RestaurantRevenue

Alexia Wells

2024-12-15

Contents

Questions

Which machine learning methods did you implement?

PCA, Penalized Linear Regression, Light-GBM Boosted Trees, Random Forest, and a Neural Network.

Discuss the key contribution of each method to your analysis. If a method didn't contribute, discuss why it didn't. A sentence or two for each method is plenty.

PCA: Didn't end up having a meaningful contribution when included in the models, actually made the root mean square error increase, but was beneficial in the EDA process. It gave more insight into which components had the potential to be important.

Penalized Linear Regression: Contributed in the sense that it was a good baseline model to compare results since it is fairly interpretable. It doesn't capture nonlinear relationships well and this data seemed to be nonlinear.

LightGBM: This model helped me realize the importance of removing the outliers in the dataset. This type of model is actually known for being sensitive to noise.

Random Forest: This was the highest performing model. The recipes for this model became most successful when I separated Date into multiple features like day of the week, month, year, and decimal.

Neural Network: Didn't contribute considering this model had a drastically higher RMSE. This may be because there is insufficient data for the neural net to perform well (only 137 entries from train), so the models that are more simple outperformed this one.

Did all methods support your conclusions or did some provide conflicting results? If so they provided conflicting results, how did you reconcile the differences?

Overall, there were more consistencies than conflicting results. All methods supported the conclusion of transforming revenue by taking the log. Another consistency was the importance of removing outliers from the training data to decrease model RMSE. Moreover, as I explored different recipes and model combinations, I learned which recipe steps had a tendency to work across all methods. Said steps were: normalization, dummy variable implementation, removal of zero variance, handling unseen and unknown levels.

To reconcile conflicting results, I focused on understanding the data and models I was using. For example, while I found steps that could be used consistently, there was no single recipe that could applied to each model. At first this was frustrating, but as I spent more time understanding the requirements and assumptions that came with implenting each model, I understood why that was. Also, I had to acknowledge the limitations that came along with using a smaller dataset. This helped me prioritize the simple/more interpretable models, instead of ones that were more complex.

Code

EDA

How the Data Looks

There are 43 columns and 137 total entries.

Data Cleaning

There is no duplicate, missing, or near zero variance present in the data. I did need to convert certain variable types for train and test.

```
# Checking for duplicates
any(duplicated(train))
```

```
## [1] FALSE
```

```
# Check if there is near zero variance
# nearZeroVar(train, saveMetrics = TRUE)

# There is no missing data
any(is.na(train))
```

```
## [1] FALSE
```

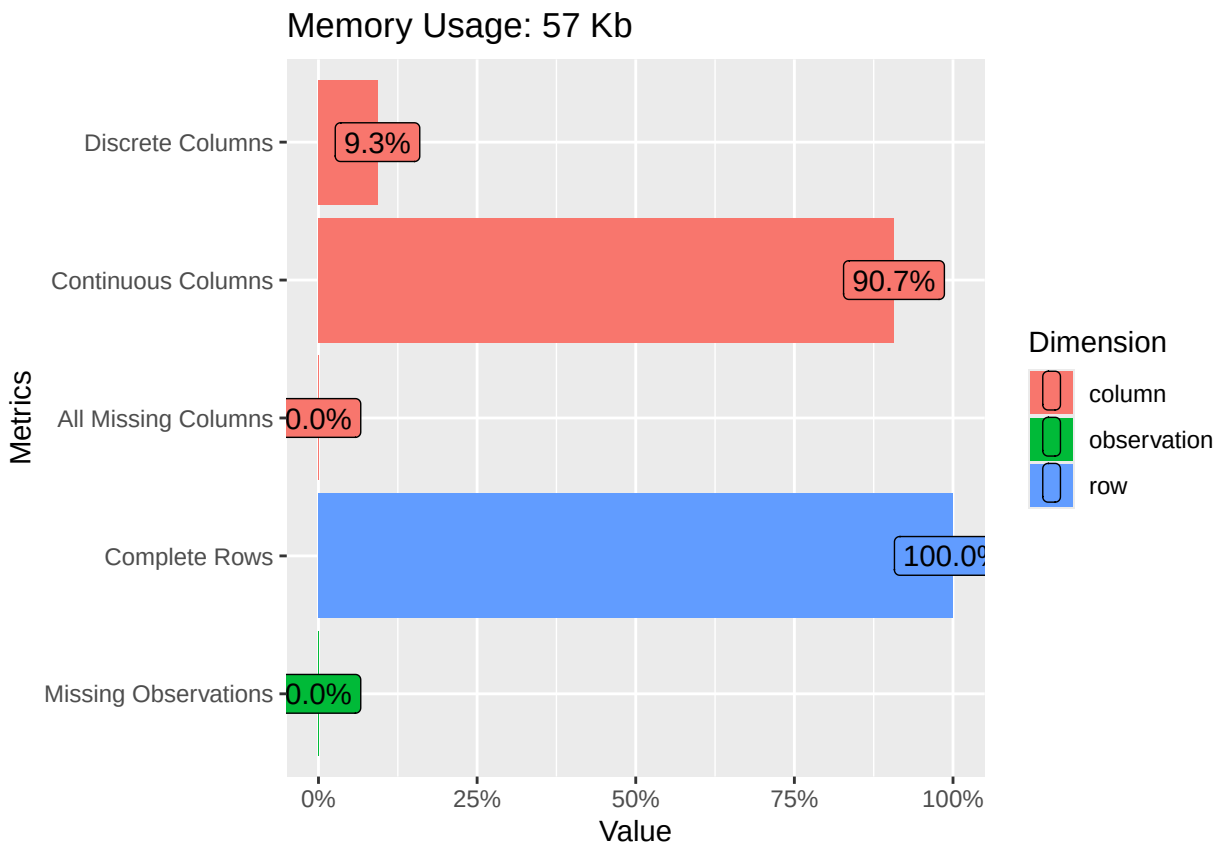
```
# Check variable types
# str(train)

# Convert variable types
train <- train |>
  mutate(Date = as.Date(`Open Date`, tryFormats = c("%m/%d/%Y")),
         across(c(City, "City Group", Type), as.factor)) |>
  dplyr::select(-`Open Date`)

# Do the same for test
test <- test |>
  mutate(Date = as.Date(`Open Date`, tryFormats = c("%m/%d/%Y")),
         across(c(City, "City Group", Type), as.factor)) |>
  dplyr::select(-`Open Date`)
```

Graphs and Tables

Intro Plot

There is no missing data!

```
train |>
  plot_intro()
```



Bar Plots

```
train |>
  plot_bar(title = "Several Bar Plots",
           ggtheme = theme_minimal())
```

City	n
İstanbul	34087
Ankara	8720
İzmir	6465
Antalya	5911
Kocaeli	4364
Mersin	2735

