Prediction of Stock Price Using Classical Machine Learning and Outlier Removal

Mark Khusidman

2023-05-23

Introduction

For reasons that are not difficult to understand, the ability to predict stock price has long been coveted. Although much time has been spent devising techniques to assist in this pursuit, its mastery has proven to be elusive for a number of reasons. The processes underlying changes in a given stock's price are usually both complex and dynamic. Movements in stock price are often subject to a significant amount of noise, especially when viewed at higher granularities. Outliers are also commonly present in price data. Finally, the quantity of data for a particular stock are often limited, and data taken from one stock are rarely applicable when modeling another. Because these inherent difficulties still pose a significant barrier to stock price prediction, further investigation into this subject is warranted. Machine learning (ML) represents a relatively new and promising approach to the modeling of stock price. This study aims to test the ability of relatively simple ML algorithms, used in combination with basic outlier removal, to model and predict a particular stock's closing price. The stock in question belongs to a video game retailer named Gamestop and trades under the symbol "GME". GME has itself been the subject of a significant amount of attention over the past couple of years due to perceived irregularities in its underlying trading patterns, making it a particularly interesting candidate for exploration. The GME data used in this study have a frequency of 1 observation per day and encompass all of the trading days from January 12th, 2021 through May 5th, 2023, of which there are 583. This represents all available data occurring after the aforementioned irregularities began. The data consist of 5 variables: *GME.Open* represents each day's opening price, *GME.Close* represents each day's closing price, *GME.Low* represents the lowest price of each day, *GME.High* represents the highest price of each day, and *GME.Volume* represents the number of GME shares traded on each day. In summary, the initial GME dataset used in this study contains 583 rows of 5 columns. Data modeling is performed using k-nearest-neighbors (k-NN) and elastic net (EN). Each algorithm is used with both an "intact" subset of the GME data and a "clipped" subset which had gone through outlier removal, yielding a total of 4 models to be evaluated.

Methods

After setting start and end dates, GME data is loaded directly into the environment using the "quantmod" package. The resulting data begins at the start date and encompasses all days up to, but not including, the end date.

```
# Load GME data using quantmod package
start <- as.Date("2021-01-12")
end <- as.Date("2023-05-06")
getSymbols("GME", from = start, to = end)
```

Data with daily frequency are used because this is the highest granularity offered by quantmod. Included in the data is the *GME.Adjusted* column, which should contain the stock's closing price after it is adjusted

for various corporate actions. In this case, however, the column is identical to *GME.Close* and is therefore removed. The rest of the data are also checked for missing values.

