Zillow Prize Modeling

Jesse Piburn

2018-07-20

Contents

Welcome

This is a *sample* book written in **Markdown**. You can use anything that Pandoc's Markdown supports, e.g., a math equation $a^2 + b^2 = c^2$.

The **bookdown** package can be installed from CRAN or Github:

```r
install.packages("bookdown")
# or the development version
# devtools::install_github("rstudio/bookdown")
```

Remember each Rmd file contains one and only one chapter, and a chapter is defined by the first-level heading `#`.

To compile this example to PDF, you need XeLaTeX. You are recommended to install TinyTeX (which includes XeLaTeX): https://yihui.name/tinytex/.

Chapter 1

Introduction

This project is meant to serve as an example workflow for making predictive models. The sections of the book are roughly broken into the major steps that are a part of the modeling process.

As with many things in life, their is rarely a one-size-fits-all, always correct, way of doing something and making a predictive model is no different. In light of that, I want to stress that the workflow and methods used in this book are meant to be illustrative not authoritative.

The only always correct answer in predictive modeling is, "It depends."

1.1 Problem

The problem we will workthrough in this book is the Zillow Prize compeitition on that took place on Kaggle. Although it is now closed, it provides a good problemset for us to work through.

From the site >>>In this million-dollar competition, participants will develop an algorithm that makes predictions about the future sale prices of homes. The contest is structured into two rounds, the qualifying round which opens May 24, 2017 and the private round for the 100 top qualifying teams that opens on Feb 1st, 2018. In the qualifying round, you'll be building a model to improve the Zestimate residual error. In the final round, you'll build a home valuation algorithm from the ground up, using external data sources to help engineer new features that give your model an edge over the competition.

1.2 Evaluation

The submission were evaluated based on the Mean Absolute Error between the predicted `logerror` and the actual `logerror`

The `logerror` is defined as

$$logerror = log(Zestimate) - log(SalePrice)$$

while Mean Absolute Error is defined as

$$MAE = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n} = \frac{\sum_{i=1}^{n} |e_i|}{n}$$

1.3 Initital Thoughts

> "Location, Location, Location."
>
> — Some Real Estate Guy

The above quote is of course the number one rule of real estate. Though out this modeling process, let's try to keep this idea in mind when we are exploring, creating new features, and modeling the data.

An additional interesting thing about this problem is that it can be thought about as creating a model to predict where Zillow's model is bad. We aren't trying to predict home prices, we are predicting where Zillow had a bad estimate of home prices, so the residuals of their Zestimate model.Based on this idea, let's create some new features

1.4 Note on Using External Features

In the original competition, you were only allowed to use the features transformations of those included in the data they provided. Since our goal is to provide an illustrative workflow for making a predictive model in general and not actually competing in the competition, we are not going to adhere to that rule and use a few external sources of information.

Chapter 2

PreProcessing

The first step for any data driven project is getting to know the data. In this section we will look at the raw data that was provided by the competition and then do a little pre processing that will make the data easier to work with for the rest of the project

2.1 The Raw Data

The raw data from zillow contains the following data (desrciptions from the the data page)

- **properties_2016.csv** - all the properties with their home features for 2016. Note: Some 2017 new properties don't have any data yet except for their parcelid's. Those data points should be populated when properties_2017.csv is available.
- **properties_2017.csv** - all the properties with their home features for 2017 (released on 10/2/2017)
- **train_2016.csv** - the training set with transactions from 1/1/2016 to 12/31/2016
- **train_2017.csv** - the training set with transactions from 1/1/2017 to 9/15/2017 (released on 10/2/2017)
- **sample_submission.csv** - a sample submission file in the correct format
- **zillow_data_dictionary.xlsx** - Field Definitions and coded value meanings

2.1.1 Saving Raw Data Using `feather`

The R binding for the feather data store provides the ability for very fast read and write of data. For speed purposes we will save all raw data into a `.feather` file format to make all other read faster

```
library(feather)
library(readr)

dir.create("data-raw/feather")

prop_16 <- read_csv("data-raw/properties_2016.csv")
prop_17 <- read_csv("data-raw/properties_2017.csv")
train_16 <- read_csv("data-raw/train_2016_v2.csv")
train_17 <- read_csv("data-raw/train_2017.csv")

write_feather(prop_16, "data-raw/feather/properties_2016.feather")
```

```r
write_feather(prop_17, "data-raw/feather/properties_2017.feather")
write_feather(train_16, "data-raw/feather/train_2016_v2.feather")
write_feather(train_17, "data-raw/feather/train_2017.feather")
```

2.2 Renaming Variables

Many of the feature names are not very consistant. To take advatange of helpful functions from the `tidyverse` set of packages, such as `starts_with()` and `one_of()` Let's rename them to something more consistant and easier to work with.

2.2.1 Renaming `properties` Features

```r
library(tidyverse)

prop_16 <- read_feather("data-raw/feather/properties_2016.feather")
prop_17 <- read_feather("data-raw/feather/properties_2017.feather")

prop_16 <- prop_16 %>%
  rename(
    id_parcel = parcelid,
    build_year = yearbuilt,
    area_basement = basementsqft,
    area_patio = yardbuildingsqft17,
    area_shed = yardbuildingsqft26,
    area_pool = poolsizesum,
    area_lot = lotsizesquarefeet,
    area_garage = garagetotalsqft,
    area_firstfloor_finished_1 = finishedfloor1squarefeet,
    area_firstfloor_finished_2 = finishedsquarefeet50,
    area_living_finished_calc = calculatedfinishedsquarefeet,
    area_base = finishedsquarefeet6,
    area_living_finished = finishedsquarefeet12,
    area_living_perimeter = finishedsquarefeet13,
    area_total = finishedsquarefeet15,
    num_unit = unitcnt,
    num_story = numberofstories,
    num_room = roomcnt,
    num_bathroom = bathroomcnt,
    num_bedroom = bedroomcnt,
    num_bathroom_calc = calculatedbathnbr,
    num_bath = fullbathcnt,
    num_75_bath = threequarterbathnbr,
    num_fireplace = fireplacecnt,
    num_pool = poolcnt,
    num_garage = garagecarcnt,
    region_county = regionidcounty,
    region_city = regionidcity,
    region_zip = regionidzip,
    region_neighbor = regionidneighborhood,
    tax_total = taxvaluedollarcnt,
```

```
    tax_building = structuretaxvaluedollarcnt,
    tax_land = landtaxvaluedollarcnt,
    tax_property = taxamount,
    tax_year = assessmentyear,
    tax_delinquency = taxdelinquencyflag,
    tax_delinquency_year = taxdelinquencyyear,
    zoning_property = propertyzoningdesc,
    zoning_landuse = propertylandusetypeid,
    zoning_landuse_county = propertycountylandusecode,
    str_flag_fireplace = fireplaceflag,
    str_flag_tub = hashottuborspa,
    str_quality = buildingqualitytypeid,
    str_framing = buildingclasstypeid,
    str_material = typeconstructiontypeid,
    str_deck = decktypeid,
    str_story = storytypeid,
    str_heating = heatingorsystemtypeid,
    str_aircon = airconditioningtypeid,
    str_arch_style = architecturalstyletypeid
  )

# use 2016 names to rename 17
names(prop_17) <- names(prop_16)
```

2.2.2 renaming train features

```
trans_16 <- read_feather("data-raw/feather/train_2016_v2.feather")
trans_17 <- read_feather("data-raw/feather/train_2017.feather")

trans_16 <- trans_16 %>%
  rename(
  id_parcel = parcelid,
  date = transactiondate,
  log_error = logerror
  )

# use 2016 names to rename 17
names(trans_17) <- names(trans_16)
```

2.2.3 Basic Transformations

Based on the definitions in the `zillow_data_dictionary.xlsx` we can recode some of the features to have be more interpretable while we are exploring.

2.2.3.1 Properties

```
library(forcats)

prop_16 <- prop_16 %>%
  mutate(
```

```
    tax_delinquency = ifelse(tax_delinquency == "Y", "Yes", "No") %>%
      as_factor(),
    str_flag_fireplace = ifelse(str_flag_fireplace == "Y", "Yes", "No") %>%
      as_factor(),
    str_flag_tub = ifelse(str_flag_tub == "Y", "Yes", "No") %>%
      as_factor(),
    zoning_landuse = factor(zoning_landuse, levels = sort(unique(zoning_landuse))),
    zoning_landuse = fct_recode(zoning_landuse,
      "Commercial/Office/Residential Mixed Used" = "31",
      "Multi-Story Store"                        = "46",
      "Store/Office (Mixed Use)"                 = "47",
      "Duplex (2 Units Any Combination)"         = "246",
      "Triplex (3 Units Any Combination)"        = "247",
      "Quadruplex (4 Units Any Combination)"     = "248",
      "Residential General"                      = "260",
      "Single Family Residential"                = "261",
      "Rural Residence"                          = "262",
      "Mobile Home"                              = "263",
      "Townhouse"                                = "264",
      "Cluster Home"                             = "265",
      "Condominium"                              = "266",
      "Cooperative"                              = "267",
      "Row House"                                = "268",
      "Planned Unit Development"                 = "269",
      "Residential Common Area"                  = "270",
      "Timeshare"                                = "271",
      "Bungalow"                                 = "273",
      "Zero Lot Line"                            = "274",
      "Manufactured Modular Prefabricated Homes" = "275",
      "Patio Home"                               = "276",
      "Inferred Single Family Residential"       = "279",
      "Vacant Land - General"                    = "290",
      "Residential Vacant Land"                  = "291"
      )
    )

prop_17 <- prop_17 %>%
  mutate(
    tax_delinquency = ifelse(tax_delinquency == "Y", "Yes", "No") %>%
      as_factor(),
    str_flag_fireplace = ifelse(str_flag_fireplace == "Y", "Yes", "No") %>%
      as_factor(),
    str_flag_tub = ifelse(str_flag_tub == "Y", "Yes", "No") %>%
      as_factor(),
    zoning_landuse = factor(zoning_landuse, levels = sort(unique(zoning_landuse))),
    zoning_landuse = fct_recode(zoning_landuse,
      "Commercial/Office/Residential Mixed Used" = "31",
      "Multi-Story Store"                        = "46",
      "Store/Office (Mixed Use)"                 = "47",
      "Duplex (2 Units Any Combination)"         = "246",
      "Triplex (3 Units Any Combination)"        = "247",
      "Quadruplex (4 Units Any Combination)"     = "248",
      "Residential General"                      = "260",
```

```
      "Single Family Residential"                = "261",
      "Rural Residence"                          = "262",
      "Mobile Home"                              = "263",
      "Townhouse"                                = "264",
      "Cluster Home"                             = "265",
      "Condominium"                              = "266",
      "Cooperative"                              = "267",
      "Row House"                                = "268",
      "Planned Unit Development"                 = "269",
      "Residential Common Area"                  = "270",
      "Timeshare"                                = "271",
      "Bungalow"                                 = "273",
      "Zero Lot Line"                            = "274",
      "Manufactured Modular Prefabricated Homes" = "275",
      "Patio Home"                               = "276",
      "Inferred Single Family Residential"       = "279",
      "Vacant Land - General"                    = "290",
      "Residential Vacant Land"                  = "291"
      )
    )
```

2.2.3.2 Transactions

The transactions tables are where our response variable `log_error` (name changed from orginal `logerror`) and the dates of the transactions are recorded.

To make them easier to work with, let's combine all the transactions into one table and create a few basic transformations of the `date` (name changed from original `transactiondate`)

```r
library(lubridate)

# combine transactions into one data frame
trans <- trans_16 %>%
  bind_rows(trans_17) %>%
  mutate(
    abs_log_error = abs(log_error),
    year = year(date),
    month_year = make_date(year(date), month(date)),
    month = month(date, label = TRUE),
    week = floor_date(date, unit = "week"),
    week_of_year = week(date),
    week_since_start = (min(date) %--% date %/% dweeks()) + 1,
    wday = wday(date, label = TRUE),
    day_of_month = day(date)
  )
```

Save our output

```r
write_feather(prop_16, "data/properties_16.feather")
write_feather(prop_17, "data/properties_17.feather")
write_feather(trans, "data/transactions.feather")
```

2.3 Extracting Geographic Information

As noted in the 1 we are going to break from the rules of the competition and use external information to (hopefully) help improve our predictions. Since the data we are using relate to locations of individual properties and we have each of their geographic coordinates, `latitude` and `longitude` let's use those to get the U.S. Census Geographies they are apart of that we can make use of when adding external information.

The original data contain the fields `rawcensustractandblock` and `censustractandblock` but after trying to parse those into a usable format and failing, I figured it was just easier to use the `latitude` and `longitude` fields and then join that to the Census information.

```r
library(sf)
library(tidycensus)

# NAD83 / California zone 5 (ftUS)
# https://epsg.io/2229
crs_id <- 2229

api_key <- Sys.getenv("CENSUS_API_KEY")
census_api_key(api_key)

# some obs have no data at all included lat/long
# the original lat / lon are mulitpled by 10e5 so divide to
# get lat lon back when converting to sf
properties <- read_feather("data-raw/properties_2017") %>%
  filter(!is.na(latitude)) %>%
  mutate(
    lat = latitude / 10e5,
    lon = longitude / 10e5
    ) %>%
  st_as_sf(
    coords = c("lon", "lat"),
    crs = 4326, # WGS 84
    remove = FALSE # keep lat/long fields
    ) %>%
  st_transform(crs_id)

census_bgs <- get_acs(
  geography = "block group",
  variables = "B19013_001",
  state = "CA",
  county = c("Los Angeles", "Orange", "Ventura"),
  geometry = TRUE,
  keep_geo_vars = TRUE
  ) %>%
  st_transform(crs_id)

# inner join
# due to lat / lon error some points didn't intersect
# with block groups left = FALSE is inner join
properties_geo <- properties %>%
  st_join(census_bgs, left = FALSE)

# find all of the points that didn't intersect
```

```
# buffer them and then join to closest block
# then add back the already joined points
# the buffer distance I just played around with until
# all points joined with a block group
properties_geo <- properties %>%
  filter(!parcelid %in% properties_geo$parcelid) %>%
  st_buffer(dist = 1500) %>% # units are in us-ft based on crs_id
  st_join(census_bgs, left = FALSE, largest = TRUE) %>%
  rbind(properties_geo)


# remove geometry b/c feather can't store lists
properties_geo <- properties_geo %>%
  select(
    id_parcel = parcelid,
    id_geo_state = STATEFP,
    id_geo_county = COUNTYFP,
    id_geo_tract = TRACTCE,
    id_geo_bg = BLKGRPCE,
    id_geo_bg_fips = GEOID,
    id_geo_bg_name = NAME.y,
    geo_bg_arealand = ALAND,
    geo_bg_areawater = AWATER,
    lat,
    lon
    ) %>%
  mutate(
    id_geo_county_fips = paste0(id_geo_state, id_geo_county),
    id_geo_tract_fips = paste0(id_geo_county_fips, id_geo_tract),
    id_geo_county_name = factor(id_geo_county) %>%
      fct_recode(
        "Los Angeles" = "037",
        "Orange"      = "059",
        "Ventura"     = "111"
      )
  )
```

Now save our geographic features

```
# remove geometry b/c feather can't store lists
# add back in when needed from lat lon
properties_geo$geometry <- NULL

write_feather(properties_geo, "data/properties_geo_only.feather")
```

Chapter 3

Exploratory Analysis

After ?? the next step is doing exploratory data analysis (EDA). I can't stess enough how critical this step is. It is tempting to want to jump right into making models and then start improving and tweaking them from there, but this can quickly take you down a rabbit hole, waste your time, and generally make you sad.

The time you spend doing EDA will pay dividends later.

Below we are first going to look at only our response variable `log_error` after that we will look at the predictor features in `properties`. Once we get a good handle on both of those, we'll look at how our response variable `log_error` varies across the predictors in `properties`

Throughout this section we will progressively pare down features in our `properties` data due to common things such as, missingness and redundancy, to only the features that we are going to continue with into the next stages.

```
library(skimr)
library(tidyverse)
library(feather)
library(DataExplorer)

trans <- read_feather("data/transactions.feather")
```

3.1 Response Variable

`log_error` something something text come back to when the processing is finished

```
skim(trans) %>%
  skimr::pander()
```

Skim summary statistics
n obs: 167888
n variables: 12

Table 3.1: Table continues below

variable	missing	complete	n	min	max
date	0	167888	167888	2016-01-01	2017-09-25
month_year	0	167888	167888	2016-01-01	2017-09-01
week	0	167888	167888	2015-12-27	2017-09-24

median	n_unique
2016-10-11	616
2016-10-01	21
2016-10-09	92

Table 3.3: Table continues below

variable	missing	complete	n	n_unique
month	0	167888	167888	12
wday	0	167888	167888	7

top_counts	ordered
Jun: 22378, May: 20448, Aug: 20412, Jul: 19437	FALSE
Fri: 44914, Thu: 34143, Wed: 32692, Tue: 31404	FALSE

Table 3.5: Table continues below

variable	missing	complete	n	mean	sd	p0
day_of_month	0	167888	167888	16.43	8.99	1
id_parcel	0	167888	167888	1.3e+07	3e+06	1.1e+07

p25	p50	p75	p100
9	16	24	31
1.2e+07	1.3e+07	1.4e+07	1.7e+08

Table 3.7: Table continues below

variable	missing	complete	n	mean	sd	p0
abs_log_error	0	167888	167888	0.069	0.15	0
log_error	0	167888	167888	0.014	0.17	-4.66
week_of_year	0	167888	167888	22.15	11.5	1
week_since_start	0	167888	167888	46.31	26.84	1
year	0	167888	167888	2016.46	0.5	2016

p25	p50	p75	p100
0.014	0.032	0.069	5.26
-0.025	0.006	0.039	5.26
13	22	31	53
23	41	72	91
2016	2016	2017	2017

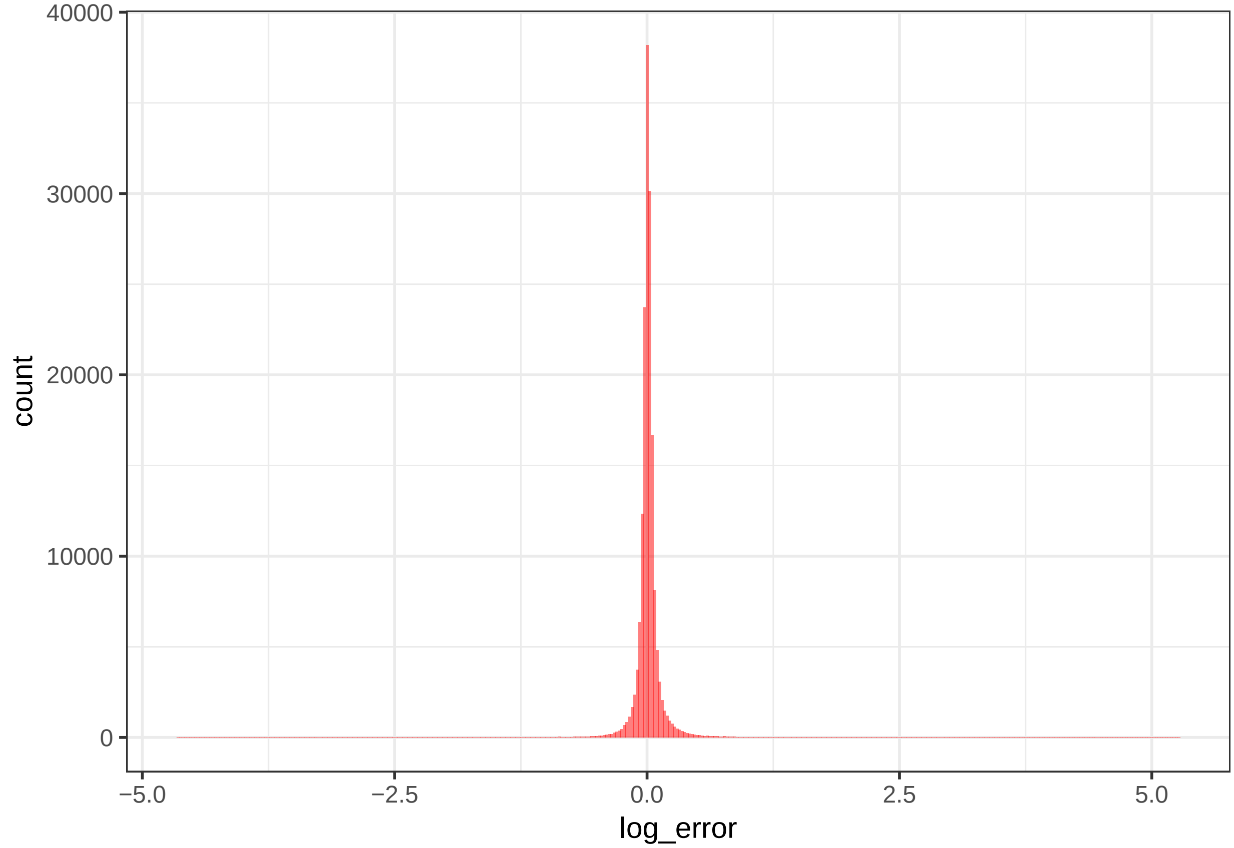


Figure 3.1: Distribution of Log Error

```r
trans %>%
  ggplot(aes(x = log_error)) +
  geom_histogram(bins=400, fill = "red", alpha = 0.5) +
  theme_bw()
```

```r
trans %>%
  filter(
    log_error > quantile(log_error, probs = c(.05)),
    log_error < quantile(log_error, probs = c(.95))
  ) %>%
  ggplot(aes(x = log_error)) +
  geom_histogram(fill = "red", alpha = 0.5) +
  theme_bw()
```

```r
trans %>%
  filter(
    log_error > quantile(abs_log_error, probs = c(.05)),
    log_error < quantile(abs_log_error, probs = c(.95))
  ) %>%
  ggplot(aes(x = abs_log_error)) +
  geom_histogram(fill = "red", alpha = 0.5) +
  theme_bw()
```