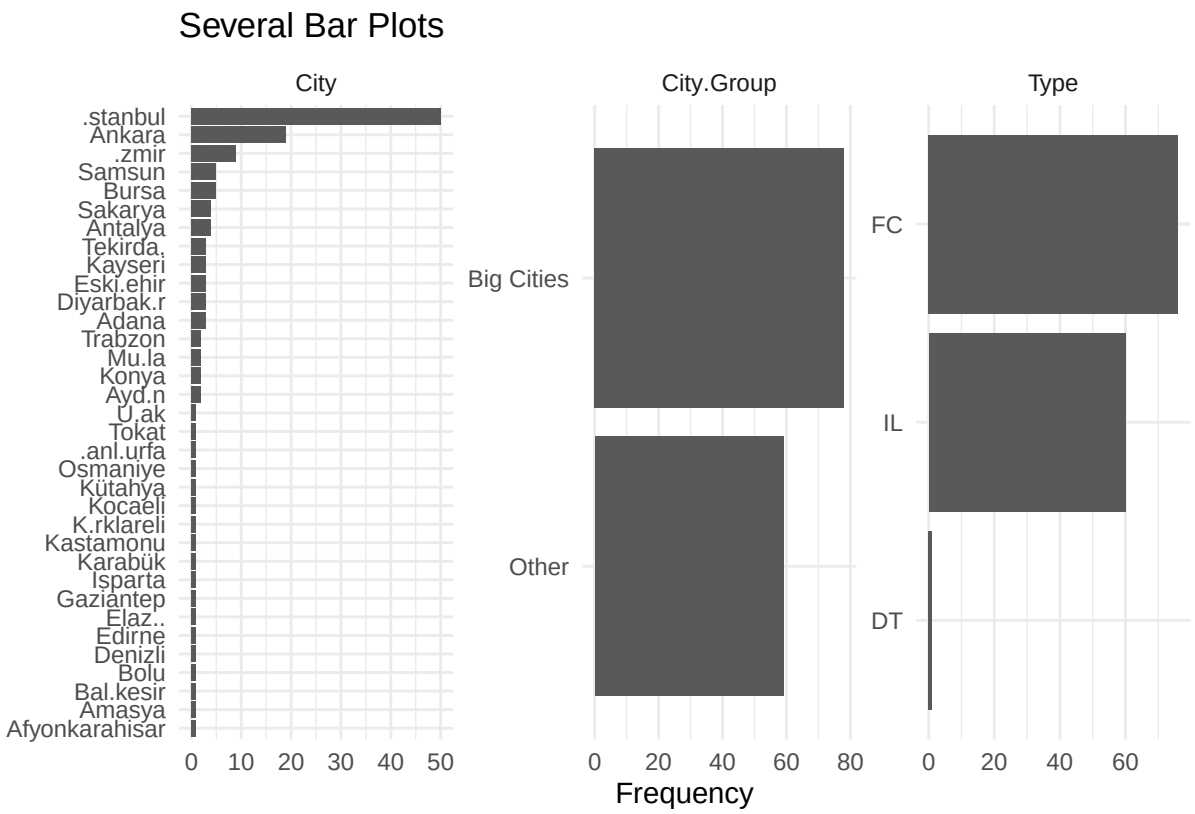
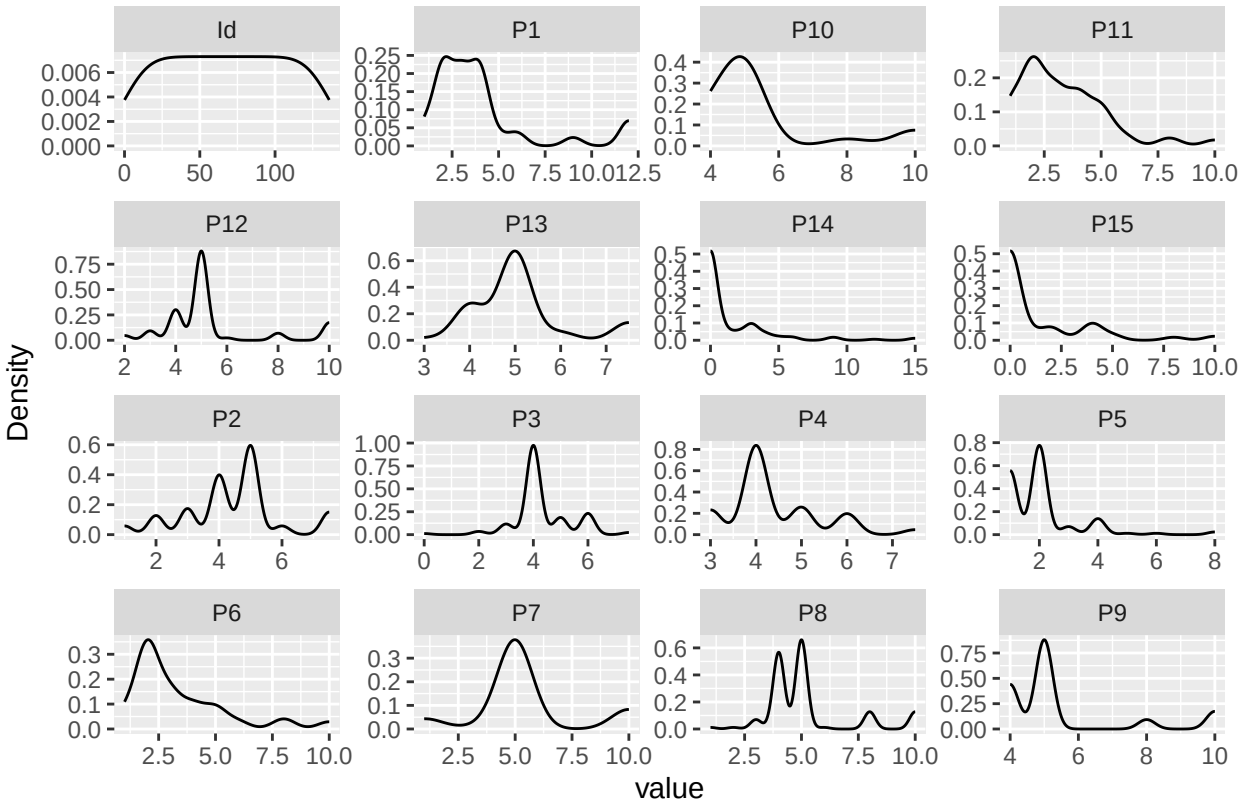


Table of City Counts for Test Data

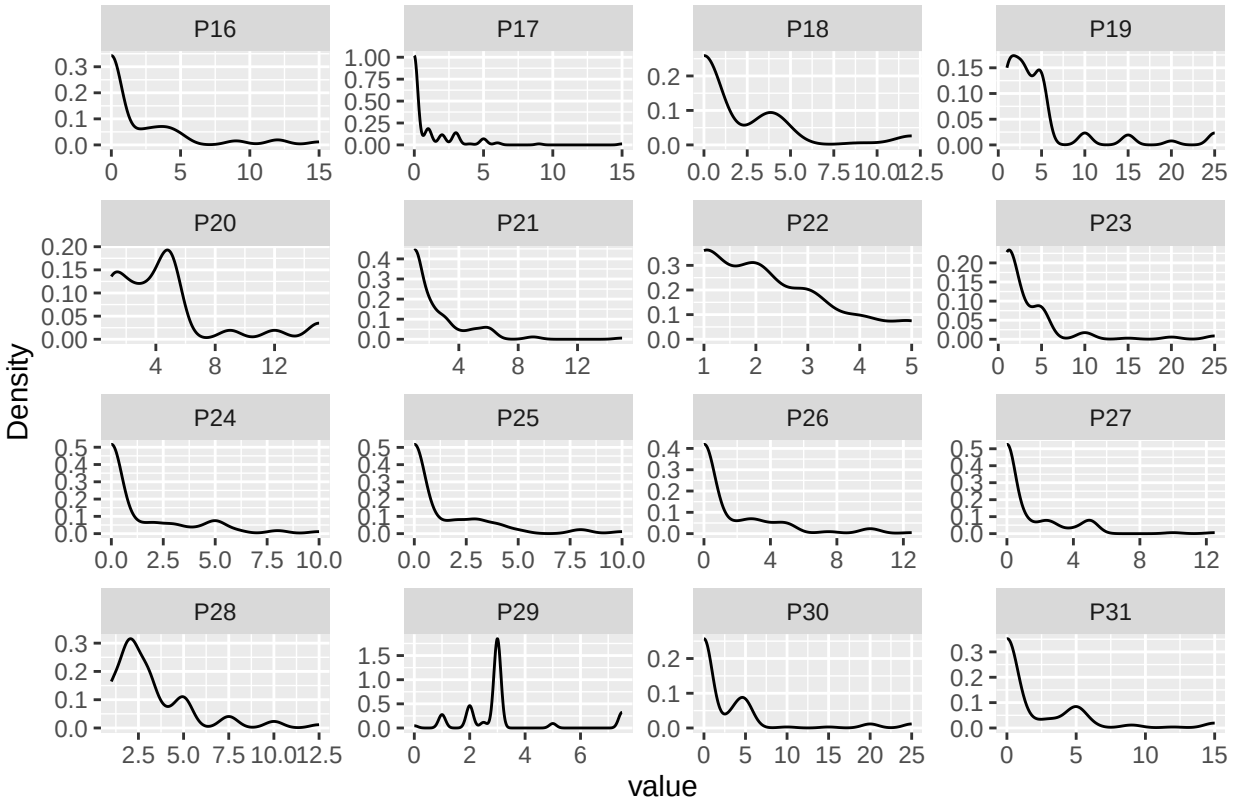
```
test %>%
  count(City) %>%  # Count occurrences of each City
  arrange(desc(n)) %>%
  head() %>%
  kbl() %>%  # Create a kable object
  kable_classic(full_width = F, html_font = "Cambria")
```

