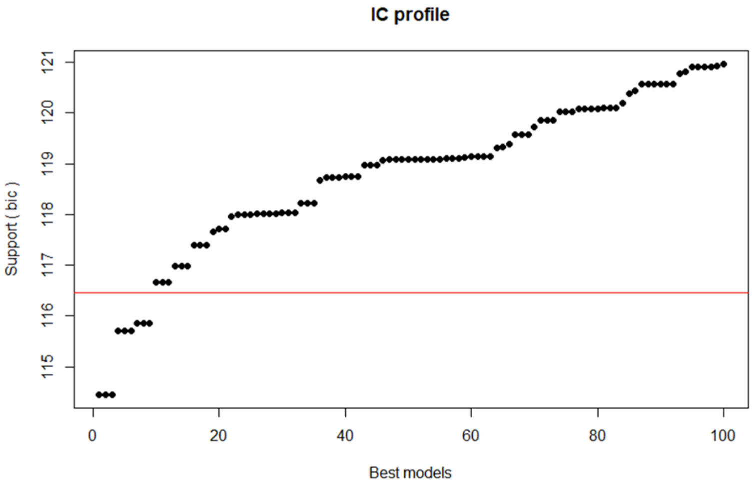
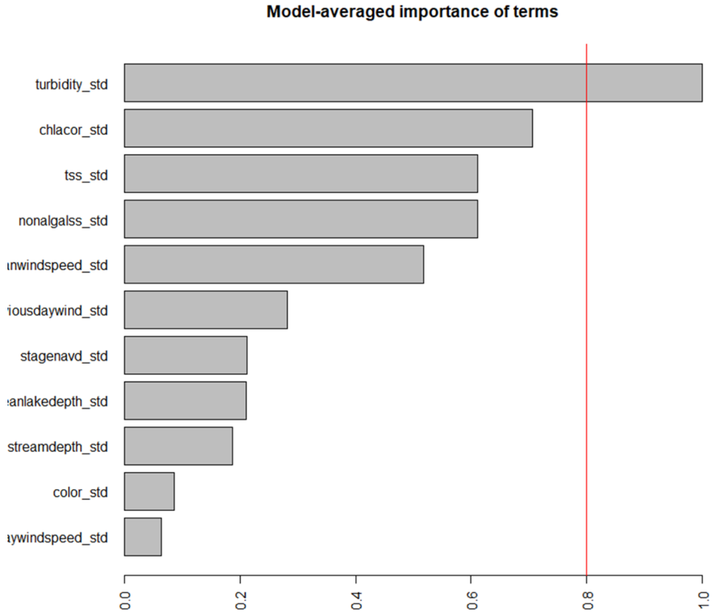
Advanced R: Statistical Machine Learning

Dan Schmutz, MS
Chief Environmental Scientist

Zoom Workshop for SJRWMD
September 24, 2020





Preprocessing Data

Ames, Iowa housing dataset

Motto: "Smart Choice"

- Install.packages(AmesHousing) # if necessary

```
# libraries
library(AmesHousing)
library(rsample)
library(caret)

# access data
ames <- AmesHousing::make_ames()

# initial dimension
dim(ames)
## [1] 2930   81

# response variable
head(ames$Sale_Price)
## [1] 215000 105000 172000 244000 189900 195500
```

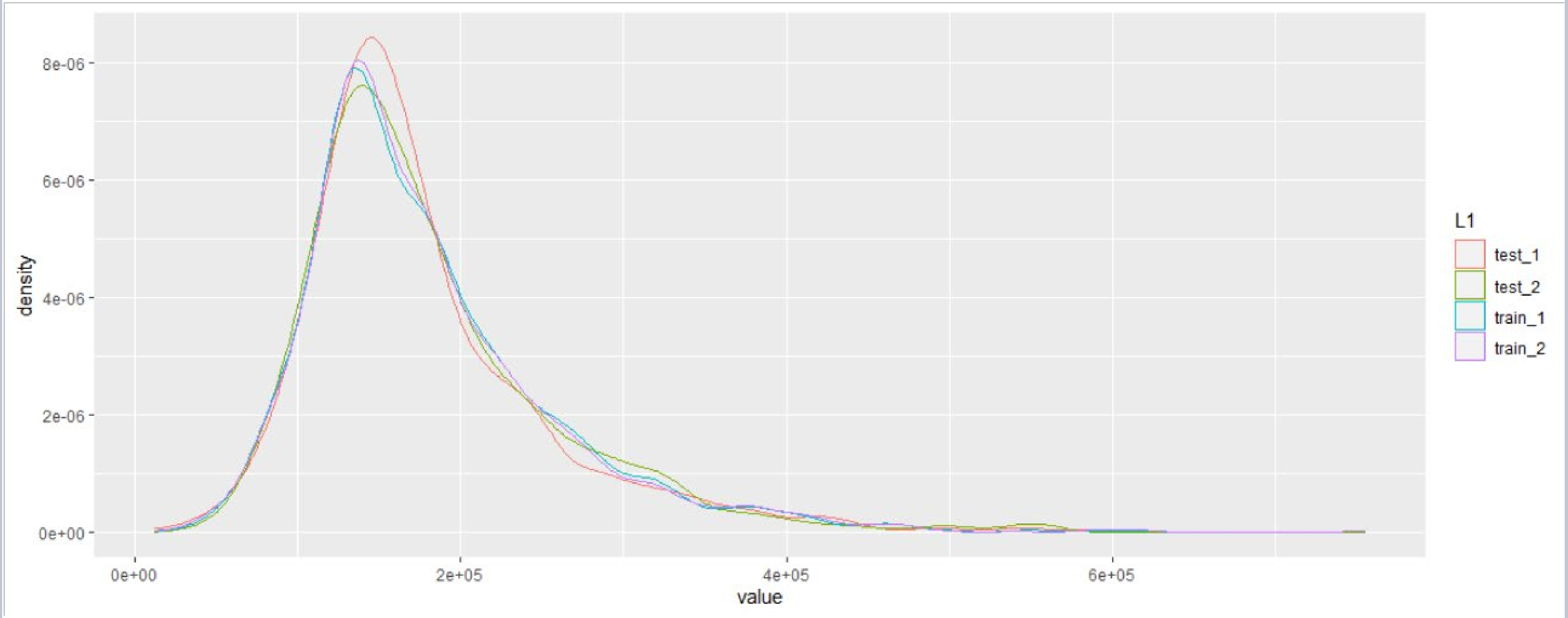
Splitting data into train and test

```r
# data partitions

# Using base R
set.seed(42)  # for reproducibility
index_1 <- sample(1:nrow(ames), round(nrow(ames) * 0.8))
train_1 <- ames[index_1, ]
test_1  <- ames[-index_1, ]

# Using caret package
set.seed(42)  # for reproducibility
index_2 <- createDataPartition(ames$Sale_Price, p = 0.8, list = FALSE)
train_2 <- ames[index_2, ]
test_2  <- ames[-index_2, ]
```

```r
library(tidyverse)
library(ggplot2)
library(reshape2)
library(rsample)
# have to use list in the next one because of different lengths of the variables, dataframe won't work
m1 <- list(train_1=train_1$Sale_Price,test_1=test_1$Sale_Price,train_2=train_2$Sale_Price, test_2=test_2$Sale_Price)
m1m<- melt(m1)
ggplot(m1m,aes(x=value,color=L1)) + geom_density(alpha=0.5)
ggplot(m1m,aes(x=value, fill=L1)) + geom_boxplot()
```

