lab04

Sean Fitch

2024-12-03

```
library(readr)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(ggplot2)
library(tidyr)
library(scales)
```

```
##
## Attaching package: 'scales'
```

```
## The following object is masked from 'package:readr':
##
##     col_factor
```

```
library(caret)
```

```
## Loading required package: lattice
```

```
library(torch)
library(randomForest)
```

```
## randomForest 4.7-1.2
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin

## The following object is masked from 'package:dplyr':
##
##     combine
```

```
data <- read_csv("~/Projects/DataAnalytics2024_Sean_Fitch/data/NYC_Citywide_Annualized_Calendar_Sales_U
```

```
## Rows: 606260 Columns: 31
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (13): BOROUGH, NEIGHBORHOOD, BUILDING CLASS CATEGORY, TAX CLASS AS OF FI...
## dbl (17): BLOCK, LOT, ZIP CODE, RESIDENTIAL UNITS, COMMERCIAL UNITS, TOTAL U...
## lgl  (1): EASE-MENT
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
# Count occurrences of each unique value in the BOROUGH column
table(data$BOROUGH)
```

```
##
##             1             2             3             4             5
##         96088         38219        123813        136547         43819
##         BRONX      BROOKLYN     MANHATTAN        QUEENS STATEN ISLAND
##         14337         48864         35636         51513         17424
```

It seems that there are 2 labels for each borough: the name and a numeric. I will find which number is which borough by viewing the overlap in neighberhoods between boroughs.

```
# Compute the overlap matrix
overlap_matrix <- data %>%
  group_by(NEIGHBORHOOD) %>%
  summarise(BOROUGHS = list(unique(BOROUGH))) %>%
  unnest_longer(BOROUGHS) %>%
  count(NEIGHBORHOOD, BOROUGHS) %>%
  pivot_wider(names_from = BOROUGHS, values_from = n, values_fill = 0) %>%
  select(-NEIGHBORHOOD) %>%
  as.matrix()

# Extract the unique borough names
borough_names <- colnames(overlap_matrix)

# Compute pairwise overlaps
pairwise_overlaps <- outer(borough_names, borough_names, Vectorize(function(b1, b2) {
  sum(overlap_matrix[, b1] > 0 & overlap_matrix[, b2] > 0)
}))

# Convert to a matrix and set row/column names as borough names
rownames(pairwise_overlaps) <- borough_names
```

```r
colnames(pairwise_overlaps) <- borough_names

# View the resulting matrix with borough names as labels
pairwise_overlaps
```

```
##                4 QUEENS  1 MANHATTAN  5 STATEN ISLAND  3 BROOKLYN  2 BRONX
## 4             62     61  0          0  1              1  1         0  0      0
## QUEENS        61     61  0          0  1              1  1         0  0      0
## 1              0      0 40         39  0              0  0         0  0      0
## MANHATTAN      0      0 39         39  0              0  0         0  0      0
## 5              1      1  0          0 59             58  0         0  0      0
## STATEN ISLAND  1      1  0          0 58             58  0         0  0      0
## 3              1      1  0          0  0              0 62        60  0      0
## BROOKLYN       0      0  0          0  0              0 60        60  0      0
## 2              0      0  0          0  0              0  0         0 40     38
## BRONX          0      0  0          0  0              0  0         0 38     38
```

```r
data <- data %>%
  mutate(BOROUGH = case_when(
    BOROUGH == "4" ~ "QUEENS",
    BOROUGH == "1" ~ "MANHATTAN",
    BOROUGH == "5" ~ "STATEN ISLAND",
    BOROUGH == "3" ~ "BROOKLYN",
    BOROUGH == "2" ~ "BRONX",
    TRUE ~ BOROUGH
  )) %>%
  mutate(BOROUGH = factor(BOROUGH))

table(data$BOROUGH)
```

```
##
##         BRONX      BROOKLYN     MANHATTAN        QUEENS STATEN ISLAND
##         52556        172677        131724        188060         61243
```

1.

```r
bronx_data <- data %>% filter(BOROUGH == "BRONX")
colnames(bronx_data) <- gsub(" ", "_", colnames(bronx_data))

# Clean the data: Replace "- 0" with 0, and remove commas
bronx_data <- bronx_data %>%
  mutate(
    LAND_SQUARE_FEET = as.numeric(gsub(",", "", gsub("-\\s0", "0", LAND_SQUARE_FEET))),
    GROSS_SQUARE_FEET = as.numeric(gsub(",", "", gsub("-\\s0", "0", GROSS_SQUARE_FEET))),
    RESIDENTIAL_UNITS = as.numeric(gsub(",", "", gsub("-\\s0", "0", RESIDENTIAL_UNITS))),
    COMMERCIAL_UNITS = as.numeric(gsub(",", "", gsub("-\\s0", "0", COMMERCIAL_UNITS))),
    Latitude = as.numeric(gsub(",", "", gsub("-\\s0", "0", Latitude))),
    Longitude = as.numeric(gsub(",", "", gsub("-\\s0", "0", Longitude))),
    TOTAL_UNITS = as.numeric(gsub(",", "", gsub("-\\s0", "0", TOTAL_UNITS))),
    YEAR_BUILT = as.numeric(gsub(",", "", gsub("-\\s0", "0", YEAR_BUILT))),
    SALE_PRICE = as.numeric(gsub(",", "", gsub("-\\s0", "0", SALE_PRICE))),
    SALE_DATE = as.Date(SALE_DATE, format = "%m/%d/%Y")
  )
```

a)

I will analyze trends in property sales values over time, focusing on seasonal variations and long-term price changes. Spatial analysis would help identify neighborhoods with higher sales activity or distinct pricing patterns. Regression models could explore relationships between property attributes (e.g., type, size) and sale price, while clustering techniques might reveal sales trends.

b)

```
# Summary statistics for overview
summary(bronx_data$SALE_PRICE)
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
##         0         0    290000    807208    600000 133400000
```

```
summary(bronx_data$LAND_SQUARE_FEET)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.     NA's
##        0     1800     2500    13846     3181 19850400     6879
```