Density Plot

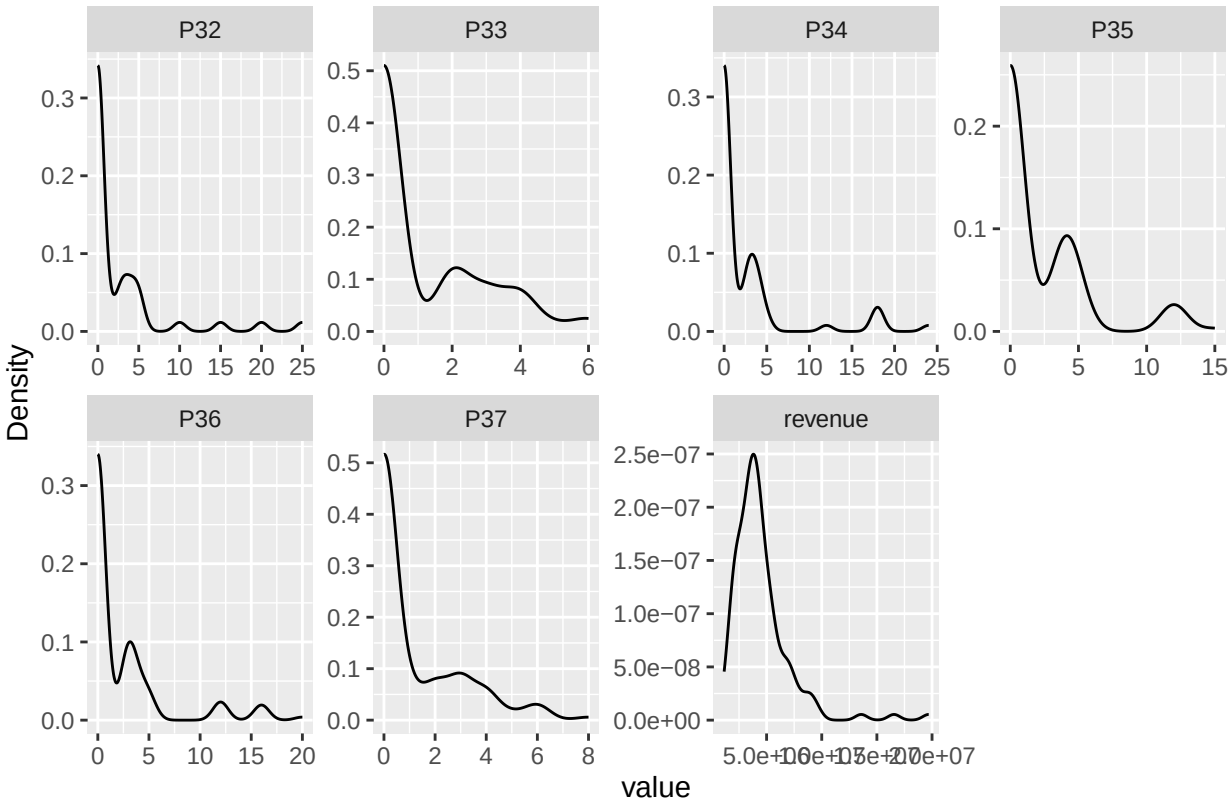
```
train |>
  plot_density()
```


Id
P1
P10
P11
P12
P13
P14
P15
P2
P3
P4
P5
P6
P7
P8
P9
Density
value

Page 1


P16
P17
P18
P19
P20
P21
P22
P23
P24
P25
P26
P27
P28
P29
P30
P31
Density
value

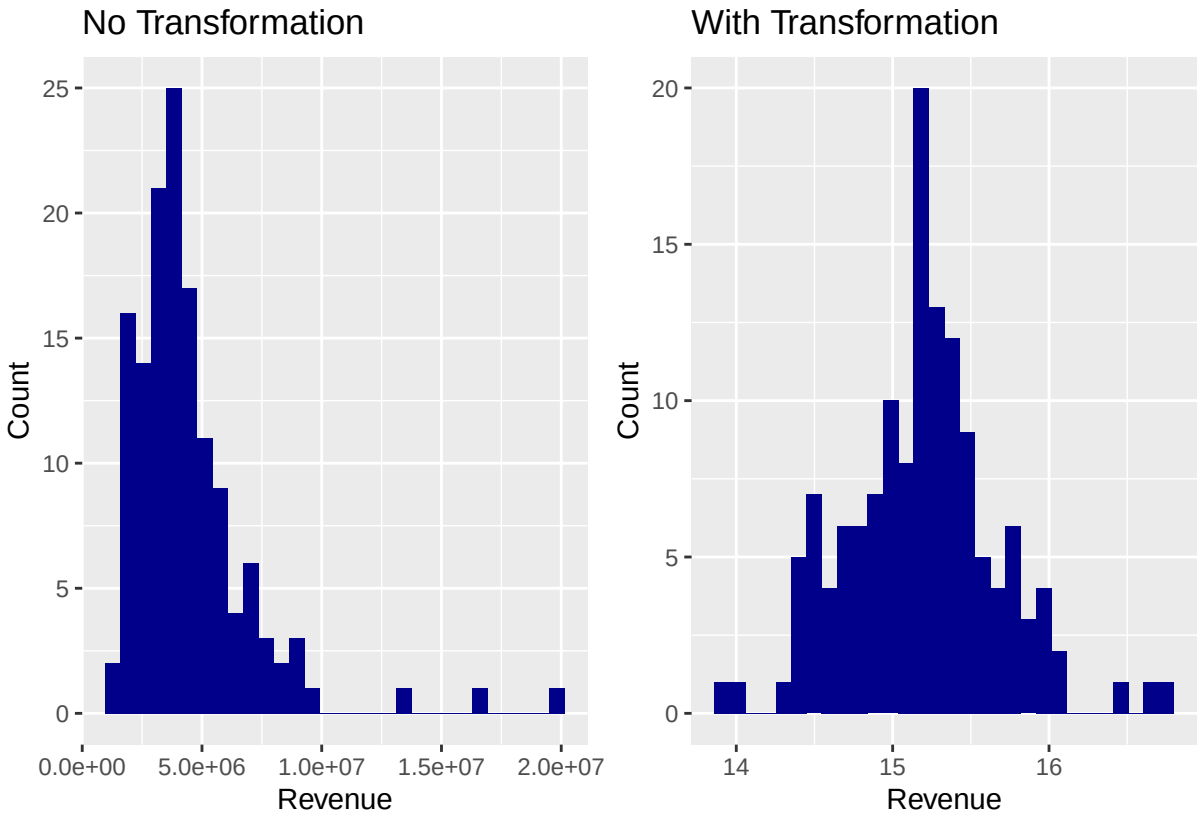
Page 2



Revenue Histograms

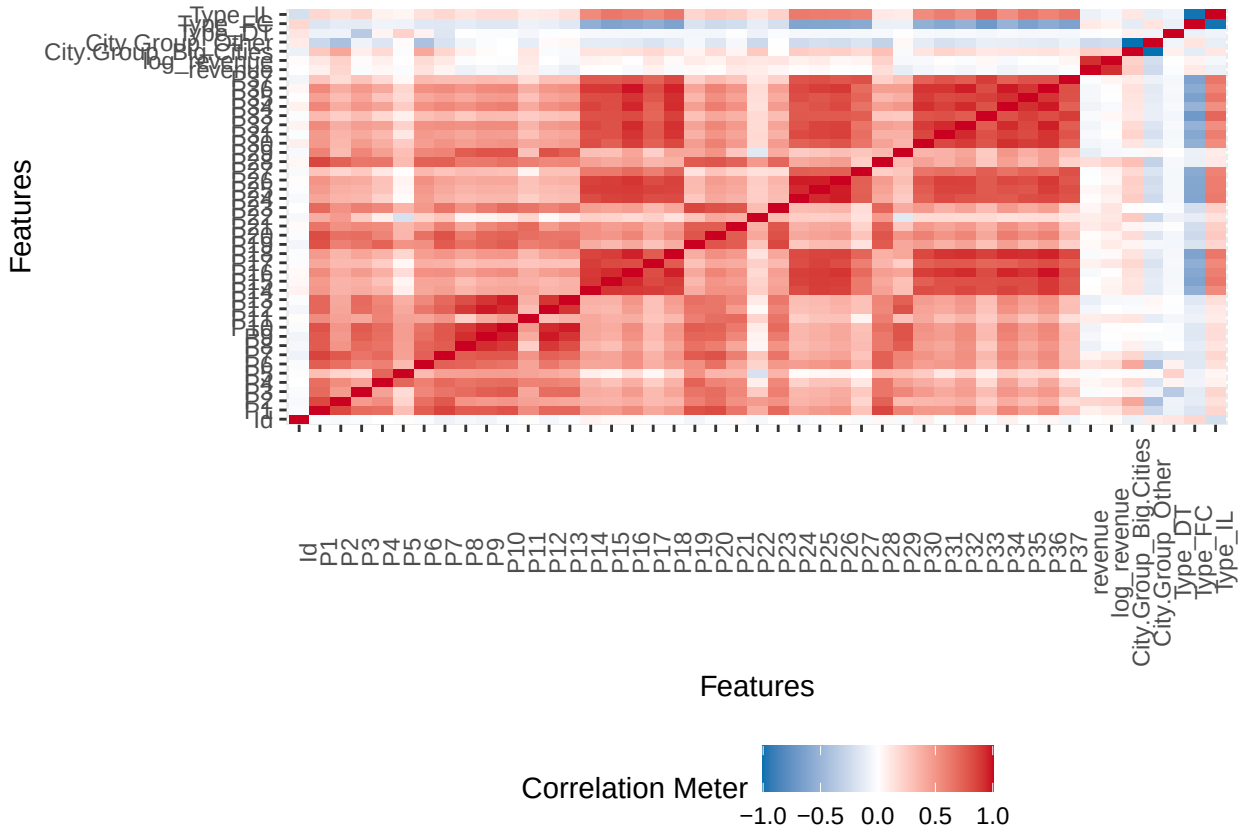
The **Feature Engineering** section, below, is where I discovered the importance of transforming Revenue. I included the histograms here to show how much of a difference said transformation makes. On top of that, it is important to note that outliers are present in the data. It may be necessary to extract them in the model workflow.

```
# Without transformation
no_transformation <- ggplot(data = train, mapping = aes(x=revenue)) +
  geom_histogram(fill = "darkblue") + xlab("Revenue") + ylab("Count") + ggtitle("No Transformation")

# With transformation
train$log_revenue <- log(train$revenue)
transformation <- ggplot(data = train, mapping = aes(x=log_revenue)) +
  geom_histogram(fill = "darkblue") + xlab("Revenue") + ylab("Count") + ggtitle("With Transformation")

# Output both histograms together
no_transformation + transformation
```