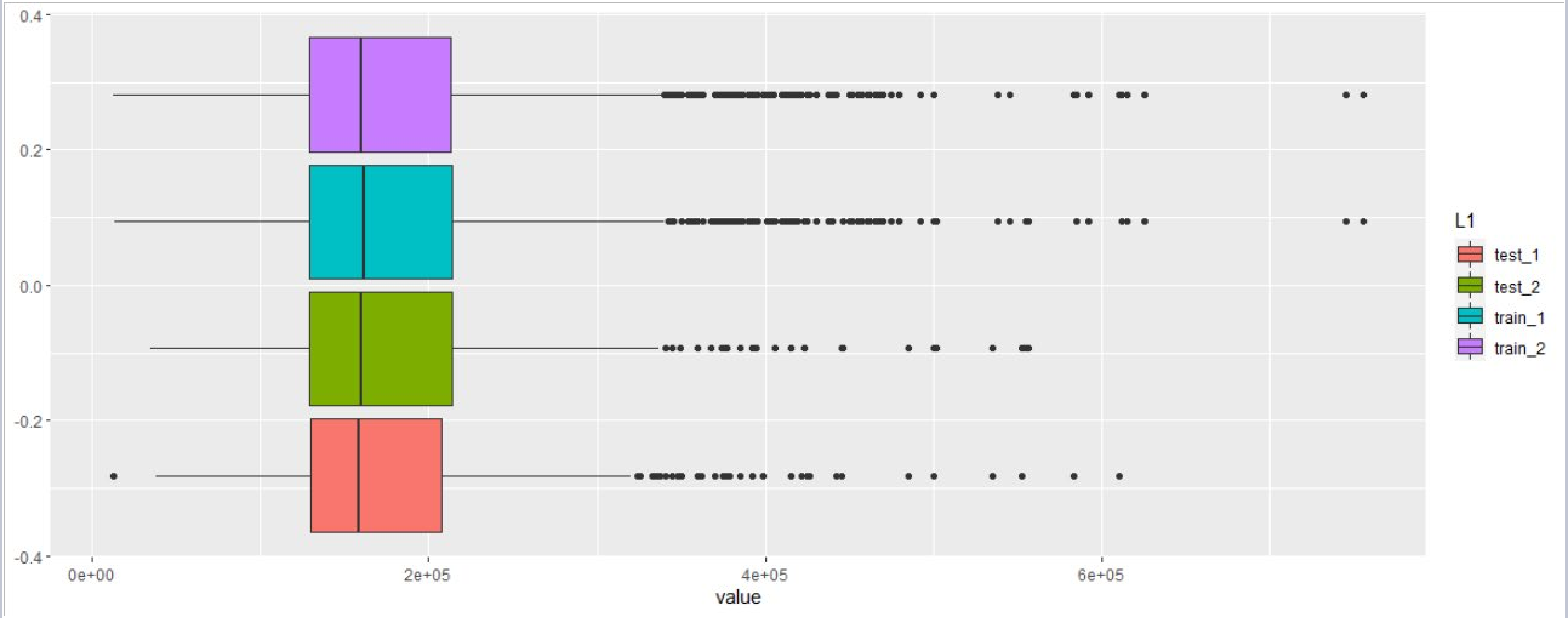
GPI

Density plots of the data partitions



GPI

Boxplots of the data partitions



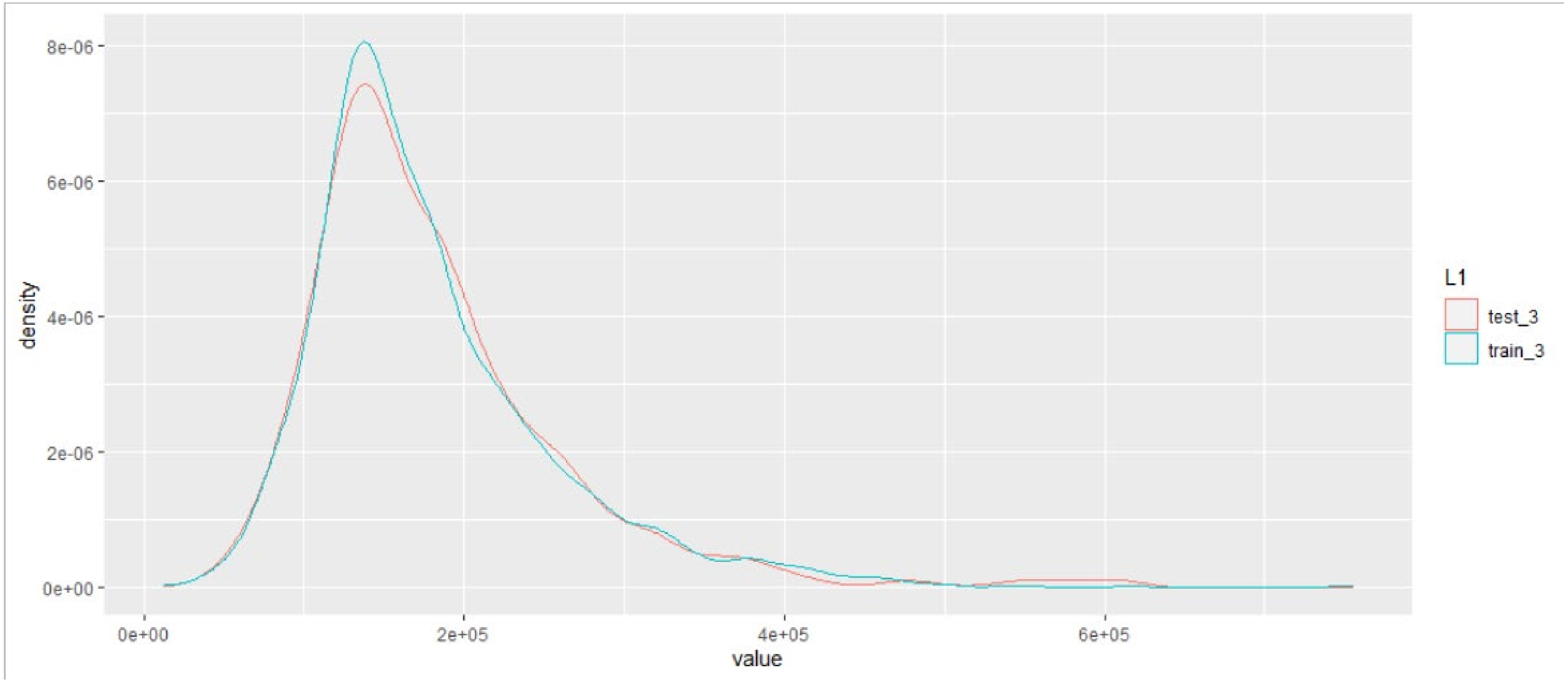
GPI

Stratified sampling to make sure train and test are representative

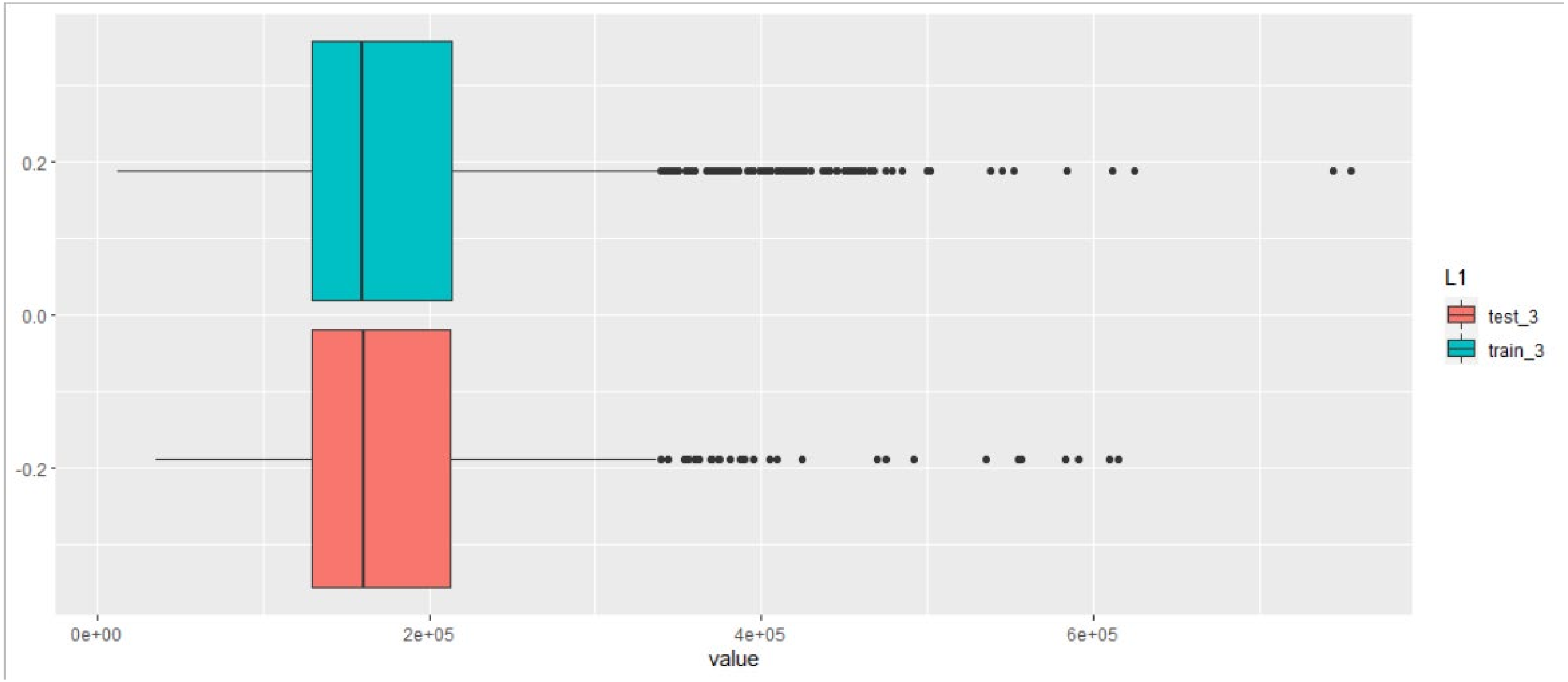
```
# stratified sampling
library(rsample)
index_3 <- initial_split(ames,0.8, strata = "Sale_Price", breaks=4)
train_3 <- training(index_3)
test_3  <- testing(index_3)
m2 <- list(train_3=train_3$Sale_Price,test_3=test_3$Sale_Price)