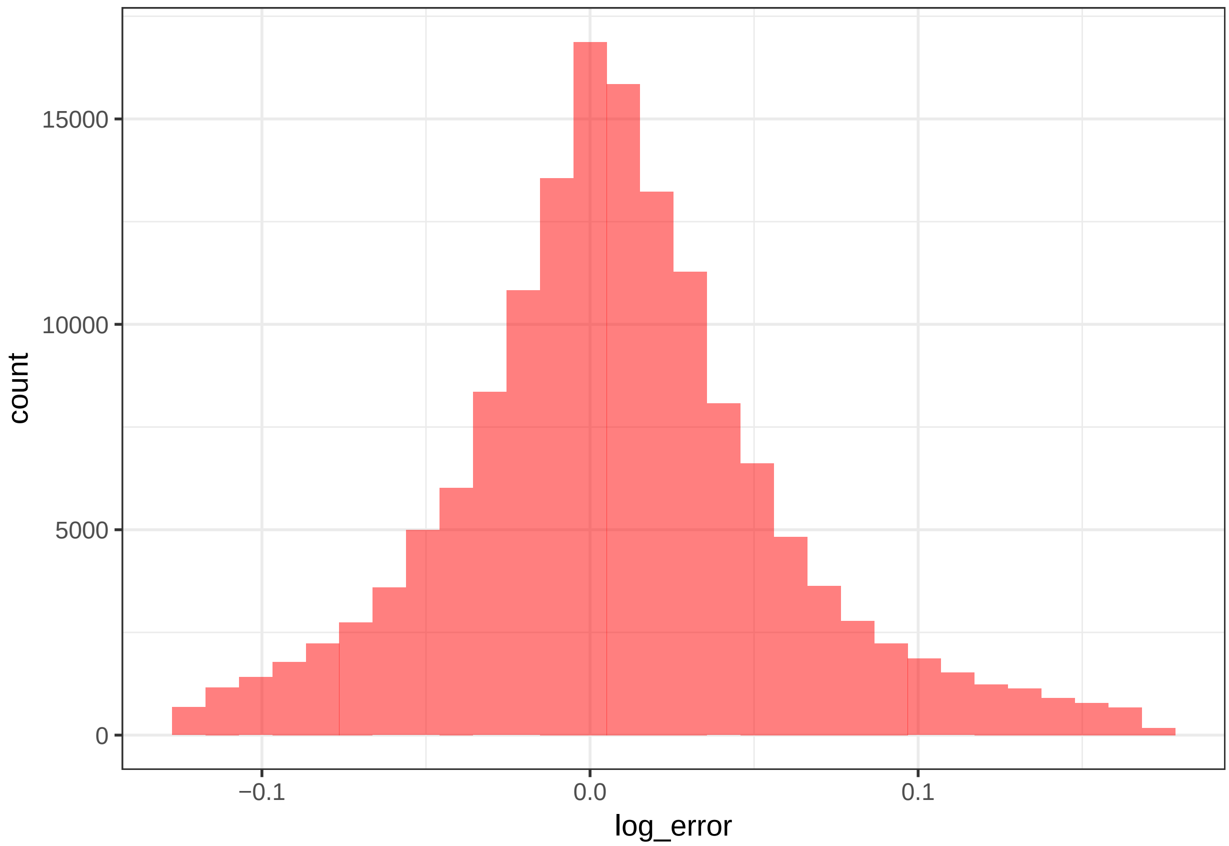


Figure 3.2: Distribution of Log Error Between 5 and 95 Percentile

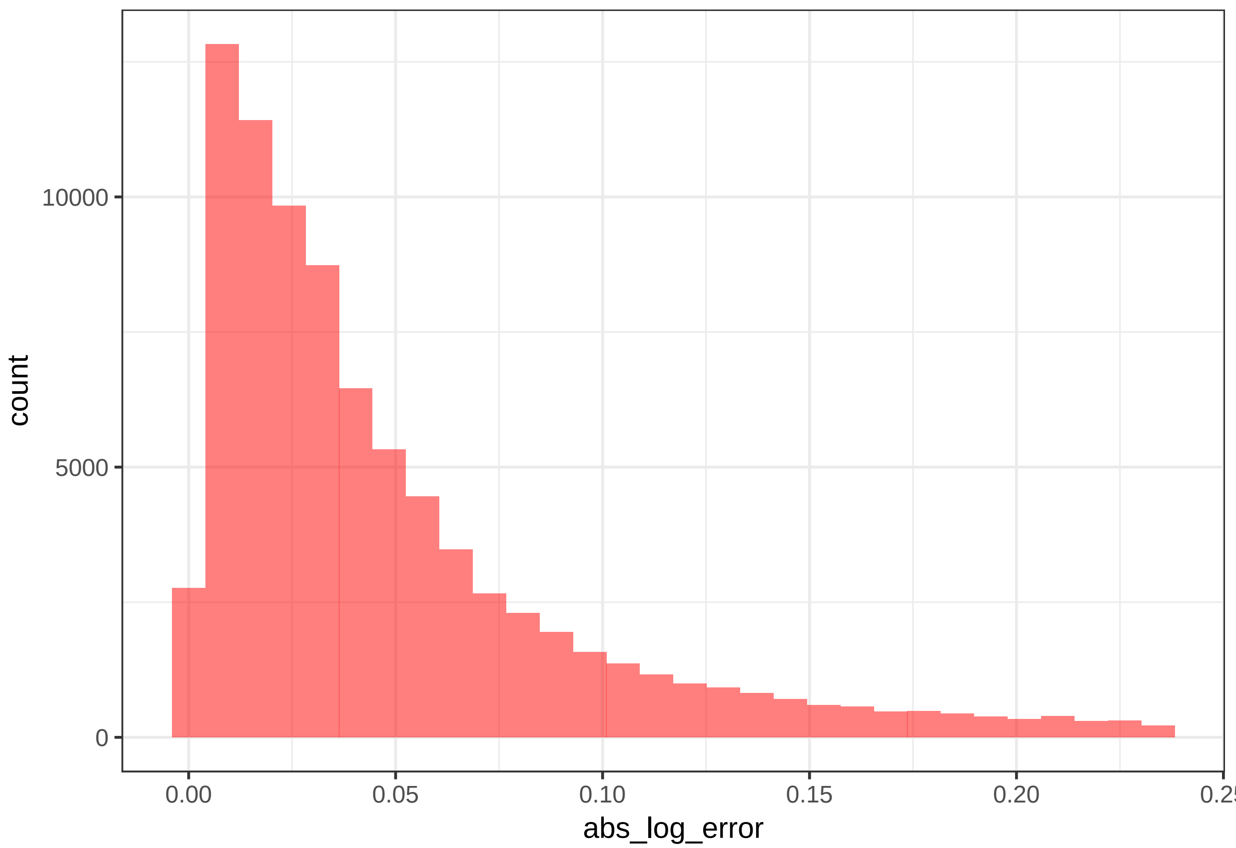


Figure 3.3: Distribution of Absolute value of Log Error Between 5 and 95 Percentile

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
trans %>%
  group_by(month_year) %>%
  summarise(mean_log_error = mean(log_error)) %>%
  ggplot(aes(x = month_year, y = mean_log_error)) +
  geom_line(size = 1, colour = "red") +
  geom_point(size = 3, colour = "red") +
  theme_bw()
```

```
trans %>%
  group_by(month_year, year, month) %>%
  summarise(mean_log_error = mean(log_error)) %>%
  ungroup() %>%
  ggplot(aes(x = as.numeric(month), y = mean_log_error)) +
  geom_path(aes(colour = as.factor(year)), size = 1) +
  theme_bw() +
  ylim(c(0, .03)) +
  scale_x_continuous(breaks = 1:12, labels = levels(trans$month)) +
  labs(
    colour = NULL,
    x = "month"
  )
```

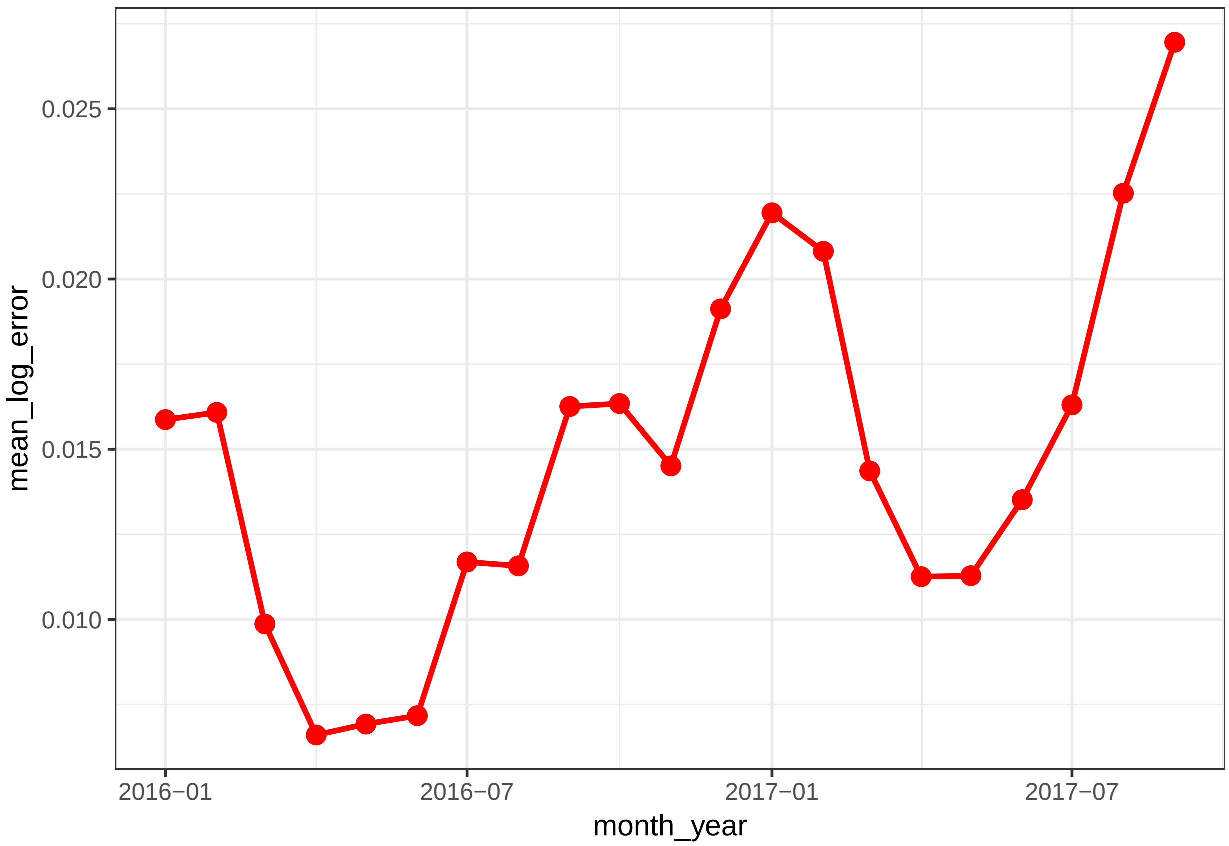


Figure 3.4: Average Log Error by Month

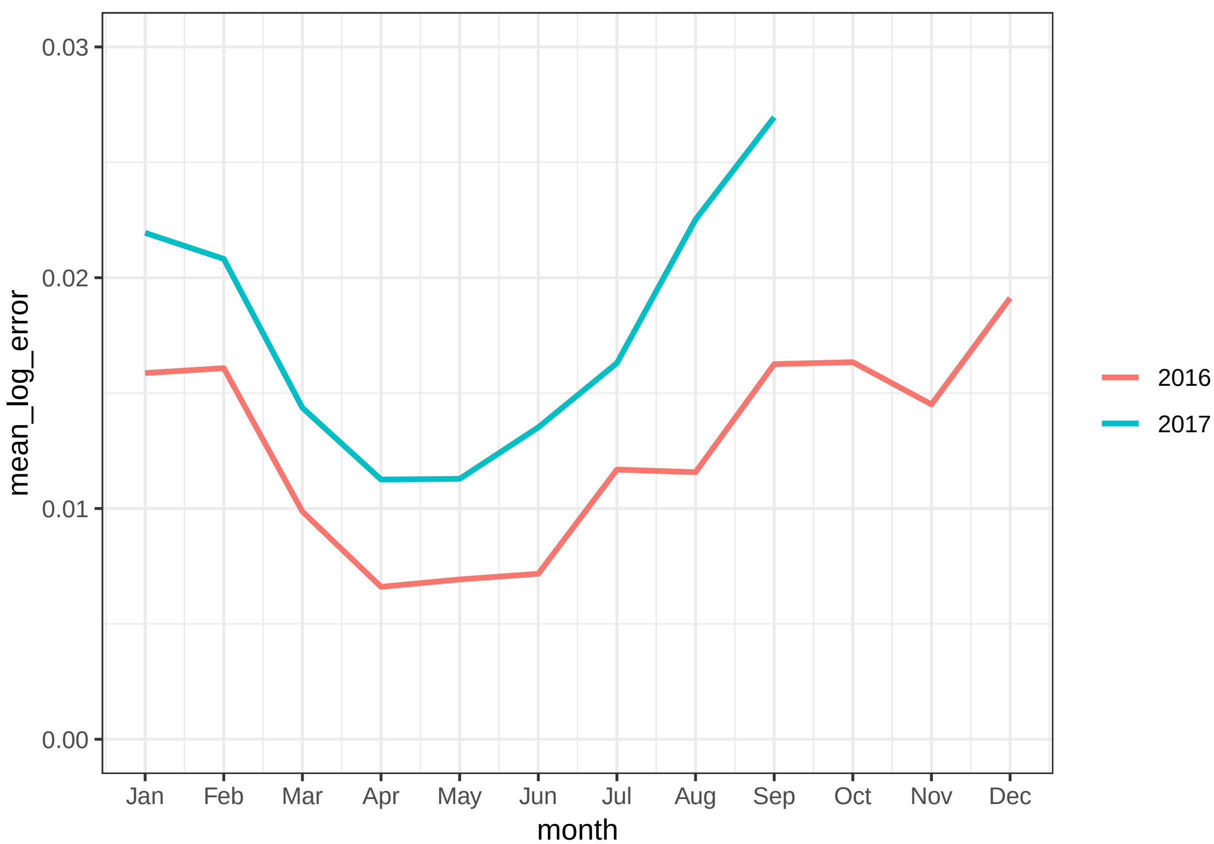


Figure 3.5: Average Log Error by Month. 2017 Looks to have a higher baseline

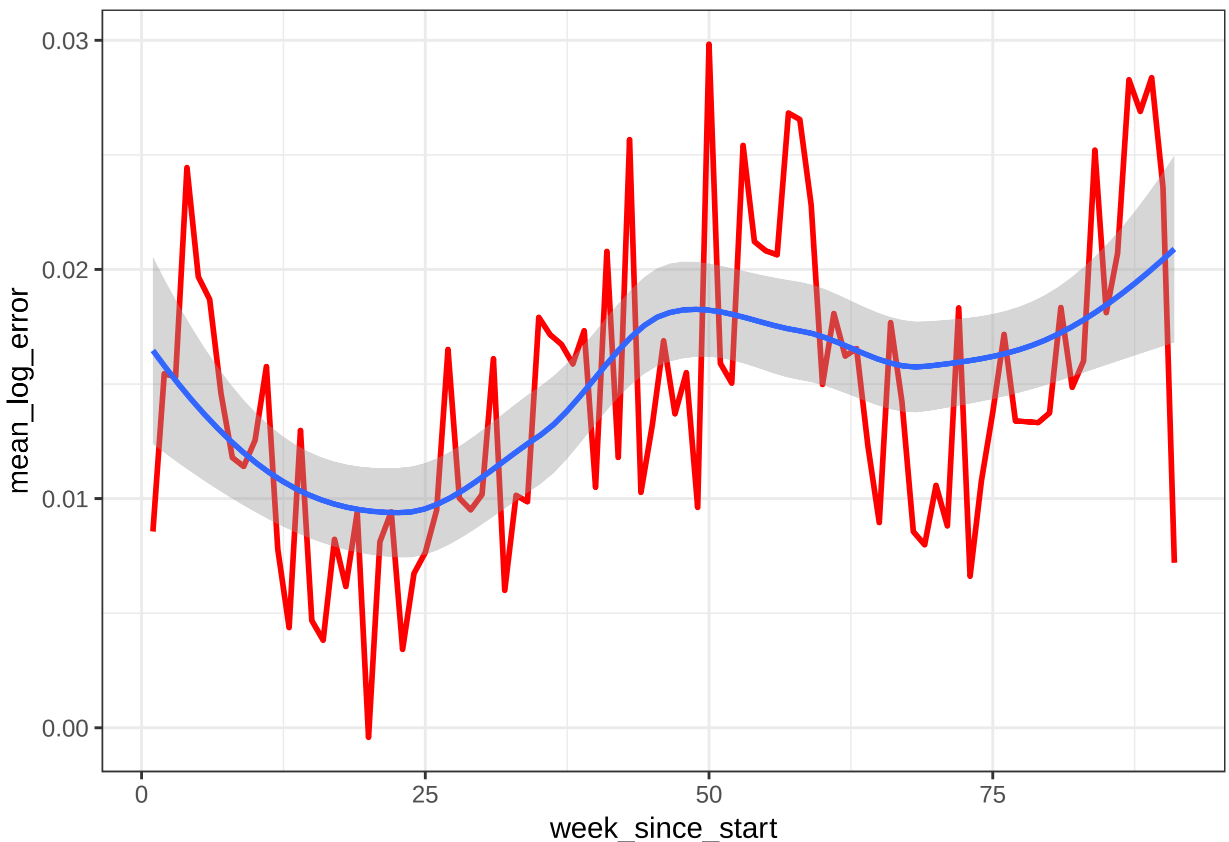


Figure 3.6: Average Log Error by Week

```
trans %>%
  group_by(week_since_start) %>%
  summarise(mean_log_error = mean(log_error)) %>%
  ggplot(aes(x = week_since_start, y = mean_log_error)) +
  geom_line(colour = "red", size = 1) +
  geom_smooth() +
  theme_bw()
```

3.1.1 Transactions Over Time