Correlations

```
train |>
  plot_correlation()
```



PCA Table

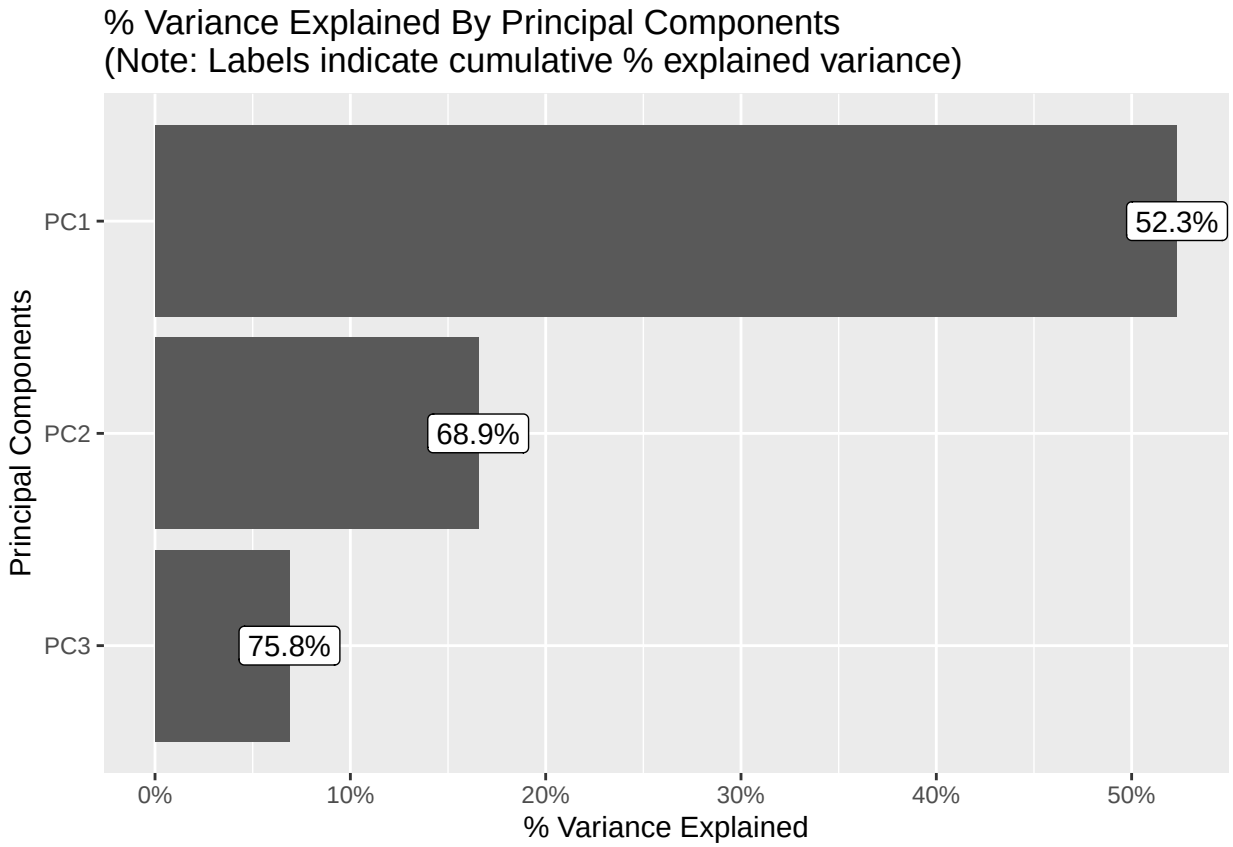
```
# Only including numeric training data for PCA
numeric_train <- train |>
  dplyr::select(-c(Id, City, `City Group`, Type, Date, -revenue))

train.pca <- prcomp(numeric_train, center = TRUE,scale. = TRUE)
summary(train.pca)
```

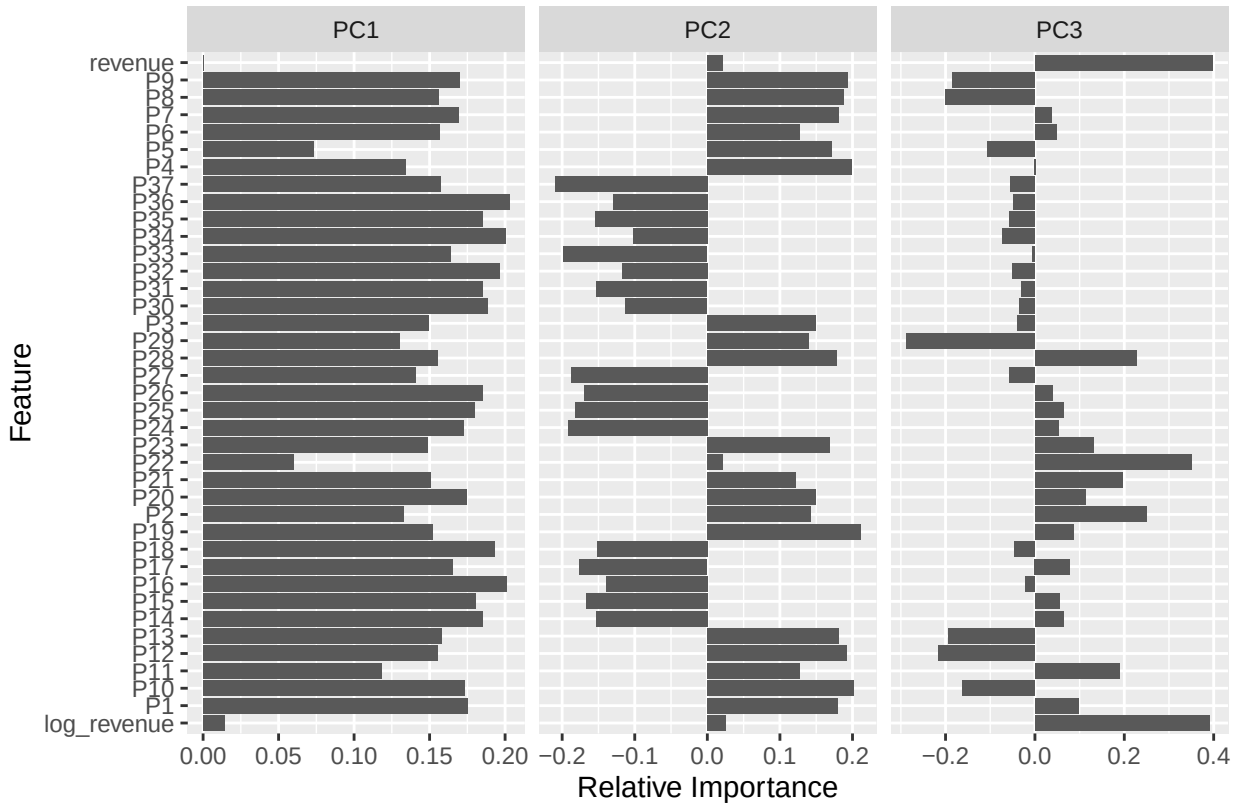
```
## Importance of components:
##                           PC1    PC2     PC3     PC4     PC5     PC6     PC7
## Standard deviation     4.5173 2.5401 1.63845 1.30557 1.20171 0.99424 0.80979
## Proportion of Variance 0.5232 0.1654 0.06883 0.04371 0.03703 0.02535 0.01681
## Cumulative Proportion  0.5232 0.6887 0.75751 0.80122 0.83825 0.86359 0.88041
##                           PC8     PC9    PC10    PC11    PC12    PC13    PC14
## Standard deviation     0.72931 0.68454 0.63276  0.6056 0.56236 0.53783 0.51960
## Proportion of Variance 0.01364 0.01202 0.01027  0.0094 0.00811 0.00742 0.00692
## Cumulative Proportion  0.89405 0.90606 0.91633  0.9257 0.93384 0.94126 0.94818
##                           PC15    PC16   PC17    PC18   PC19    PC20    PC21
## Standard deviation     0.48423 0.46572 0.4462 0.41030 0.3952 0.37506 0.34892
## Proportion of Variance 0.00601 0.00556 0.0051 0.00432 0.0040 0.00361 0.00312
## Cumulative Proportion  0.95419 0.95975 0.9649 0.96917 0.9732 0.97679 0.97991
##                           PC22    PC23    PC24    PC25   PC26    PC27    PC28
## Standard deviation     0.33108 0.31178 0.28973 0.28568 0.2651 0.24741 0.24126
## Proportion of Variance 0.00281 0.00249 0.00215 0.00209 0.0018 0.00157 0.00149
## Cumulative Proportion  0.98272 0.98521 0.98736 0.98946 0.9913 0.99283 0.99432
##                           PC29    PC30   PC31    PC32    PC33    PC34   PC35
```

```
## Standard deviation     0.20223 0.19881 0.1872 0.17365 0.13645 0.12894 0.1253
## Proportion of Variance 0.00105 0.00101 0.0009 0.00077 0.00048 0.00043 0.0004
## Cumulative Proportion  0.99537 0.99638 0.9973 0.99805 0.99853 0.99896 0.9994
##                           PC36    PC37    PC38    PC39
## Standard deviation     0.10443 0.08145 0.06403 0.05729
## Proportion of Variance 0.00028 0.00017 0.00011 0.00008
## Cumulative Proportion  0.99964 0.99981 0.99992 1.00000
```