m2m<- melt(m2)
ggplot(m2m,aes(x=value,color=L1)) + geom_density(alpha=0.5)
ggplot(m2m,aes(x=value, fill=L1)) + geom_boxplot()
```

GPI

Density plots of the stratified data partition



Boxplots of the stratified data partition



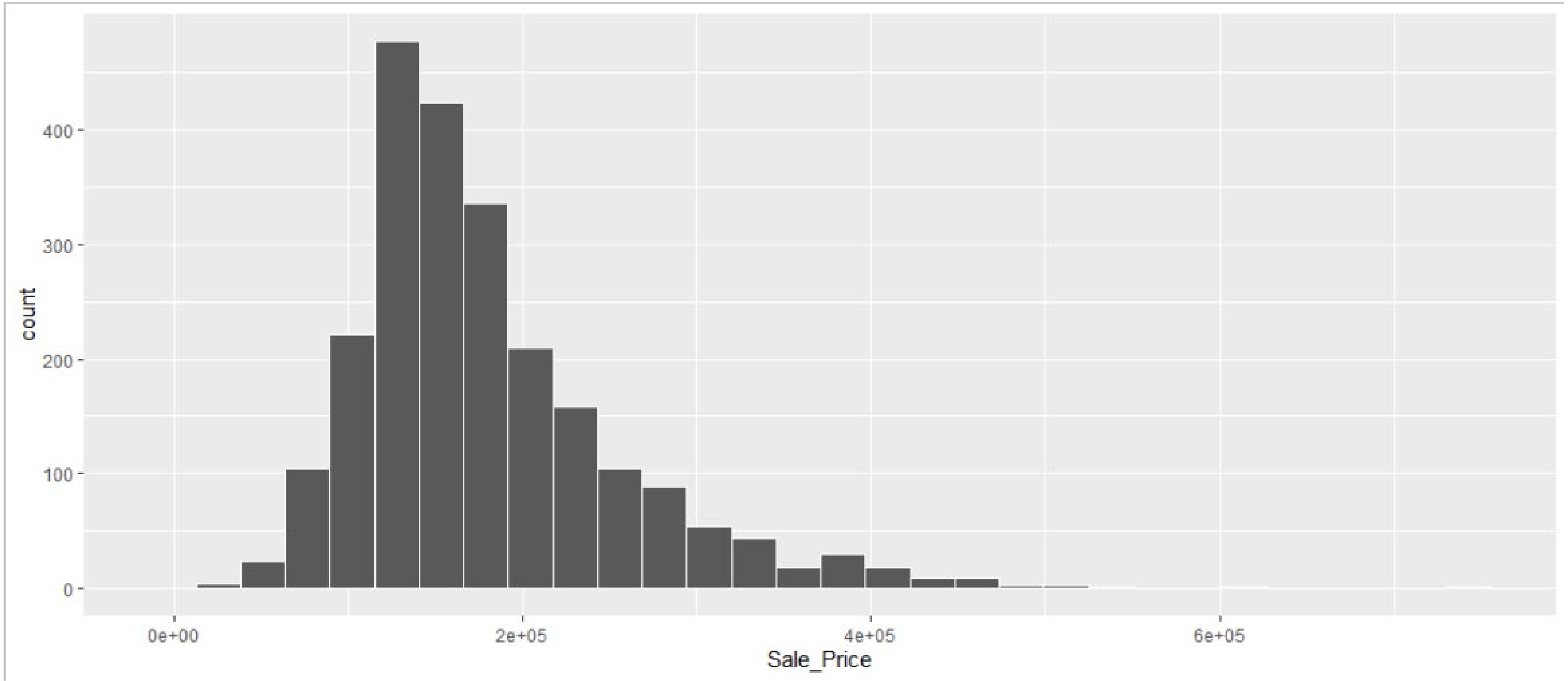
May be useful to save copy for use in another R project

```
# saving copies of the final processed files to use in other projects
write.csv(train_3,file='train_3.csv',row.names=F)
write.csv(test_3,file='test_3.csv',row.names=F)
```