```
trans %>%
  group_by(week_since_start) %>%
  summarise(n = n()) %>%
  ggplot(aes(x = week_since_start, y = n)) +
  geom_line(colour = "red", size = 1) +
  theme_bw() +
  labs(
    y = "Numeber of Transactions"
  )
```

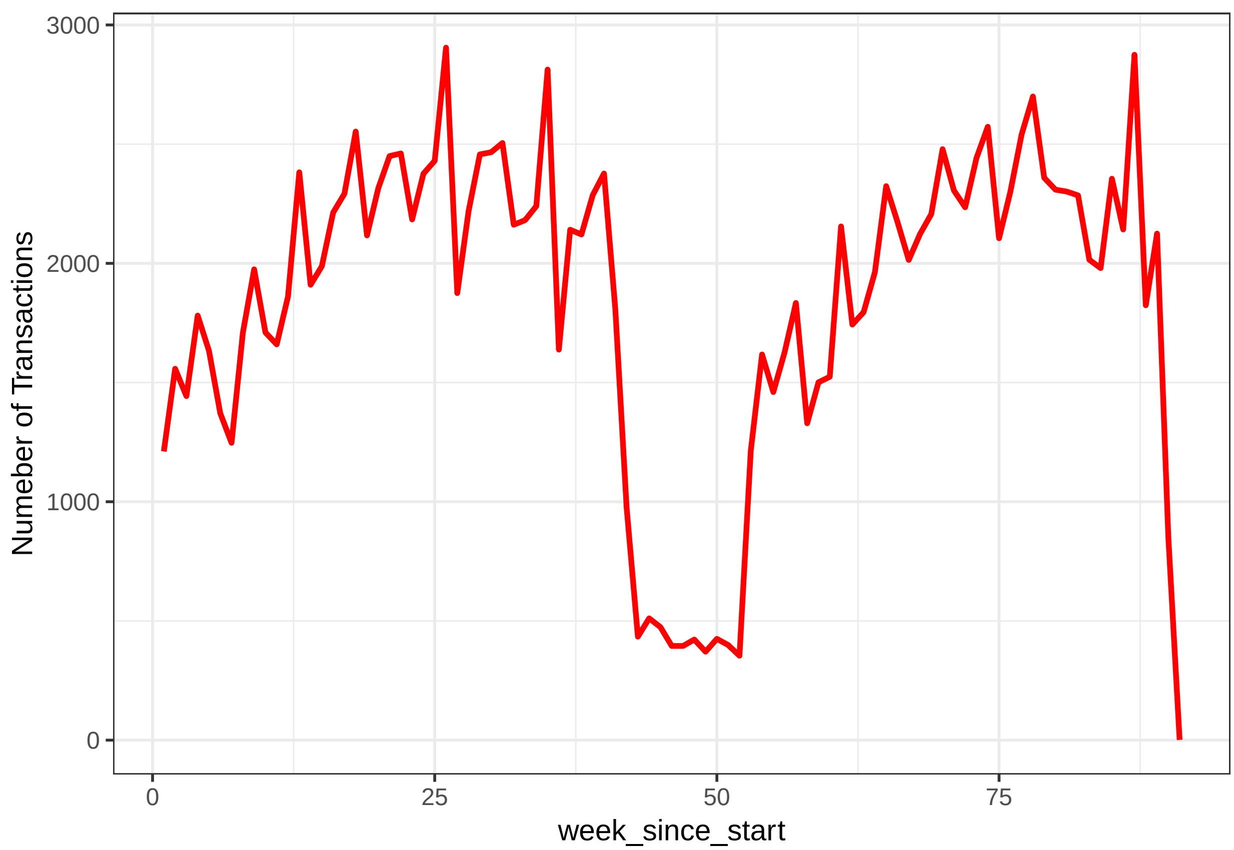


Figure 3.7: Number of Transactions per Week. The dip in the middle coorisponds to the hold out testing data

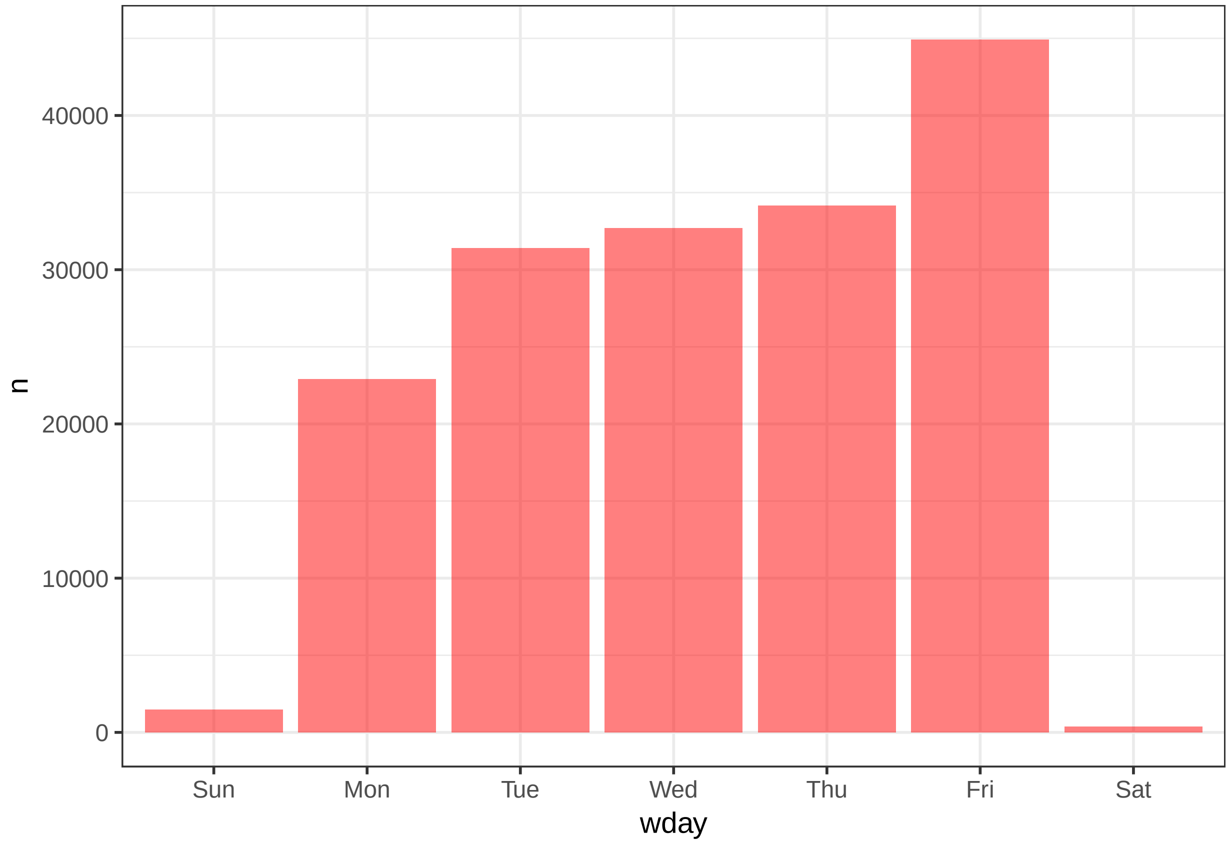


Figure 3.8: Number of Transactions by Day of the Week.

```
trans %>%
  group_by(wday) %>%
  count() %>%
  ggplot(aes(x = wday, y = n)) +
  geom_bar(stat = "identity", fill = "red", alpha = 0.5) +
  theme_bw()
```

3.1.2 Spatial Distribution of `log_error`

```
library(leaflet)
library(leaflet.extras)

read_feather("data/properties_geo_only.feather") %>%
  right_join(trans, by = "id_parcel") %>%
  filter(
    !is.na(lat)
    ) %>%
leaflet() %>%
  addProviderTiles(providers$CartoDB.DarkMatter) %>%
  addHeatmap(lng=~lon, lat=~lat, intensity = ~log_error,
             radius = 5, minOpacity = 0.2, cellSize = 6)
```

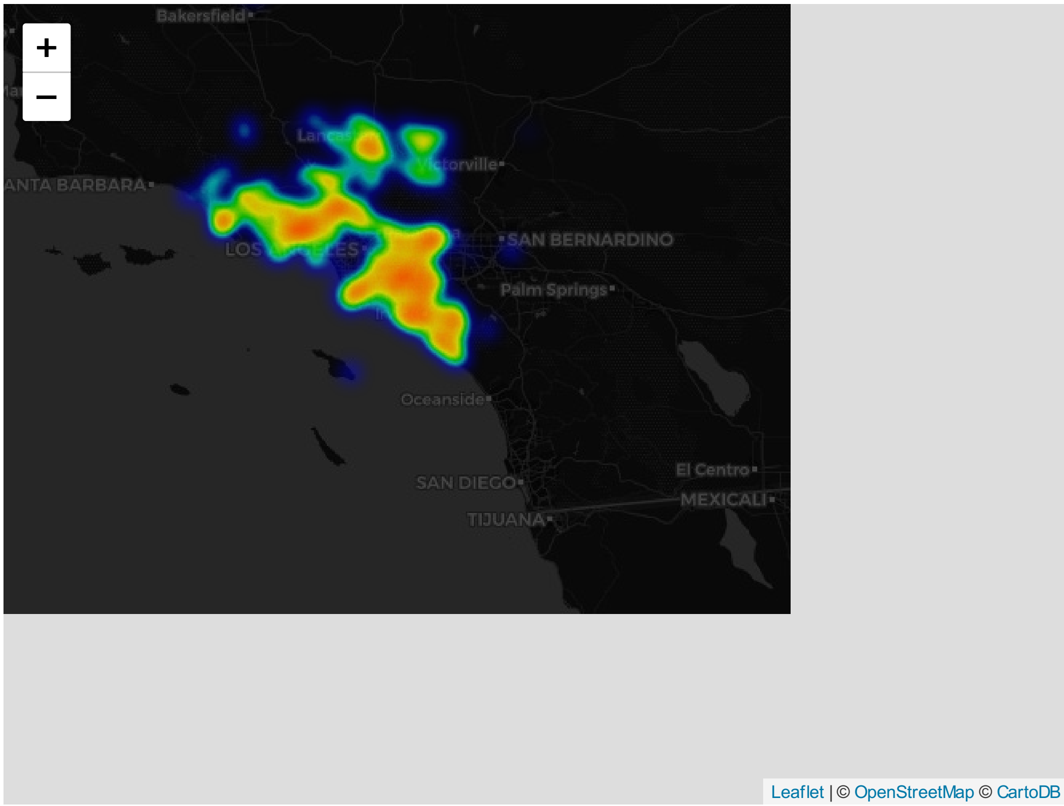


Figure 3.9: Distribution of Log Errors

3.2 Predictor Variables

Now lets take a look at the `properties` dataset. Based on the descriptions from Kaggle, it seem's like the `properties_17.csv` has updated information and is a replacement from that of `properties_16.csv`. For our purposes we are only going to use `properties_17.csv`, however given more space, it would be interesting to look into the differences in these files to see if there were any patterns that could be useful.

```
properties <- read_feather("data/properties_17.feather")
```

```
skim(properties) %>%
  skimr::pander()
```

```
## Warning: Skimr's histograms incorrectly render with pander on Windows.
## Removing them. Use kable() if you'd like them rendered.
```

Skim summary statistics
n obs: 2985217
n variables: 58

variable	missing	complete	n	min	max	empty	n_unique
fips	2932	2982285	2985217	5	5	0	3
pooltypeid10	2968211	17006	2985217	1	1	0	1
pooltypeid2	2952161	33056	2985217	1	1	0	1
rawcensustractandblock	2932	2982285	2985217	12	16	0	100529
str_arch_style	2979156	6061	2985217	1	2	0	8
str_material	2978471	6746	2985217	1	2	0	5

variable	missing	complete	n	min	max	empty	n_unique
zoning_landuse_county	2999	2982218	2985217	1	4	0	234
zoning_property	1002746	1982471	2985217	1	10	0	5651

Table 3.10: Table continues below

variable	missing	complete	n	n_unique
str_flag_fireplace	2980054	5163	2985217	1
str_flag_tub	2935155	50062	2985217	1
tax_delinquency	2928702	56515	2985217	1
zoning_landuse	2932	2982285	2985217	16

top_counts	ordered
NA: 2980054, No: 5163	FALSE
NA: 2935155, No: 50062	FALSE
NA: 2928702, Yes: 56515	FALSE
Sin: 2152863, Con: 483789, Dup: 114415, Pla: 61559	FALSE

Table 3.12: Table continues below

variable	missing	complete	n	mean
area_base	2963735	21482	2985217	2427.56
area_basement	2983590	1627	2985217	647.22
area_firstfloor_finished_1	2781459	203758	2985217	1379.78
area_firstfloor_finished_2	2781459	203758	2985217	1392.03
area_garage	2094209	891008	2985217	383.16
area_living_finished	264431	2720786	2985217	1764.04
area_living_perimeter	2977546	7671	2985217	1178.92
area_patio	2903629	81588	2985217	321.54
area_pool	2957259	27958	2985217	519.72
area_shed	2982571	2646	2985217	278.37
area_total	2795032	190185	2985217	2754.87
id_parcel	0	2985217	2985217	1.3e+07
latitude	2932	2982285	2985217	3.4e+07
longitude	2932	2982285	2985217	-1.2e+08
num_75_bath	2668860	316357	2985217	1.01
num_bath	117156	2868061	2985217	2.25
num_fireplace	2672093	313124	2985217	1.17
num_garage	2094209	891008	2985217	1.83
num_pool	2445585	539632	2985217	1
num_story	2299541	685676	2985217	1.4
num_unit	1004175	1981042	2985217	1.18
pooltypeid7	2479322	505895	2985217	1
region_city	62128	2923089	2985217	34987.66
region_county	2932	2982285	2985217	2569.09
region_neighbor	1828476	1156741	2985217	193538.7
region_zip	12714	2972503	2985217	96553.29
str_aircon	2169855	815362	2985217	1.95

variable	missing	complete	n	mean
str__deck	2967838	17379	2985217	66
str__framing	2972486	12731	2985217	3.73
str__heating	1116053	1869164	2985217	4.08
str__quality	1043822	1941395	2985217	6.28
str__story	2983594	1623	2985217	7
tax__delinquency__year	2928700	56517	2985217	13.89
tax__year	2933	2982284	2985217	2016

sd	p0	p25	p50	p75	p100
7786.19	117	1072	2008	3411	952576
538.79	20	272	535	847.5	8516
634.42	1	1010	1281	1615	31303
682.32	3	1012	1284	1619	41906
246.22	0	312	441	494	7749
1031.38	1	1198	1542	2075	427079
357.09	120	960	1296	1440	2688
236.88	10	190	270	390	7983
191.33	19	430	495	594	17410
369.78	10	96	168	320	6141
5999.38	112	1696	2173	2975	820242
7909966.39	1.1e+07	1.2e+07	1.3e+07	1.4e+07	1.7e+08
243515.71	3.3e+07	3.4e+07	3.4e+07	3.4e+07	3.5e+07
345591.77	-1.2e+08	-1.2e+08	-1.2e+08	-1.2e+08	-1.2e+08
0.12	1	1	1	1	7
0.99	1	2	2	3	32
0.46	1	1	1	1	9
0.61	0	2	2	2	25
0	1	1	1	1	1
0.54	1	1	1	2	41
2.49	1	1	1	1	997
0	1	1	1	1	1
50709.68	3491	12447	25218	45457	4e+05
788.68	1286	1286	3101	3101	3101
165725.27	6952	46736	118920	274800	764167
3680.82	95982	96180	96377	96974	4e+05
3.16	1	1	1	1	13
0	66	66	66	66	66
0.5	1	3	4	4	5
3.29	1	2	2	7	24
1.73	1	5	6	8	12
0	7	7	7	7	7
2.56	0	14	14	15	99
0.06	2000	2016	2016	2016	2016

Table 3.14: Table continues below

variable	missing	complete	n	mean
area__living__finished__calc	45097	2940120	2985217	1831.46
area__lot	272706	2712511	2985217	22603.76

variable	missing	complete	n	mean
build_year	47833	2937384	2985217	1964.44
censustractandblock	74985	2910232	2985217	6e+13
num_bathroom	2957	2982260	2985217	2.22
num_bathroom_calc	117156	2868061	2985217	2.3
num_bedroom	2945	2982272	2985217	3.09
num_room	2969	2982248	2985217	1.47
tax_building	46464	2938753	2985217	178142.89
tax_land	59926	2925291	2985217	268455.77
tax_property	22752	2962465	2985217	5408.95
tax_total	34266	2950951	2985217	443527.93

sd	p0	p25	p50	p75	p100
1954.2	1	1215	1574	2140	952576
249983.63	100	5683	7000	9893	3.7e+08
23.64	1801	1950	1963	1981	2016
3.2e+11	-1	6e+13	6e+13	6.1e+13	4.8e+14
1.08	0	2	2	3	32
1	1	2	2	3	32
1.27	0	2	3	4	25
2.84	0	0	0	0	96
460050.31	1	77666	127066	2e+05	2.6e+08
486509.71	1	79700	176619	326100	9.4e+07
9675.57	0.24	2468.62	4007.62	6230.5	3823175.65
816336.63	1	188220	321161	514072	3.2e+08

3.2.1 Missingness

Missing values in data is a cold cruel reality. It is one of the most contraining factors there is when it comes to predictive power. Having a good understanding of the prevalence of missing values and any patterns to them is needed to make the most out of what data you do have.

```r
missing_data <- plot_missing(properties, theme = theme_bw())
```

```r
missing_data
```

```
## # A tibble: 58 x 4
##    feature            num_missing pct_missing group
##    <fct>                    <int>       <dbl> <chr>
##  1 id_parcel                    0    0        Good
##  2 str_aircon             2169855    0.727    Bad
##  3 str_arch_style         2979156    0.998    Remove
##  4 area_basement          2983590    0.999    Remove
##  5 num_bathroom              2957    0.000991 Good
##  6 num_bedroom               2945    0.000987 Good
##  7 str_framing            2972486    0.996    Remove
##  8 str_quality            1043822    0.350    OK
##  9 num_bathroom_calc       117156    0.0392   Good
## 10 str_deck               2967838    0.994    Remove
## # ... with 48 more rows
```

There seem to be quite a lot of missing features. For now lets remove the ones that are over 50% and continue

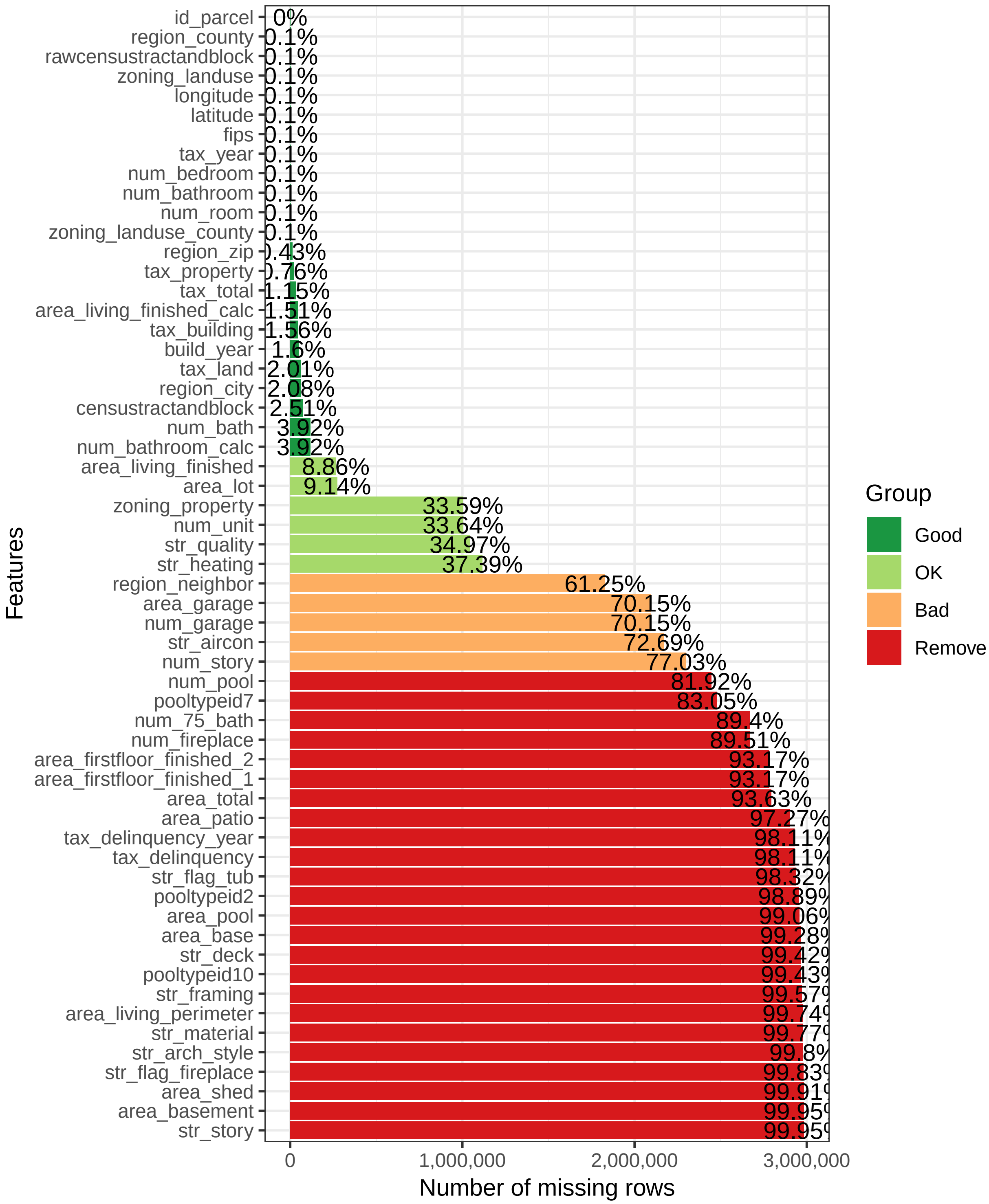


Figure 3.10: Completeness by Feature. Many are extremely sparse

on with those. We could come back to the ones we dropped and try to recover some of those missing values with more sophisticated methods, for example we could impute the missing values based on their spatial neighbors but for now we will continue with the ones that have over 50% of their values.

A few of the features, `rawcensustractandblock`, `fips`, and `censustractandblock`, and `region_county` are ID fields for their census geography units. Since we have already extracted that information earlier in `properties_geo` we will drop them here as well since we can add the information contained in those features in a cleaner format later.

Additionally, based on the descriptions, `zoning_landuse`, `zoning_landuse_county`, and `zoning_property` all seem to contain pretty similar information.Since the number of unique categories are fairly large for each one, if they are redundant they could add needless complexity and computation time to our model. Let's use a chi-squared test to see what it looks like

