MC-D607 Final Project HUD-API

Marco Castro

2024-11-26

Fair Market Rents (FMR) vs Market Rate Rents

This project aims to ascertain whether the market rate rental costs for 1 and 2 bedroom apartments are consistent with the federal government's Fair Market Rents (FMR) — the 40th percentile of gross rents for typical rental units paid by tenants that moved within the last 20 months. FMR is used to calculate benefits such as the Housing Choice Vouchers used to help unhoused individuals find permanent housing.

As an annually-released estimate, FMR data does not capture the real-time economic realities that drive local housing markets. This means that some individuals that are approved for housing assistance benefits are unable to find an apartment locally that they can pay for with their approved voucher amount based on their area's FMR. In other words, market rates tend to outpace FMR estimates when market conditions quickly drive rental prices up in certain areas.

This project uses FMR data pulled from HUD using their publicly available API. Market rate data was scraped from the rental listing site Trulia in three distinct metropolitan areas: Atlanta, GA, Buffalo, NY and San Diego, CA to obtain an up-to-date snapshot of real rental costs that can be compared against data from HUD's FMR Dataset. The comparison will help to determine if the FMR data released earlier this year for fiscal year Oct 2024-Sept 2025 is still relevant to current housing costs in these areas. For the purposes of this project, we will work with data only for apartments with less than 3 bedrooms.

```r
# A1: declare the cities we will analyze
target_cities <- c('Atlanta-Sandy Springs-Roswell','Buffalo-Cheektowaga-Niagara Falls','San Diego-Carls
max_bedrooms <- 3
```

A: Getting FMR data using the HUD API

The section below declares the global variables and functions for to call the HUD FMR & IL API using the `httr2` package using a HUD API key (A2).

```r
# A2: Custom function to call HUD FMR & IL API
call_hud <- function(endpoint) {
  # declare constants
  domain <- "https://www.huduser.gov"
  method   <- "fmr" # alt method is il or mtspil
  path <-paste("hudapi/public/", method, "/", endpoint,sep="")

  # init request
  req <- request(domain) |>
    req_headers("Accept" = "application/json") |>
    req_auth_bearer_token(token) |>
    req_url_path(path) |>
    req_user_agent("Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Ch

  # call api
  resp <- req_perform(req)
  # parse response
  json_resp <- resp |>
    resp_body_string() |>
    fromJSON()

  # return parsed response
  return(json_resp)
}
```

Retrieving the Metro Ids

The custom function `call_hud` is used in the section below (A3) to request the HUD metropolitan area id (`entity_id`) for each of our target cities. This id will then be used to retrieve the FMR for each of the zipcodes that make up the area as well as the estimate for the area overall. The request returns a json obj for all metropolitan areas. After converting the response to a dataframe (A4), we will need to separate the **area_name** column to cities, states, and area type. Finally, we filter for using the target cities array declared earier.

```r
  # A3: Fetch Metro Area Ids from API
  metros <- call_hud("listMetroAreas")

  # A4: cleanup and filter list of metro areas
  #     for target areas
  metro_areas <- as.data.frame(metros) |>
    # split area_name col into city and type
    separate_wider_regex(
      area_name, c(area_name= ".*", ",\\s", type=".*")
    ) |>
    # split type col into state and type
    mutate(
      state = substring(type,1,2),
    ) |>
    # filter by our target areas
    filter(area_name %in% target_cities) |>
    select(c('cbsa_code',  'area_name', 'state'))

glimpse(metro_areas)
```

```
## Rows: 3
## Columns: 3
## $ cbsa_code <chr> "METRO12060M12060", "METRO15380M15380", "METRO41740M41740"
## $ area_name <chr> "Atlanta-Sandy Springs-Roswell", "Buffalo-Cheektowaga-Niagar~
## $ state     <chr> "GA", "NY", "CA"
```