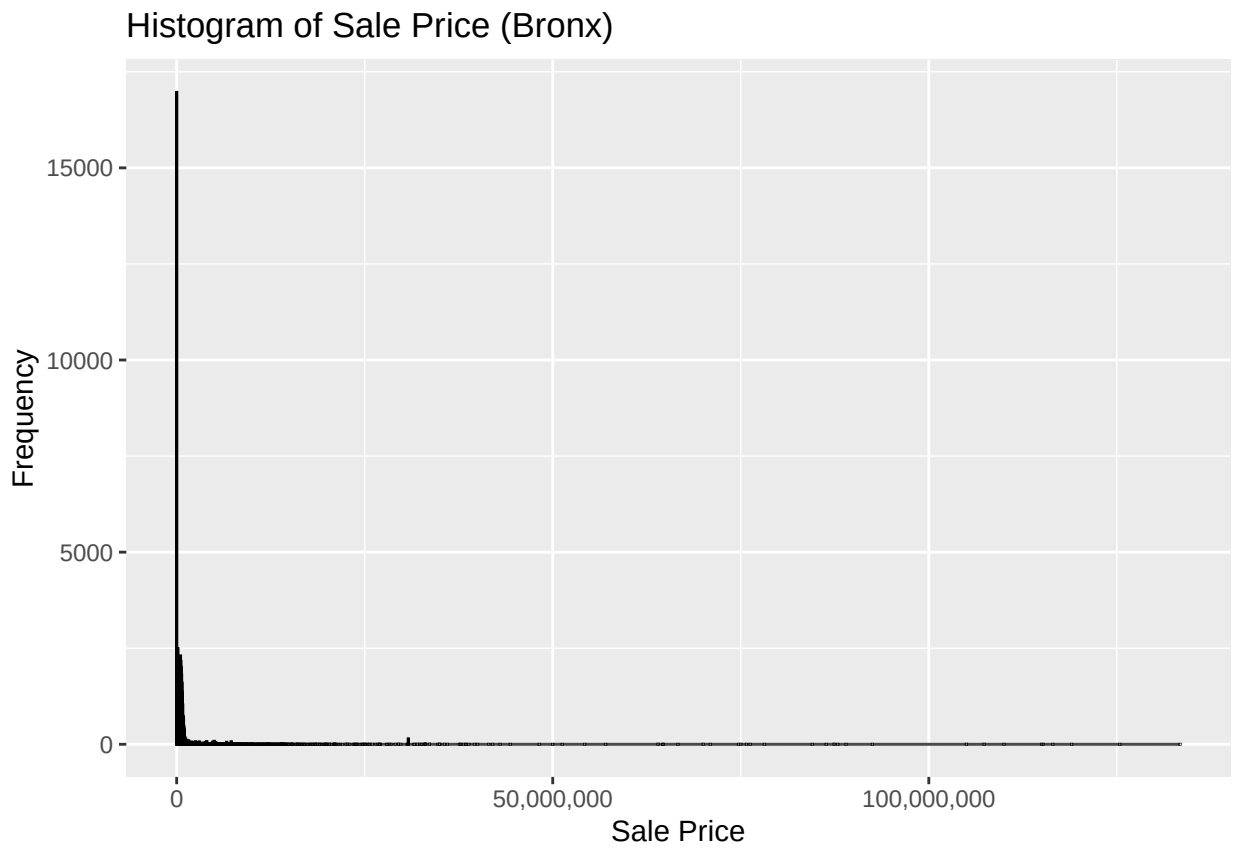
```
sum(bronx_data$LAND_SQUARE_FEET == 0 | is.na(bronx_data$LAND_SQUARE_FEET))
```

```
## [1] 13652
```

```
sum(bronx_data$LAND_SQUARE_FEET != 0 & !is.na(bronx_data$LAND_SQUARE_FEET))
```

```
## [1] 38904
```

```
# Histogram for SALE PRICE distribution
ggplot(bronx_data, aes(x = SALE_PRICE)) +
  geom_histogram(binwidth = 50000, fill = "blue", color = "black") +
  scale_x_continuous(labels = comma) +
  labs(title = "Histogram of Sale Price (Bronx)", x = "Sale Price", y = "Frequency")
```



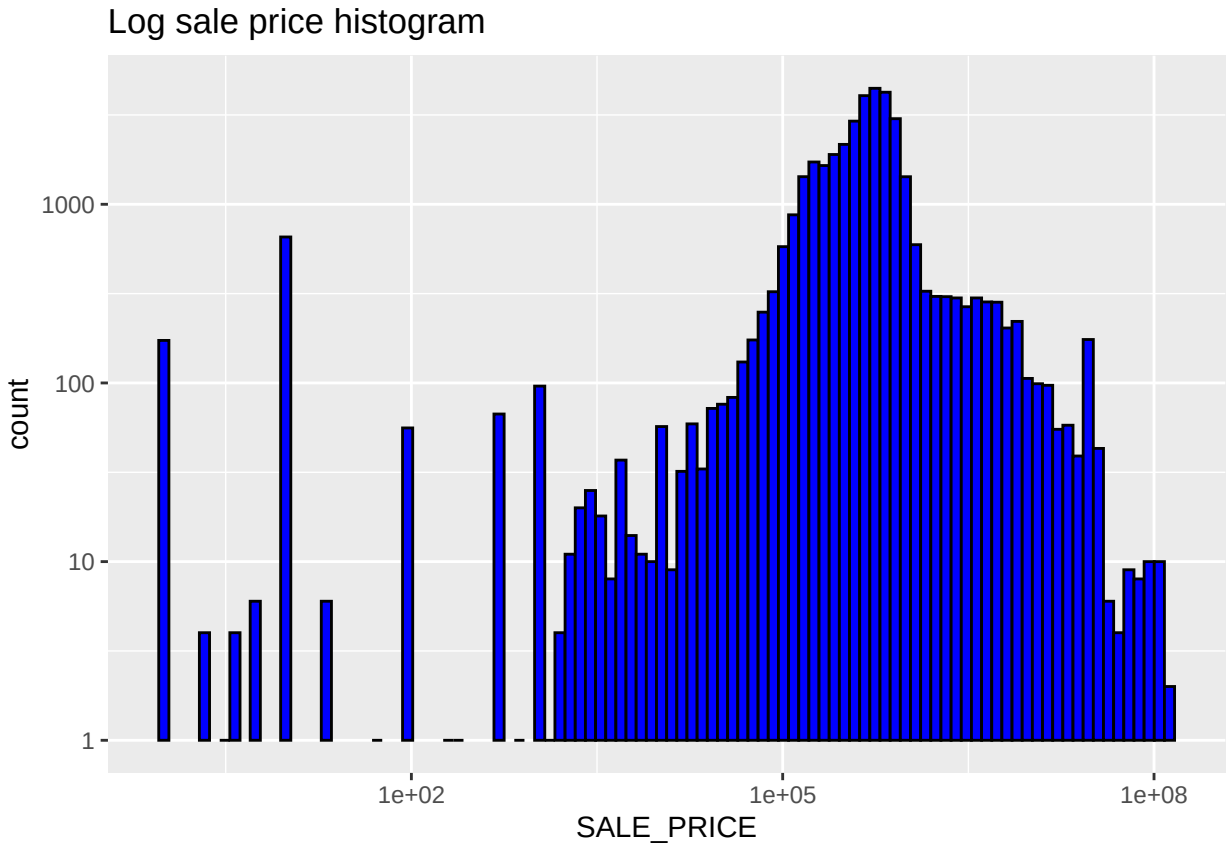

```
# Log histogram for SALE PRICE distribution
ggplot(bronx_data, aes(x = SALE_PRICE)) +
                  geom_histogram(bins=100, fill = "blue", color = "black") +
                  scale_x_log10() + scale_y_log10() +
                  labs(title = "Log sale price histogram")
```

```
## Warning in scale_x_log10(): log-10 transformation introduced infinite values.
```

```
## Warning: Removed 15502 rows containing non-finite outside the scale range
## (`stat_bin()`).
```