```
chisq.test(properties$zoning_landuse, properties$zoning_property)
```

```
##
##  Pearson's Chi-squared test
##
## data:  properties$zoning_landuse and properties$zoning_property
## X-squared = 4377000, df = 73450, p-value < 2.2e-16
```

```
chisq.test(properties$zoning_landuse, properties$zoning_landuse_county)
```

```
##
##  Pearson's Chi-squared test
##
## data:  properties$zoning_landuse and properties$zoning_landuse_county
## X-squared = 37455000, df = 3262, p-value < 2.2e-16
```

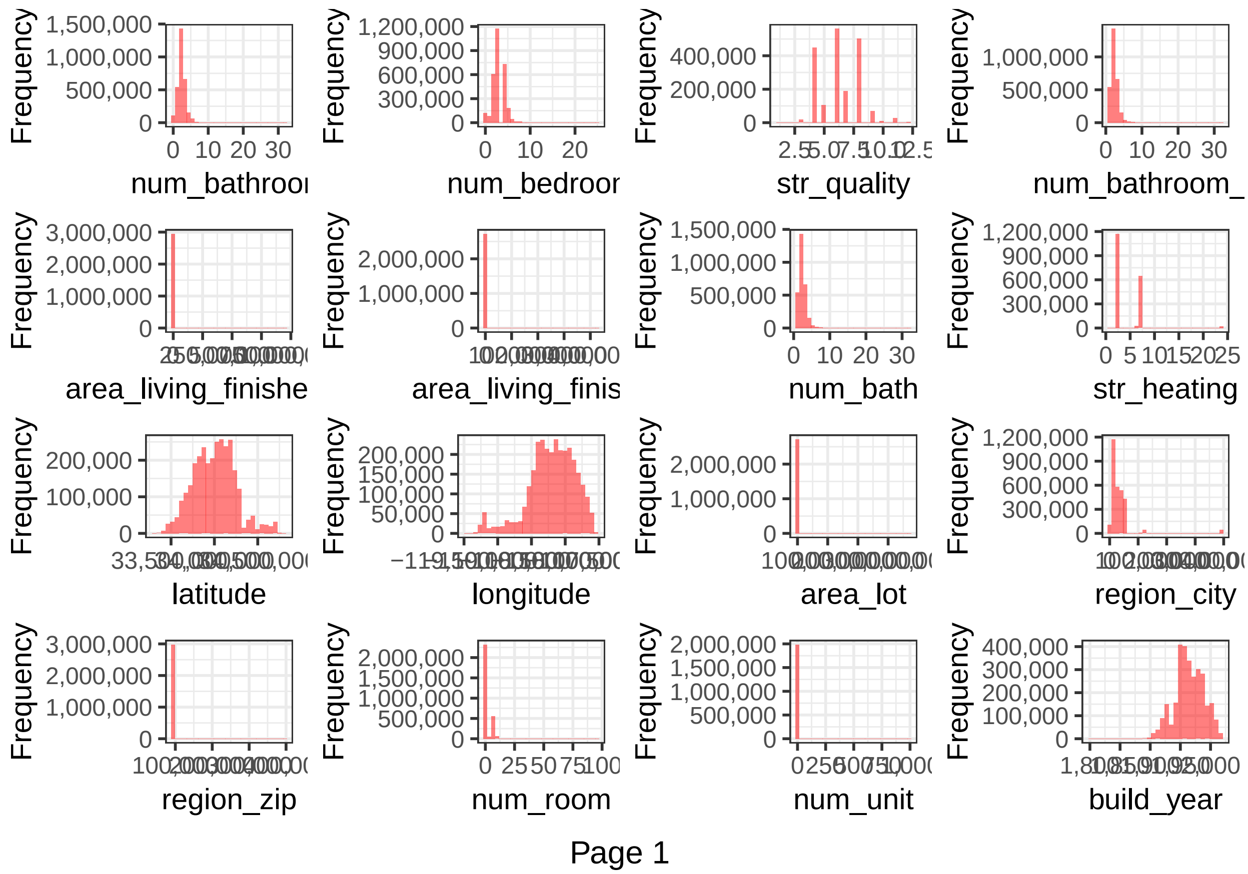
Based on that, let's remove `zoning_property` and `zoning_landuse_county`

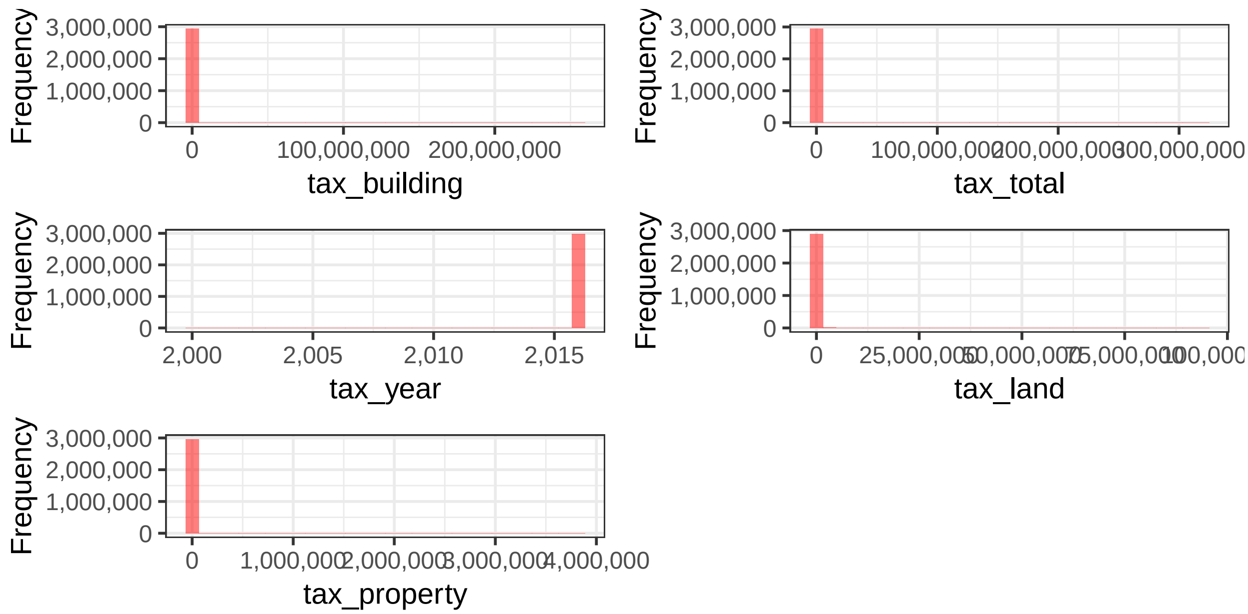
```
features_to_keep <- missing_data %>%
  filter(
    pct_missing <= .50,
    !feature %in% c("rawcensustractandblock", "fips",
                    "censustractandblock", "region_county",
                    "zoning_property", "zoning_landuse_county")
    ) %>%
  select(feature) %>%
  .$feature %>%
  as.character()

properties <- properties %>%
  select(features_to_keep)
```

3.2.2 Numeric Features

Lets look at the histograms of all the numeric features

```
properties %>%
  select(
    -id_parcel
  ) %>%
plot_histogram(ggtheme = theme_bw(),  fill = "red", alpha = 0.5)
```


Frequency
num_bathroom
num_bedroom
str_quality
num_bathroom_
area_living_finishe
area_living_finis
num_bath
str_heating
latitude
longitude
area_lot
region_city
region_zip
num_room
num_unit
build_year
Page 1



Page 2

Looking at the histograms a few things become obvious. There are huge outliers in many of the features and there are some features that are currently encoded as numeric but should not be treated as such. For example, `str_quality` is an ordinal scale 1 (best qaulity) to 12 (worst) but if we leave them as numeric they will be treated as ratio. `str_heating` is nominal so the order doesn't have meaning. Other that need to be changed are `region_city`, `region_zip`

Once we do this, we'll look again at the relationships between our numeric features.

```
properties <- properties %>%
  mutate(
    str_quality = factor(str_quality,
                         levels = min(str_quality, na.rm = TRUE):max(str_quality, na.rm = TRUE),
                         ordered = TRUE),
    str_heating = factor(str_heating,
                         levels = na.omit(unique(str_heating)),
                         ordered = FALSE),
    region_city = factor(region_city,
                         levels = na.omit(unique(region_city)),
                         ordered = FALSE),
    region_zip = factor(region_zip,
                         levels = na.omit(unique(region_zip)),
                         ordered = FALSE)
  )
```

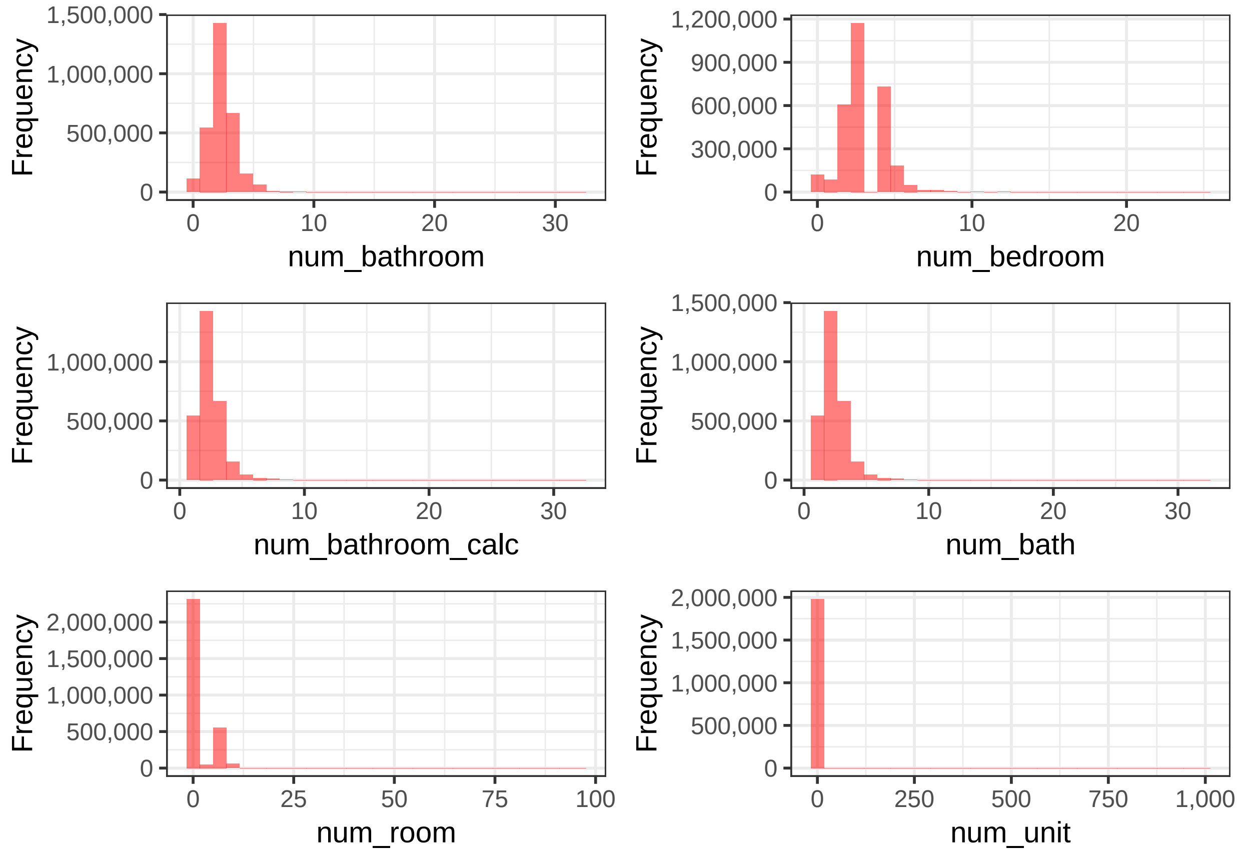


Figure 3.11: Distriubtions of 'num_*' Features

3.2.3 Numeric Outliers

Based on the histograms there looks to be lots of outliers in many of our numeric features. Two groups of features pop out, the `num_*` features and the `tax_*`features. Let's take a closer look.

```
properties %>%
  select(starts_with("num_")) %>%
plot_histogram(ggtheme = theme_bw(),  fill = "red", alpha = 0.5)
```

Looking at the `num_bathroom`, `num_bathroom_calc`, `num_bath` is pretty interesting. `num_bathroom` was one of the most complete features we had however, looking at the distributions, it seems strange that there would be so many houses with `0` bathrooms.

```
sum(properties$num_bathroom == 0, na.rm = TRUE)
```

```
## [1] 113470
```

```
sum(properties$num_bathroom_calc == 0, na.rm = TRUE)
```

```
## [1] 0
```

Now for comparing all 3

```
properties %>%
  group_by(
    num_bathroom_calc,
```

```
    num_bath,
    num_bathroom
    ) %>%
  count() %>%
  DT::datatable()
```

Show 10 entries Search:

	num_bathroom_calc	num_bath	num_bathroom	n
1	1	1	1	499324
2	1.5	1	1.5	45427
3	2	1	2	40
4	2	2	2	1219759
5	2.5	1	2.5	1
6	2.5	2	2.5	208577
7	3	2	3	559
8	3	3	3	632529
9	3.5	1	3.5	2
10	3.5	2	3.5	21

Showing 1 to 10 of 96 entries Previous 1 2 3 4 5 … 10 Next

If you sort by descending by `n` you'll see that one of the most frequent combinations is blank values of `num_bathroom_calc` and `num_bath` which are `NA` values and `0` for `num_bathroom`. Based on this I am interpreting that as either `0` being a coded value for `NA` or it just being wrong. Either way it looks like `num_bathroom_calc` is the one to keep out of all 3, since it has calculations of half-baths as well.

Applying the same logic to `num_room` and `num_bedroom` we can set all values equal to `0` to `NA`. One side effect of this is that the `num_room` feature is now almost 100% missing and not very useful anymore. So we will just remove it.

Quickly looking at `area_living_finished_calc` and `area_living_finished` reveals a similar `*_calc` being a corrected version of the feature. Becasue of this we will go ahead and remove `area_living_finished` as well

```
properties <- properties %>%
  select(
    -num_bath,
    -num_bathroom,
    -num_room,
    -area_living_finished
    ) %>%
  mutate(
    num_bedroom = ifelse(num_bedroom == 0, NA, num_bedroom)
    )
```

Now let's look at the tax related features

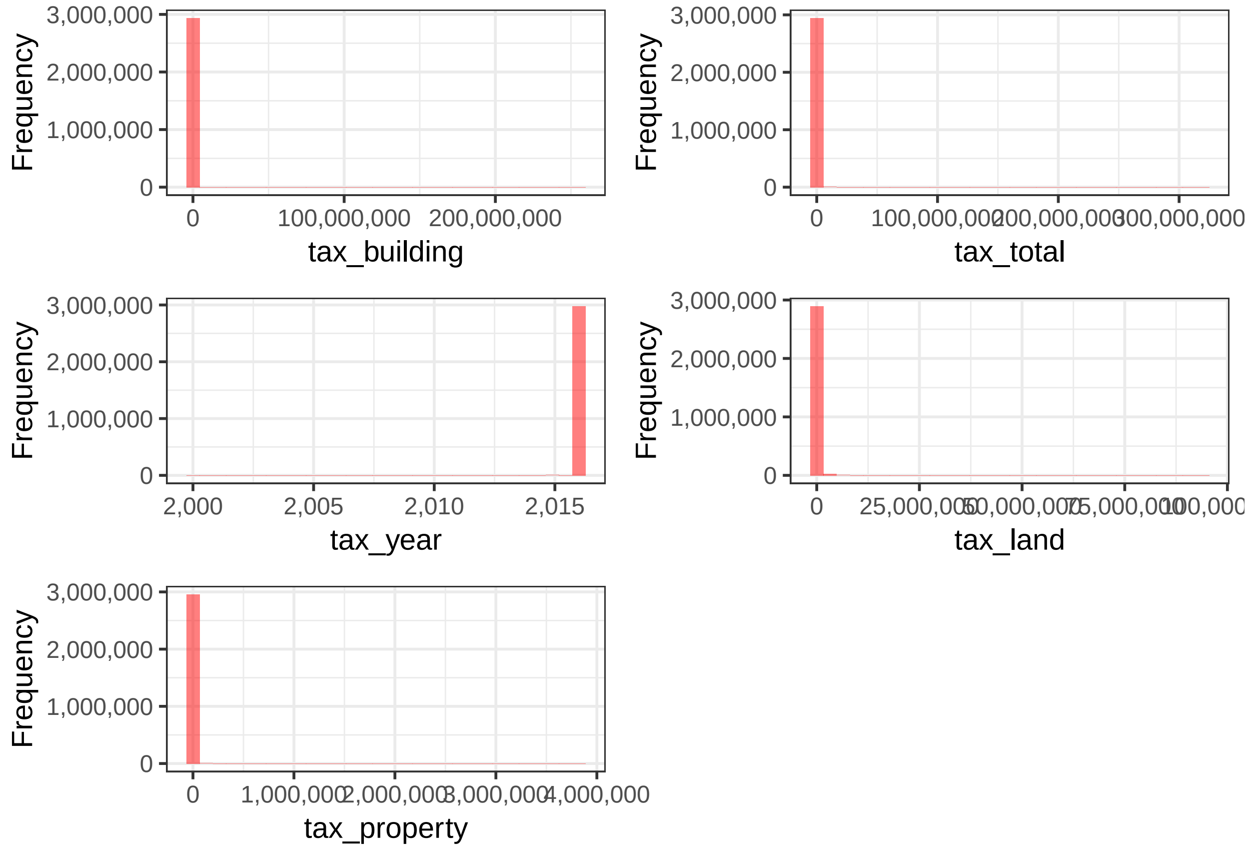


Figure 3.12: Distriubtions of 'tax_*' Features

```r
properties %>%
  select(starts_with("tax_")) %>%
plot_histogram(ggtheme = theme_bw(),  fill = "red", alpha = 0.5)
```

Lets look at the highest values for `tax_total` and see if something jumps out

```r
properties %>%
  mutate(tax_rank = rank(desc(tax_total))) %>%
  filter(tax_rank <= 20) %>%
  select(
    zoning_landuse,
    starts_with("area_"),
    starts_with("tax_")
    ) %>%
  arrange(tax_rank) %>%
  DT::datatable(
    extensions = 'FixedColumns',
    options = list(
    dom = 't',
    scrollX = TRUE,
    scrollCollapse = TRUE
    )
    )
```

	zoning_landuse	area_living_finished_calc	area_lot	tax_building	tax_total	tax_year	tax_land	ta
1	Residential General			228000000	319622473	2016	91622473	
2	Store/Office (Mixed Use)	520825		255321161	287098486	2016	31777325	
3	Store/Office (Mixed Use)	472363		222334475	271004605	2016	48670130	
4	Residential General			157500000	224345565	2016	66845565	
5	Residential General			74000000	164246219	2015	90246219	
6	Commercial/Office/Residential Mixed Used	522511		141417470	151895087	2016	10477617	
7	Residential General			55128075	149139154	2016	94011079	
8	Commercial/Office/Residential Mixed Used	319611		123371992	146257183	2016	22885191	
9	Residential General			136062640	137493421	2016	1430781	
10	Residential General			54300000	134574879	2015	80274879	

While the values are extremely large, they appear to look legitimate. We won't remove these, but it does indicate that we should perhaps apply some transformations to our tax features before we start applying our model.

Now a look at the relationships between our remaining numeric features