Retrieving FMRs for Target Cities

Now that we have the ids for the metropolitan areas, we can call the `data` method of HUD's API to retrieve the FMR values for each metropolitan area and its corresponding zip codes (A5). I created a custom function `get_metro_data` that requests the FMR estimates the API and clean up the response. This function is called from a loop that iterates through each of our target areas. Finally, the loop uses `rbind` to merge the dataframes for each target city into a single dataframe

```r
# A5: Call, clean-up and merge metro area data
# Custom function to call and clean up metro area data
get_metro_data <- function(row) {
  # use "cbsa_code" col as entity_id for API call
  entity_resp <- call_hud(paste("data", row$cbsa_code, sep="/"))

  # clean up data
  df <- as_tibble(entity_resp$data$basicdata) |>
    rename_all(~tolower(str_trim(str_replace_all(., "-", "_")))) |>
    rename(studio = efficiency) |>
    mutate(
      area_name = row$area_name
    ) |>
    relocate(area_name, .before=zip_code)
}

# init df with first row
metro_fmrs <- tibble()

# loop through all other metro areas
for (i in 1:nrow(metro_areas)) {
  metro_fmrs <- rbind(metro_fmrs, get_metro_data(metro_areas[i,]))
}

head(metro_fmrs)
```

```
## # A tibble: 6 x 7
##   area_name   zip_code studio one_bedroom two_bedroom three_bedroom four_bedroom
##   <chr>       <chr>     <int>       <int>       <int>         <int>        <int>
## 1 Atlanta-Sa~ MSA lev~   1591        1653        1830          2205         2653
## 2 Atlanta-Sa~ 30002      1100        1150        1270          1530         1840
## 3 Atlanta-Sa~ 30003      1730        1800        1990          2400         2890
## 4 Atlanta-Sa~ 30004      1830        1910        2110          2540         3060
## 5 Atlanta-Sa~ 30005      2030        2110        2340          2820         3390
## 6 Atlanta-Sa~ 30006      1660        1730        1910          2300         2770
```

```
zip_codes <- metro_fmrs |>
  filter(zip_code != 'MSA level') |>
  select(c("area_name", "zip_code"))
```

Tidying the FMR data

In section A6, I used `pivot_longer` to break up each FMR cost estimate into its own row broken up by apartment type in each apartment. I then used `mutate` to convert the bedroom classifications to an integer representation based on the number of bedrooms. For example, a studio was given a value of 0, `one_bedroom` was given a value of 1, etc.

```
# A6:  tidy data
metro_fmr_by_zip_and_num_bds <- metro_fmrs |>
  pivot_longer(
    cols = studio:four_bedroom,
    names_to = c("bedrooms"),
    values_to = "fmr"
  ) |>
  mutate(
    bedrooms = case_when(
      bedrooms == 'studio' ~ as.integer(0),
      bedrooms == 'one_bedroom' ~ as.integer(1),
      bedrooms == 'two_bedroom' ~ as.integer(2),
      bedrooms == 'three_bedroom' ~ as.integer(3),
      bedrooms == 'four_bedroom' ~ as.integer(4)
    )
  ) |>
  arrange(zip_code) |>
  filter(bedrooms < max_bedrooms)

metro_fmr_msa_by_num_bds <- metro_fmr_by_zip_and_num_bds |>
  filter(zip_code == 'MSA level')

head(metro_fmr_by_zip_and_num_bds)
```

```
## # A tibble: 6 x 4
##   area_name                         zip_code bedrooms   fmr
##   <chr>                             <chr>       <int> <int>
## 1 Buffalo-Cheektowaga-Niagara Falls 14001           0   860
## 2 Buffalo-Cheektowaga-Niagara Falls 14001           1   900
## 3 Buffalo-Cheektowaga-Niagara Falls 14001           2  1050
## 4 Buffalo-Cheektowaga-Niagara Falls 14004           0   860
## 5 Buffalo-Cheektowaga-Niagara Falls 14004           1   900
## 6 Buffalo-Cheektowaga-Niagara Falls 14004           2  1050
```