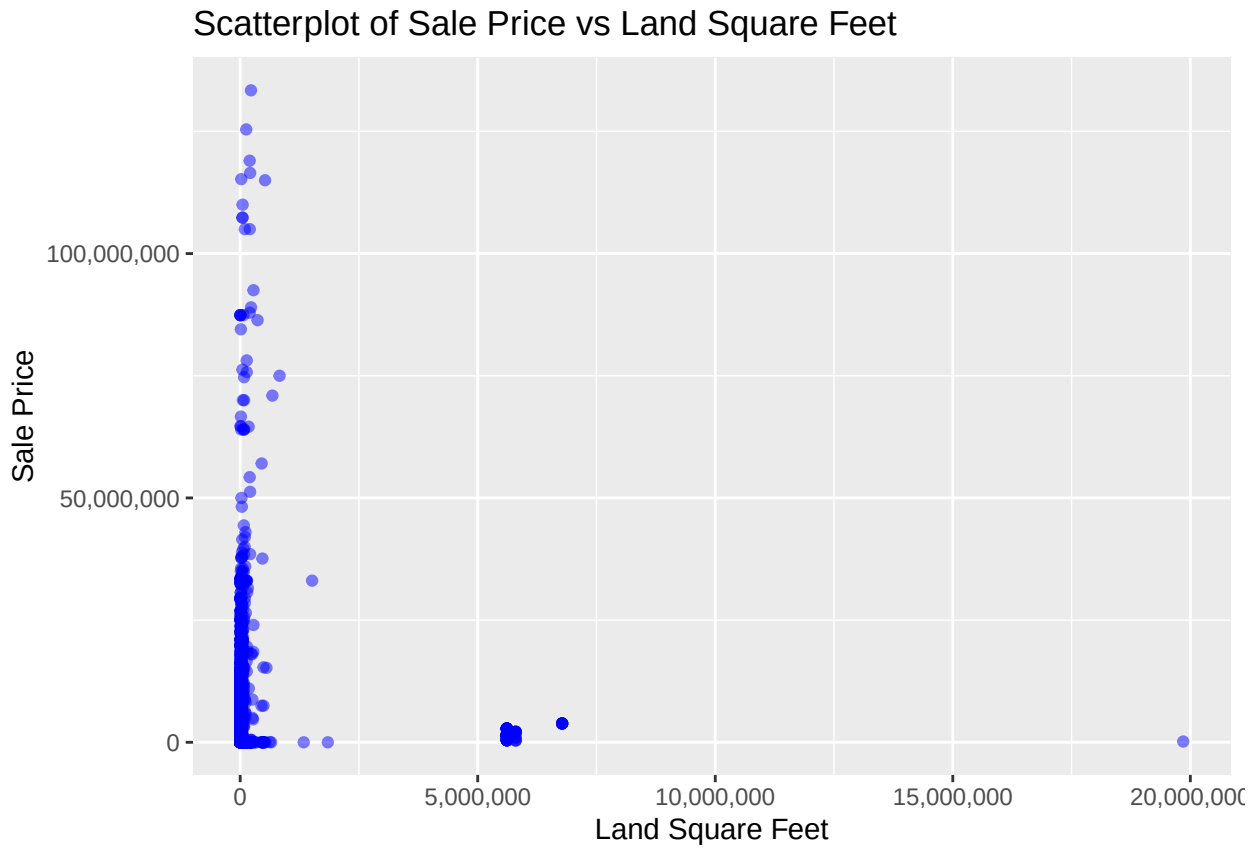
```
## Warning in scale_y_log10(): log-10 transformation introduced infinite values.
```

```
## Warning: Removed 24 rows containing missing values or values outside the scale range
## (`geom_bar()`).
```




```
land_bronx_data <- bronx_data %>% filter(LAND_SQUARE_FEET != 0 & !is.na(LAND_SQUARE_FEET))

# Scatterplot to visualize SALE PRICE against LAND SQUARE FEET
ggplot(land_bronx_data, aes(x = LAND_SQUARE_FEET, y = SALE_PRICE)) +
  geom_point(alpha = 0.5, color = "blue") +
  scale_x_continuous(labels = comma) +
  scale_y_continuous(labels = comma) +
  labs(title = "Scatterplot of Sale Price vs Land Square Feet",
       x = "Land Square Feet", y = "Sale Price")
```