```
library(heatmaply)

properties %>%
  select(-id_parcel) %>%
  select_if(is.numeric) %>%
  cor(use = "pairwise.complete.obs") %>%
  heatmaply_cor()
```

3.2.4 Categorical Features

```
plot_bar(properties, ggtheme = theme_bw())
```

```
## 2 columns ignored with more than 50 categories.
## region_city: 187 categories
## region_zip: 404 categories
```

The distribution across categories are extremely non-uniform, especially `str_heating` and `zoning_landuse`. This imbalance could cause use some pain later one when trying to fit our model. One way we can avoid some of this pain is by collapsing some of the rare categories into an `other` category. The number of categories we collapse to is not a hard and fast decision, it can be based on number of observations, subject matter expertise, heterogentity of the response variable within categories, or some mix of all of these).

Let's look at what the distribution of `log_error` looks like across these categories.

Figure 3.13: Correlation of Numeric Features

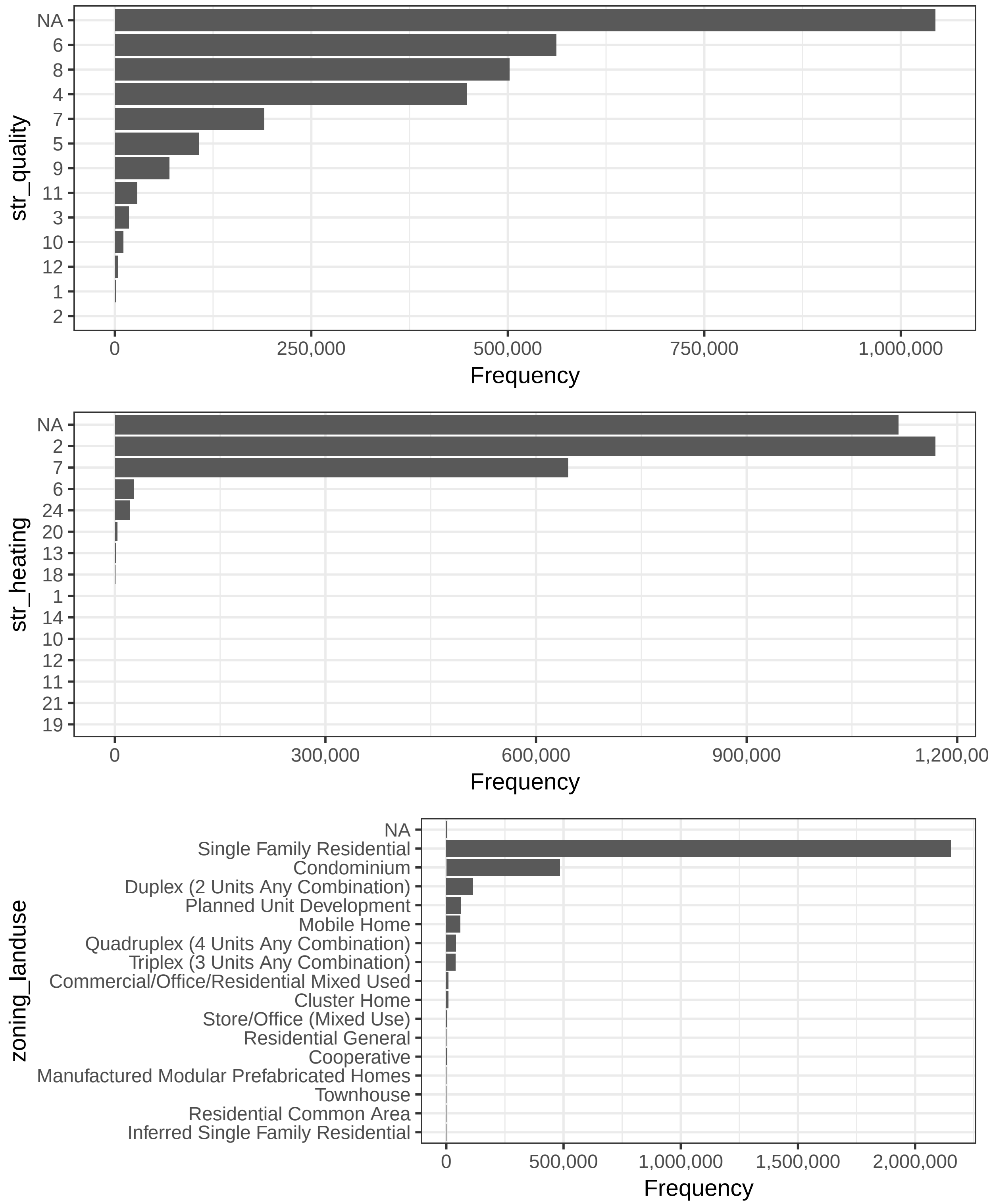


Figure 3.14: Distriubtions of All Categorical Features

```
library(ggridges)

properties %>%
  select(
    id_parcel,
    str_quality
  ) %>%
  right_join(trans, by = "id_parcel") %>%
  ggplot(aes(x = log_error, y = fct_reorder(str_quality, log_error), fill = factor(..quantile..))) +
  stat_density_ridges(
    geom = "density_ridges_gradient",
    calc_ecdf = TRUE,
    quantiles = c(0.05, 0.95)
    ) +
  scale_fill_manual(
    name = "Probability",
    values = c("#FF0000A0", "#A0A0A0A0", "#0000FFA0"),
    labels = c("(0, 0.05]", "(0.05, 0.95]", "(0.95, 1]")
    ) +
  xlim(c(-0.5, 0.5)) +
  theme_bw() +
  labs(
    y = "str_quality"
  )
```

```
library(ggridges)

properties %>%
  select(
    id_parcel,
    str_heating
  ) %>%
  right_join(trans, by = "id_parcel") %>%
  ggplot(aes(x = log_error, y = fct_reorder(str_heating, log_error), fill = factor(..quantile..))) +
  stat_density_ridges(
    geom = "density_ridges_gradient",
    calc_ecdf = TRUE,
    quantiles = c(0.05, 0.95)
    ) +
  scale_fill_manual(
    name = "Probability",
    values = c("#FF0000A0", "#A0A0A0A0", "#0000FFA0"),
    labels = c("(0, 0.05]", "(0.05, 0.95]", "(0.95, 1]")
    ) +
  xlim(c(-0.5, 0.5)) +
  theme_bw() +
  labs(
    y = "str_heating"
  )
```

```
library(ggridges)

properties %>%
  select(
```

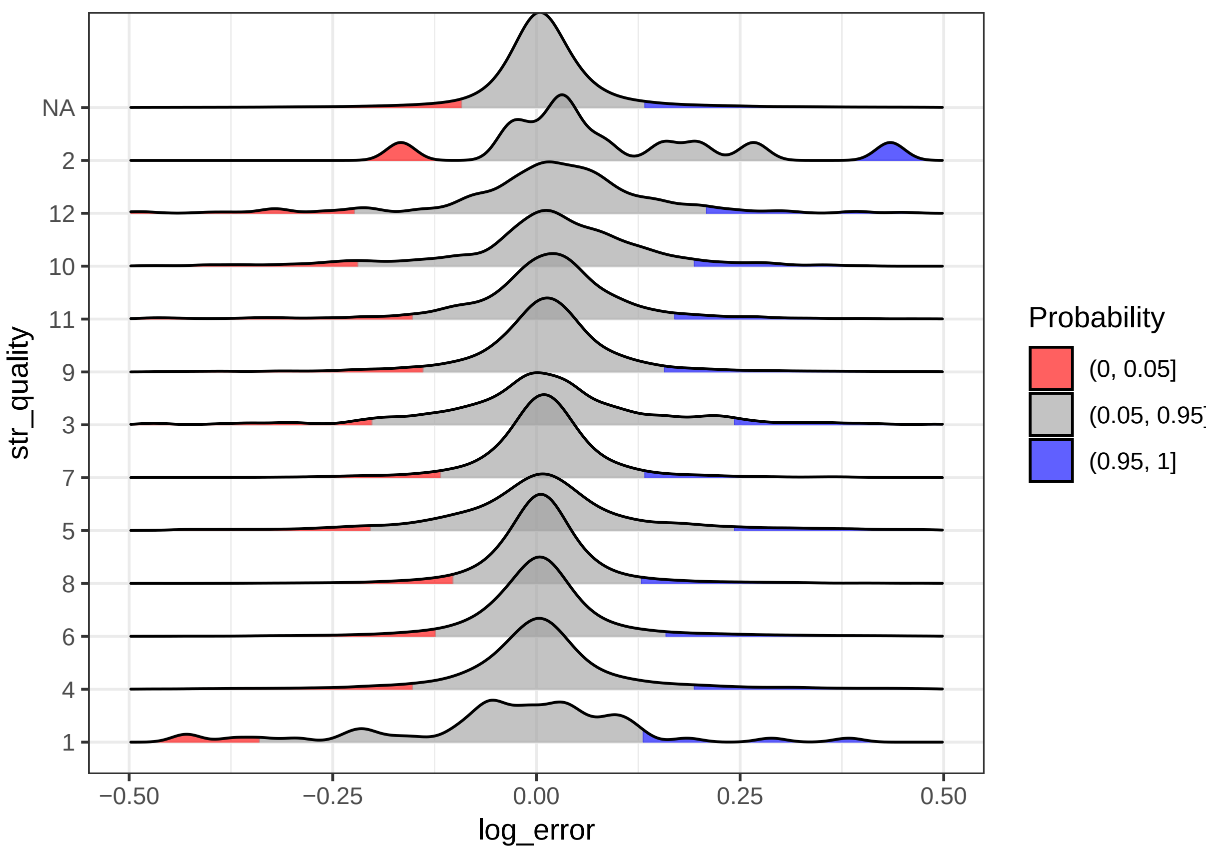


Figure 3.15: Distribution of Log Error Across Structure Quality Feature

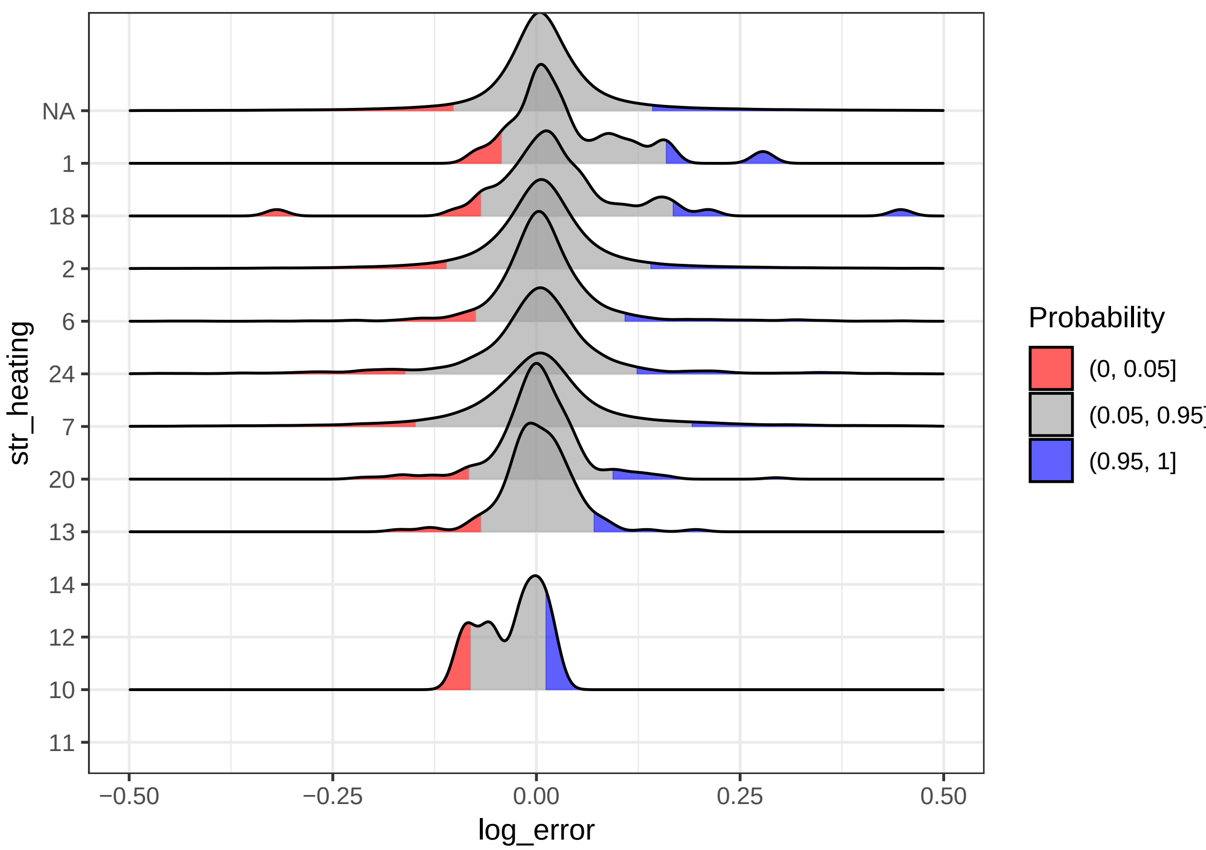


Figure 3.16: Distribution of Log Error Across Heating Type Feature

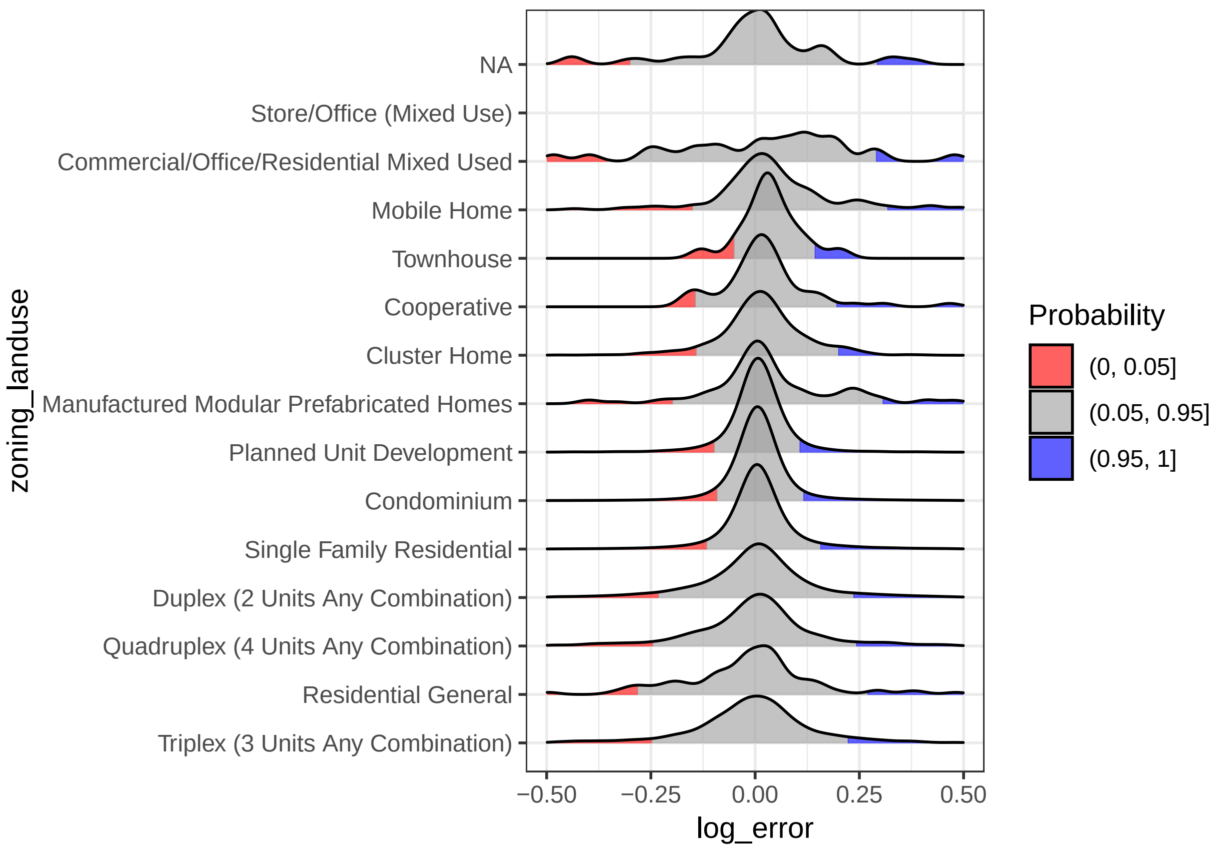


Figure 3.17: Distribution of Log Error Across Zoning Feature

```
    id_parcel,
    zoning_landuse
  ) %>%
  right_join(trans, by = "id_parcel") %>%
  ggplot(aes(x = log_error, y = fct_reorder(zoning_landuse, log_error), fill = factor(..quantile..))) +
  stat_density_ridges(
    geom = "density_ridges_gradient",
    calc_ecdf = TRUE,
    quantiles = c(0.05, 0.95)
    ) +
  scale_fill_manual(
    name = "Probability",
    values = c("#FF0000A0", "#A0A0A0A0", "#0000FFA0"),
    labels = c("(0, 0.05]", "(0.05, 0.95]", "(0.95, 1]")
    ) +
  xlim(c(-0.5, 0.5)) +
  theme_bw() +
  labs(
    y = "zoning_landuse"
  )
```

Since the distributions of `log_error` within each category seems well behaved, we will recode them based on number of observations

```
properties <- properties %>%
  mutate(
    str_heating = fct_lump(str_heating, n = 6),
    zoning_landuse = fct_lump(zoning_landuse, n = 8),
    str_heating = fct_recode(str_heating,
      "Central" = "2",
      "Floor/Wall" = "7",
      "Solar" = "20",
      "Forced Air" = "6",
      "Yes - Type Unknown" = "24",
      "None" = "13"
    )
  )
```

3.3 Exploring log_error A little More

Now let's join the properties and properties_geo tables to our trans table of tranactions and their log_error's and explore those

```
trans_prop <- read_feather("data/properties_geo_only.feather") %>%
  right_join(trans, by = "id_parcel") %>%
  left_join(properties, by = "id_parcel")
```

```
trans_prop %>%
  group_by(id_geo_bg_fips, id_geo_county_name) %>%
  summarise(
    n = n(),
    mean_abs_error = mean(abs_log_error)
    ) %>%
  ungroup() %>%
  mutate(
    trans_pert = cut(n, breaks = c(seq(0, 100, 10), 350))
    ) %>%
  ggplot(aes(x = trans_pert, y = mean_abs_error, colour = id_geo_county_name)) +
  geom_boxplot(outlier.size = 1.5, outlier.alpha = 1/3) +
  theme_bw() +
  labs(
    subtitle = "Block Group Average Mean Absolute Error",
    colour = NULL,
    x = "Number of Total Transactions per Block Group",
    y = "Mean Absolute Log Error"
  )
```

It looks like Los Angeles is largerly the only county that has information populated for str_qaulity