PCA Visualization using DataExplorer

```
numeric_train |>
  plot_prcomp()
```



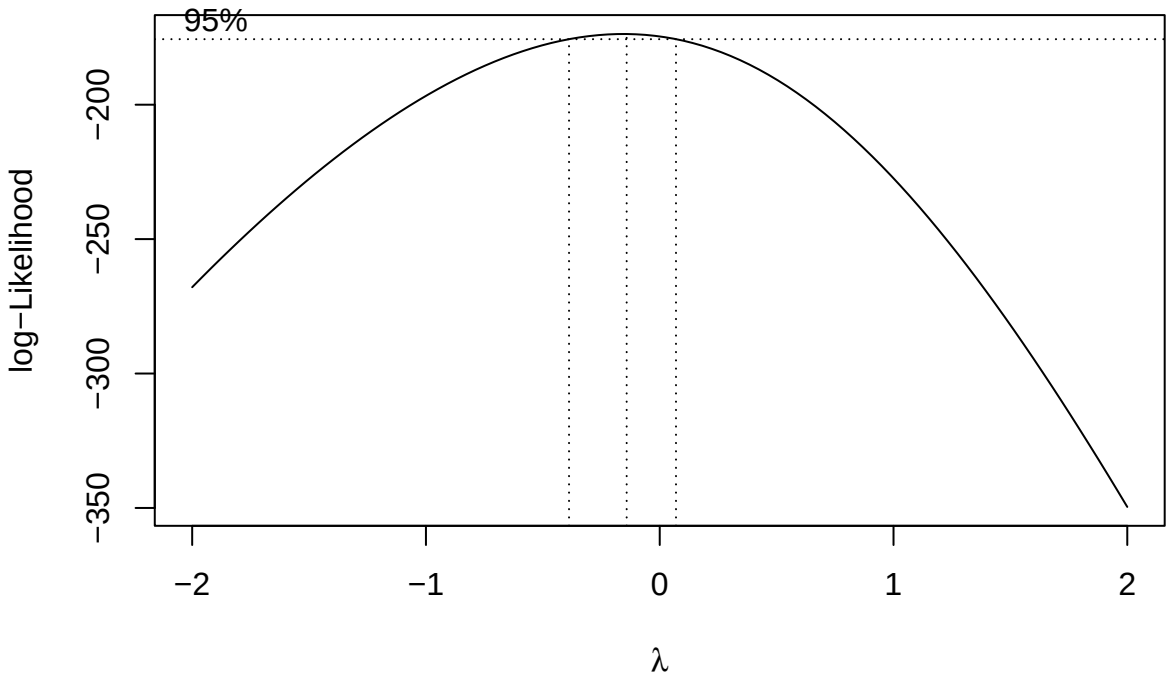


Feature Engineering

Transformations

While exploring the data, it became apparent that revenue had a right-skewed distribution. Therefore, I figured it would be a good idea. Considering that lambda is -0.1414141, the proper transformation will be log.

```
# 1. Transformations: changing the value of a single column log(), sqrt(), categories from numeric, sca
train_noID <- train |>
  dplyr::select(-Id, -log_revenue)

simple.lm <- lm(revenue ~ ., data = train_noID)
boxcox_results <-boxcox(simple.lm)
```




```
lambda <- boxcox_results$x[which.max(boxcox_results$y)]
lambda
```

```
## [1] -0.1414141
```

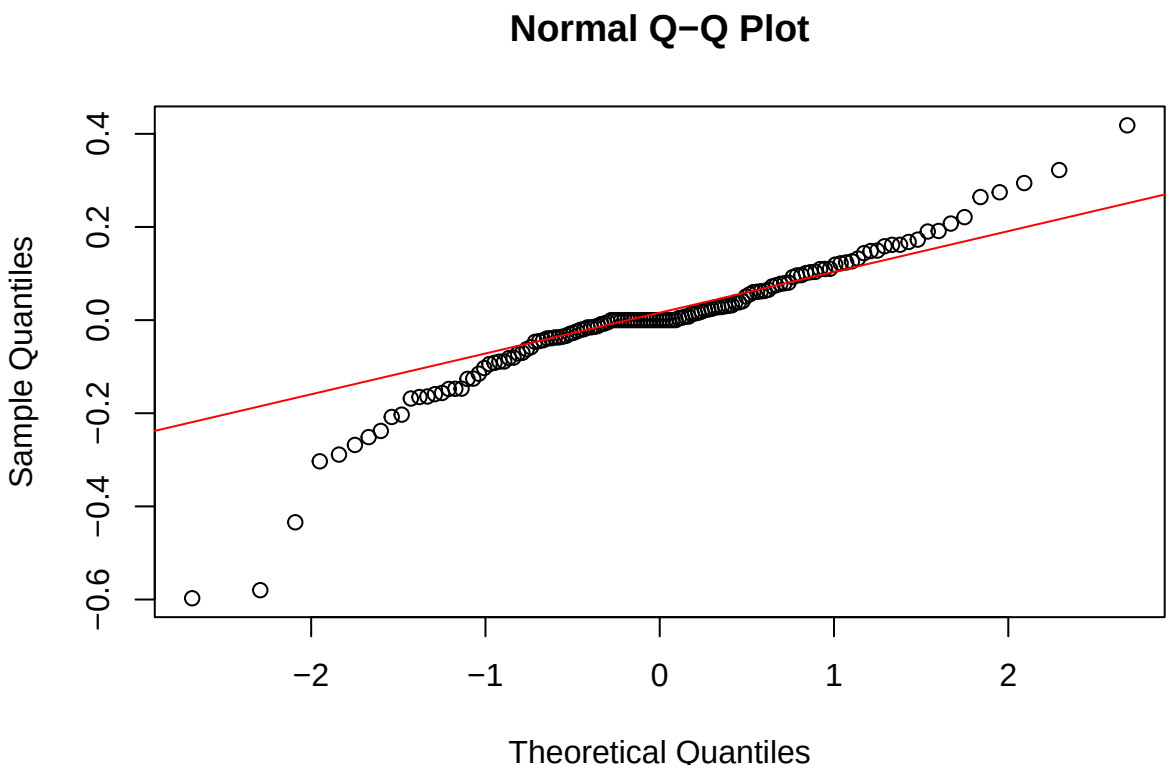
Model Assumptions

Linearity is met with the log transformation, this can be seen in the **Revenue Histograms**. The other assumptions are below.