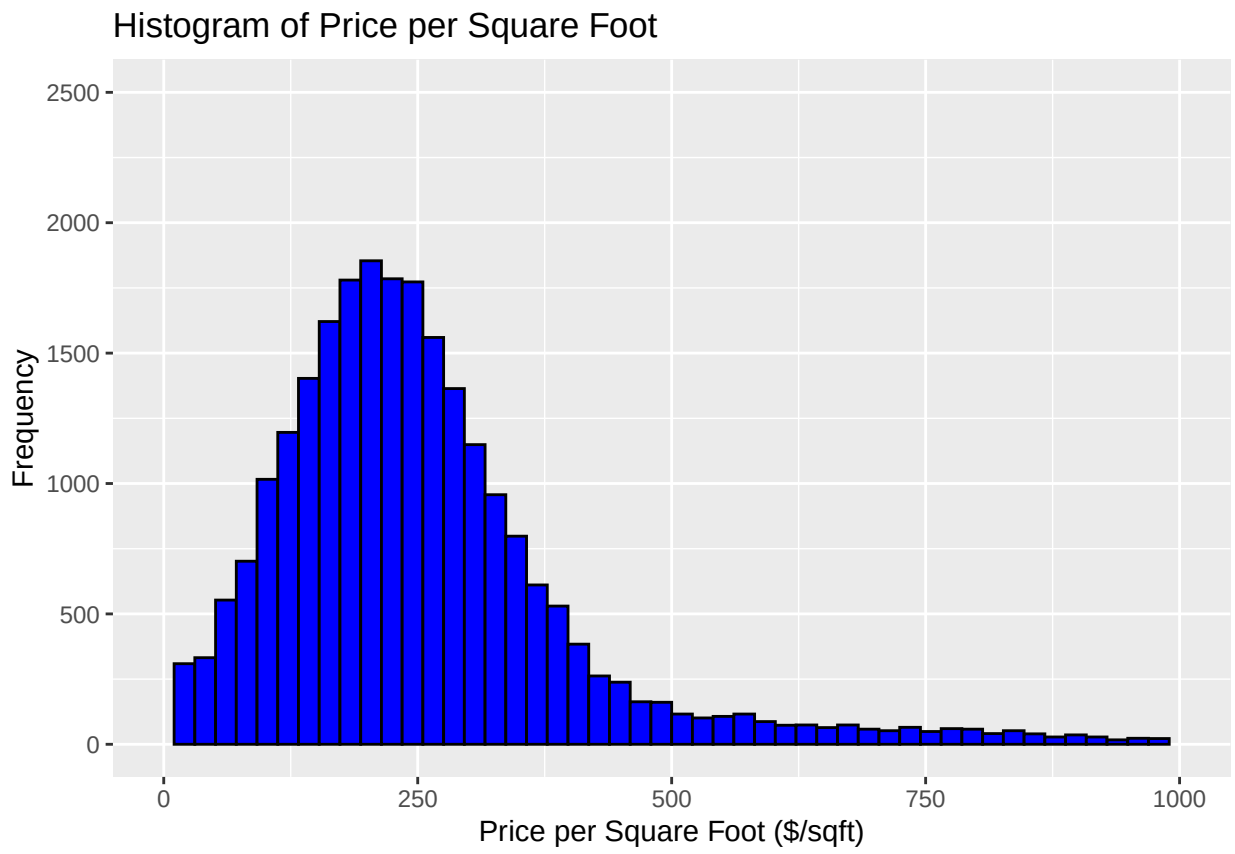

```
# Calculate price per square foot ($/sqft)
land_bronx_data <- land_bronx_data %>%
  mutate(SQFT_PRICE = SALE_PRICE / LAND_SQUARE_FEET)

# Histogram of $/sqft distribution
ggplot(land_bronx_data, aes(x = SQFT_PRICE)) +
  geom_histogram(bins = 50, fill = "blue", color = "black") +
  scale_x_continuous(labels = comma) +
  labs(title = "Histogram of Price per Square Foot", x = "Price per Square Foot ($/sqft)", y = "Frequen
  xlim(0, 1000) +
  ylim(0, 2500)
```

```
## Scale for x is already present.
## Adding another scale for x, which will replace the existing scale.
```