GPI

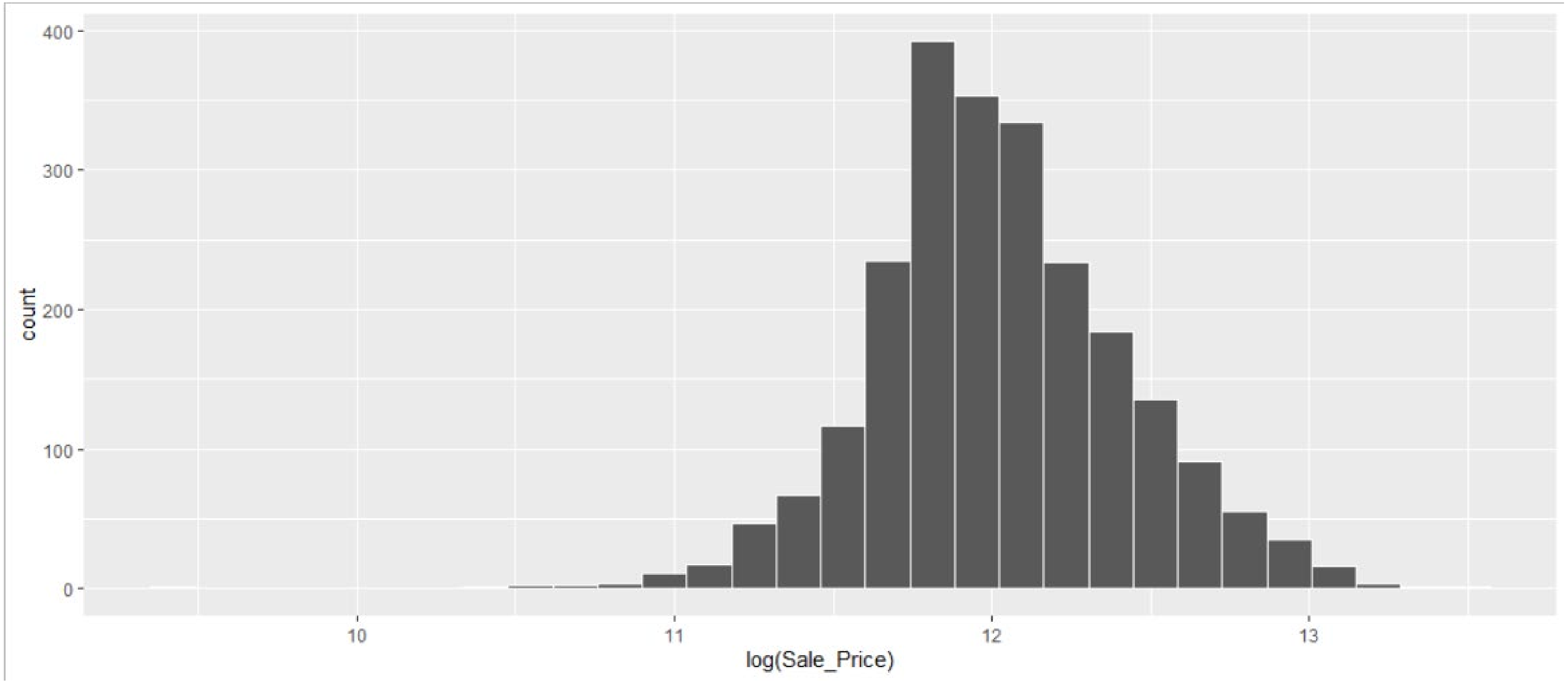
Target feature engineering

ggplot(train_3, aes(x=Sale_Price))+geom_histogram(color='white')



Target feature engineering

ggplot(train_3, aes(x=log(Sale_Price)))+geom_histogram(color='white')