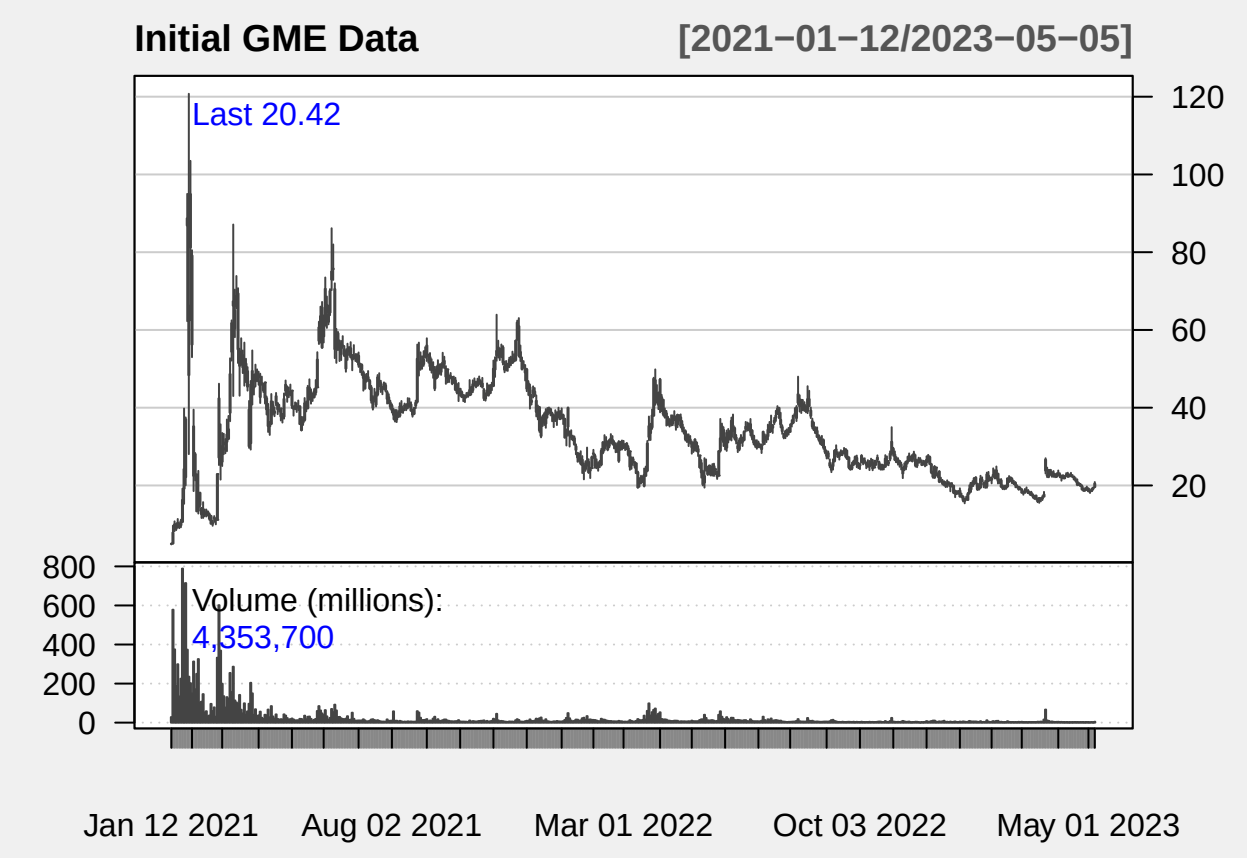
```
# Drop GME.Adjusted
sprintf("Are GME.Adjusted and GME.Close identical?: %s",
              all(GME$GME.Close == GME$GME.Adjusted))
GME$GME.Adjusted <- NULL

# Make sure there are no missing values in data
sprintf("Any missing values in data?: %s",any(is.na(GME)))
```

```
## [1] "Are GME.Adjusted and GME.Close identical?: TRUE"
## [1] "Any missing values in data?: FALSE"
```

This yields an initial dataset which, as previously mentioned, consists of 583 rows and 5 columns. At this point, it is helpful to visualize the data. A candle chart is a common method of visualizing stock data which has the advantage of being able to incorporate all of the columns in the initial dataset.

```
# Visualize the initial data witch a candle chart
candleChart(GME, up.col = "blue", dn.col = "red", theme = "white",
            name = "Initial GME Data")
```



Although the individual candles are difficult to make out, the chart still provides a decent sense of how GME's price and volume changed over the associated time-frame. One thing that is immediately clear is that both price and volume are non-stationary. Data is considered non-stationary if either its mean or

variance change over time. In this case, both the mean and the variance of all columns in the initial dataset fluctuate over time. This poses a challenge, as non-stationary data is generally more difficult to model with ML algorithms than stationary data.

Before this issue can be addressed, however, the data must be split into a training set and a test set.

```
# Split data into training and test sets
training <- GME[1: 450,]
test <- GME[451: nrow(GME),]
```

Rather than splitting the data via the removal of a random sample, the initial dataset is split into two contiguous groups. A contiguous set of test data is better than a random sample at approximating the characteristics of future input data, thus leading to a better estimate of a model's generalization error. The training set yielded from this split is pictured below.

```
# Visualize training data with a candle chart
candleChart(as.xts(training), up.col = "blue", dn.col = "red", theme = "white",
            name = "Raw Training Data")
```

Now that the initial dataset has been split, a baseline model can be defined based on the training set. This model will serve as a useful reference point when first evaluating the k-NN and EN models. The specific baseline model chosen in this instance is the naive model, also known as the random walk model. Predictions using this model are always equal to the previously observed value, meaning that the predicted series looks like the observed series shifted forward by one time-step. This model is particularly useful for time series with no consistent trend or apparent seasonality. It should be noted that the first 16 values of the training data are not used as observations when evaluating the baseline model. For reasons that will later be made clear, this allows for a better comparison to future evaluations of ML models.

```r
# Evaluate baseline random walk model on training set
observed <- as.numeric(GME$GME.Close[17:450])
baseline_pred_train <- as.numeric(GME$GME.Close[16:449])
baseline_rmse_train <- sqrt(mean((baseline_pred_train - observed)^2))
print(sprintf("Baseline training RMSE: %f",baseline_rmse_train))
```

```
## [1] "Baseline training RMSE: 2.974267"
```

As shown above, the baseline model yields an RMSE of approximately 2.974.