B: Scraping Market Rate Data

In this section, we use the `httr` package to scrape Trulia, a real estate listing portal, for rental listings in the zipcodes we retrieved for each of our target cities. In the block below (B1), we initialize the an empty tibble and declare the varible types that we will be using to populate with the scaped rental listing data. We also set our user agent headers to mimick a web browser. Finally, we declare to custom functions: 1. **throttled_GET**: throttle the HTML requests to Trulia. This is used to send a delayedGET request using a timer using the `slowly` function in order to minimize possibly being blocked 2. **cleanup_price**: converts the string value from Trulia to an integer by removing any non-numerical characters. For example, the string "$1,000/mo" would return 1000.

```
# B1: Declare parser constants

# init market rate tibble
real_estate_df <- tibble(
  zip_code = character(),
  bedrooms = integer(),
  market_rate = integer()
)

# update user agents
user_agent <- "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/
set_config(add_headers(`User-Agent` = user_agent))

delay <- 10       # number of seconds to delay each GET request

# custom function for delayed scraping
throttled_GET <- slowly(
  ~read_html(.),
  rate = rate_delay(delay)
)
```

```r
# custom function to remove misc chars
cleanup_price <- function(s) {
  t <- str_replace_all(s, "([$,])", "") #remove dollar signs and commas
  x <- gsub("^([0-9]+).*", "\\1", t) #only return first group of integers
  return(as.integer(x))
}
```

Looping through our Zip Codes

In the following section, we use a for loop to iterate through each of the zip codes retrieved from HUD for our target cities to retrieve current (B2). We set each zip code (`zip`), the number of bedrooms (`beds`), and a page counter (`current_page`) as paramaters for our GET request, which is called via the custom `throttled_GET` function (B3). Upon retrieving the HTML from our response, we use `html_element` to find all listings on the price, then save the monthly rental cost in a list `listings` (B4). We use another for loop through this list and append the values as new rows in the `real_estate_df` tibble (B5). Finally, we save the results to a CSV called "trulia_data.csv" for future use (B6).

```r
# B2: Scrape Real Estate Data

# can be altered for multipage scraping
current_page <- 1 # page counter
start <- 1         # start index
end <- 1           # end index. Use #nrow(zip_codes) to use zip_code length

# loop through zip codes list
for (beds in 0:max_bedrooms) {

  for (i in start:end) {
    # get zip code at current index
    zip <- zip_codes[i,]$zip_code

    # B3: build path based on the current zip, beds and page vars
    # and send to our throttled GET request function
    page_reponse <- throttled_GET(paste("https://www.trulia.com/for_rent/", zip, "_zip/", beds, "_beds/

    # find price from current page html
    result_container <- page_reponse %>%
      html_element(xpath='//ul[@data-testid="search-result-list-container"]' )

    # B4: save our rental prices to Trulia
    listings <- result_container %>%
      html_elements(xpath = '//li//div[@data-testid="property-price"]' ) %>%
      html_text()

    # B5: loop through all listing prices
    # and append to market rate tibble
    for (s in listings)
      real_estate_df <- real_estate_df |>
        add_row(zip_code = zip, bedrooms = beds, market_rate = cleanup_price(s))
  }
}

head(real_estate_df, n=10)
```

```
## # A tibble: 10 x 3
##    zip_code bedrooms market_rate
##    <chr>       <int>       <int>
##  1 30002           0        1400
##  2 30002           0        1397
##  3 30002           0        1303
##  4 30002           0        1575
##  5 30002           1         925
##  6 30002           1        1198
##  7 30002           1        1500
##  8 30002           1        1565
##  9 30002           1        1175
## 10 30002           1        1100
```

```r
# B6: write rental listings to csv
write_csv(real_estate_df, "trulia_data_temp.csv")
```