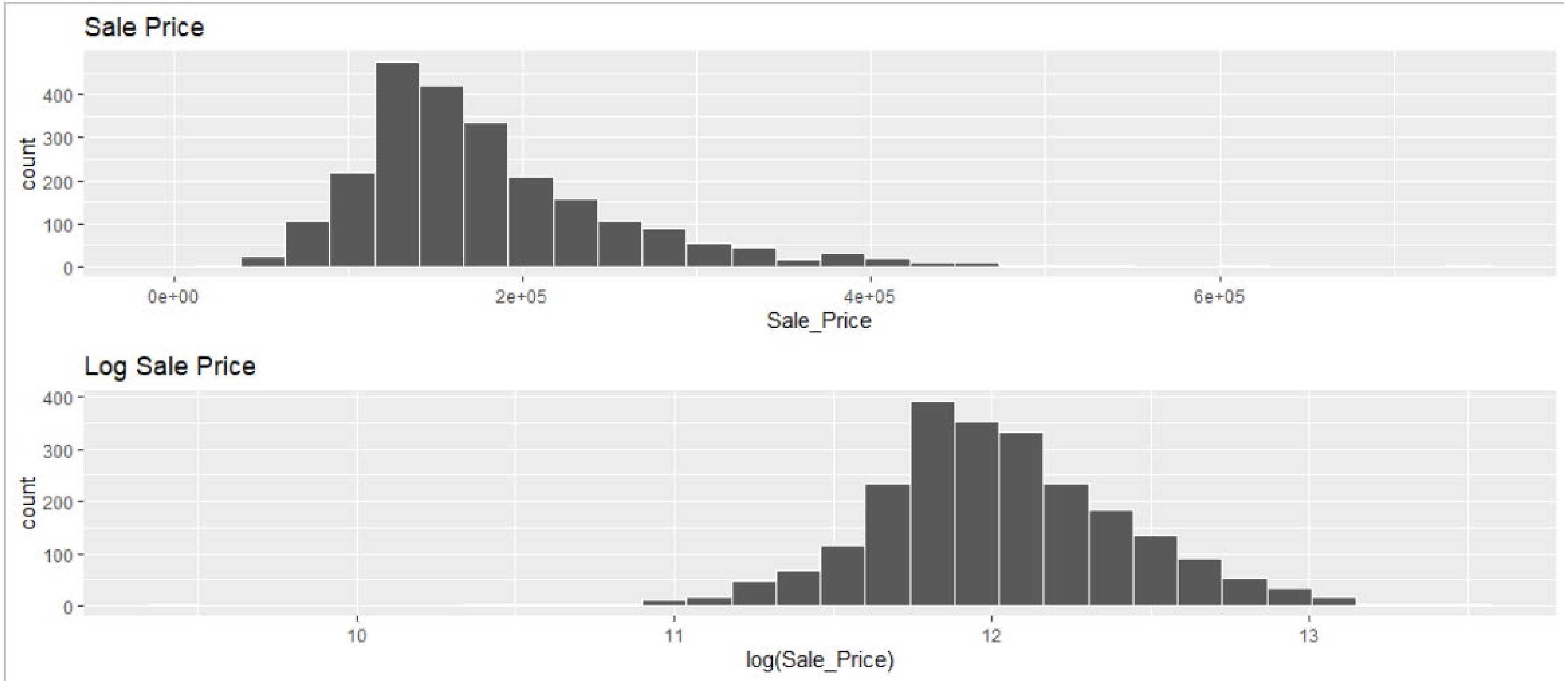


Taking log appears to improve normality. Linear regression assumption is normality of residuals, but this usually improves prediction too.

GPI

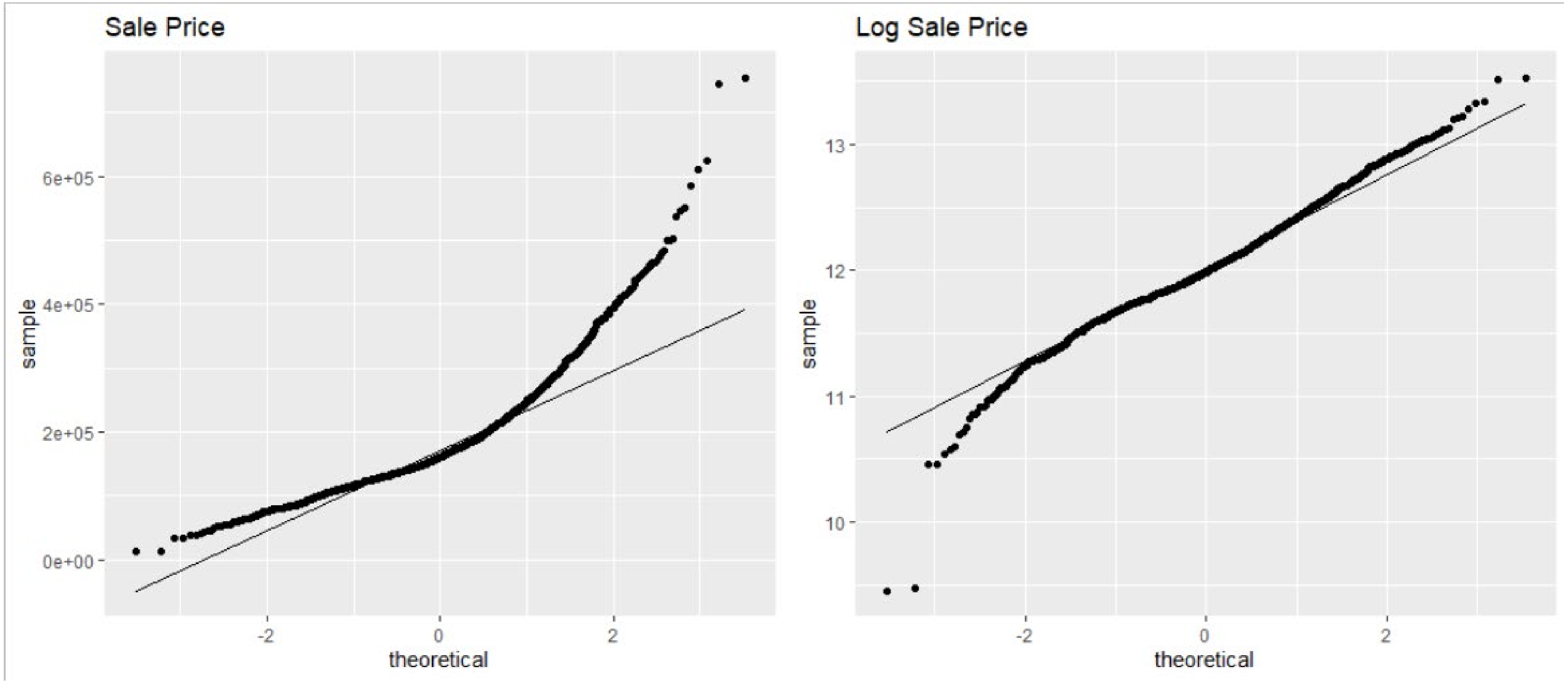
Target feature engineering

```
plot_1<-ggplot(train_3, aes(x=Sale_Price))+geom_histogram(color='white')+labs(title='Sale Price')
plot_2<-ggplot(train_3, aes(x=log(Sale_Price)))+geom_histogram(color='white')+labs(title='Log Sale Price')
grid.arrange(plot_1,plot_2, ncol=1)
```



Target feature engineering

```
plot_1b <- ggplot(train_3, aes(sample = Sale_Price))+ stat_qq() + stat_qq_line()+labs(title='Sale Price')
plot_2b <- ggplot(train_3, aes(sample = log(Sale_Price)))+ stat_qq() + stat_qq_line()+labs(title='Log Sale Price')
grid.arrange(plot_1b,plot_2b, ncol=2)
```



GPI

Normality testing typically not useful

- Normality tests tend to be very powerful at large sample sizes

```
> shapiro.test(train_3$Sale_Price)

        Shapiro-Wilk normality test

data:  train_3$Sale_Price
W = 0.88195, p-value < 2.2e-16

> shapiro.test(log(train_3$Sale_Price))

        Shapiro-Wilk normality test

data:  log(train_3$Sale_Price)
W = 0.98408, p-value = 1.536e-15
```

GPI

Box Cox finds optimal power transformation to approach normality*

$$y_i^{(\lambda)} = \begin{cases} \dfrac{y_i^\lambda - 1}{\lambda} & \text{if } \lambda \neq 0, \\ \ln y_i & \text{if } \lambda = 0, \end{cases}$$

```r
# Box Cox transform
library(recipes) # allows pre-processing of variables prior to modeling
simple_trans_rec <- recipe(Sale_Price ~ ., data = train_3) %>%
  step_BoxCox(Sale_Price) %>%
  prep(training = train_3)

simple_trans_result <- bake(simple_trans_rec, train_3)

library(EnvStats) # Box Cox in EnvStats
box_1<-boxcox(train_3$Sale_Price, optimize=T)
print(box_1)

# Box Cox using the package forecast
library(forecast)
box_2<-BoxCox.lambda(train_3$Sale_Price)
box_2
train_3t<-BoxCox(train_3$Sale_Price,lambda= "auto")
train_3t_df<-data.frame(train_3t)
colnames(train_3t_df)[1]<-"Box_Cox_Sale_Price"

plot_1<-ggplot(train_3, aes(x=Sale_Price))+
  geom_histogram(color='white')+labs(title='Sale Price')
plot_2<-ggplot(train_3, aes(x=log(Sale_Price)))+
  geom_histogram(color='white')+labs(title='Log Sale Price')
plot_3<-ggplot(train_3t_df, aes(x=Box_Cox_Sale_Price))+
  geom_histogram(color='white')+labs(title='Box Cox Sale Price')
grid.arrange(plot_1,plot_2,plot_3, ncol=1)
# InvBoxCox(x, lambda, biasadj = FALSE, fvar = NULL) will get variable back
```

* If your variable has negative values then check out the Yeo-Johnson transform.

