_replace(county_full,"(.+) County.*","\\1")) %>%
  select(county, B01003_001) %>%
```

```
  rename("population_2015" = B01003_001)

pop_2014 <- as.data.frame(bigPop_2014@estimate) %>%
  mutate(county_full = rownames(.),
         county = str_replace(county_full,"(.+) County.*","\\1")) %>%
  select(county, B01003_001) %>%
  rename("population_2014" = B01003_001)

pop_2013 <- as.data.frame(bigPop_2013@estimate) %>%
  mutate(county_full = rownames(.),
         county = str_replace(county_full,"(.+) County.*","\\1")) %>%
  select(county, B01003_001) %>%
  rename("population_2013" = B01003_001)

pop_2012 <- as.data.frame(bigPop_2012@estimate) %>%
  mutate(county_full = rownames(.),
         county = str_replace(county_full,"(.+) County.*","\\1")) %>%
  select(county, B01003_001) %>%
  rename("population_2012" = B01003_001)

pop_2011 <- as.data.frame(bigPop_2011@estimate) %>%
  mutate(county_full = rownames(.),
         county = str_replace(county_full,"(.+) County.*","\\1")) %>%
  select(county, B01003_001) %>%
  rename("population_2011" = B01003_001)

pop_2010 <- as.data.frame(bigPop_2010@estimate) %>%
  mutate(county_full = rownames(.),
         county = str_replace(county_full,"(.+) County.*","\\1")) %>%
  select(county, B01003_001) %>%
  rename("population_2010" = B01003_001)

pop_2009 <- as.data.frame(bigPop_2009@estimate) %>%
  mutate(county_full = rownames(.),
         county = str_replace(county_full,"(.+) County.*","\\1")) %>%
  select(county, B01003_001) %>%
  rename("population_2009" = B01003_001)

income_2015 <- as.data.frame(bigIncome_2015@estimate) %>%
  mutate(county_full = rownames(.),
         county = str_replace(county_full,"(.+) County.*","\\1")) %>%
  select(county,B19013_001) %>%
  rename("income_2015" = B19013_001)

income_2014 <- as.data.frame(bigIncome_2014@estimate) %>%
  mutate(county_full = rownames(.),
         county = str_replace(county_full,"(.+) County.*","\\1")) %>%
  select(county,B19013_001) %>%
  rename("income_2014" = B19013_001)

income_2013 <- as.data.frame(bigIncome_2013@estimate) %>%
  mutate(county_full = rownames(.),
         county = str_replace(county_full,"(.+) County.*","\\1")) %>%
```

```
  select(county,B19013_001) %>%
  rename("income_2013" = B19013_001)

income_2012 <- as.data.frame(bigIncome_2012@estimate) %>%
  mutate(county_full = rownames(.),
         county = str_replace(county_full,"(.+) County.*","\\1")) %>%
  select(county,B19013_001) %>%
  rename("income_2012" = B19013_001)

income_2011 <- as.data.frame(bigIncome_2011@estimate) %>%
  mutate(county_full = rownames(.),
         county = str_replace(county_full,"(.+) County.*","\\1")) %>%
  select(county,B19013_001) %>%
  rename("income_2011" = B19013_001)

income_2010 <- as.data.frame(bigIncome_2010@estimate) %>%
  mutate(county_full = rownames(.),
         county = str_replace(county_full,"(.+) County.*","\\1")) %>%
  select(county,B19013_001) %>%
  rename("income_2010" = B19013_001)

income_2009 <- as.data.frame(bigIncome_2009@estimate) %>%
  mutate(county_full = rownames(.),
         county = str_replace(county_full,"(.+) County.*","\\1")) %>%
  select(county,B19013_001) %>%
  rename("income_2009" = B19013_001)
```