```
trans_prop %>%
  ggplot(aes(x = str_quality, y = log_error, colour = id_geo_county_name)) +
  geom_boxplot(outlier.size = 1.5, outlier.alpha = 1/3) +
  theme_bw() +
  labs(
    colour = NULL
  )
```

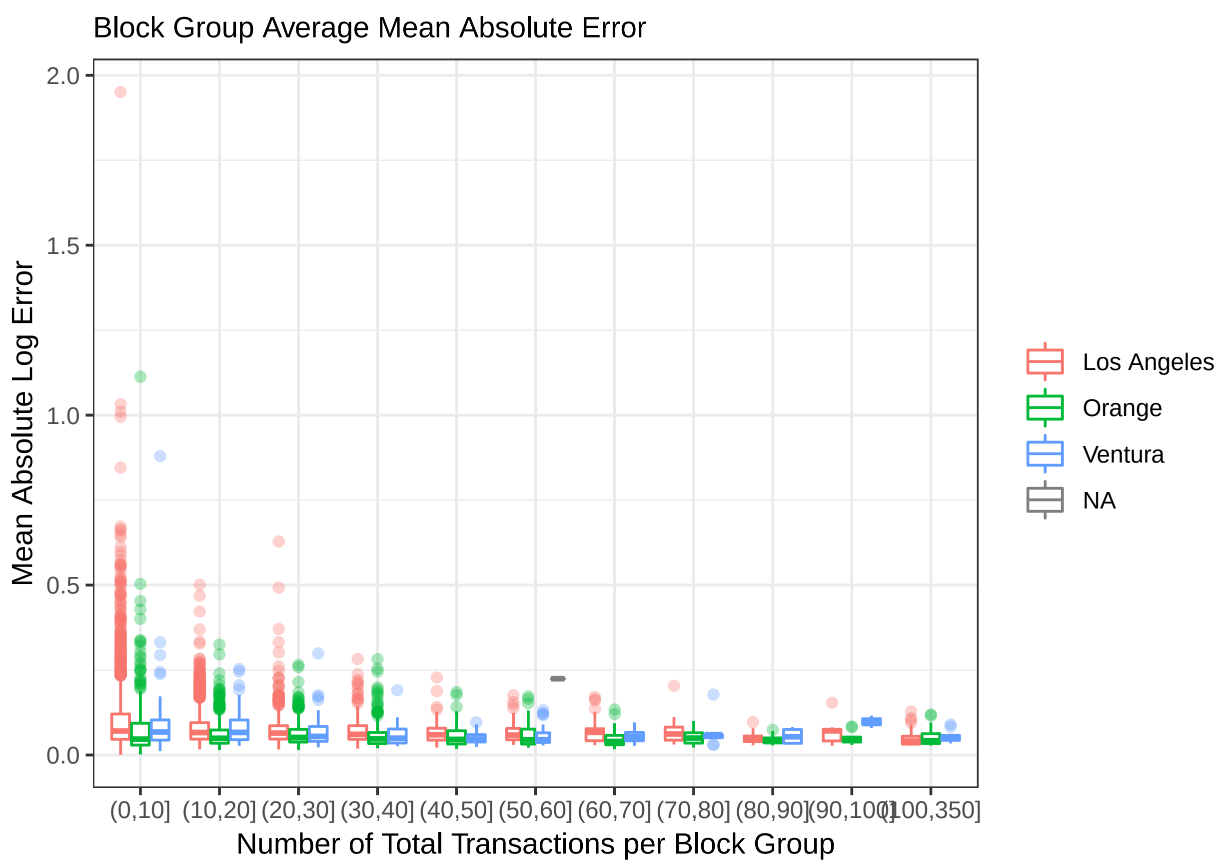


Figure 3.18: Outliers and Variability of Mean Absolute Error Dreceases When Neighborhood Sales Increase

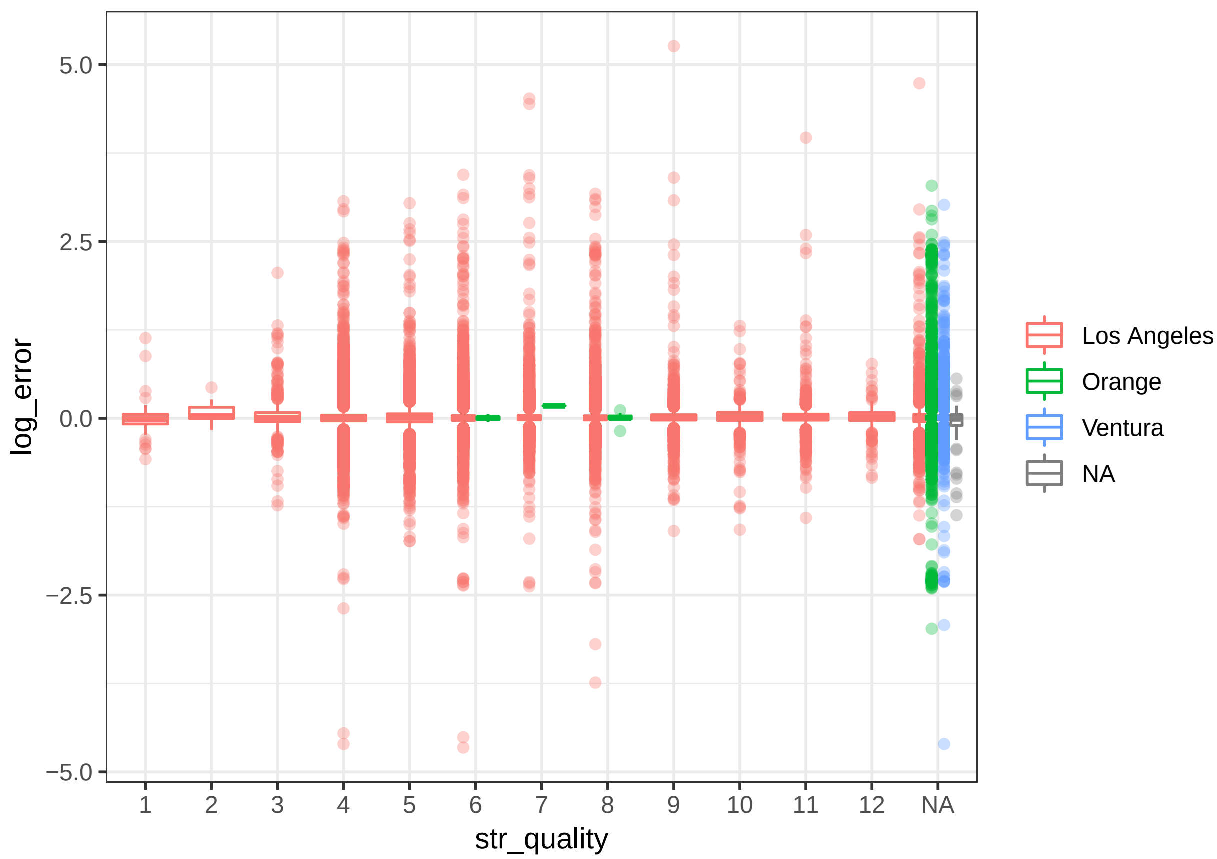


Figure 3.19: Log Error by Structure Quality

```
library(ggmap)

trans_prop_tmp <- trans_prop %>%
  filter(!is.na(id_geo_county_name)) %>%
  group_by(
    id_parcel,
    id_geo_county_name
    ) %>%
  mutate(
    log_error_parcel_avg = mean(log_error)
    ) %>%
  ungroup() %>%
  mutate(
    outlier = ifelse(log_error < quantile(log_error, probs  = .1) |
                     log_error > quantile(log_error, probs  = .9),
                     "Outlier", "Normal")
    )

error_map <- get_map(location = "Los Angeles, CA",
                     color="bw",
                     crop = FALSE,
                     zoom = 9)

ggmap(error_map) +
  stat_density2d(
    data = trans_prop_tmp,
    aes(x = lon, y = lat,
        fill = ..level..,
        alpha = ..level..),
    geom = "polygon",
    size = 0.001,
    bins = 100
    ) +
  scale_fill_viridis_c() +
  scale_alpha(range = c(0.05, 0.95), guide = FALSE) +
  facet_wrap(~outlier)
```

Now lets look at the spatiotemporal distribution of `log_error` outliers

```
trans_prop %>%
  filter(
    !is.na(lat),
    (
      log_error <= quantile(log_error, probs  = .1) |
      log_error >= quantile(log_error, probs  = .9)
      )
    ) %>%
  mutate(
    lon = round(lon/0.5, digits = 1) * 0.5,
    lat = round(lat/0.5, digits = 1) * 0.5
  ) %>%
  group_by(lon, lat, month_year) %>%
  summarise(
    n = n()
```

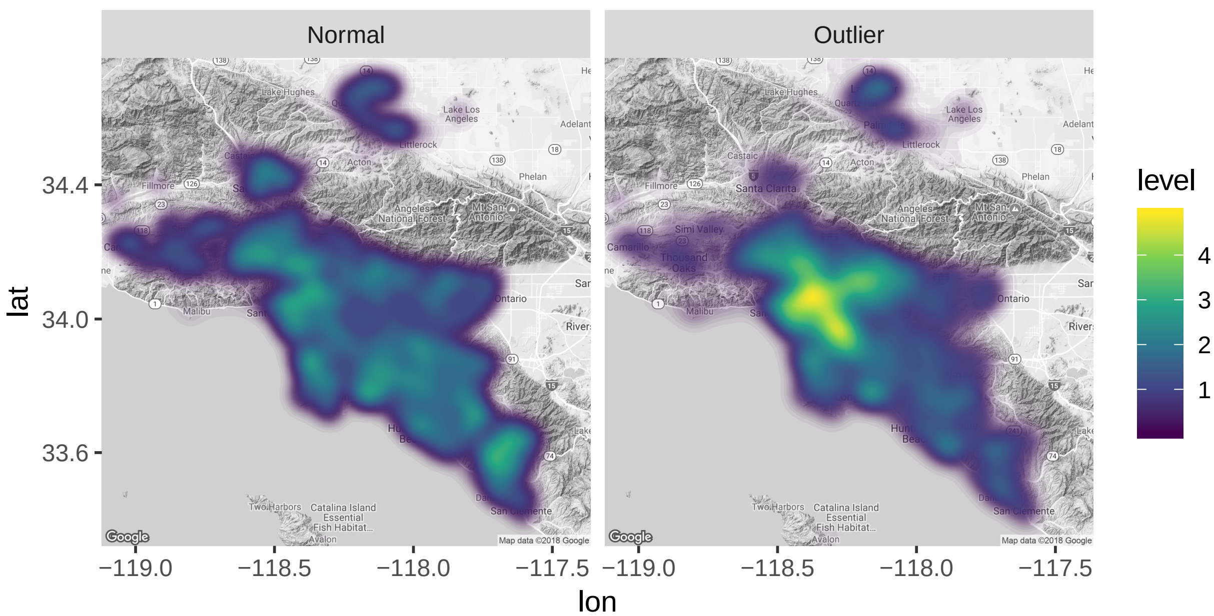


Figure 3.20: Spatial Distribution of Log Error Outliers

```
  ) %>%
ggplot(aes(lon, lat)) +
  geom_raster(aes(fill = n)) +
  scale_fill_viridis_c() +
  facet_wrap(~month_year, ncol = 3) +
  coord_quickmap() +
  theme_dark() +
  labs(
    subtitle = "Downtown Los Angeles looks to be consistently bad",
    fill = "Count"
  ) +
  theme(
    axis.text = element_text(size = 5)
    )
```

At first glance there looks to be a strong spatial and temporal correlation to `log_error`. Let's look more into the spatial correlation.

Moran's I and its variant Local Moran's I, provide a useful measure of the amount of spatial autocorrelation in a variable.

```
library(spatstat)
library(spdep)

d <- trans_prop %>%
  filter(
    !is.na(lat),
    (
      log_error <= quantile(log_error, probs  = .1) |
      log_error >= quantile(log_error, probs  = .9)
      )
    ) %>%
  mutate(
    lon = round(lon/0.1, digits = 1) * 0.1,
    lat = round(lat/0.1, digits = 1) * 0.1
  ) %>%
  group_by(lon, lat) %>%
  summarise(
    n = n()
  )

coordinates(d) <- ~lon + lat
w <- knn2nb(knearneigh(d, k = 10, longlat = TRUE))
moran.test(d$n, nb2listw(w))
```

```
##
##  Moran I test under randomisation
##
## data:  d$n
## weights: nb2listw(w)
##
## Moran I statistic standard deviate = 71.112, p-value < 2.2e-16
## alternative hypothesis: greater
## sample estimates:
## Moran I statistic       Expectation          Variance
```

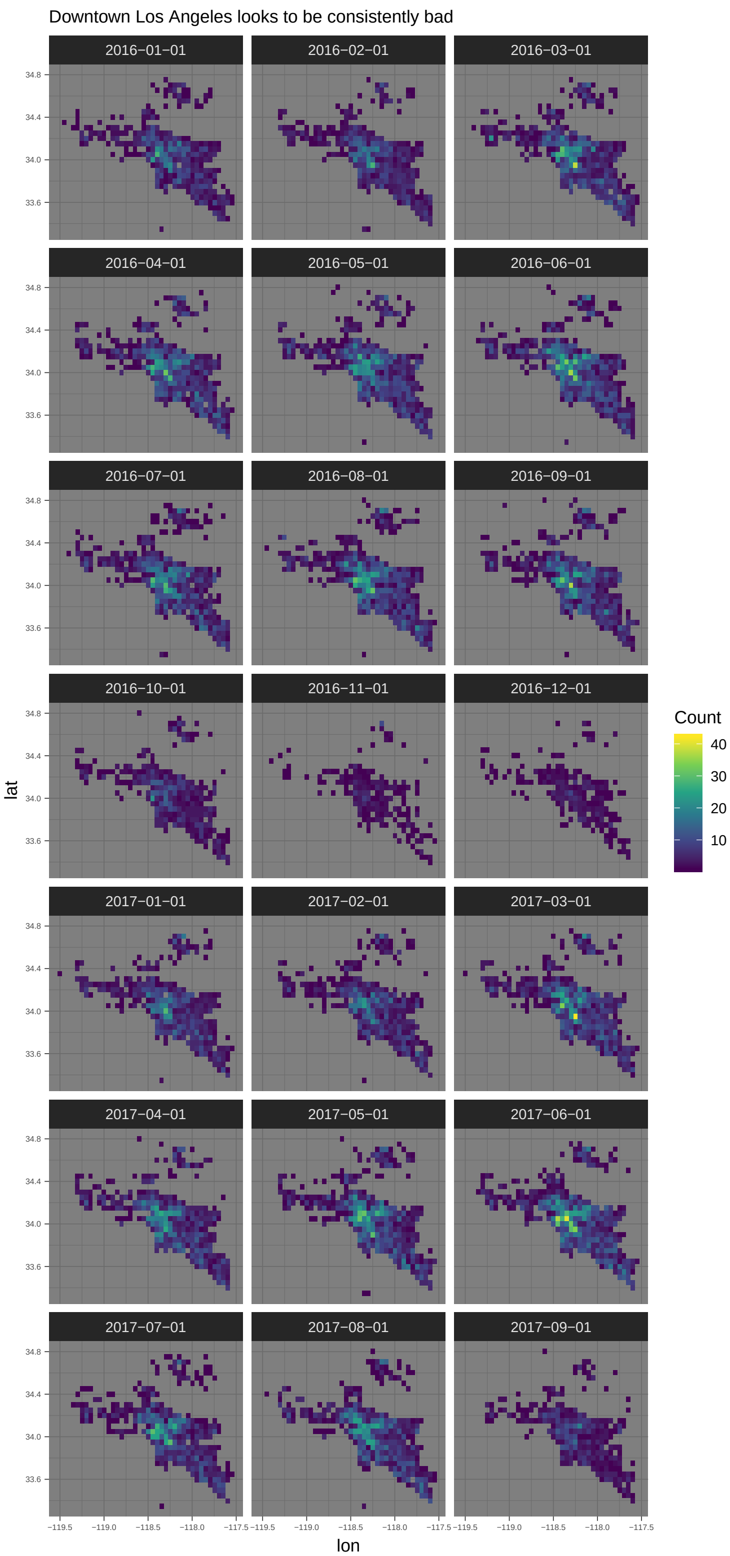


Figure 3.21: SpatioTemporal Distribution of Log Error Outliers

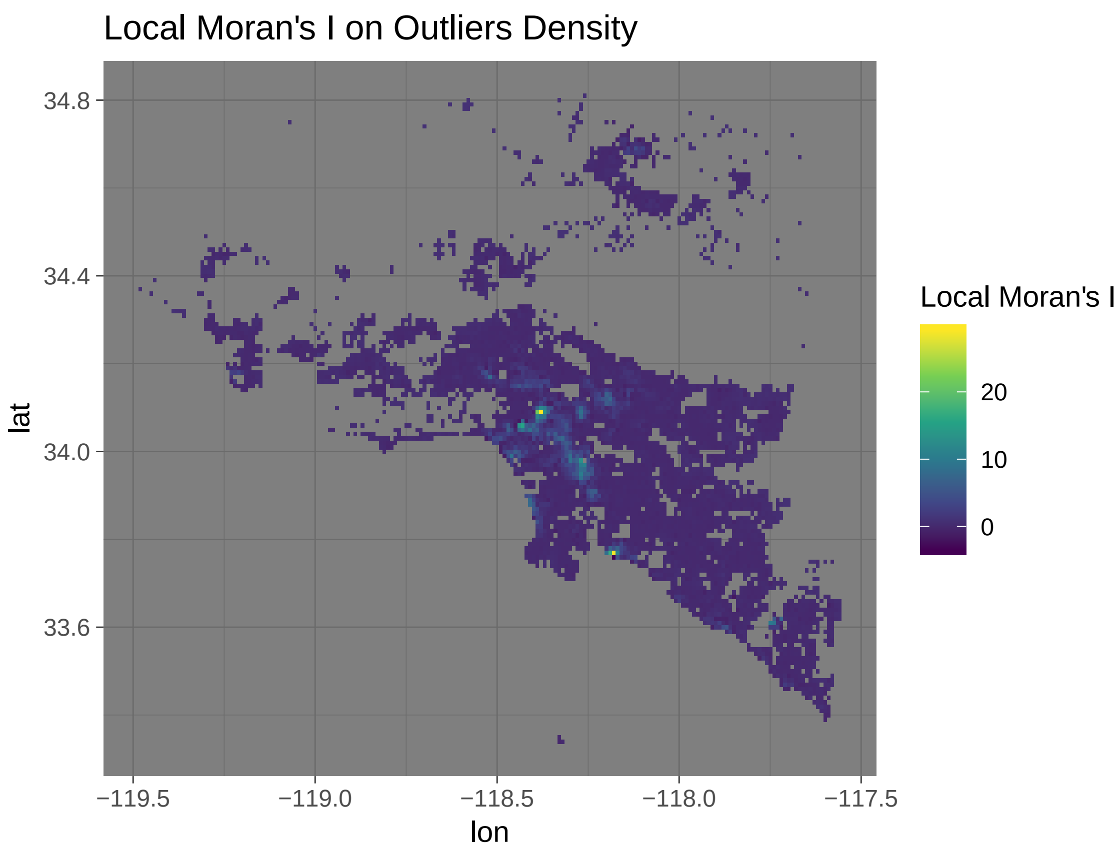


Figure 3.22: Spatial Autocorrelation of Log Error Outliers

```
##      4.308120e-01     -1.952362e-04      3.673489e-05
```

```
local_moran <- as.data.frame(localmoran(d$n, nb2listw(w)))

d %>%
  as.data.frame() %>%
  cbind(local_moran) %>%
  ggplot(aes(lon, lat)) +
  geom_raster(aes(fill = Ii)) +
  scale_fill_viridis_c() +
  coord_quickmap() +
  theme_dark() +
  labs(
    title = "Local Moran's I on Outliers Density",
    fill = "Local Moran's I"
  )
```

Let’s save our pared down `properties` table and then get into feature engineering

```
write_feather(properties, "data/properties_17_filtered.feather")
```

Chapter 4

Feature Engineering

After we have done an intital EDA of our data we can start doing some feature engineering, this is where we can create new features such as interaction variables, apply transformations such as centering and scaling, choice how we want to encode our categorical features, and also bring in new external information.

Just as in 3, throughout this section we will progressively updating our `properties` data to include new and transformed features that we are going to continue with into the next stages.

4.1 Creating New Features

> "Everything is related to everything else, but near things are more related than distant things."
>
> - Waldo Tobler

This "First Law of Geography" is something we can take advantage of for creating new features based on our existing ones. In this section we will create based on both data from the Kaggle competition and also examples of external sources as well

4.1.1 Internal Features

Since we have the neighborhood, as defined by `id_geo_bg_fips`, that each parcel is apart of, we can use this to create neighborhood average features.

There are many ways one could define neighborhood for the purposes of using near by parcels, knn for example. Perhaps a more rigourous and certainly more computationly intensive approach would be to estimate the radius at which the spatial autocorrelation of `log_error` is no longer statistically significant using something such as a bootstrapped spline correlogram such as the function `spline.correlog()` provided by the `ncf` package

```
ncf::spline.correlog(x = lon, y = lat, z = log_error)
```

For now we will stick with defining our neighborhood by the census block group (`id_geo_bg_fips`) that each parcel is apart of

4.1.1.1 Neigborhood Average `properties` Features

```
bg_avg_features <- properties %>%
  group_by(id_geo_bg_fips) %>%
  select(-id_parcel) %>%
  select_if(is.numeric) %>%
  summarise_all(mean, na.rm = TRUE) %>%
  filter(
    !is.na(id_geo_bg_fips)
    )

names(bg_avg_features) <- paste0("bg_avg_", names(bg_avg_features))
names(bg_avg_features)[1] <- "id_geo_bg_fips"

# update the properties table
properties <- properties %>%
  left_join(bg_avg_features, by = "id_geo_bg_fips")
```

4.1.1.2 Rolling Local Average `log_error`

There is a strong spatial and temporal autocorrelation to our response variable `log_error`. To take advantage of this, let's create a few new features based on the rolling average of the local `log_error` values.

Because these features will have values for every day from `min(trans$date)` to `max(trans$date)` we won't join them to our data yet.