GPI

Box Cox as part of recipes package

- Powerful idea, part of the tidymodels approach, to apply transformations to your train variables, then use the same recipe on your test data (to avoid information leakage).

```
# Box Cox transform
library(recipes) # allows pre-processing of variables prior to modeling
simple_trans_rec <- recipe(Sale_Price ~ ., data = train_3) %>%
  step_BoxCox(Sale_Price) %>%
  prep(training = train_3)

simple_trans_result <- bake(simple_trans_rec, train_3)
```

GPI

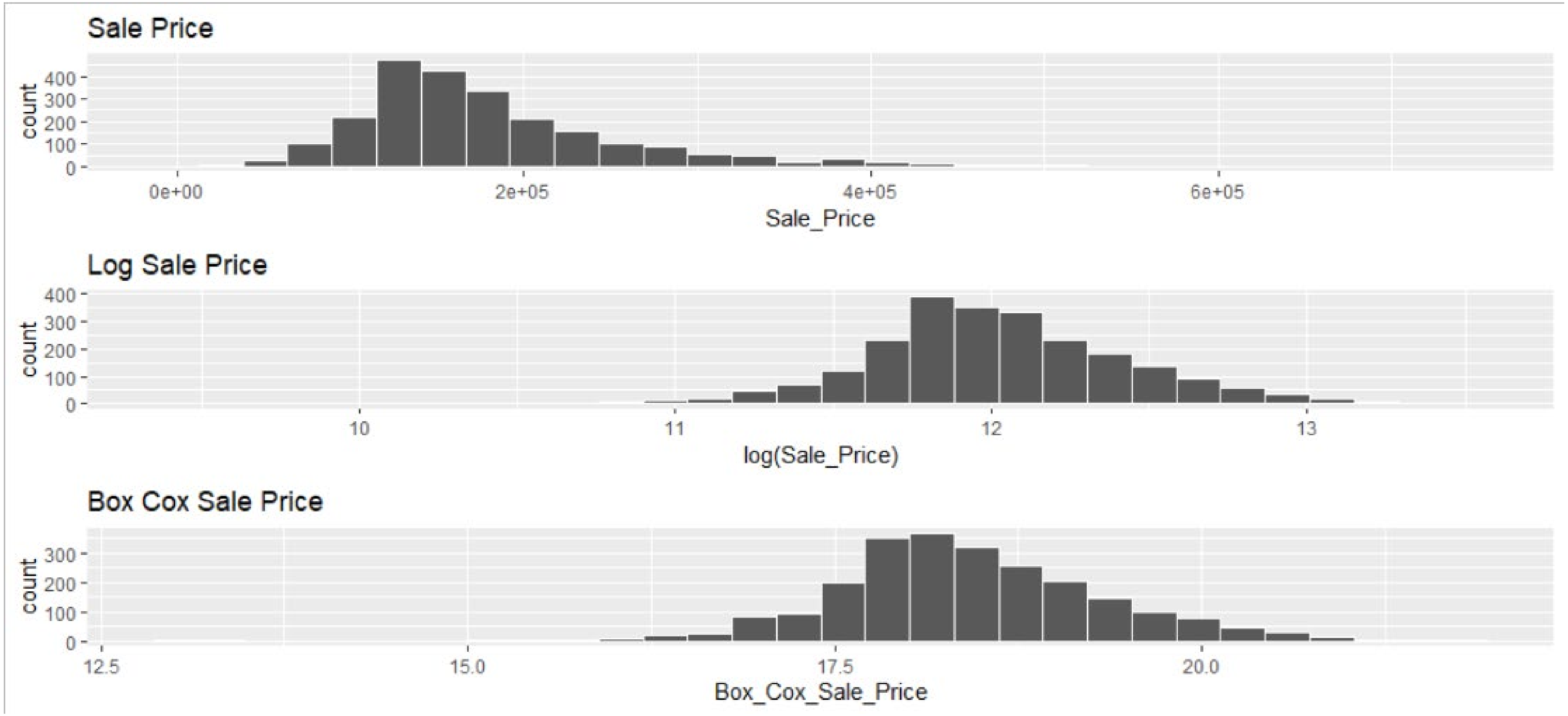
Box Cox EnvStats package

```
> print(box_1)
$lambda
[1] 0.07854492

$objective
[1] 0.9922288

$objective.name
[1] "PPCC"

$optimize
[1] TRUE

$optimize.bounds
lower upper
   -2     2

$eps
[1] 2.220446e-16

$data
   [1] 215000 105000 172000 244000 189900 195500 213500 191500 236500 175900
  [11] 185000 171500 212000 538000 164000 394432 141000 210000 190000 170000
  [21] 216000 149900 142000 126000 115000 184000  96000 105500  88000 127500
  [31] 149900 146000 376162 306000 395192 220000 275000 214000 611657 224000
  [41] 500000 320000 319900 175500 199500 160000 192000 184500 216500 185088
  [51] 180000 222500 333168 355000 260400 325000 221000 410000 204500 254900
```

Lambda close to 0, which would be the log (natural log) transform.

GPI

Box Cox results using forecase package—not much different from log



GPI

Any missing values in ames? How about in the raw data?

- Two different ways to count missing values in your dataset

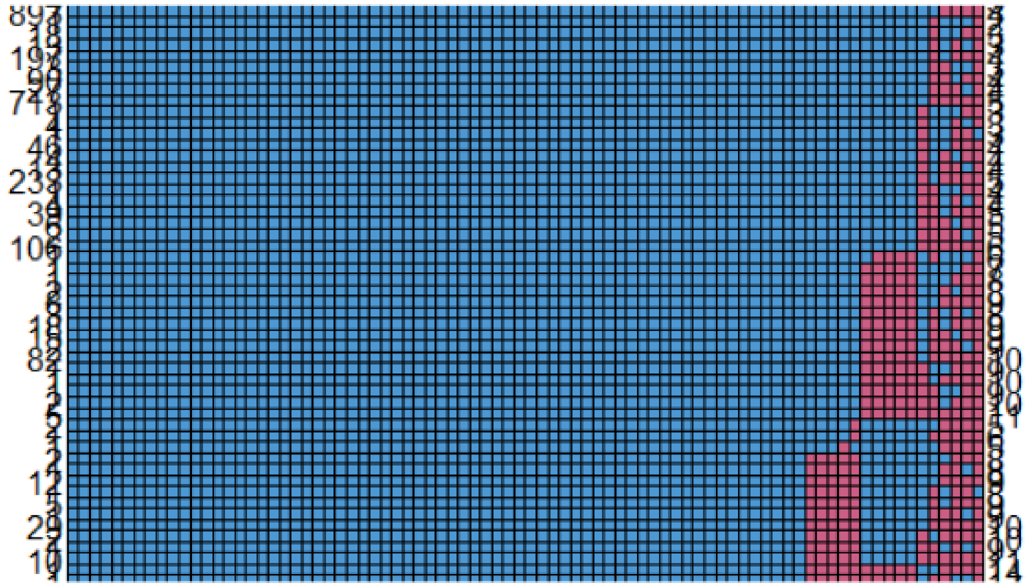
```
# are any values missing?
is.na(ames) %>% table()
sum(is.na(ames))

sum(is.na(AmesHousing::ames_raw)) # back to the preprocessed version
```