C: Working with Combined Data

In this section, we combine the FMR data we retrieved using HUD's API and the market rate data we scraped from Trulia. First, we read in the data scraped from Trulia from section B from the CSV file "trulia_data.csv" and filter out any major outliers (C1). Next we `group_by` using the zip codes and number of bedrooms and `summarise` to calculate the 40th percentile of rental costs for every type of apartment size within each zip code and assign the grouped results to the dataframe `market_rate_by_zip_and_num_bds`.

```r
# C1: Read previously parsed file from Trulia
# add mean for market rate data based on zipcode and # of bedrooms
# then add metro ids
market_rates <- read_csv(
    file = "trulia_data.csv",
    col_names = TRUE,
    col_types = cols(.default = col_character(), market_rate = "i", bedrooms = "i"),
    show_col_types = FALSE
  ) |>
  filter(
    bedrooms < 3 &
    market_rate < 10000 # remove outliers
  )

# C2: group by zip codes and # bedrooms
market_rate_by_zip_and_num_bds <- market_rates |>
  group_by(across(all_of(c("zip_code", "bedrooms")))) |>
  summarise(
    market_rate = round(quantile(market_rate,probs=0.4))
  )
```

```
## `summarise()` has grouped output by 'zip_code'. You can override using the
## `.groups` argument.
```

Combining FMR and Market Rate Dataframes By Zip Code

Now that we have the calculations for the 40th percential of market rate rents, we can combine the data with the FMR data. We use the `right_join` function to join the `metro_fmr_by_zip_and_num_bds` dataframe to the `market_rate_by_zip_and_num_bds` dataframe by zip codes (C3). Because some of the zip codes did not have any active listings on the date we scraped Trulia, using `right_join` will ensure that only the zip codes that had listings will be preserved in our `combined_df_by_zip` dataframe. Next, we use `pivot_longer` to break up each estimated rental cost as a unique observation that can be grouped or filtered by area_name, zip code, apartment size (# of bedrooms), and estimate type (fmr vs market rate).

```
### C3: Combine FMR and Market Rate Dataframes By Zip Code
combined_df_by_zip <- metro_fmr_by_zip_and_num_bds |>
  right_join(market_rate_by_zip_and_num_bds, by= c('zip_code'='zip_code', 'bedrooms'='bedrooms')) |>
  mutate(bedrooms = as.factor(bedrooms))

# C4: pivot longer by estimate type
combined_df_by_zip_and_type <- combined_df_by_zip |>
  pivot_longer(
    cols = c(fmr, market_rate),
    names_to = c("type"),
    values_to = "rent"
  ) |>
  mutate(
    type = str_replace_all(type, "_", " ")
  )

head(combined_df_by_zip_and_type)
```

```
## # A tibble: 6 x 5
##   area_name                          zip_code bedrooms type         rent
##   <chr>                              <chr>    <fct>    <chr>       <dbl>
## 1 Buffalo-Cheektowaga-Niagara Falls 14001    1        fmr           900
## 2 Buffalo-Cheektowaga-Niagara Falls 14001    1        market rate  1040
## 3 Buffalo-Cheektowaga-Niagara Falls 14001    2        fmr          1050
## 4 Buffalo-Cheektowaga-Niagara Falls 14001    2        market rate  1145
## 5 Buffalo-Cheektowaga-Niagara Falls 14004    1        fmr           900
## 6 Buffalo-Cheektowaga-Niagara Falls 14004    1        market rate  1748
```