```
acs_pop <- pop_2015 %>%
  inner_join(pop_2014, by = "county") %>%
  inner_join(pop_2013, by = "county") %>%
  inner_join(pop_2012, by = "county") %>%
  inner_join(pop_2011, by = "county") %>%
  inner_join(pop_2010, by = "county") %>%
  inner_join(pop_2009, by = "county") %>%
  gather("year", "population", 2:8) %>%
  mutate(year = gsub("population_", "", year)) %>%
  mutate(year = as.numeric(year))

acs_income <- income_2015 %>%
  inner_join(income_2014, by = "county") %>%
  inner_join(income_2013, by = "county") %>%
  inner_join(income_2012, by = "county") %>%
  inner_join(income_2011, by = "county") %>%
  inner_join(income_2010, by = "county") %>%
  inner_join(income_2009, by = "county") %>%
  gather("year", "income", 2:8) %>%
  mutate(year = gsub("income_", "", year)) %>%
  mutate(year = as.numeric(year))

acs <- acs_pop %>%
  inner_join(acs_income, by = c("county", "year"))

names(acs)[1] <- "County"
```

```
save(acs, file="acs.Rda")
```

We saved the acs data as an Rda file and used the data further in nyhousing_ingest.Rmd to incorporate it with the real estate transaction data.

# Real Estate Transactions

## Web Scraper

The housing data served as the motivation for many of the questions we asked and the visualizations we developed. The data is freely available at http://rochester.nydatabases.com/database/real-estate-database and is updated weekly by the Gannett-operated newspapers of New York State. While the data is easy to view, there is no API, so we had to scrape the data from the website in order to bring it into our analysis. One substantial difficulty that we ran into was the way that the dataset was coded into the website. Typical web scraping packages like `rvest` work well when searching HTML code for specific tags of interest. However, this website used a combination of different iterations of JavaScript that made the tags that we needed inaccessible without clicking on each page of the table, meaning that the usual web scraping packages would not work in this situation. To address this, we were able to use a Python library, Selenium, to open the web page in a browser, and then automate searching for each page and clicking on it. The code we used to do so is below.

```
import time
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.common.exceptions import TimeoutException
from selenium.common.exceptions import StaleElementReferenceException
import MySQLdb

url = 'http://rochester.nydatabases.com/database/real-estate-database'
countyName = ""
```

In the code chunk above, we simply imported the necessary libraries and set a url variable. Below, we defined functions to start the web browser and load the web page.

```
def init_driver():
    driver = webdriver.Firefox()
    driver.wait = WebDriverWait(driver, 5)
    return driver

def load(driver):
    driver.get(url)
    try:
        button = driver.wait.until(EC.element_to_be_clickable(
            (By.ID, "search")))
        button.click()
    except TimeoutException:
        print("Button not found")
```

```
def nextPage(driver, current):
    try:
        button = driver.wait.until(EC.element_to_be_clickable(
```

```
            (By.XPATH, '//span[@start={0}]'.format(str(current)))))
        button.click()
        time.sleep(4)
        return(current+1)
    except TimeoutException:
        print("No more pages found")
        return(current)

def newYear(driver, current):
    try:
        button = driver.wait.until(EC.element_to_be_clickable(
            (By.XPATH, '//select[@id="sYear"]/option[@value={0}]'.format(str(current-1)))))
        button.click()
        search = driver.wait.until(EC.element_to_be_clickable(
            (By.ID, "search")))
        search.click()
        time.sleep(4)
        return(current-1)
    except TimeoutException:
        print("No more years")
        return(current)

def newCounty(driver, current):
    try:
        counties = driver.find_elements_by_xpath('//select[@id="cnty"]')
        for county in counties:
            names = county.find_elements_by_tag_name('option')
            global countyName
            countyName = [name.text for name in names][current-1]
        button = driver.wait.until(EC.element_to_be_clickable(
            (By.XPATH, '//select[@id="cnty"]/option[@value={0}]'.format(str(current)))))
        button.click()
        search = driver.wait.until(EC.element_to_be_clickable(
            (By.ID, "search")))
        search.click()
        time.sleep(4)
        return(current+1)
    except TimeoutException:
        print("No more counties")
        return(current)
```

We then defined functions to change to the next page in the table, switch the year that we are filtering the data by, and change the county that we are filtering by. The pages, counties, and years in the table were all indexed numerically in the HTML code, so we were able to loop through all the pages in each year and all the years for each county until we collected all the data for each particular county. For each page, once the code rendered, we scraped all values in the table shown and saved them into a vector. Then once we had run through each page in the year, we added the data in the vector to a table in a MySQL database. The following code executed this process.