```
library(tibbletime)

trans_prop <- properties_geo %>%
  right_join(trans, by = "id_parcel") %>%
  select(
    id_parcel,
    id_geo_bg_fips,
    id_geo_tract_fips,
    date,
    log_error
  )

# create rolling functions ------------------------------------------------

rolling_sum_7 <- rollify(~sum(.x, na.rm = TRUE), window = 7)
rolling_sum_28 <- rollify(~sum(.x, na.rm = TRUE), window = 28)

# by block group ----------------------------------------------------------

roll_bg <- create_series(min(trans_prop$date) ~ max(trans_prop$date),
                         'daily', class = "Date") %>%
  tidyr::expand(
    date,
    id_geo_bg_fips = unique(trans_prop$id_geo_bg_fips)
  ) %>%
  full_join(trans_prop) %>%
  group_by(id_geo_bg_fips, date) %>%
  summarise(
    sum_log_error = sum(log_error, na.rm = TRUE),
```

```
    sales_total = sum(!is.na(log_error))
    ) %>%
  ungroup() %>%
  group_by(id_geo_bg_fips) %>%
  mutate(
    sum_log_error_7days = rolling_sum_7(sum_log_error),
    sum_log_error_28days  =  rolling_sum_28(sum_log_error),
    roll_bg_trans_total_7days = rolling_sum_7(sales_total),
    roll_bg_trans_total_28days  =  rolling_sum_28(sales_total),
    roll_bg_avg_log_error_7days = sum_log_error_7days / roll_bg_trans_total_7days,
    roll_bg_avg_log_error_28days = sum_log_error_28days / roll_bg_trans_total_28days,
    date = date + lubridate::days(1) # to not include the current day in avg
  ) %>%
  select(
    id_geo_bg_fips,
    date,
    roll_bg_trans_total_7days,
    roll_bg_trans_total_28days,
    roll_bg_avg_log_error_7days,
    roll_bg_avg_log_error_28days
  ) %>%
  mutate(
    roll_bg_trans_total_7days = ifelse(is.na(roll_bg_trans_total_7days),
                                       0, roll_bg_trans_total_7days),
    roll_bg_trans_total_28days = ifelse(is.na(roll_bg_trans_total_28days),
                                        0, roll_bg_trans_total_28days),
    roll_bg_avg_log_error_7days = ifelse(is.nan(roll_bg_avg_log_error_7days),
                                         0, roll_bg_avg_log_error_7days),
    roll_bg_avg_log_error_28days = ifelse(is.nan(roll_bg_avg_log_error_28days),
                                          0, roll_bg_avg_log_error_28days),
    roll_bg_avg_log_error_7days = as.numeric(forecast::na.interp(roll_bg_avg_log_error_7days)),
    roll_bg_avg_log_error_28days = as.numeric(forecast::na.interp(roll_bg_avg_log_error_28days))
  )

# by tract ----------------------------------------------------------------

roll_tract <- create_series(min(trans_prop$date) ~ max(trans_prop$date),
                         'daily', class = "Date") %>%
  tidyr::expand(
    date,
    id_geo_tract_fips = unique(trans_prop$id_geo_tract_fips)
  ) %>%
  full_join(trans_prop) %>%
  group_by(id_geo_tract_fips, date) %>%
  summarise(
    sum_log_error = sum(log_error, na.rm = TRUE),
    sales_total = sum(!is.na(log_error))
  ) %>%
  ungroup() %>%
  group_by(id_geo_tract_fips) %>%
  mutate(
    sum_log_error_7days = rolling_sum_7(sum_log_error),
    sum_log_error_28days  =  rolling_sum_28(sum_log_error),
```

```
    roll_tract_trans_total_7days = rolling_sum_7(sales_total),
    roll_tract_trans_total_28days  =  rolling_sum_28(sales_total),
    roll_tract_avg_log_error_7days = sum_log_error_7days / roll_tract_trans_total_7days,
    roll_tract_avg_log_error_28days = sum_log_error_28days / roll_tract_trans_total_28days,
    date = date + lubridate::days(1) # to not include the current day in avg
  ) %>%
  select(
    id_geo_tract_fips,
    date,
    roll_tract_trans_total_7days,
    roll_tract_trans_total_28days,
    roll_tract_avg_log_error_7days,
    roll_tract_avg_log_error_28days
  ) %>%
  mutate(
    roll_tract_trans_total_7days = ifelse(is.na(roll_tract_trans_total_7days),
                                          0, roll_tract_trans_total_7days),
    roll_tract_trans_total_28days = ifelse(is.na(roll_tract_trans_total_28days),
                                           0, roll_tract_trans_total_28days),
    roll_tract_avg_log_error_7days = ifelse(is.nan(roll_tract_avg_log_error_7days),
                                            0, roll_tract_avg_log_error_7days),
    roll_tract_avg_log_error_28days = ifelse(is.nan(roll_tract_avg_log_error_28days),
                                             0, roll_tract_avg_log_error_28days),
    roll_tract_avg_log_error_7days = as.numeric(forecast::na.interp(roll_tract_avg_log_error_7days)),
    roll_tract_avg_log_error_28days = as.numeric(forecast::na.interp(roll_tract_avg_log_error_28days))
  )

prop_geo_ids <- properties_geo %>%
  select(
    id_parcel,
    id_geo_bg_fips,
    id_geo_tract_fips
  )

write_feather(roll_bg, "data/external-features/roll_features_blockgroup.feather")
write_feather(roll_tract, "data/external-features/roll_features_tract.feather")
```

4.1.2 External Features

Breaking from the rules of the actual Kaggle competition, we're going to add in some external features as an example of bringing in other information

4.1.2.1 American Community Survey

The American Community Survey is a great source of demographic and household data. As an example of using this data let's bring in a few features related to our area of interest.

In our example here, we are completely ignoring the margin or error for each feature, given more time investigating the information contained in these fields is most likely worth your while.

There are literally thousands you can explore in the ACS. For our example, we are going to stop a little short of that and only add the following

- Fill this in
    - sub group 1
    - sub group 2
- fill this in

```
library(tidycensus)

api_key <- Sys.getenv("CENSUS_API_KEY")
census_api_key(api_key)

acs_var_list <- load_variables(2016, "acs5", cache = TRUE)

acs_bg_vars <- c("B25034_001E", "B25034_002E", "B25034_003E",
                 "B25034_004E", "B25034_005E", "B25034_006E",
                 "B25034_007E", "B25034_008E", "B25034_009E",
                 "B25034_010E", "B25034_011E", "B25076_001E",
                 "B25077_001E", "B25078_001E", "B25056_001E",
                 "B25002_001E", "B25002_003E", "B25001_001E")

acs_bg_home_value <- acs_var_list %>%
  filter(grepl("B25075_", x = name))

acs_bg_home_value_vars <- acs_bg_home_value$name

acs_bg_vars <- c(acs_bg_vars, acs_bg_home_value_vars)

acs_bg_data <- get_acs(
  geography = "block group",
  variables = acs_bg_vars,
  state = "CA",
  county = c("Los Angeles", "Orange", "Ventura"),
  output = "wide",
  geometry = FALSE,
  keep_geo_vars = TRUE
)

acs_bg_data1 <- acs_bg_data %>%
  select(
    id_geo_bg_fips = GEOID,
    acs_str_yr_total = B25034_001E,
    acs_str_yr_2014_later = B25034_002E,
    acs_str_yr_2010_2013 = B25034_003E,
    acs_str_yr_2000_2009 = B25034_004E,
    acs_str_yr_1990_1999 = B25034_005E,
    acs_str_yr_1980_1989 = B25034_006E,
    acs_str_yr_1970_1979 = B25034_007E,
    acs_str_yr_1960_1969 = B25034_008E,
    acs_str_yr_1950_1959 = B25034_009E,
    acs_str_yr_1940_1949 = B25034_010E,
    acs_str_yr_1939_earlier = B25034_011E,
    acs_home_value_lwr = B25076_001E,
    acs_home_value_med = B25077_001E,
    acs_home_value_upr = B25078_001E,
    acs_num_of_renters_total = B25056_001E,
```

```
    acs_num_of_house_units = B25001_001E,
    acs_occ_status_total = B25002_001E,
    acs_occ_status_vacant = B25002_003E,
    acs_home_value_cnt_total = B25075_001E,
    acs_home_value_cnt_less_10k = B25075_002E,
    acs_home_value_cnt_10k_15k = B25075_003E,
    acs_home_value_cnt_15k_20k = B25075_004E,
    acs_home_value_cnt_20k_25k = B25075_005E,
    acs_home_value_cnt_25k_30k = B25075_006E,
    acs_home_value_cnt_30k_35k = B25075_007E,
    acs_home_value_cnt_35k_40k = B25075_008E,
    acs_home_value_cnt_40k_50k = B25075_009E,
    acs_home_value_cnt_50k_60k = B25075_010E,
    acs_home_value_cnt_60k_70k = B25075_011E,
    acs_home_value_cnt_70k_80k = B25075_012E,
    acs_home_value_cnt_80k_90k = B25075_013E,
    acs_home_value_cnt_90k_100k = B25075_014E,
    acs_home_value_cnt_100k_125k = B25075_015E,
    acs_home_value_cnt_125k_150k = B25075_016E,
    acs_home_value_cnt_150k_175k = B25075_017E,
    acs_home_value_cnt_175k_200k = B25075_018E,
    acs_home_value_cnt_200k_250k = B25075_019E,
    acs_home_value_cnt_250k_300k = B25075_020E,
    acs_home_value_cnt_300k_400k = B25075_021E,
    acs_home_value_cnt_400k_500k = B25075_022E,
    acs_home_value_cnt_500k_750k = B25075_023E,
    acs_home_value_cnt_750k_1000k = B25075_024E,
    acs_home_value_cnt_1000k_1500k = B25075_025E,
    acs_home_value_cnt_1500k_2000k = B25075_026E,
    acs_home_value_cnt_2000k_more = B25075_027E
  ) %>%
  mutate_at(
    vars(starts_with("acs_home_value_cnt")), function(x) round(x / .$acs_home_value_cnt_total, digits =
    ) %>%
  mutate_at(
    vars(starts_with("acs_str_yr")), function(x) round(x / .$acs_str_yr_total, digits = 5)
  ) %>%
  mutate(
    acs_per_renters = round(acs_num_of_renters_total / acs_num_of_house_units, digits = 5),
    acs_per_vacant = round(acs_occ_status_vacant / acs_occ_status_total, digits = 5)
  ) %>%
  select(
    -acs_occ_status_total,
    -acs_home_value_cnt_total,
    -acs_str_yr_total
    )

acs_features <- properties_geo %>%
  select(
    id_parcel,
    id_geo_bg_fips
    ) %>%
  left_join(acs_bg_data1, by = "id_geo_bg_fips") %>%
```

```
  select(-id_geo_bg_fips)

properties <- properties %>%
  left_join(acs_features, by = "id_parcel")
```

4.1.2.2 Economic Indicators

The value of a home is not only influenced by itself and its neighbors, but also larger economic trends. To help account for this in our model we are going to add in the following economic indicators

- 30-Year Fixed Rate Mortgage Average in the United States
- S&P/Case-Shiller CA-Los Angeles Home Price Index
- Unemployment Rate in Los Angeles County, CA

```
library(alfred)

# 30-Year Fixed Rate Mortgage Average in the United States (weekly)
mort30 <- get_fred_series("MORTGAGE30US") %>%
  mutate(
    date_month = floor_date(date, unit = "month"),
    date_week = floor_date(date, unit = "week")
    )

# S&P/Case-Shiller CA-Los Angeles Home Price Index (monthly)
spcs <- get_fred_series("LXXRNSA")

# Unemployment Rate in Los Angeles County, CA (monthly)
unemployment <- get_fred_series("CALOSA7URN")

econ_features <- create_series(min(mort30$date) ~ max(mort30$date),
                               'daily', class = "Date") %>%
  mutate(date_week = floor_date(date, unit = "week")) %>%
  left_join(mort30, by = c("date_week" = "date_week")) %>%
  left_join(spcs, by = c("date_month" = "date")) %>%
  left_join(unemployment, by = c("date_month" = "date")) %>%
  select(
    date = date.x,
    econ_mort_30 = MORTGAGE30US,
    econ_case_shiller = LXXRNSA,
    econ_unemployment = CALOSA7URN
  ) %>%
  filter(
    date >= date("2015-12-01"),
    date <= date("2018-01-01")
    )

write_feather(econ_features, "data/external-features/econ_features.feather")
```

Ok, so a check in on where we are. We currently have 5 data frames of interest, our old friends `properties` which now contains new features from our `bg_avg_features` neighborhood data and the `acs_features` which contain a few indicators from the American Community Survey and `trans` which contains our response variable `log_error` as well as the transaction date and a few date based features.

The other 3 data frames we have are seperated from the `properties` and `trans` data currently because they contain features that have different values for each day and depending on what our transaction dates are for the partiuclar set of observations we will use filter those values down and join them at the time of training.

4.2 Handling Missing Data

There are many ways to handle missing data from simple mean or median imputation to more complex methods such as knn or multiple imputation or even constructing other predictive models for predicting missing features. We will not explore that topic in depth here and use a somewhat simple approach that takes advantage of the spatial relationships of our data.

We will use median (or modal for nominal features) imputation but instead of doing global median values for all observations, we are going to break our observations into subspaces based on `zoning_landuse`, `area_lot`, and increasingly larger neigborhood windows. The reasoning behind this choice is that the values for many of the other features can vary widely across these categories. For example it wouldn't make sense to include the `tax_building` values for mobile homes if we are imputing the `tax_building` value of a commercial office building. The is true for `area_lot` the tax burden on a large commercial office will be larger then the one of a smaller office.

So we will look at increaingly larger neighborhoods of `zoning_landuse` and (a discretized version of) `area_lot` combinations, if there are any none `NA` observations of that combination in the missing observations block group, fill its values with the block group median (mode), if there are no non-missing observations with that combination in that block group, then look at the tract level, if there are none at the tract look at the county level, and finally if there are no non-missing observations in the County that the parcel belongs to, an unlikely event, then use the "global" values to impute.

To do this we first need to impute the values for `zoning_landuse` and `area_lot`. For this we will just use the increasing neighborhood search for `zoning_landuse` and then use the increasing neighborhood search broken down by `zoning_landuse` to fill in `area_lot`

For some reason their are no built in functions for calculating the mode. Make a simple helper function to do so

```
# simple helper function to find the mode
fct_mode <- function(f) {

  f_no_na <- na.omit(f)
  fct_tab <- table(f_no_na)

  # if everything NA return NA
  if (length(fct_tab) == 0) return(NA)

  modal_fct <- names(fct_tab)[which(fct_tab == max(fct_tab))]
  modal_fct <- modal_fct[1] # in case of ties, go with first one

  modal_fct
}
```

Some parcels have no information at all including geographic ids. Randomly assign them a `id_geo_bg_fips` value based block group frequency

```
parcels_no_info <- properties %>%
  filter(
    is.na(id_geo_bg_fips)
    ) %>%
  select(id_parcel)
```

```
bg_probs <- table(properties$id_geo_bg_fips) %>%
  as.data.frame()

bg_assignments <- sample(bg_probs$Var1,
                         size = nrow(parcels_no_info),
                         replace = TRUE,
                         prob = bg_probs$Freq)

bg_assignments <- as.character(bg_assignments)

parcels_no_info_row_id <- properties$id_parcel %in% parcels_no_info$id_parcel

properties[parcels_no_info_row_id , "id_geo_bg_fips"] <- bg_assignments

# fill in the missing tract and county based on bg
properties <- properties %>%
  group_by(id_geo_bg_fips) %>%
  mutate(
    id_geo_tract_fips = fct_mode(id_geo_tract_fips)
    ) %>%
  ungroup() %>%
  group_by(id_geo_tract_fips) %>%
  mutate(
    id_geo_county_fips = fct_mode(id_geo_county_fips)
  ) %>%
  ungroup()
```

Now that all observations have at least geo id values we can impute `zoning_landuse` using the increasing neighborhood search

```
properties <- properties %>%
  group_by(id_geo_bg_fips) %>%
  mutate(
    zoning_landuse = replace_na(zoning_landuse, fct_mode(zoning_landuse))
  ) %>%
  ungroup() %>%
  group_by(id_geo_tract_fips) %>%
  mutate(
    zoning_landuse = replace_na(zoning_landuse, fct_mode(zoning_landuse))
  ) %>%
  ungroup() %>%
  group_by(id_geo_county_fips) %>%
  mutate(
    zoning_landuse = replace_na(zoning_landuse, fct_mode(zoning_landuse))
  ) %>%
  ungroup()
```

Now for `area_lot`

```
properties <- properties %>%
  group_by(
    id_geo_bg_fips,
    zoning_landuse
    ) %>%
  mutate(
```

```
    area_lot = replace_na(area_lot, median(area_lot, na.rm = TRUE))
  ) %>%
  ungroup() %>%
  group_by(
    id_geo_tract_fips,
    zoning_landuse
    ) %>%
  mutate(
    area_lot = replace_na(area_lot, median(area_lot, na.rm = TRUE))
  ) %>%
  ungroup() %>%
  group_by(
    id_geo_county_fips,
    zoning_landuse
    ) %>%
  mutate(
    area_lot = replace_na(area_lot, median(area_lot, na.rm = TRUE))
  ) %>%
  ungroup()
```

OK, now for the rest of them. Becasue we used `median()` to fill in `area_lot` the `quantile()` are not unique, so add a little bit of noise `area_lot_jitter` and base the breaks on those.

```
# impute all other features based on neighborhood, zoning_landuse, and area_lot
properties <- properties %>%
  mutate(
    area_lot_jitter = area_lot + runif(n = n(), min = -1, max = 1),
    area_lot_quantile = cut(area_lot_jitter,
                            breaks = quantile(area_lot_jitter, probs = seq(0, 1, 0.1), na.rm = TRUE))
    ) %>%
  group_by(
    id_geo_bg_fips,
    zoning_landuse,
    area_lot_quantile
  ) %>%
  mutate_if(
    is.numeric, .funs = function(x) replace_na(x, median(x, na.rm = TRUE))
    ) %>%
  mutate_if(
    is.factor, .funs = function(x) replace_na(x, fct_mode(x))
  ) %>%
  ungroup() %>%
  group_by(
    id_geo_tract_fips,
    zoning_landuse,
    area_lot_quantile
  ) %>%
  mutate_if(
    is.numeric, .funs = function(x) replace_na(x, median(x, na.rm = TRUE))
  ) %>%
  mutate_if(
    is.factor, .funs = function(x) replace_na(x, fct_mode(x))
  ) %>%
  ungroup() %>%
```

```
  group_by(
    id_geo_county_fips,
    zoning_landuse,
    area_lot_quantile
  ) %>%
  mutate_if(
    is.numeric, .funs = function(x) replace_na(x, median(x, na.rm = TRUE))
  ) %>%
  mutate_if(
    is.factor, .funs = function(x) replace_na(x, fct_mode(x))
  ) %>%
  ungroup() %>%
  group_by(
    zoning_landuse,
    area_lot_quantile
  ) %>%
  mutate_if(
    is.numeric, .funs = function(x) replace_na(x, median(x, na.rm = TRUE))
  ) %>%
  mutate_if(
    is.factor, .funs = function(x) replace_na(x, fct_mode(x))
  ) %>%
  ungroup()
```

At this point we now have all the original features we are going to use and have filled in all missing values. The next step is to transform our features, create interaction features, and then move unto feature selection.

4.3 Feature Transformation

Combine all of our data and remove a handful of features that we aren't going to use

```
# have to remove id_geo after joins because of time features
d <- trans %>%
  left_join(properties, by = "id_parcel") %>%
  left_join(econ_features, by = "date") %>%
  left_join(roll_bg, by = c("id_geo_bg_fips", "date")) %>%
  left_join(roll_tract, by = c("id_geo_tract_fips", "date")) %>%
  select(
    -id_parcel,
    -abs_log_error,
    -week,
    -region_city,
    -region_zip,
    -area_lot_jitter,
    -area_lot_quantile,
    -lat, # we have other lat/lon features
    -lon,
    -starts_with("id_geo")
  ) %>%
  mutate(
    date = as.numeric(date),
    year = factor(year, levels = sort(unique(year)), ordered = TRUE),
    month_year = factor(
```

```
      as.character(month_year),
      levels = as.character(unique(sort(month_year))),
      ordered = TRUE),
    str_quality = factor(str_quality, levels = 12:1, ordered = TRUE)
  )
```

Here we are going to use the fantastic package `recipes` to handle all of our feature transformations

```
library(recipes)

rec <- recipe(d) %>%
  add_role(log_error, new_role = 'outcome') %>%
  add_role(-log_error, new_role = 'predictor') %>%
  step_meanimpute(starts_with("roll_")) %>%
  step_zv(all_numeric()) %>%
  step_BoxCox(
    starts_with("num_"),
    starts_with("area_"),
    starts_with("tax_"),
    starts_with("bg_avg_num_"),
    starts_with("bg_avg_area_"),
    starts_with("bg_avg_tax_")
  ) %>%
  step_dummy(all_nominal(), one_hot = TRUE) %>%
  step_interact(~starts_with("tax_"):starts_with("area_")) %>%
  step_center(all_numeric(), -all_outcomes()) %>%
  step_scale(all_numeric(), -all_outcomes()) %>%
  step_pca(starts_with("acs_home_value_cnt"), prefix = "acs_home_value_cnt_PC") %>%
  step_zv(all_numeric())

rec_prepped <- prep(rec, training = d)
```

Chapter 5

Feature Selection

Now that we have our data in something Based on the data frame we created in `d` and the transformation recipe we made we are going to do some initital analysis on which features we want to keep or drop. the `xgboost` package provides a function, `xgb.importance()` that gives a summary of how important each feature was in a model estimated by `xgb.train()`

To have a little more robustness in our selection, we will use v fold cross validation to get mutliple samples from `d` and investigate the importance of the features across all samples.