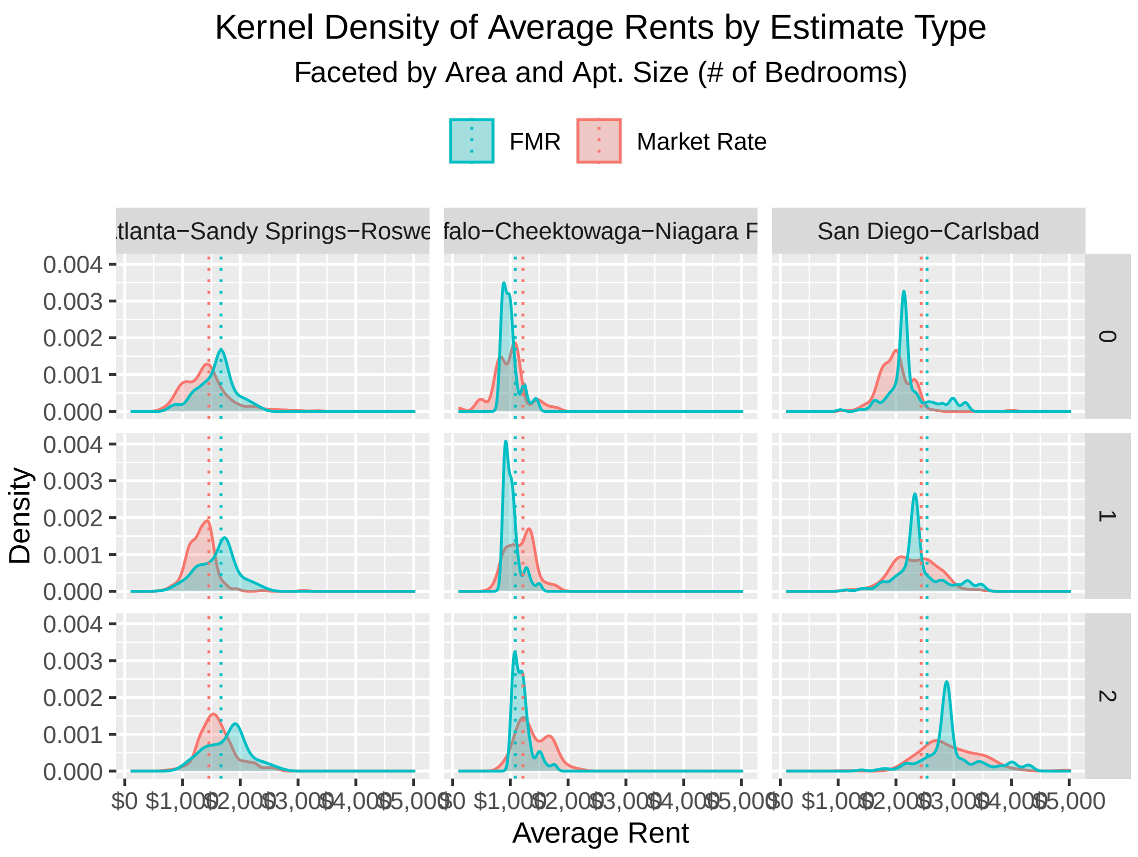
Kernel Density of Estimate Types

With the dataframe tidied, we can start analysing our data. The chart below shows a kernel density of the distribution of average rents by estimate types (FMR vs Market Rent). Using facets, we can see group data by city and apartment size measured in number of bedrooms. The chart suggests that market rates generally trail FMR costs for all apartment sizes in the Atlanta Metropolitan area. In the Buffalo metro, market rates appear to be somewhat similar for studios, but market rate rents have a wider spread are somewhat higher than FMR for 1 and 2 bedrooms. In San Diego, the median FMR are somewhat similar

```
mean_rents <- combined_df_by_zip_and_type |>
  group_by(type, area_name, 1) |>
  summarise(mean_rent= mean(rent)
)
```