Normality

Looking pretty good, there are outliers present, but these can be removed later

```
# QQplot with transformation
trans.simple.lm <- lm(log_revenue ~ ., data = train)
qqnorm(residuals(trans.simple.lm))
qqline(residuals(trans.simple.lm), col = "red")
```

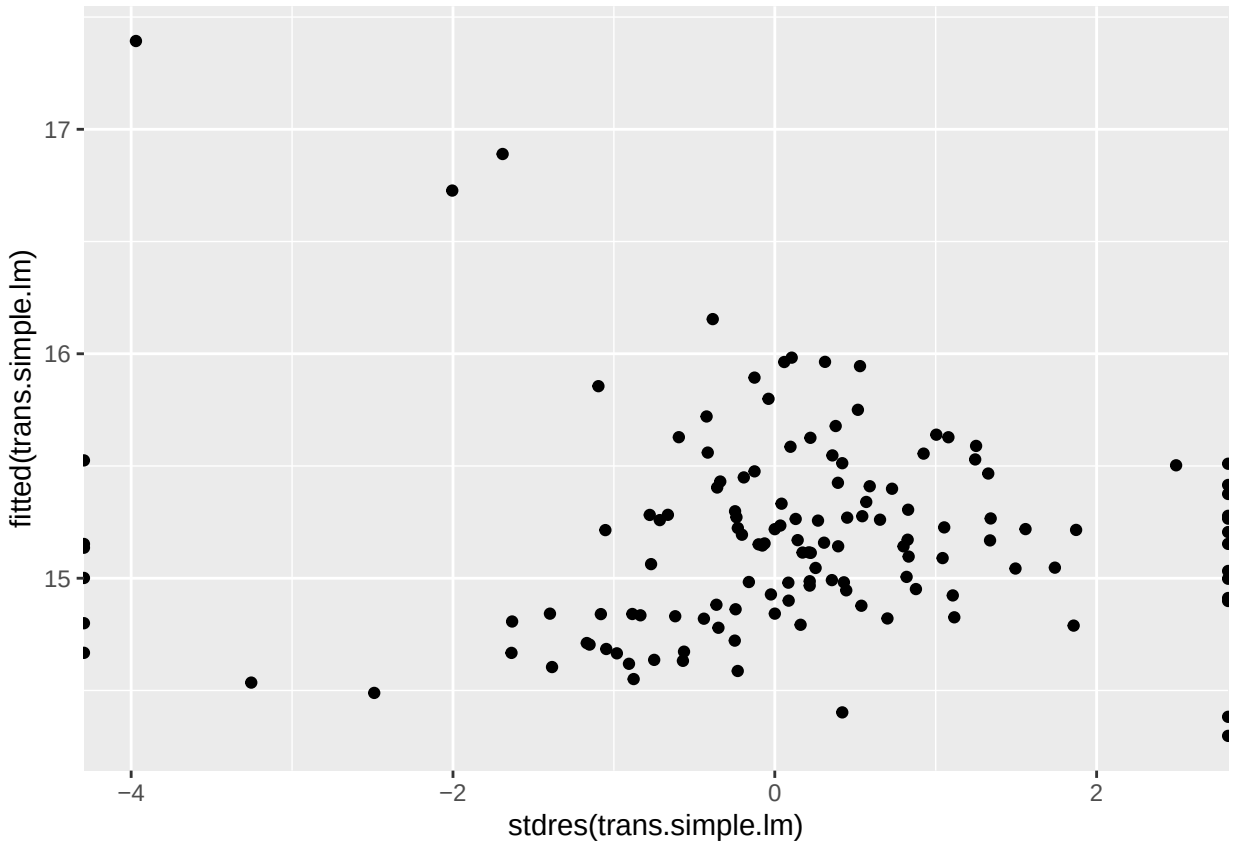


Equal variance

Not perfect, but good enough for our purposes to continue.

```
ggplot(data = train, mapping = aes(x = stdres(trans.simple.lm), y = fitted(trans.simple.lm))) + geom_po
```

```
## Warning in sqrt((n - p - sr^2)/(n - p - 1)): NaNs produced
```



Check for Multicollinearity

Multicollinearity is certainly present considering the error we are given, "there are aliased coefficients in the model."

```
# Using same lm from earlier
# VIF(trans.simple.lm)
```

Outlier Removal

```
# Take the log revenue
no_outliers <- train_noID
no_outliers$revenue <- log(no_outliers$revenue)

# Removing outliers post transformation, does this improve the scores?
# Calculate IQR bounds
Q1 <- quantile(no_outliers$revenue, 0.25)
Q3 <- quantile(no_outliers$revenue, 0.75)
IQR <- Q3 - Q1
lower_bound <- Q1 - 1.5 * IQR
upper_bound <- Q3 + 1.5 * IQR

# Get rid of outliers
no_outliers <- no_outliers |>
  mutate(across(revenue, ~ pmin(upper_bound, pmax(lower_bound, .))))

# Trying to keep the results the same
```

```r
set.seed(123)
```

Models

Model 1 - Penalized Linear Regression

```r
# 1869141.90811
my_recipe <- recipe(revenue ~ ., data = no_outliers) %>%
  step_date(Date, features = "year") %>% # Extracts features like 'year' from Date
  step_unknown(all_nominal_predictors()) %>% # Handles unknown levels in categorical predictors
  step_novel(all_nominal_predictors()) %>% # Handles unseen levels
  step_dummy(all_nominal_predictors()) %>% # Converts categories to dummy variables
  step_zv(all_predictors()) %>% # Removes zero-variance predictors
  step_nzv(all_predictors()) %>% # Removes near-zero-variance predictors
  step_corr(all_numeric_predictors(), threshold = 0.70) %>% # Removes highly correlated numeric predict
  step_normalize(all_numeric_predictors()) %>% # Normalizes numeric predictors
  step_rm(Date) # Removes the original 'Date' column

# 1837562.77602, here I increased the threshold for removing correlating variables
my_recipe <- recipe(revenue ~ ., data = no_outliers) %>%
  step_date(Date, features = "year") %>% # Extracts features like 'year' from Date
  step_unknown(all_nominal_predictors()) %>% # Handles unknown levels in categorical predictors
  step_novel(all_nominal_predictors()) %>% # Handles unseen levels
  step_dummy(all_nominal_predictors()) %>% # Converts categories to dummy variables
  step_zv(all_predictors()) %>% # Removes zero-variance predictors
  step_nzv(all_predictors()) %>% # Removes near-zero-variance predictors
  step_corr(all_numeric_predictors(), threshold = 0.90) %>% # Removes highly correlated numeric predict
  step_normalize(all_numeric_predictors()) %>% # Normalizes numeric predictors
  step_mutate(Date = as.numeric(Date)) |>
  step_log(Date)

preg_model <- linear_reg(penalty=tune(), mixture=tune()) %>%
  set_engine("glmnet")

# Set workflow
preg_wf <- workflow() %>%
  add_recipe(my_recipe) %>%
  add_model(preg_model)

# Split data for CV (5-10 groups)
cv_folds <- vfold_cv(no_outliers, v=10, repeats = 1)

# Grid of values to tune over
grid_of_tuning_params <- grid_regular(penalty(),
                                      mixture(range = c(0, 1)), levels = 20)
# Run the CV
CV_results <- preg_wf %>%
  tune_grid(resamples=cv_folds,
            grid=grid_of_tuning_params,
            metrics=metric_set(rmse,mae,rsq)) # or leave metrics null
```

```
## > A | warning: A correlation computation is required, but `estimate` is constant and has 0
##                standard deviation, resulting in a divide by 0 error. `NA` will be returned.
```

```
## There were issues with some computations   A: x1There were issues with some computations   A: x2There
```

```
# Find best tuning parameters
bestTune <- CV_results %>%
  select_best(metric = "rmse")

# Finalize the workflow and fit it
final_wf <- preg_wf %>%
  finalize_workflow(bestTune) %>%
  fit(data=no_outliers)

predictions <- predict(final_wf, new_data = test)
```

```
## Warning: Novel levels found in column 'City': 'Aksaray', 'Artvin', 'Batman',
## 'Bilecik', 'Çanakkale', 'Çankırı', 'Çorum', 'Düzce', 'Erzincan', 'Erzurum',
## 'Giresun', 'Hatay', 'Kahramanmaraş', 'Kars', 'Kırıkkale', 'Kırşehir',
## 'Malatya', 'Manisa', 'Mardin', 'Mersin', 'Nevşehir', 'Niğde', 'Ordu', 'Rize',
## 'Siirt', 'Sivas', 'Tanımsız', 'Yalova', 'Zonguldak'. The levels have been
## removed, and values have been coerced to 'NA'.
```

```
## Warning: Novel levels found in column 'Type': 'MB'. The levels have been
## removed, and values have been coerced to 'NA'.
```

```
predictions <- exp(predictions)

# Prepare for Kaggle Submission
OG  <- vroom::vroom("test.csv")
```

```
## Rows: 100000 Columns: 42
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (4): Open Date, City, City Group, Type
## dbl (38): Id, P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13, P14, P...
##
## 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.
```

```
kaggle_submission <- predictions %>%
   bind_cols(OG) %>%
  dplyr::select(Id, .pred) %>%
  rename(Prediction=.pred) %>%
  mutate(Prediction=pmax(0, Prediction))

vroom::vroom_write(x=kaggle_submission, file="./PenalizedPreds1.csv", delim=",")
```