GPI

Patterns in missing data

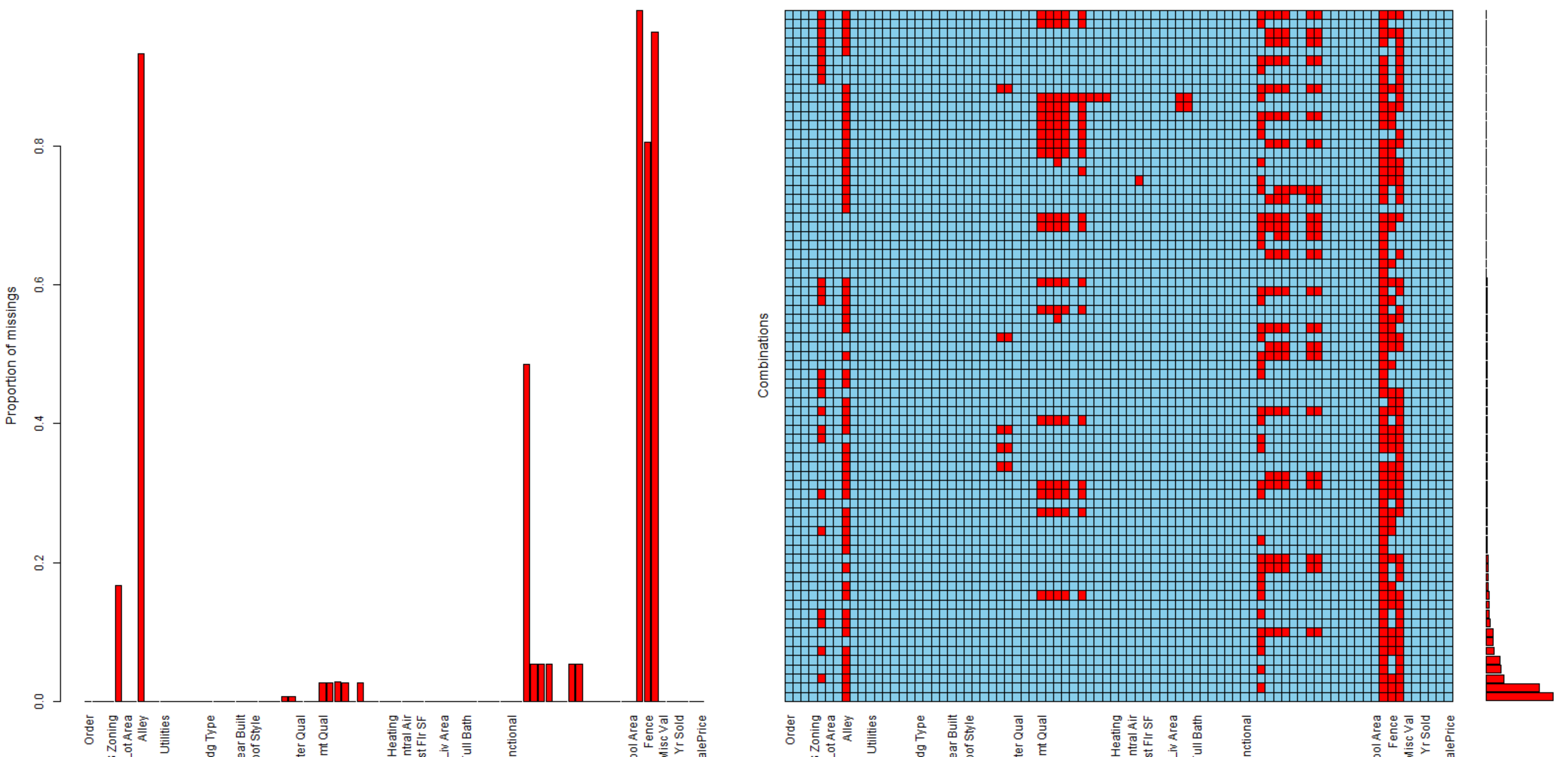
mpat<-md.pattern(ames_raw)

Pretty hard to see at this scale but useful for smaller datasets



```
png(file="mdpat.png",width=1600,height=800)
aggr(ames_raw)
dev.off()
matrixplot(ames_raw)
```

A little more useful



- Listing the features with missing values

```
mdcount<-sapply(ames_raw, function(x) sum(is.na(x)))
mdcount %>% data.frame(mdcount) %>% arrange(desc(mdcount))
```

131:1 (Top Level)