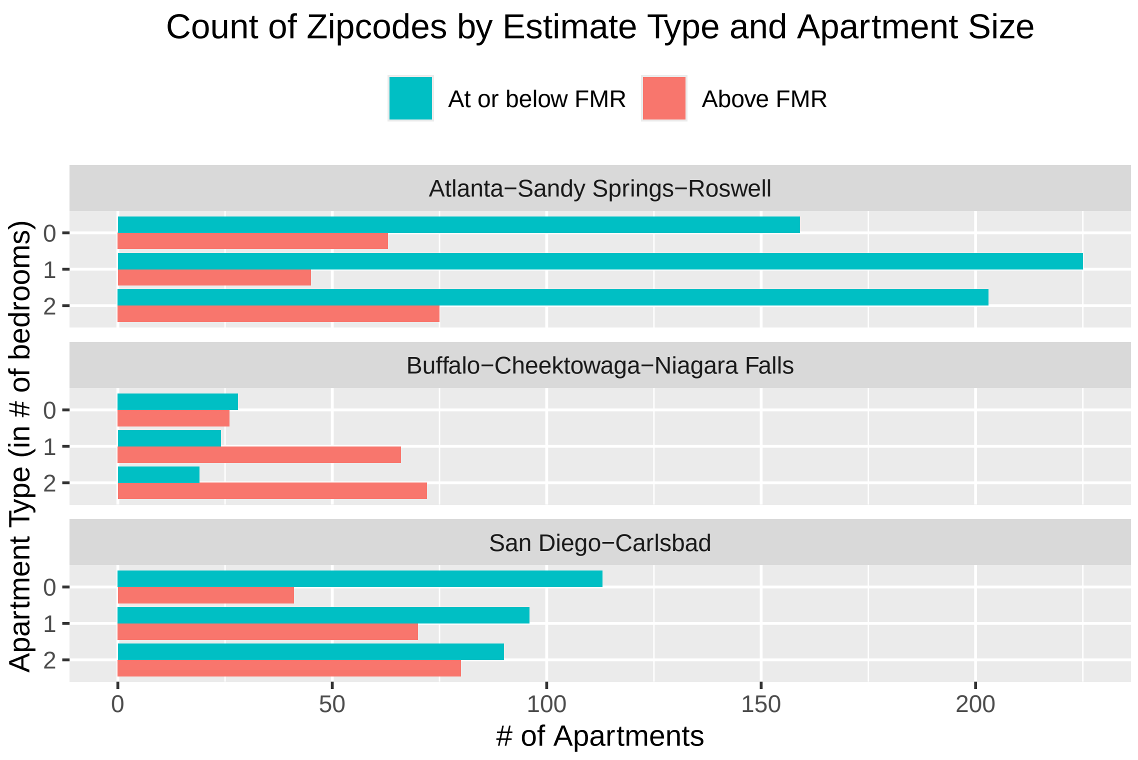
```
## `summarise()` has grouped output by 'type', 'area_name'. You can override using
## the `.groups` argument.
```

```
# distribution of fmr vs market reate

ggplot(combined_df_by_zip_and_type, aes(rent, fill=fct_rev(type), color=fct_rev(type))) +
  geom_density(alpha = .3) +
  geom_vline(data=mean_rents, aes(xintercept=mean_rent, col=type), linetype = "dotted") +
  theme(
    legend.position = "top",
    plot.title = element_text(hjust = 0.5),
    plot.subtitle = element_text(hjust = 0.5)
  ) +
  labs(
    title = "Kernel Density of Average Rents by Estimate Type",
    subtitle = "Faceted by Area and Apt. Size (# of Bedrooms)",
    x = "Average Rent",
    y = "Density"
  ) +
  scale_x_continuous(labels = scales::dollar_format()) +
  scale_fill_discrete(
    name = element_blank(),
    labels=c('Market Rate', 'FMR'),
    guide = guide_legend(reverse = TRUE)
  ) +
  scale_color_discrete(
    name = element_blank(),
    labels=c('Market Rate', 'FMR'),
    guide = guide_legend(reverse = TRUE)
  ) +
  facet_grid(vars(bedrooms), vars(area_name))
```