```
if __name__ == "__main__":
    driver = init_driver()
    load(driver)
    time.sleep(5)
```

```
    county = newCounty(driver, 5)

    db = MySQLdb.connect(user="root", passwd="stat231project", db="nyhousing", host="localhost")
    cur = db.cursor()

    db.set_character_set('utf8')
    cur.execute('SET NAMES utf8;')
    cur.execute('SET CHARACTER SET utf8;')
    cur.execute('SET character_set_connection=utf8;')

    while True:
        curyear = newYear(driver, 2018)
        if(county == 6):
            curyear = newYear(driver, 2014)
        while True:
            pagenum = 1
            while True:
                data = []
                trs = driver.find_elements_by_tag_name('tr')
                if trs:
                    for tr in trs:
                        tds = tr.find_elements_by_tag_name('td')
                        if tds:
                            text = []
                            text.append(countyName)
                            for td in tds:
                                text.append(td.text)
                            data.append(text)
                cur.executemany("INSERT INTO transactions (County, Town, Address, ZipCode, SalePrice, Sa
                print "County: {} Year: {} Page: {}".format(county, curyear, pagenum)
                newpage = nextPage(driver, pagenum)
                if (newpage > pagenum):
                    pagenum = newpage
                else:
                    break
            db.commit()
            newyear = newYear(driver, curyear)
            if (newyear < curyear):
                curyear = newyear
            else:
                break
        newcounty = newCounty(driver, county)
        if(newcounty > county):
            county = newcounty
        else:
            break

    driver.quit()
    cur.close()
    db.close()
```

## Geocoding

The real estate database included the street address and town name, but not the latitude and longitude coordinates that would be necessary to plot the locations on a map. We used the `geocode` function in the `ggmap` package to access these coordinates. To do so, we used the `RMySQL` package to connect to our MySQL database and read in the data. The code chunk below shows the process of connecting to the database, reading the data to a `data.frame` object, and then creating a variable merging together the entire street address to feed into the `geocode` function.

```
db <- dbConnect(MySQL(), user = 'stat231', password = 'stat231project', dbname = "nyhousing", host = '14
transactions_df <- dbReadTable(db, "transactions")
transactions_df <- transactions_df %>%
  mutate(address_full = paste(Address, Town, "NY", sep = ", "))
```

Because the `geocode` function takes an address and returns latitude and longitude coordinates without any dependency between observations, this process was ideal for us to run in parallel in order to reduce the run time. We used the `pbapply` package to generate a progress bar to indicate approximately how long the function would take to complete, and also used the `tryCatch` function to allow the function to work despite invalid addresses that would generate missing values.

```
no_cores <- detectCores() - 1
registerDoParallel(cores=no_cores)
cluster <- makeCluster(no_cores, type="FORK")

full_address <- transactions_df %>%
  select(address_full) %>%
  unlist() %>%
  as.vector()

geocodes <- pblapply(full_address, function(x) tryCatch(geocode(x, output = "latlon", messaging = FALSE
stopCluster(cluster)

full_address <- data.frame(full_address)
names(full_address) <- c("address_full")
geocodes <- unlist(geocodes) %>%
  matrix(ncol = 2, byrow = TRUE) %>%
  data.frame
names(geocodes) <- c("lon", "lat")
geocodes <- bind_cols(full_address, geocodes) %>%
  distinct()
geocodes$address_full <- as.character(geocodes$address_full)
transactions2 <- transactions_df %>%
  left_join(geocodes, by = "address_full")

full_address <- NULL
transactions_df <- NULL
geocodes <- NULL
```

```
counties <- c("Albany", "Allegany", "Broome", "Cattaraugus", "Cayuga",
              "Chautauqua", "Chemung", "Chenango", "Clinton", "Columbia",
              "Cortland", "Delaware", "Dutchess", "Erie", "Essex", "Franklin",
              "Fulton", "Genesee", "Greene", "Hamilton", "Herkimer", "Jefferson",
              "Lewis", "Livingston", "Madison", "Monroe", "Montgomery",
              "Nassau", "Niagara", "Oneida", "Onondaga", "Ontario", "Orange",
              "Orleans", "Oswego", "Otsego", "Putnam", "Rensselaer", "Rockland",
```