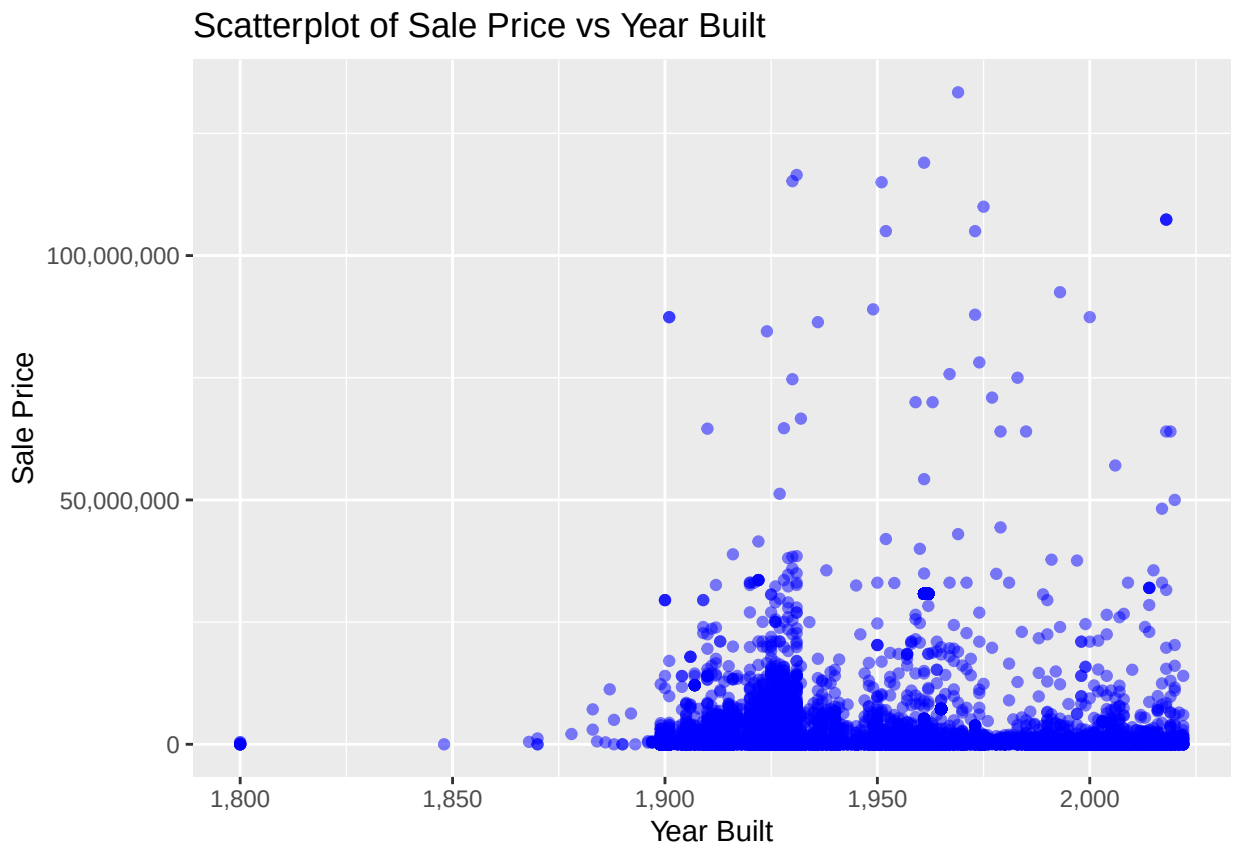
```
## Warning: Removed 546 rows containing non-finite outside the scale range
## (`stat_bin()`).
```

```
## Warning: Removed 2 rows containing missing values or values outside the scale range
## (`geom_bar()`).
```

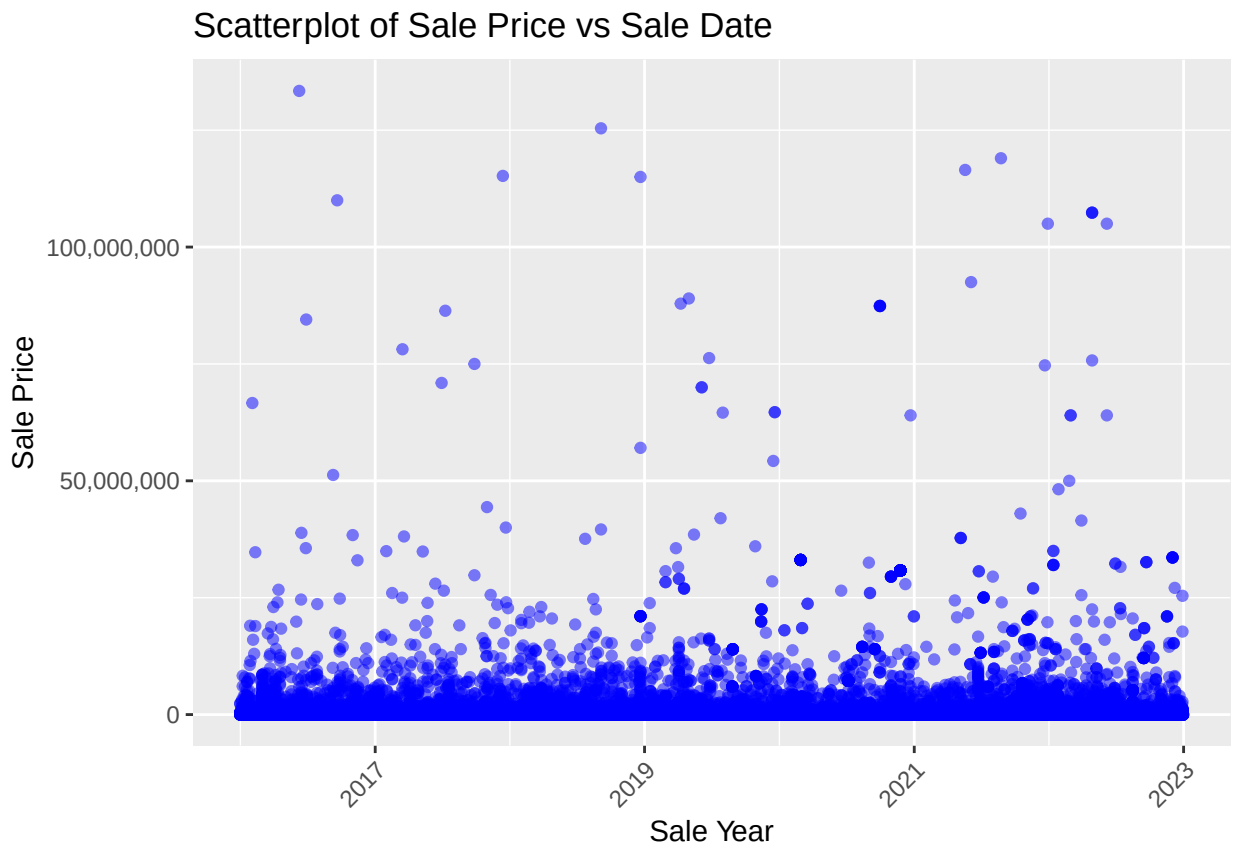



```r
year_bronx_data <- bronx_data %>% filter(YEAR_BUILT != 0 & !is.na(YEAR_BUILT))

# Scatterplot to visualize SALE PRICE against year built
ggplot(year_bronx_data, aes(x = YEAR_BUILT, y = SALE_PRICE)) +
  geom_point(alpha = 0.5, color = "blue") +
  scale_x_continuous(labels = comma) +
  scale_y_continuous(labels = comma) +
  labs(title = "Scatterplot of Sale Price vs Year Built",
       x = "Year Built", y = "Sale Price")
```




```
# Scatterplot to visualize SALE PRICE against sale date
ggplot(bronx_data, aes(x = SALE_DATE, y = SALE_PRICE)) +
  geom_point(alpha = 0.5, color = "blue") +
  scale_x_date(labels = scales::date_format("%Y"), breaks = "2 years") +
  scale_y_continuous(labels = scales::comma) +
  labs(title = "Scatterplot of Sale Price vs Sale Date",
       x = "Sale Year", y = "Sale Price") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



I started by summarizing the key variables of interest, such as SALE_PRICE and LAND_SQUARE_FEET, to get an overview of the dataset. I created histograms for the SALE_PRICE distribution to visualize the frequency of different price ranges, both in the raw form and on a log scale for better interpretation of the skewed data. A scatterplot of SALE_PRICE vs LAND_SQUARE_FEET was plotted to examine the relationship between land size and sale price, followed by a calculation of the price per square foot to identify any trends or pricing anomalies. The distribution of price per square foot was visualized using a histogram to observe any potential outliers or unusual pricing patterns. Additionally, I created scatterplots for SALE_PRICE against the YEAR_BUILT and SALE_DATE to identify temporal trends and observe how prices have evolved over time.

From the analysis, it was surprising to find some sales below $100, which could be outliers or data errors. These outliers are likely caused by missing or incorrect data entries, and further investigation or data cleaning may be necessary to address them.

c)