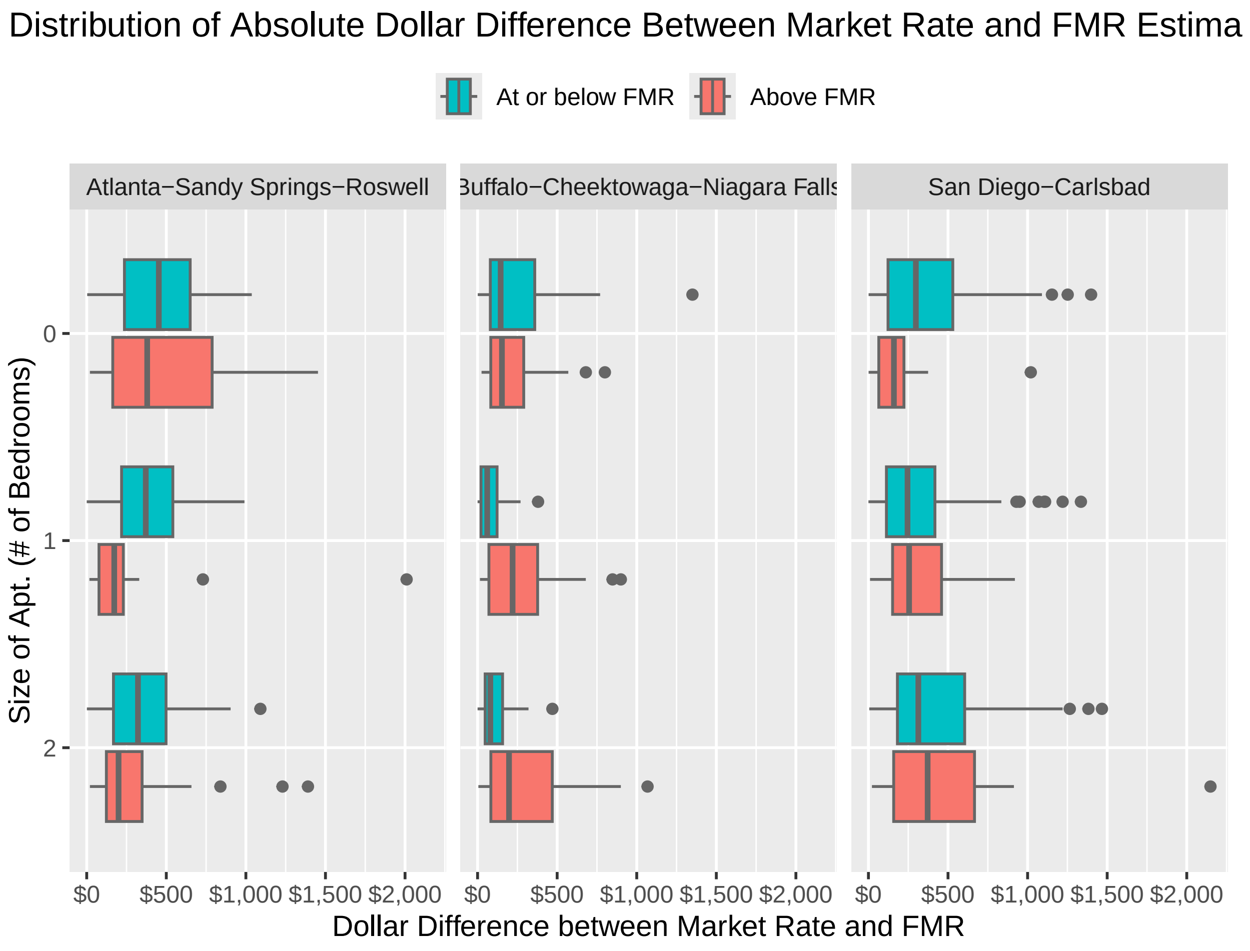
```
# C3: Calculate Rent Differences by Zip
# create df with the margin of error
combined_df_by_zip_diffs <- combined_df_by_zip |>
```

```
  mutate(
    margin_of_error = market_rate - fmr,
    at_or_below_fmr = as.factor(fmr >= market_rate)
  ) |>
  select(-c(fmr, market_rate))


ggplot(combined_df_by_zip_diffs, aes(y=fct_rev(bedrooms), fill=at_or_below_fmr)) +
  geom_bar(position = position_dodge()) +
  theme(
    legend.position = "top",
    plot.title = element_text(hjust = 0.5)
  ) +
  labs(
    title = "Count of Zipcodes by Estimate Type and Apartment Size",
    x = "# of Apartments",
    y = "Apartment Type (in # of bedrooms)"
  ) +
  scale_fill_discrete(
    name = element_blank(),
    labels=c('Above FMR', 'At or below FMR'),
    guide = guide_legend(reverse = TRUE)
  ) +
  facet_wrap(~area_name, ncol=1)
```



```
#  facet_grid(vars(bedrooms), vars(area_name))
```

```r
ggplot(combined_df_by_zip_diffs, aes(abs(margin_of_error), fct_rev(bedrooms), fill=at_or_below_fmr)) +
  geom_boxplot(color="#666666") +
  theme(
    legend.position="top",
    plot.title = element_text(hjust = 0.5)
  ) +
  labs(
    title = "Distribution of Absolute Dollar Difference Between Market Rate and FMR Estimates",
    x = "Dollar Difference between Market Rate and FMR",
    y = "Size of Apt. (# of Bedrooms)"
  ) +
  scale_x_continuous(labels = scales::dollar_format()) +
  scale_fill_discrete(
    name = element_blank(),
    labels=c('Above FMR', 'At or below FMR'),
    guide = guide_legend(reverse = TRUE)
  ) +
  facet_wrap(~area_name, ncol=3, shrink=FALSE)
```



```r
  # facet_grid(vars(bedrooms), vars(area_name))
```