```
              "St. Lawrence", "Saratoga", "Schenectady", "Schoharie",
              "Schuyler", "Seneca", "Steuben", "Suffolk", "Sullivan",
              "Tioga", "Tompkins", "Ulster", "Warren", "Washington", "Wayne",
              "Westchester", "Wyoming", "Yates")

long_min <- c(-74.3, -78.3, -76.2, -79.1, -76.8, -79.8, -77.0, -75.9, -74.0, -74.0,
              -76.3, -75.5, -74.0, -79.2, -74.4, -74.8, -74.8, -78.5, -74.6, -74.9,
              -75.3, -76.5, -75.9, -78.1, -76.0, -78.0, -74.8, -73.8, -79.1, -75.9,
              -76.5, -77.7, -74.8, -78.5, -76.7, -75.5, -74.0, -73.8, -74.3, -75.9,
              -74.2, -74.4, -74.8, -77.1, -77.0, -77.8, -73.5, -75.2, -76.6, -76.7,
              -74.8, -74.3, -73.7, -77.4, -74.0, -78.5, -77.4)

long_max <- c(-73.6, -77.7, -75.3, -78.3, -76.2, -79.0, -76.5, -75.3, -73.3, -73.3,
              -75.8, -74.4, -73.4, -78.4, -73.3, -73.9, -74.1, -77.9, -73.7, -74.0,
              -74.6, -75.4, -75.1, -77.4, -75.2, -77.3, -74.0, -73.4, -78.4, -75.0,
              -75.8, -76.9, -73.9, -77.9, -75.7, -74.6, -73.5, -73.2, -73.9, -74.5,
              -73.5, -73.8, -74.1, -76.6, -76.6, -76.9, -71.8, -74.3, -76.0, -76.2,
              -73.9, -73.4, -73.2, -76.7, -73.4, -77.9, -76.9)

lat_min <- c(42.4, 42.0, 42.0, 42.0, 42.6, 42.0, 42.0, 42.2, 44.4, 41.9, 42.4, 41.8,
             41.4, 42.4, 43.7, 44.0, 42.9, 42.8, 42.0, 43.2, 42.8, 43.6, 43.4, 42.4,
             42.7, 42.9, 42.7, 40.5, 43.0, 42.8, 42.7, 42.5, 41.1, 43.1, 43.1, 42.3,
             41.3, 42.4, 40.9, 44.0, 42.7, 42.7, 42.3, 42.2, 42.5, 42.0, 40.6, 41.4,
             42.0, 42.2, 41.5, 43.2, 42.9, 43.0, 40.8, 42.5, 42.4)

lat_max <- c(42.9, 42.6, 42.5, 42.6, 43.5, 42.6, 42.3, 42.8, 45.0, 42.5, 42.8, 42.6,
             42.1, 43.1, 44.6, 45.0, 43.3, 43.2, 42.5, 44.2, 44.1, 44.4, 44.3, 43.0,
             43.2, 43.4, 43.1, 41.0, 43.4, 43.6, 43.3, 43.0, 41.7, 43.4, 43.7, 42.9,
             41.6, 43.0, 41.4, 45.1, 43.4, 43.0, 42.9, 42.6, 43.1, 42.6, 41.3, 42.0,
             42.5, 42.7, 42.2, 43.8, 43.8, 43.4, 41.4, 42.9, 42.8)

counties_lat_lon <- data.frame(counties, long_min, long_max, lat_min, lat_max)
```

Unfortunately, the housing database included addresses that, when geocoded, translated to valid latitude and longitude coordinates that were not in the correct county, or even in the right part of the world. In order to resolve this issue, we filtered out all locations that had latitude or longitude coordinates outside of their county boundaries. The code chunk above shows the development of a dataset containing each county in the real estate data and their maximum and minimum latitude and longitude values. Below, we join this `data.frame` with the general transactions dataset and filter out all locations outside of the county boundaries.