```
bronx_data_clean <- bronx_data %>%
  select(-BOROUGH, -ADDRESS, -APARTMENT_NUMBER, -`EASE-MENT`, -Census_Tract_2020)%>%
  filter(!is.na(SALE_PRICE)) %>%
  mutate_all(~replace(., is.na(.), 0))

# Convert these columns to factors
categorical_columns <- sapply(bronx_data_clean, function(x) is.character(x) | is.logical(x) | is.factor
bronx_data_clean[categorical_columns] <- lapply(bronx_data_clean[categorical_columns], factor)

model <- lm(SALE_PRICE ~ ., data = bronx_data_clean)
summary(model)$adj.r.squared
```

```
## [1] 0.2521503
```

Adjusted R-squared of 0.2522 means that approximately 25% of the variance in SALE_PRICE is explained by the predictor variables included in the model. This indicates that there are important factors affecting SALE_PRICE that are not captured by the model.

```
# Split the dataset into two based on the median of YEAR_BUILT
median_year_built <- median(bronx_data_clean$YEAR_BUILT, na.rm = TRUE)
data_before_median <- bronx_data_clean %>% filter(YEAR_BUILT <= median_year_built)
data_after_median <- bronx_data_clean %>% filter(YEAR_BUILT > median_year_built)

model_before <- lm(SALE_PRICE ~ ., data = data_before_median)
model_after <- lm(SALE_PRICE ~ ., data = data_after_median)

# Summaries of the models
summary(model_before)$adj.r.squared
```

```
## [1] 0.2965673
```

```
summary(model_after)$adj.r.squared
```

```
## [1] 0.2945144
```

Splitting the data on YEAR_BUILT improved the predictive power of each model by about 17% on both models. This indicates that there are likely non-linear relationships among YEAR_BUILT and SALE_PRICE. There may also be similar relationships among other variables.

d)

```
# Drop rare classes and factor columns
bronx_data_clean <- bronx_data_clean %>%
  group_by(BUILDING_CLASS_CATEGORY) %>%
  filter(n() >= 100) %>%
  ungroup() %>%
  mutate(
    SALE_DATE = as.numeric(SALE_DATE - min(SALE_DATE))  # Days since the earliest date
  )

# Keep only the target column (BUILDING_CLASS_CATEGORY) as a factor
bronx_data_clean <- bronx_data_clean %>%
  mutate(BUILDING_CLASS_CATEGORY = factor(BUILDING_CLASS_CATEGORY)) %>%
  select(-where(is.factor), BUILDING_CLASS_CATEGORY)

# Step 2: Stratified Sampling
set.seed(42)

# Get indices for each class
classes <- levels(bronx_data_clean$BUILDING_CLASS_CATEGORY)
train_indices <- c()
test_indices <- c()
```

```r
for (class in classes) {
  class_indices <- which(bronx_data_clean$BUILDING_CLASS_CATEGORY == class)
  n_class_samples <- length(class_indices)

  n_train <- floor(0.8 * n_class_samples)

  # Randomly sample train and test indices for this class
  train_indices <- c(train_indices, sample(class_indices, size = n_train))
  test_indices <- c(test_indices, setdiff(class_indices, train_indices))
}

# Create train and test datasets
train <- bronx_data_clean[train_indices, ]
test <- bronx_data_clean[test_indices, ]

# Scale numeric columns
scaler <- preProcess(train, method = c("center", "scale"))
train <- predict(scaler, train)
test <- predict(scaler, test)

# Step 3: Prepare PyTorch Tensors
# Split into features (X) and target (Y)
x_train1 <- train %>% select(-BUILDING_CLASS_CATEGORY)
y_train1 <- as.integer(train$BUILDING_CLASS_CATEGORY)
x_test1 <- test %>% select(-BUILDING_CLASS_CATEGORY)
y_test1 <- as.integer(test$BUILDING_CLASS_CATEGORY)

# Convert to PyTorch tensors
x_train <- torch_tensor(as.matrix(x_train1), dtype = torch_float())
x_test <- torch_tensor(as.matrix(x_test1), dtype = torch_float())
y_train <- torch_tensor(as.array(y_train1), dtype = torch_long())
y_test <- torch_tensor(as.array(y_test1), dtype = torch_long())
```

```r
train_nn <- function(x_train, y_train, learning_rate = 0.001) {
  model <- nn_module(
    initialize = function() {
      self$fc1 <- nn_linear(ncol(x_train), 64)
      self$dropout1 <- nn_dropout(0.3)
      self$fc2 <- nn_linear(64, 128)
      self$dropout2 <- nn_dropout(0.3)
      self$fc3 <- nn_linear(128, 64)
      self$fc4 <- nn_linear(64, 33)
      self$batch_norm1 <- nn_batch_norm1d(64)
      self$batch_norm2 <- nn_batch_norm1d(128)
    },

    forward = function(x) {
      x %>%
        self$fc1() %>%
        self$batch_norm1() %>%
        nnf_relu() %>%
        self$dropout1() %>%
        self$fc2() %>%
```