Model 2 - Boosting

Tried lightgbm and xgboost, suprisingly xgboost performed worse.

```
# This recipe performs better here - 1770795.43113, but the highest it performed on random forest was 1
my_recipe <- recipe(revenue ~ ., data = no_outliers) %>%
  step_date(Date, features = "year") %>% # Extracts features like 'year' from Date
  step_unknown(all_nominal_predictors()) %>% # Handles unknown levels in categorical predictors
  step_novel(all_nominal_predictors()) %>% # Handles unseen levels
  step_dummy(all_nominal_predictors()) %>% # Converts categories to dummy variables
  step_zv(all_predictors()) %>% # Removes zero-variance predictors
  step_nzv(all_predictors()) %>% # Removes near-zero-variance predictors
```

```
  step_corr(all_numeric_predictors(), threshold = 0.70) %>% # Removes highly correlated numeric predict
  step_normalize(all_numeric_predictors()) %>% # Normalizes numeric predictors
  step_rm(Date) # Removes the original 'Date' column

# Boosted Trees
boost_model <- boost_tree(tree_depth=tune(), trees=tune(), learn_rate=tune()) %>%
set_engine("lightgbm") %>% #or "xgboost" but lightgbm is faster
set_mode("regression")

# Create workflow for the boosted tree model
boost_wf <- workflow() %>%
  add_recipe(my_recipe) %>%
  add_model(boost_model)

# Grid of values to tune over, helped decrease score
grid_of_tuning_params <- grid_regular(tree_depth(), trees(), learn_rate(), levels = 5)

# Run the CV
boost_CV_results <- boost_wf %>%
  tune_grid(resamples=cv_folds,
            grid=grid_of_tuning_params,
            metrics=metric_set(rmse,mae,rsq))

# Find best tuning parameters
bestTune <- boost_CV_results %>%
  select_best(metric = "rmse")

# Finalize the workflow and fit it
final_wf <- boost_wf %>%
  finalize_workflow(bestTune) %>%
  fit(data=no_outliers)

## Make predictions
predictions <- predict(final_wf, new_data=test)
predictions <- exp(predictions)

# Prepare data for kaggle submission
kaggle_submission <- predictions %>%
   bind_cols(OG) %>%
  dplyr::select(Id, .pred) %>%
  rename(Prediction=.pred) %>%
  mutate(Prediction=pmax(0, Prediction))

# Write out file
vroom::vroom_write(x=kaggle_submission, file="./LightGBMPreds.csv", delim=",")
```

Model 3 - Bagging, random Forest

I tried pca, didn't improve my recipes. but increasing my correlation threshold did improve the recipe. So did doing a log(Date), I saw that some Kaggle submission did this, so I attempted it myself. I removed outliers post transformation and this seemed to help a lot!

```
#  1791723.07814 doing log(Date) to 1776260.31064 with excluding outliers!
```

```
# 1776260.31064 with outliers removed post transformation
my_recipe <- recipe(revenue ~ ., data = no_outliers) %>%
  step_date(Date, features = "year") %>% # Extracts features like 'year' from Date
  step_unknown(all_nominal_predictors()) %>% # Handles unknown levels in categorical predictors
  step_novel(all_nominal_predictors()) %>% # Handles unseen levels
  step_dummy(all_nominal_predictors()) %>% # Converts categories to dummy variables
  step_zv(all_predictors()) %>% # Removes zero-variance predictors
  step_nzv(all_predictors()) %>% # Removes near-zero-variance predictors
  step_corr(all_numeric_predictors(), threshold = 0.90) %>% # Removes highly correlated numeric predict
  step_normalize(all_numeric_predictors()) %>% # Normalizes numeric predictors
  step_mutate(Date = as.numeric(Date)) |>
  step_log(Date)

# 1726644.14318 became worse when added step_corr - 1758307.79119
my_recipe <- recipe(revenue ~ ., data = no_outliers) %>%
  step_date(Date, features = c("dow", "month", "year", "decimal")) %>%
  step_rm(Date) %>%
  step_mutate_at(Date_dow, fn = factor) %>%
  step_mutate_at(Date_month, fn = factor) %>%
  step_mutate_at(Date_year, fn = factor) %>%
  step_novel(all_nominal_predictors()) %>%
  step_unknown(all_nominal_predictors()) %>% # Handles unknown levels in categorical predictors
  step_dummy(all_nominal_predictors()) %>%
  step_zv(all_numeric_predictors()) %>%
  step_normalize(all_numeric_predictors())

rf_model <- rand_forest(min_n=tune(), trees=tune()) %>%
  set_engine("ranger", importance = "impurity") %>%
  set_mode("regression")

rf_wf <- workflow() %>%
  add_recipe(my_recipe) %>%
  add_model(rf_model)

## Grid of values to tune over
grid_of_tuning_params <- grid_regular(trees(), min_n(), levels = 5)

## Run the CV
rf_CV_results <- rf_wf %>%
  tune_grid(resamples=cv_folds,
            grid=grid_of_tuning_params,
            metrics=metric_set(rmse,mae,rsq))

## Find Best Tuning Parameters
bestTune <- rf_CV_results %>%
  select_best(metric = "rmse")

## Finalize the Workflow & fit it
final_wf <-rf_wf %>%
  finalize_workflow(bestTune) %>%
  fit(data=no_outliers)

## Make predictions
```

```
predictions <- predict(final_wf, new_data=test)
```

```
## Warning: Novel levels found in column 'City': 'Aksaray', 'Artvin', 'Batman',
## 'Bilecik', 'Çanakkale', 'Çankırı', 'Çorum', 'Düzce', 'Erzincan', 'Erzurum',
## 'Giresun', 'Hatay', 'Kahramanmaraş', 'Kars', 'Kırıkkale', 'Kırşehir',
## 'Malatya', 'Manisa', 'Mardin', 'Mersin', 'Nevşehir', 'Niğde', 'Ordu', 'Rize',
## 'Siirt', 'Sivas', 'Tanımsız', 'Yalova', 'Zonguldak'. The levels have been
## removed, and values have been coerced to 'NA'.
```

```
## Warning: Novel levels found in column 'Type': 'MB'. The levels have been
## removed, and values have been coerced to 'NA'.
```

```
rf_predictions <- exp(predictions)

# Prepare data for kaggle submission
kaggle_submission <- rf_predictions %>%
   bind_cols(OG) %>%
  dplyr::select(Id, .pred) %>%
  rename(Prediction=.pred) %>%
  mutate(Prediction=pmax(0, Prediction))

# Write out file
vroom::vroom_write(x=kaggle_submission, file="./RandomForestPreds.csv", delim=",")
```

Model 4 - Neural Net

```
# Staying with a simple recipe gave the same results as a complex recipe here.
my_recipe <- recipe(revenue ~ ., data = no_outliers) %>%
  step_rm(Date) %>%
  step_novel(all_nominal_predictors()) %>%
  step_unknown(all_nominal_predictors()) %>% # Handles unknown levels in categorical predictors
  step_dummy(all_nominal_predictors()) %>%
  step_zv(all_numeric_predictors()) %>%
  step_normalize(all_numeric_predictors())

nn_model <- mlp(hidden_units = tune()) %>%
  set_engine("keras") %>%
  set_mode("regression")

# Set up neural network workflow
nn_wf <- workflow() %>%
  add_model(nn_model) %>%
  add_recipe(my_recipe)

n_tuneGrid <- grid_regular(hidden_units(range=c(1, 100)), levels=3)

tuned_results <- nn_wf |>
  tune_grid(resamples = cv_folds, grid = n_tuneGrid, metrics =
              metric_set(rmse)) # Suppress training output

best_params <- tuned_results |>
  select_best(metric = "rmse")

nn_final_wf <- nn_wf %>%
```

```
  finalize_workflow(best_params) %>%
  fit(no_outliers)

# Prepare predictions and Kaggle submission
predictions <- predict(nn_final_wf, new_data = test)
predictions <- exp(predictions)

# Prepare data for kaggle submission
kaggle_submission <- predictions %>%
   bind_cols(OG) %>%
  dplyr::select(Id, .pred) %>%
  rename(Prediction=.pred) %>%
  mutate(Prediction=pmax(0, Prediction))

# Write out file
vroom::vroom_write(x=kaggle_submission, file="./NNPreds.csv", delim=",")
```

Appendix

Graphs and Tables for Paper

Glimpse of Dataset

```
first_column <- train[, "P1", drop = FALSE]  # P1
last_column <- train[, "P37", drop = FALSE]  # P37

# Create a column of ellipses
ellipsis_column <- data.frame("..." = rep("...", nrow(train)))

# Select columns that are NOT labeled as 'P1' to 'P37'
non_P_columns <- train[, !grepl("^P", colnames(train)), drop = FALSE]  # non-P columns

non_P_columns <- non_P_columns |>
  dplyr::rename(Revenue = revenue) |>
  dplyr::select(-log_revenue)

# Combine the selected columns: P1, ellipses, P37, and non-P columns
train_summary <- cbind(non_P_columns, first_column, ellipsis_column, last_column)

# Display the table using kable with the custom summary
head(train_summary) |>
  kbl(caption = "Glimpse of the Dataset") %>%
  kable_classic(full_width = F, html_font = "Cambria")
```