```
transactions2 <- transactions2 %>%
  left_join(counties_lat_lon, by = c("County" = "counties"))

transactions2 <- transactions2 %>%
  mutate(lon = ifelse((lon > long_min & lon < long_max & lat > lat_min & lat < lat_max), lon, NA),
         lat = ifelse((lon > long_min & lon < long_max & lat > lat_min & lat < lat_max), lat, NA))
```

In the chunk below, we convert the SalePrice variable to a valid numeric variable, select the variables we needed for the further analysis, and update a separate MySQL table.

```
transactions2 <- transactions2 %>%
  mutate(SalePrice = gsub(",", "", SalePrice),
         FullSaleDate = SaleDate) %>%
  separate(FullSaleDate, c("month", "day", "year"), sep = "/")
```

```
transactions2 <- transactions2 %>%
  select(County, Town, Address, ZipCode, SalePrice, SaleDate, SellerName, BuyerName,
         id, address_full, lon, lat, month, day, year)

dbWriteTable(db, "transactions2", transactions2, overwrite = TRUE)
```

## Ingesting Data

In order to prepare the data to visualize, we pulled the data from the updated table in our MySQL database, completed wrangling steps to convert the relevant variables to the correct format, merged it with the ACS data, and saved it to a new Rda file to enter into our Shiny app. The following code chunks complete this process.

```
db <- dbConnect(MySQL(), user = 'stat231', password = 'stat231project', dbname = "nyhousing", host = '14
transactions <- tbl(db, "transactions2")
```

```
transactions <- collect(transactions)
transactions <- transactions %>%
mutate(SaleDate = mdy(SaleDate),
month = as.numeric(month),
day = as.numeric(day),
year = as.numeric(year),
SalePrice = as.numeric(SalePrice),
Long = lon,
Lat = lat)
```

```
transactions <- transactions %>%
select(County, Town, Address, ZipCode, SalePrice, SaleDate, SellerName, BuyerName,
id, address_full, Long, Lat, month, day, year)
```

```
load("acs.Rda")

transactions <- transactions %>%
  left_join(acs, by = c("County", "year"))
```

```
save(transactions, file="nyhousing_transactions.Rda")
```

## Visualization

In order to successfully visualize the data, we needed to be able to zoom in to be able to properly discern individual points on the map. To do this, we used the `leaflet` package to integrate an interactive map into our Shiny app. Additionally, in the Shiny app, we made it possible to filter by county and by year. This serves two purposes: to allow the user to focus on the locations and time periods that they are most interested in, and to decrease the amount of time that it takes for the Shiny app to process the commands given to it, making for a slightly smoother user experience. We added the reference year value to the app to allow for a comparison between real estate prices in each county and the population and income values gathered from the ACS data. We hope that with access to more extensive data in both the ACS and real estate realms, we will be able to use the reference year feature of the app to help draw some conclusive decisions about the relationship between economic estimates and real estate transactions. This would be helpful in testing our hypotheses for statistical significance. The chunk below includes all code necessary to run the Shiny app.

```r
load("nyhousing_transactions.Rda")

transactions <- transactions %>%
  select(year, month, day, County, Address, Town, ZipCode, SalePrice, Long, Lat) %>%
  mutate(County = as.factor(County),
         Town = as.factor(Town),
         ZipCode = as.factor(ZipCode),
         SalePrice = as.numeric(SalePrice),
         month = as.numeric(month),
         day = as.numeric(day),
         year = as.numeric(year))

# create codebook
description <- c("Year", "Month", "Day", "County", "Street Address", "Town Name",
                 "Zip Code", "Sale Price", "Longitude", "Latitude")
codebook <- data.frame(name=names(transactions), description)
names(codebook) <- c("Variable", "Variable Description")

pal <- colorNumeric(
  palette = "YlOrRd",
  domain = log(transactions$SalePrice))