```
      self$batch_norm2() %>%
      nnf_relu() %>%
      self$dropout2() %>%
      self$fc3() %>%
      nnf_relu() %>%
      self$fc4() %>%
      nnf_log_softmax(dim = 1)
  }
)

# Instantiate the model
model <- model()

# Define loss function and optimizer
criterion <- nn_cross_entropy_loss()
optimizer <- optim_adam(model$parameters, lr = learning_rate)

# Training loop with early stopping
best_loss <- Inf
patience_counter <- 0
max_patience <- 10
min_lr <- 1e-6

for (epoch in 1:100) {
  model$train()
  optimizer$zero_grad()

  # Forward pass
  output <- model(x_train)
  loss <- criterion(output, y_train)

  # Backward pass and optimization
  loss$backward()
  optimizer$step()

  # Manual learning rate adjustment if loss plateaus
  if (loss$item() >= best_loss) {
    patience_counter <- patience_counter + 1
    if (patience_counter >= 5) {
      current_lr <- optimizer$param_groups[[1]]$lr
      if (current_lr > min_lr) {
        new_lr <- current_lr * 0.5
        optimizer$param_groups[[1]]$lr <- new_lr
        cat(sprintf("Reducing learning rate to %f\n", new_lr))
      }
      patience_counter <- 0
    }
  } else {
    best_loss <- loss$item()
    patience_counter <- 0
  }

  # Early stopping check
```

```
    if (optimizer$param_groups[[1]]$lr <= min_lr && patience_counter >= max_patience) {
      cat(sprintf("Early stopping at epoch %d\n", epoch))
      break
    }

    # Print progress
    if (epoch %% 10 == 0) {
      cat(sprintf("Epoch [%d/100], Loss: %.4f, LR: %.6f\n",
                  epoch, loss$item(), optimizer$param_groups[[1]]$lr))
    }
  }

  return(model)
}
eval_model <- function(model, x_test, y_test){
  # Evaluate the model
  model$eval()
  with_no_grad({
    output <- model(x_test)
    predicted_probs <- output$exp()
    max_result <- predicted_probs$max(dim = 2)
    predicted_classes <- max_result[[2]]

    # Calculate accuracy
    correct <- sum(predicted_classes == y_test)
    accuracy <- correct / length(y_test)

    # Ensure accuracy is a numeric value (not a tensor)
    accuracy <- as.numeric(accuracy)
  })

  return(accuracy)
}
```

```
nn_model <- train_nn(x_train, y_train)
```

```
## Epoch [10/100], Loss: 3.2882, LR: 0.001000
## Epoch [20/100], Loss: 2.8904, LR: 0.001000
## Epoch [30/100], Loss: 2.5377, LR: 0.001000
## Epoch [40/100], Loss: 2.2136, LR: 0.001000
## Epoch [50/100], Loss: 2.0137, LR: 0.001000
## Epoch [60/100], Loss: 1.8751, LR: 0.001000
## Epoch [70/100], Loss: 1.7243, LR: 0.001000
## Epoch [80/100], Loss: 1.6242, LR: 0.001000
## Epoch [90/100], Loss: 1.5768, LR: 0.001000
## Epoch [100/100], Loss: 1.4717, LR: 0.001000
```

```
eval_model(nn_model, x_test, y_test)
```

```
## [1] 0.4808008
```

```r
rf_model <- randomForest(BUILDING_CLASS_CATEGORY ~ ., data = train, importance = TRUE, ntree = 500)

# Print the model accuracy
rf_predictions <- predict(rf_model, newdata = test)
confusion_matrix <- table(Predicted = rf_predictions, Actual = test$BUILDING_CLASS_CATEGORY)
accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)
cat("Accuracy: ", accuracy, "\n")
```

```
## Accuracy:  0.9734017
```

The data preprocessing involved removing rare classes, transforming SALE_DATE to days since the earliest sale, and converting BUILDING_CLASS_CATEGORY into a factor. Stratified sampling split the data into training and test sets, and numeric columns were scaled. A neural network model with fully connected layers, dropout, batch normalization, and early stopping was trained, with a learning rate schedule to avoid overfitting. A random forest model with 500 trees and feature importance calculation was also trained. The neural network achieved 43.6% accuracy, while the random forest performed better with 97.34%, suggesting the NN needs further tuning, but the NN already took 3+ hours to get actual predictions so I'm just happy it works.

2.

```r
# apply same preprocessing to all data
colnames(data) <- gsub(" ", "_", colnames(data))

# Clean the data: Replace "- 0" with 0, and remove commas
data <- data %>%
  mutate(
    LAND_SQUARE_FEET = as.numeric(gsub(",", "", gsub("-\\s0", "0", LAND_SQUARE_FEET))),
    GROSS_SQUARE_FEET = as.numeric(gsub(",", "", gsub("-\\s0", "0", GROSS_SQUARE_FEET))),
    RESIDENTIAL_UNITS = as.numeric(gsub(",", "", gsub("-\\s0", "0", RESIDENTIAL_UNITS))),
    COMMERCIAL_UNITS = as.numeric(gsub(",", "", gsub("-\\s0", "0", COMMERCIAL_UNITS))),
    Latitude = as.numeric(gsub(",", "", gsub("-\\s0", "0", Latitude))),
    Longitude = as.numeric(gsub(",", "", gsub("-\\s0", "0", Longitude))),
    TOTAL_UNITS = as.numeric(gsub(",", "", gsub("-\\s0", "0", TOTAL_UNITS))),
    YEAR_BUILT = as.numeric(gsub(",", "", gsub("-\\s0", "0", YEAR_BUILT))),
    SALE_PRICE = as.numeric(gsub(",", "", gsub("-\\s0", "0", SALE_PRICE))),
    SALE_DATE = as.Date(SALE_DATE, format = "%m/%d/%Y")
  )

data <- data %>%
  select(-BOROUGH, -ADDRESS, -APARTMENT_NUMBER, -`EASE-MENT`, -Census_Tract_2020)%>%
  filter(!is.na(SALE_PRICE)) %>%
  mutate_all(~replace(., is.na(.), 0))