```
library(broom)
library(purrr)
library(xgboost)

importance_results <- function(splits) {

  x <- bake(rec_prepped, newdata = analysis(splits))
  y <- x$log_error

  d <- model.matrix(log_error ~., data = x)
  d <- xgb.DMatrix(d, label = y)

  mdl <- xgb.train(data = d, label = y, nrounds = 1000, nthread = 4)
  print(summary(mdl))

  mdl_importance <- as.data.frame(xgb.importance(model = mdl))

  mdl_importance
}
```

```
library(rsample)

resamples <- vfold_cv(d, v = 10, repeats = 5)

resamples$results <- map(resamples$splits,
                         importance_results)


importance_df <- bind_rows(resamples$results)

feature_avg <- importance_df %>%
```

```
  group_by(Feature) %>%
  summarise(
    mean = mean(Gain),
    sd = sd(Gain),
    n = n()
    )
```

```
feature_avg %>%
  ggplot(aes(x = forcats::fct_reorder(Feature, mean), y = mean)) +
  geom_hline(aes(yintercept = 0.001), colour = "red", size = 1, alpha = 0.5) +
  geom_point(size = 1) +
  geom_errorbar(aes(ymin = mean - sd * 2, ymax = mean + sd * 2)) +
  coord_flip() +
  theme_bw() +
  theme(
    axis.text=element_text(size = 6)
    ) +
  labs(
    x = "Feature",
    y = "Mean Gain"
  )
```

To reduce the complexity and computation time our of modeling, we are going to remove the feature that consistantly did not provide much value by cutting off the number of features we'll use at a mean gain at `0.001` (red line).

```
features_to_use <- feature_avg %>%
  filter(mean >= 0.001) %>%
  .$Feature
```

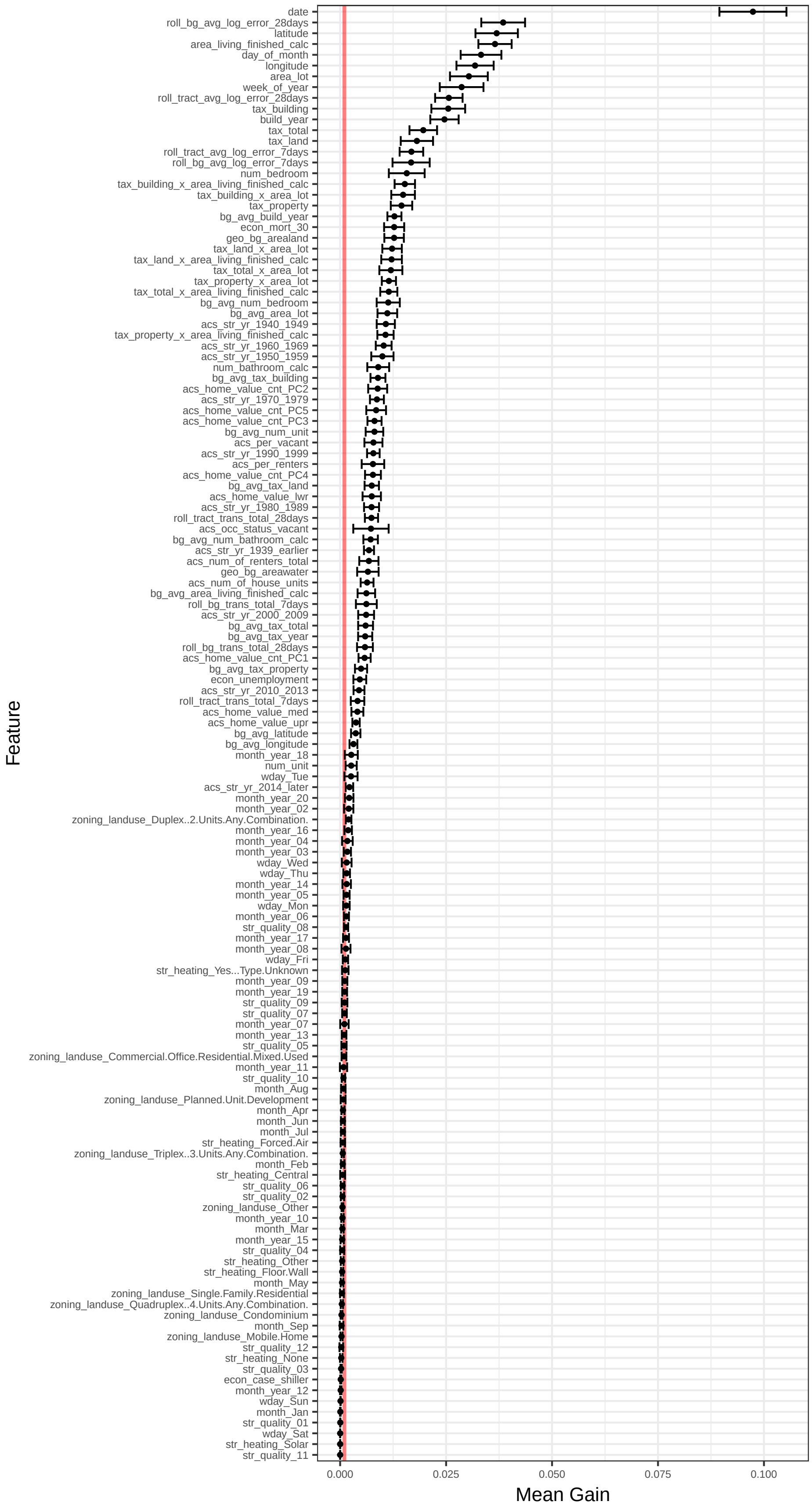


Figure 5.1: Mean Feature Importance Based on Cross Validation Using Basic XGBoost Model

Chapter 6

Modeling

For our first pass a submission, we are going to use the XGBoost model. This model has seen much success in Kaggle competition due to its flexiblity and range of modeling tasks it can be applied to.

6.0.1 XGBoost

In gerenal XGBoost works like this…

As a reminder our recipe for our transformations are stored in `rec`

```
rec
```

```
## Data Recipe
##
## Inputs:
##
##       role #variables
##    outcome          1
##  predictor         98
##
## Operations:
##
## Mean Imputation for starts_with("roll_")
## Zero variance filter on all_numeric()
## Box-Cox transformation on 6 items
## Dummy variables from all_nominal()
## Interactions with starts_with("tax_"):starts_with("area_")
## Centering for all_numeric(), -all_outcomes()
## Scaling for all_numeric(), -all_outcomes()
## PCA extraction with starts_with("acs_home_value_cnt")
## Zero variance filter on all_numeric()
```

and the data we have is in `d`

```
# some kind of d summary
```

6.0.2 creating our scoring function

```
# create scoring function -------------------------------------------------

xgboost_regress_score <- function(train_df, target_var, params, eval_df, ...) {

    X_train <- train_df %>%
      select(features_to_use) %>%
      as.matrix()

    y_train <- train_df[[target_var]]
    xgb_train_data <- xgb.DMatrix(X_train, label = y_train)

    X_eval <- eval_df %>%
      select(features_to_use) %>%
      as.matrix()

    y_eval <- eval_df[[target_var]]

    xgb_eval_data <- xgb.DMatrix(X_eval, label = y_eval)

    model <- xgb.train(params = params,
                       data = xgb_train_data,
                       watchlist = list(train = xgb_train_data, eval = xgb_eval_data),
                       objective = 'reg:linear',
                       verbose = FALSE,
                       nthread = 20,
                       ...)

    preds <- predict(model, xgb_eval_data)

    list(mae = MAE(preds, y_eval))

  }
```

Make the parameter set that we are going to search over

```
library(ParamHelpers)

# make parameter set -----------------------------------------------------

xgboost_random_params <-
  makeParamSet(
    makeIntegerParam('max_depth', lower = 1, upper = 15),
    makeNumericParam('eta', lower = 0.01, upper = 0.1),
    makeNumericParam('gamma', lower = 0, upper = 5),
    makeIntegerParam('min_child_weight', lower = 1, upper = 100),
    makeNumericParam('subsample', lower = 0.25, upper = 0.9),
    makeNumericParam('colsample_bytree', lower = 0.25, upper = 0.9)
  )
```

set up our cv

```
# create cv resampling ---------------------------------------------------
resamples <- vfold_cv(d, v = 5)
```

Use Ranger model for a progressive zoom into parameter space for tuning

```r
library(MLmetrics)
library(xgboost)
library(tidytune)
# perform surrogate search over parameters --------------------------------

n <- c(10, 5, 3, 2)
n_candidates <- c(0, 10, 100, 1000)

search_results <-
  surrogate_search(
    resamples = resamples,
    recipe = rec,
    param_set = xgboost_random_params,
    n = n,
    scoring_func = xgboost_regress_score,
    nrounds = 1000,
    early_stopping_rounds = 20,
    eval_metric = 'mae',
    input = NULL,
    surrogate_target = 'mae',
    n_candidates = n_candidates,
    top_n = 5
  )
```

```r
search_summary <-
  search_results %>%
  group_by_at(getParamIds(xgboost_random_params)) %>%
  summarise(mae = mean(mae)) %>%
  arrange(mae)
```

	max_depth	eta	gamma	min_child_weight	subsample	colsample_bytree	
1	5	0.0753141567087732	2.00997113599442	100	0.882115679827984	0.528489873954095	0.0685
2	5	0.0254947707173415	2.04678243258968	38	0.842303937417455	0.55284611225361	0.0685
3	5	0.0119649214693345	2.05316238454543	26	0.866748715774156	0.553988792502787	0.0685
4	5	0.0227248513093218	1.79130512056872	87	0.890362443809863	0.436710486758966	0.068
5	8	0.0179595448542386	2.05562060466036	93	0.891160581179429	0.521792834519874	0.068
6	6	0.0184877650788985	2.47866926947609	64	0.83313264483586	0.812140957009979	0.0686
7	6	0.0597224851348437	2.54486419609748	85	0.858863256021868	0.88247439750703	0.0686
8	5	0.0427106887870468	2.11250258260407	94	0.745799078559503	0.638513505784795	0.0686
9	5	0.0147431414970197	2.22801433177665	19	0.798432543617673	0.84205840973882	0.0686
10	3	0.0400677399151027	1.93769115372561	65	0.837546187068801	0.774923810071778	0.0686

```r
search_results %>%
  group_by_at(
    c("surrogate_run",
      "surrogate_iteration",
      "param_id",
       getParamIds(xgboost_random_params)
      )
    ) %>%
  summarise(mae = mean(mae)) %>%
  ungroup() %>%
  mutate(surrogate_run = factor(surrogate_run)) %>%
  arrange(
    surrogate_run,
    surrogate_iteration
  ) %>%
  mutate(
    iteration = row_number()
  ) %>%
ggplot(aes(x = iteration, y = mae)) +
    geom_smooth(alpha = 0.2, size = 0.8, colour = "grey") +
  geom_point(aes(col = surrogate_run)) +
  theme_bw() +
  labs(
    y = "MAE",
    x = "iteration",
    col = "Surrogate Run"
  )
```

6.1 Make Predictions with Tuned Parameters

```r
tuned_params <- search_summary %>%
  ungroup() %>%
  filter(mae == min(mae)) %>%
  select(getParamIds(xgboost_random_params)) %>%
  as.list()

tuned_params
```

```
## $max_depth
## [1] 5
##
## $eta
## [1] 0.07531416
##
## $gamma
## [1] 2.009971
##
## $min_child_weight
## [1] 100
##
## $subsample
```

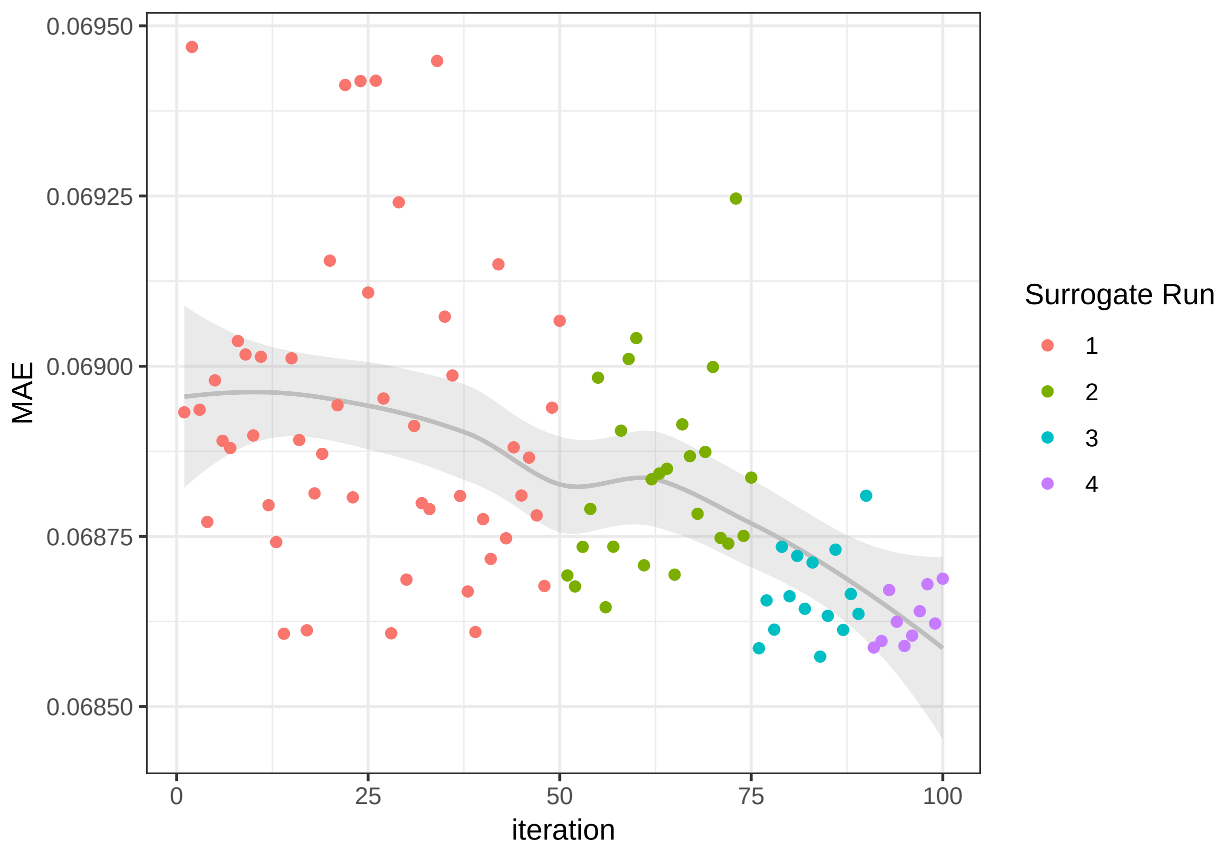


Figure 6.1: Mean Absolute Error Progressively Decreasing with Each Surrogate Run

```
## [1] 0.8821157
##
## $colsample_bytree
## [1] 0.5284899
```

Train the model using the tuned parameters

```r
d_prepped <- prep(rec)

train_df <- bake(d_prepped, newdata = d)

x_train <- train_df %>%
  select(features_to_use) %>%
  as.matrix()

y_train <- train_df$log_error

xgb_train_data <- xgb.DMatrix(x_train, label = y_train)

model <- xgb.train(params = tuned_params,
                   data = xgb_train_data,
                   objective = 'reg:linear',
                   verbose = FALSE,
                   nthread = 4,
                   nrounds = 1000)
```

Now we need to make our predictions. We'll make a helper function `predict_date()` to do this.

```r
predict_date <- function(parcel_id, predict_date, mdl) {

d_predict_ids <- properties %>%
  filter(id_parcel %in% parcel_id) %>%
  crossing(date = predict_date)

d_predict <- d_predict_ids %>%
  mutate(
    year = year(date),
    month_year = make_date(year(date), month(date)),
    month = month(date, label = TRUE),
    week = floor_date(date, unit = "week"),
    week_of_year = week(date),
    week_since_start = (min(date) %--% date %/% dweeks()) + 1,
    wday = wday(date, label = TRUE),
    day_of_month = day(date)
  ) %>%
  left_join(econ_features, by = "date") %>%
  left_join(roll_bg, by = c("id_geo_bg_fips", "date")) %>%
  left_join(roll_tract, by = c("id_geo_tract_fips", "date")) %>%
  select(
    -id_parcel,
    -week,
    -region_city,
    -region_zip,
    -area_lot_jitter,
    -area_lot_quantile,
    -lat, # we have other lat/lon features
```

```
    -lon,
    -starts_with("id_geo")
  ) %>%
  mutate(
    date = as.numeric(date),
    year = factor(year, levels = sort(unique(year)), ordered = TRUE),
    month_year = factor(
      as.character(month_year),
      levels = as.character(unique(sort(month_year))),
      ordered = TRUE),
    str_quality = factor(str_quality, levels = 12:1, ordered = TRUE)
  )

eval_df <- bake(d_prepped, newdata = d_predict)

x_eval <- eval_df %>%
  select(features_to_use) %>%
  as.matrix()

preds <- predict(mdl, x_eval)

properties_predict <- d_predict_ids %>%
  select(
    id_parcel,
    date
  ) %>%
  mutate(
    pred = preds
    ) %>%
  spread(date, pred)

names(properties_predict) <- c("ParcelId", "201610","201611","201612", "201710","201711","201712")

properties_predict

}
```

The submission requires a prediction for Oct-Dec 2016 and Oct-Dec 2017. This means that the prediction is for any day in that month. For our example first submission, we are going to just set the date to the first wednesday in each month. This is completely arbitrary. Another approach would be to make predictions for every day in each month and submit the mean prediction for each month. We'll save this for later work.

```
# first wednesday in each month
predict_dates <- date(c("2016-10-06","2016-11-02","2016-12-07", "2017-10-04","2017-11-01","2017-12-06"))

# split parcels to chunck our predictions
id_parcels <- properties$id_parcel
id_parcel_splits <- split(id_parcels, ceiling(seq_along(id_parcels) / 5000))


predict_list <- lapply(id_parcel_splits, function(i) {

  pred_df <- predict_date(
    parcel_id = i,
```

```
    predict_date = predict_dates,
    mdl = model
    )
  })

# they only evaluate to 4 decimcals so round to save space
# Convert ParcelId to integer to prevent Sci Notation that causes
# issues with submission
predict_df <- bind_rows(predict_list) %>%
  mutate_at(vars(`201610`:`201712`), round, digits = 4) %>%
  mutate(ParcelId = as.integer(ParcelId)) %>%
  as.data.frame()

write_csv(predict_df, "data/submit01.csv")
```

This model produced a MAE of `0.0651839` on the public leaderboard, which is honestly not that great, but it is a starting point that we can now start iterarting from. My first thought is perhaps we are overfitting on our training data, we could start exploring how differnt tuning affect actual submissions and not just cross validation based on resampling. Based on this we can tract how our performance changes and start narrow down what what tuning are most performant for this task.

Another approach would be to only use

Chapter 7

Summary

7.1 Key Findings

7.2 Weak Points of Analysis

imputation method need to test more models could include more geo related features like interstate density should do a base line comparision