Date Table

```
# Date Information
train |>
  dplyr::select(Date) |>
  arrange(desc(Date)) |>
  head() %>%
  kbl() %>%  # Create a kable object
```

Table 1: Glimpse of the Dataset

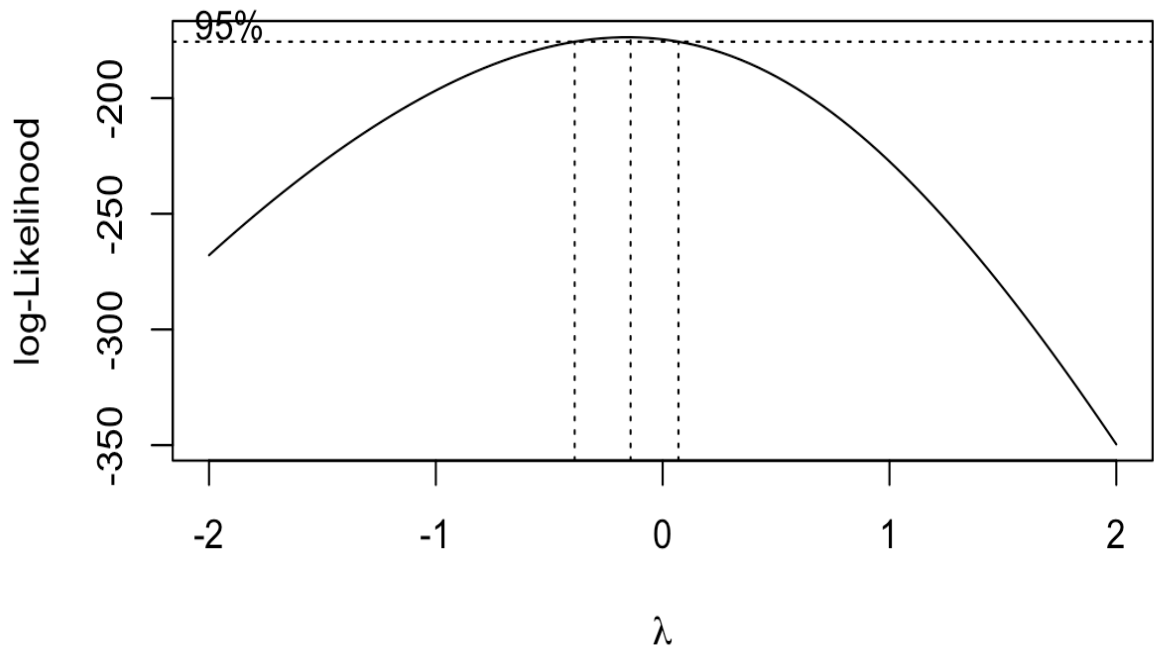
Id	City	City Group	Type	Revenue	Date	P1	...	P37
0	İstanbul	Big Cities	IL	5653753	1999-07-17	4	...	4
1	Ankara	Big Cities	FC	6923131	2008-02-14	4	...	0
2	Diyarbakır	Other	IL	2055379	2013-03-09	2	...	0
3	Tokat	Other	IL	2675511	2012-02-02	6	...	6
4	Gaziantep	Other	IL	4316715	2009-05-09	3	...	3
5	Ankara	Big Cities	FC	5017319	2010-02-12	6	...	0

Date
2014-01-25
2014-01-03
2013-12-21
2013-11-12
2013-10-25
2013-08-09

```
  kable_classic(full_width = F, html_font = "Cambria")
```

Box-Cox Results

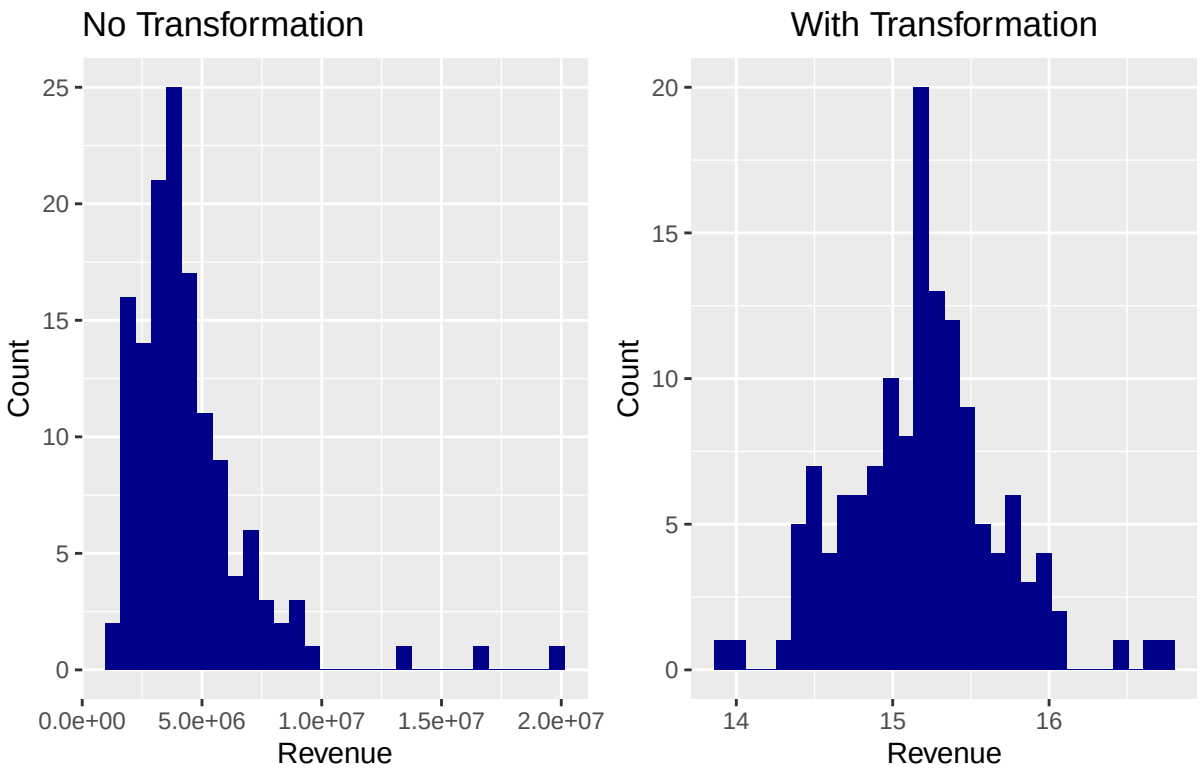
```
# Output both histograms together
img <- readPNG("Boxcox2.png")

# Step 3: Create a ggplot object for the image using annotation_raster
img_plot <- ggplot() +
  annotation_raster(img, xmin = -Inf, xmax = Inf, ymin = -Inf, ymax = Inf) +
  theme_void() + ggtitle("Box-Cox Results") + theme(plot.title = element_text(hjust = 0.5))

img_plot
```



Log Transformation

```
no_transformation + transformation + plot_annotation(
  title = 'Impact of Log Transformation on Revenue',
) + theme(plot.title = element_text(hjust = 0.5))
```

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



Model Results Table

```
model_performance <- data.frame(
  Model_Type = c("Penalized Linear Regression", "LightGBM Boosted Trees","Random Forest"),
  Model_Description = c(
    "Baseline model with auto hyperparameter tuning for penalty and mixture",
    "Includes auto hyperparameter tuning for tree_depth, trees, and  learn_rate",
    "Includes auto hyperparameter tuning for min_n and trees"
  ),
  RMSE = c("1837562.77602", "1770795.43113", "1726644.14318")
)

# Here is the actual table printed out
kable(model_performance,
      caption = "Model Performance Comparison",
      col.names = c("Model Type", "Model Description", "RMSE")) %>%
  kable_classic(full_width = T, html_font = "Cambria", bootstrap_options = c("striped", "hover", "conde
  column_spec(1, bold = F) %>%
  column_spec(3, width = "7em", color = "black") %>%
  row_spec(0, bold = TRUE, background = "#D3D3D3")
```

Profit associated with Random Forest

```
total_sum <- 0

for (i in rf_predictions$.pred) {
  total_sum <- total_sum + i
```

```
}

# total_sum
```

Random Forest Feature Importance

```
final_wf <-rf_wf %>%
  finalize_workflow(bestTune) %>%
  fit(data=no_outliers) %>%
  extract_fit_parsnip() %>%
  vip(num_features = 10)

final_wf + plot_annotation(
  tag_levels = list(c('Random Forest Feature Importance')),
)
```

Random Forest Feature Importance