# Convert these columns to factors
categorical_columns <- sapply(data, function(x) is.character(x) | is.logical(x) | is.factor(x))
data[categorical_columns] <- lapply(data[categorical_columns], factor)
```

```r
# Apply model based on median year split
# Retrain models without factors that may have new levels
data_before_median <- data_before_median %>% select(-NEIGHBORHOOD, -NTA, -NTA_Code)
data_after_median <- data_after_median %>% select(-NEIGHBORHOOD, -NTA, -NTA_Code)
```

```r
model_before <- lm(SALE_PRICE ~ ., data = data_before_median)
model_after <- lm(SALE_PRICE ~ ., data = data_after_median)

# Summaries of the models
summary(model_before)$adj.r.squared
```

```
## [1] 0.2851321
```

```r
summary(model_after)$adj.r.squared
```

```
## [1] 0.2636392
```

```r
# Split the dataset into two based on the median of YEAR_BUILT
new_data_before_median <- data %>%
  filter(YEAR_BUILT <= median_year_built) %>%
  select(-NEIGHBORHOOD) %>%
  filter(BUILDING_CLASS_CATEGORY %in% data_before_median$BUILDING_CLASS_CATEGORY) %>%
  filter(TAX_CLASS_AS_OF_FINAL_ROLL %in% data_before_median$TAX_CLASS_AS_OF_FINAL_ROLL) %>%
  filter(BUILDING_CLASS_AS_OF_FINAL_ROLL %in% data_before_median$BUILDING_CLASS_AS_OF_FINAL_ROLL) %>%
  filter(BUILDING_CLASS_AT_TIME_OF_SALE %in% data_before_median$BUILDING_CLASS_AT_TIME_OF_SALE)
new_data_after_median <- data %>%
  filter(YEAR_BUILT > median_year_built) %>%
  select(-NEIGHBORHOOD) %>%
  filter(BUILDING_CLASS_CATEGORY %in% data_after_median$BUILDING_CLASS_CATEGORY) %>%
  filter(TAX_CLASS_AS_OF_FINAL_ROLL %in% data_after_median$TAX_CLASS_AS_OF_FINAL_ROLL) %>%
  filter(BUILDING_CLASS_AS_OF_FINAL_ROLL %in% data_after_median$BUILDING_CLASS_AS_OF_FINAL_ROLL) %>%
  filter(BUILDING_CLASS_AT_TIME_OF_SALE %in% data_after_median$BUILDING_CLASS_AT_TIME_OF_SALE)

# Predictions for the new dataset
pred_before <- predict(model_before, new_data_before_median)
```

```
## Warning in predict.lm(model_before, new_data_before_median): prediction from
## rank-deficient fit; attr(*, "non-estim") has doubtful cases
```

```r
pred_after <- predict(model_after, new_data_after_median)
```

```
## Warning in predict.lm(model_after, new_data_after_median): prediction from
## rank-deficient fit; attr(*, "non-estim") has doubtful cases
```

```r
# Calculate adjusted R-squared
ss_total_before <- sum((new_data_before_median$SALE_PRICE - mean(new_data_before_median$SALE_PRICE))^2)
ss_res_before <- sum((new_data_before_median$SALE_PRICE - pred_before)^2)
n_before <- nrow(new_data_before_median)
p_before <- length(model_before$coefficients)
1 - ((ss_res_before / (n_before - p_before)) / (ss_total_before / (n_before - 1)))
```

```
## [1] -0.04081149
```

```r
# Calculate adjusted R-squared
ss_total_after <- sum((new_data_after_median$SALE_PRICE - mean(new_data_after_median$SALE_PRICE))^2)
ss_res_after <- sum((new_data_after_median$SALE_PRICE - pred_after)^2)
n_after <- nrow(new_data_after_median)
p_after <- length(model_after$coefficients)
1 - ((ss_res_after / (n_after - p_after)) / (ss_total_after / (n_after - 1)))
```

```
## [1] -22.50324
```

Here I apply the median split regression models to the entire dataset. They both perform worse than the null model. This is likely due to multicolinearity, as seen in the warning. Additionally, Many values will be outside the scope of the original training data, such as latitude and longitude. This will lead to poor or no generalization. If I were to try to fix this, I would use methods to check for multicolinearity and remove colinear columns from the model.

```r
# drop classes not in bronx
# Drop rare classes and factor columns
data <- data %>%
  filter(BUILDING_CLASS_CATEGORY %in% classes) %>%
  mutate(
    SALE_DATE = as.numeric(SALE_DATE - min(SALE_DATE))  # Days since the earliest date
  )

# Keep only the target column (BUILDING_CLASS_CATEGORY) as a factor
data <- data %>%
  mutate(BUILDING_CLASS_CATEGORY = factor(BUILDING_CLASS_CATEGORY)) %>%
  select(-where(is.factor), BUILDING_CLASS_CATEGORY)

data <- predict(scaler, data)

X <- data %>% select(-BUILDING_CLASS_CATEGORY)
Y <- as.integer(data$BUILDING_CLASS_CATEGORY)
x <- torch_tensor(as.matrix(X), dtype = torch_float())
y <- torch_tensor(as.array(Y), dtype = torch_long())
```

```r
# apply models
eval_model(nn_model, x, y)
```

```
## [1] 0.3208344
```

```r
rf_predictions <- predict(rf_model, newdata = data)
confusion_matrix <- table(Predicted = rf_predictions, Actual = data$BUILDING_CLASS_CATEGORY)
accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)
cat("Accuracy: ", accuracy, "\n")
```

```
## Accuracy:  0.7883349
```

The models show a significant drop in performance when applied to data from all five boroughs compared to the Bronx only dataset, with the neural network's accuracy decreasing from 43.6% to 20.32% and the random forest's from 97.34% to 78.83%. This suggests that the models do not generalize well to the larger, more diverse dataset. A likely reason is that some items are out of bounds of the original data, and may include different patterns or distributions not represented by Bronx.