# UI
ui <- fluidPage(
  titlePanel("Map of Transactions"),
  sidebarLayout(

    # Input(s)
    sidebarPanel(
      selectInput("transactionyear", "Year of Transaction", choices = 1993:2017, selected = 2017),
      checkboxGroupInput("transactioncounties", "Counties:", choices = c("All", unique(sort(as.characte
                         selected = "All")
      #checkboxInput("crosslistedenrollment", "Include Cross-Listed Enrollments", TRUE)
      #checkboxGroupInput("subject", "Departments",
      #                   choices = unique(sort(as.character(fakeenrolls$subject))),
      #                   selected = unique(sort(as.character(fakeenrolls$subject)))[1]
      #),
      #checkboxGroupInput("type", "Type of course",
      #                   choices = unique(sort(as.character(fakeenrolls$type))),
      #                   selected = unique(sort(as.character(fakeenrolls$type)))
      #)
    ),
    # Output(s)
    mainPanel(

      tabsetPanel(id = "tabspanel", type = "tabs",
                  tabPanel(title = "Data Table",
                           br(),
                           DT::dataTableOutput(outputId = "transactionstable")),
```

```
                  tabPanel(title = "Map of Real Estate Transactions",
                           leafletOutput(outputId = "map"),
                           br(),
                           h4(uiOutput(outputId = "n1"))),
                  #tabPanel(title = "Course Enrollments (by dept and level)",
                  #         plotOutput(outputId = "boxplot2"),
                  #         br(),
                  #         h4(uiOutput(outputId = "n2"))),
                  # New tab panel for Codebook
                  tabPanel("Codebook",
                           br(),
                           DT::dataTableOutput(outputId = "codebook"))
      )
    )
  )
)

# Server
server <- function(input, output) {

  # Create reactive data frame
  sales <- reactive({
    newval <- transactions %>%
      filter(year == input$transactionyear)

    if("All" %in% input$transactioncounties){

    } else{
      newval <- newval %>%
        filter(County %in% input$transactioncounties)
    }
    #if (input$crosslistedenrollment) {
    #  newval <- newval %>%
    #    mutate(displayenroll = totalenroll)
    #} else {
    #  newval <- newval %>%
    #    mutate(displayenroll = enroll)
    #}
    #if (! input$specialtopicshonors) {
    #  newval <- newval %>%
    #    filter(! number %in% c(290, 390, 490))
    #}
    return(newval)
  })

  output$map <- renderLeaflet({
    m <- sales() %>%
      select(Long, Lat, SalePrice) %>%
      leaflet() %>%
      addTiles() %>%
      addCircles(lng = ~Long, lat = ~Lat, radius = 10,
                 color = ~pal(log(SalePrice)), opacity = 5)
```

```
    return(m)
  })

  #Create data table
  output$transactionstable <- DT::renderDataTable({
    DT::datatable(data = sales(),
                  options = list(pageLength = 20),
                  rownames = FALSE)
  })

  # Create data table
  output$codebook <- DT::renderDataTable({
    DT::datatable(data = codebook,
                  options = list(pageLength = 8),
                  rownames = FALSE)
  })
}

shinyApp(ui = ui, server = server)
```

## Future Work

One way to expand our findings would be to gather more data. Because the website where we gathered data from crashed, we were not able to gather real estate data from all upstate counties. Further the ACS only provided data from 2009 until 2016. However, the census Bureau has data from 2000 on. Finding a way to incorporate this data into our project could provide a more complete picture of the New York state economy. Including this data would allow us to more thoroughly examining the link between economic indicators and housing transactions. For example we could investigate causation, and answer the question of causation in the link between income, population and housing variables. Further, we could look at the time lag between changes in variables. We could attempt to see how long it takes for a drop in population or income to be reflected in housing prices

Another extension would be using our data to answer questions about the real estate market of Upstate New York. Our data provides a platform for a close analysis of any county in New York. It could be used to attempt to determine the effect of a specific event, such as a factory opening or closing or a period of increased immigration had on a specific area. This analysis could either pair our real estate data with data from by ACS package, or another data set more suited to the specific task.