C3: Combine FMR and Market Rate for each MSA

```
### C3: Combine FMR and Market Rate for each MSA

# calculate average market rate for MSA
market_rate_msa_by_num_bds <- market_rates |>
  mutate(
    area_name = case_when(
      as.integer(zip_code) > 90000 ~ 'San Diego-Carlsbad',
      as.integer(zip_code) < 20000 ~ 'Buffalo-Cheektowaga-Niagara Falls',
      TRUE ~ 'Atlanta-Sandy Springs-Roswell'
    )
  ) |>
  group_by(across(all_of(c("area_name", "bedrooms")))) |>
  summarise(
    market = mean(market_rate)
  )
```

```
## `summarise()` has grouped output by 'area_name'. You can override using the
## `.groups` argument.
```

```
# join fmr and market rate dataframes at the MSA level
combined_df_by_msa <- metro_fmr_msa_by_num_bds |>
  right_join(market_rate_msa_by_num_bds, by=c("area_name", "bedrooms"))

combined_df_by_msa_long <- combined_df_by_msa |>
  pivot_longer(
    cols = c(fmr, market),
    names_to = c("type"),
    values_to = "rent"
  )

head(combined_df_by_msa_long)
```

```
## # A tibble: 6 x 5
##   area_name                     zip_code  bedrooms type    rent
##   <chr>                         <chr>        <int> <chr>  <dbl>
## 1 Atlanta-Sandy Springs-Roswell MSA level        0 fmr    1591
## 2 Atlanta-Sandy Springs-Roswell MSA level        0 market 1519.
## 3 Atlanta-Sandy Springs-Roswell MSA level        1 fmr    1653
## 4 Atlanta-Sandy Springs-Roswell MSA level        1 market 1406.
## 5 Atlanta-Sandy Springs-Roswell MSA level        2 fmr    1830
## 6 Atlanta-Sandy Springs-Roswell MSA level        2 market 1724.
```

Summary