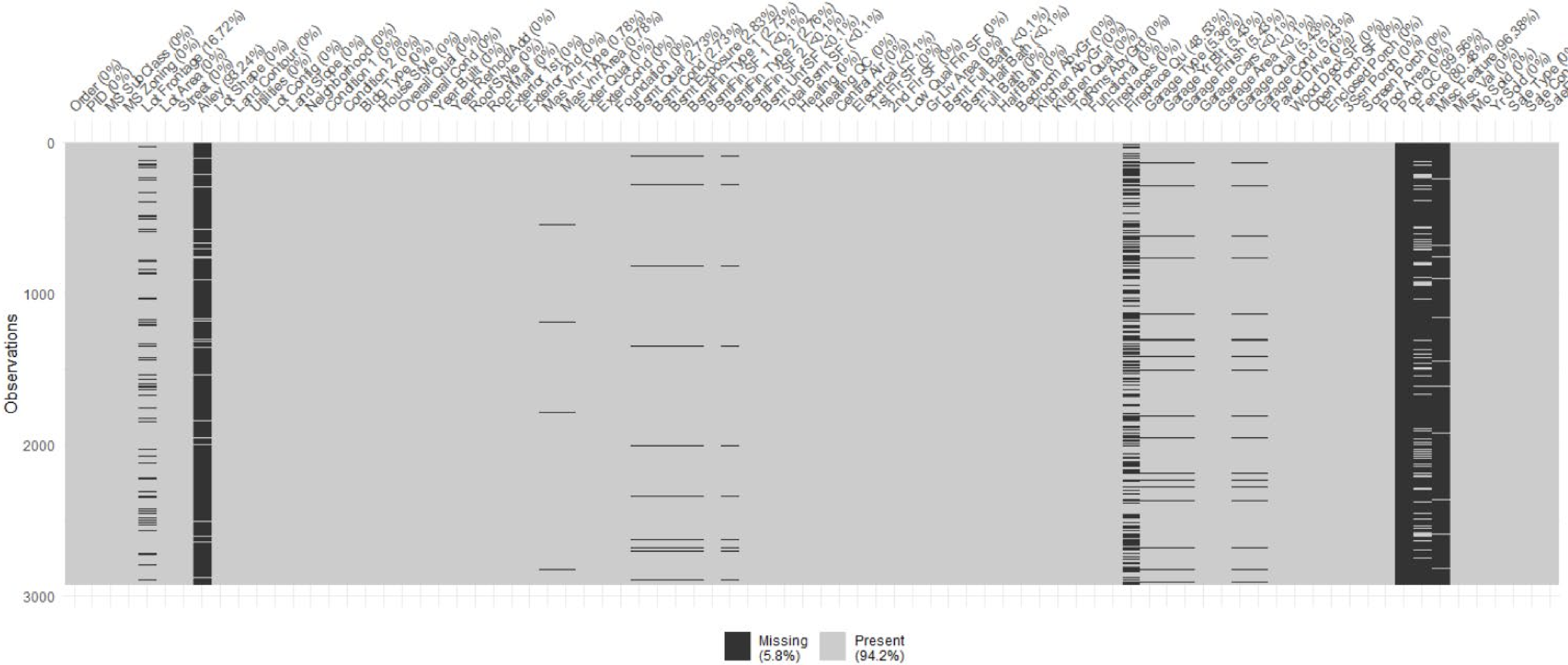
Console | Terminal | Jobs

D:/Personal/R_Training_SJRWMD/Day_2_ML/R_review_for_Kaggle_upload/

```
                     0            0
> mdcount %>% data.frame(mdcount) %>% arrange(desc(mdcount))
                    . mdcount
Pool QC          2917    2917
Misc Feature     2824    2824
Alley            2732    2732
Fence            2358    2358
Fireplace Qu     1422    1422
Lot Frontage      490     490
Garage Yr Blt     159     159
Garage Finish     159     159
Garage Qual       159     159
Garage Cond       159     159
Garage Type       157     157
Bsmt Exposure      83      83
BsmtFin Type 2     81      81
Bsmt Qual          80      80
Bsmt Cond          80      80
BsmtFin Type 1     80      80
Mas Vnr Type       23      23
Mas Vnr Area       23      23
Bsmt Full Bath      2       2
Bsmt Half Bath      2       2
BsmtFin SF 1        1       1
BsmtFin SF 2        1       1
Bsmt Unf SF         1       1
Total Bsmt SF       1       1
Electrical          1       1
Garage Cars         1       1
Garage Area         1       1
```

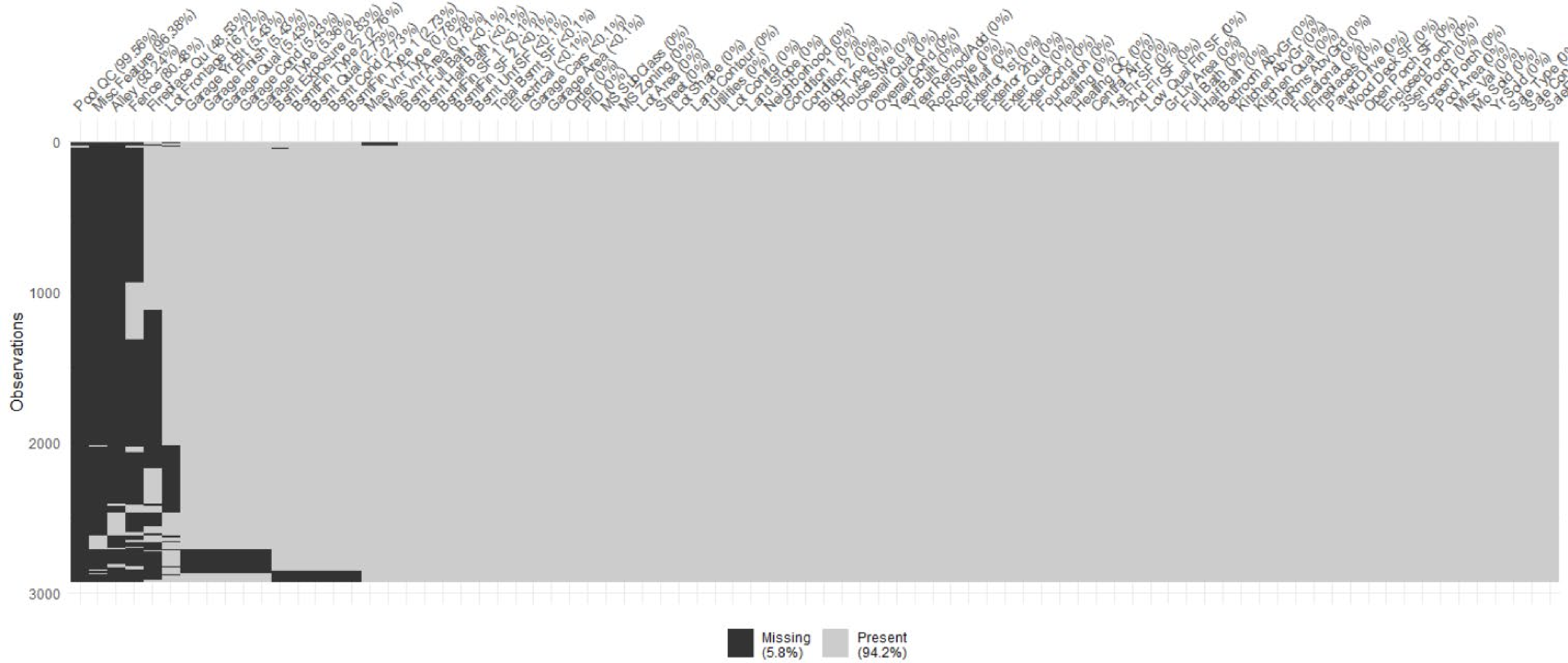
```
library(visdat)
vis_miss(ames_raw)
```

Beautiful!



vis_miss(ames_raw, cluster=T, sort_miss=T)

Even better, sorted.



Handling Missing Data

- If more than 20% of cases are missing, probably need to consider discarding the variable, unless another predictor or combination of predictors is highly predictive of the variable
- Sometimes missingness is informative, so coding as a missing category or an extreme numeric value may be o.k. depending on the type of model (i.e., not leaving it as NA)
- Consider data imputation using information from train dataset only to avoid information leakage
  - Median
  - Mean
  - k-nearest neighbor interpolation (near in data space)
  - random forest interpolation
  - Time series might use linear interpolation, or substituting last value

GPI

Other variable processing

- Drop zero variance or near zero variance predictors
- Some models have trouble with nominal variables (i.e., categorical), so they need to be encoded as integers
  - One-hot encoding (can blow up the number of features)
  - Label encoding (give different integer to each level); if there is a meaningful order, then please use it.
- Possibly
  - Center
  - Scale

Color
Red
Red
Yellow
Green
Yellow

Red	Yellow	Green
1	0	0
1	0	0
0	1	0
0	0	1

https://www.quora.com/What-is-one-hot-encoding-and-when-is-it-used-in-data-science