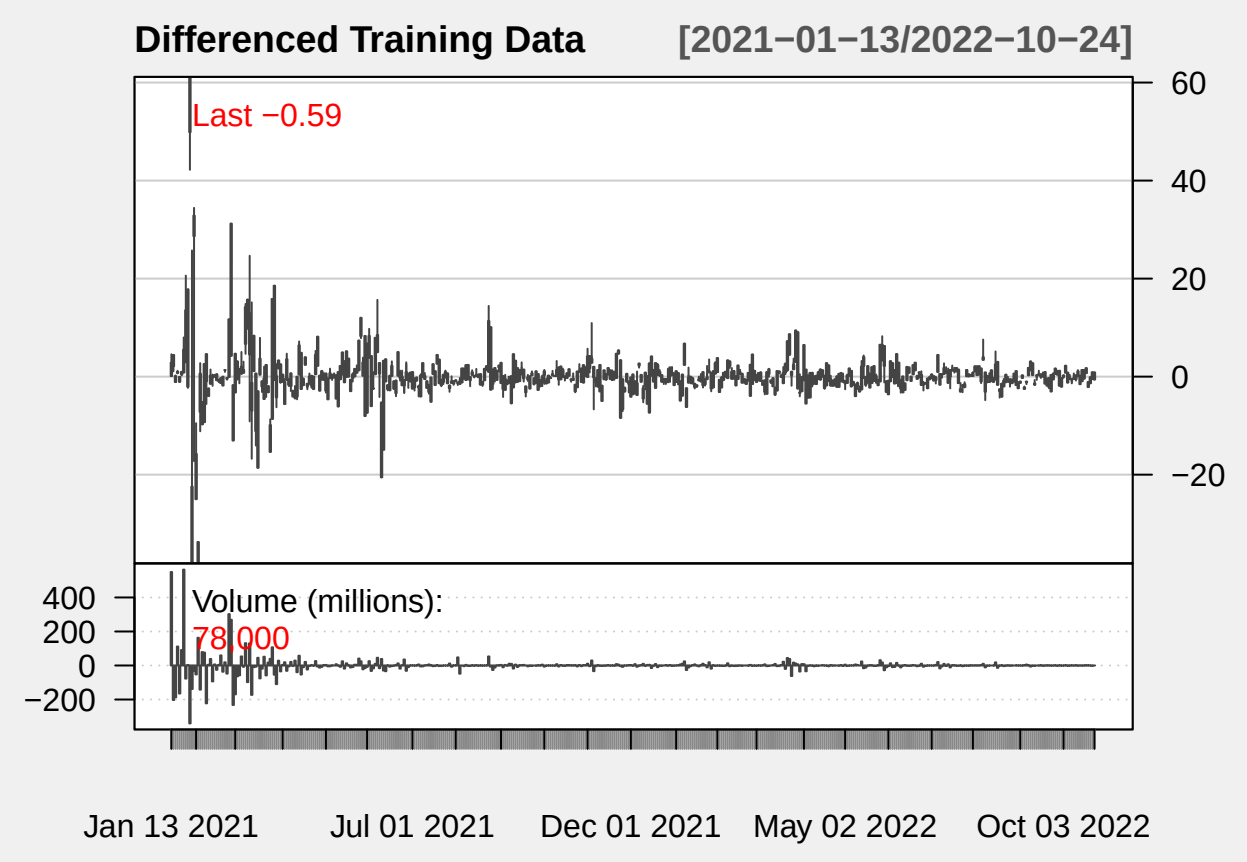
After the baseline model is defined, preprocessing of the training set begins. First, the aforementioned issue regarding the data's non-stationary nature will be addressed to an extent. This is accomplished by "differencing" the data. This means that every data point is replaced by the difference between said point and its preceding value. It is important to note that this technique only serves to stabilize the data's mean; it is not effective at stabilizing variance.

```r
# Calculate differenced training data
training <- diff(as.matrix(training))
```

The resulting array has one fewer row than before.

Now that the data have been differenced, it is useful to again visualize them. A true candle chart of these data cannot be made because the meaning of individual candles would be ambiguous. However, the previously used "candleChart" function can still utilize the differenced data to create a chart which, like before, gives a decent impression of how the data are shaped over time.

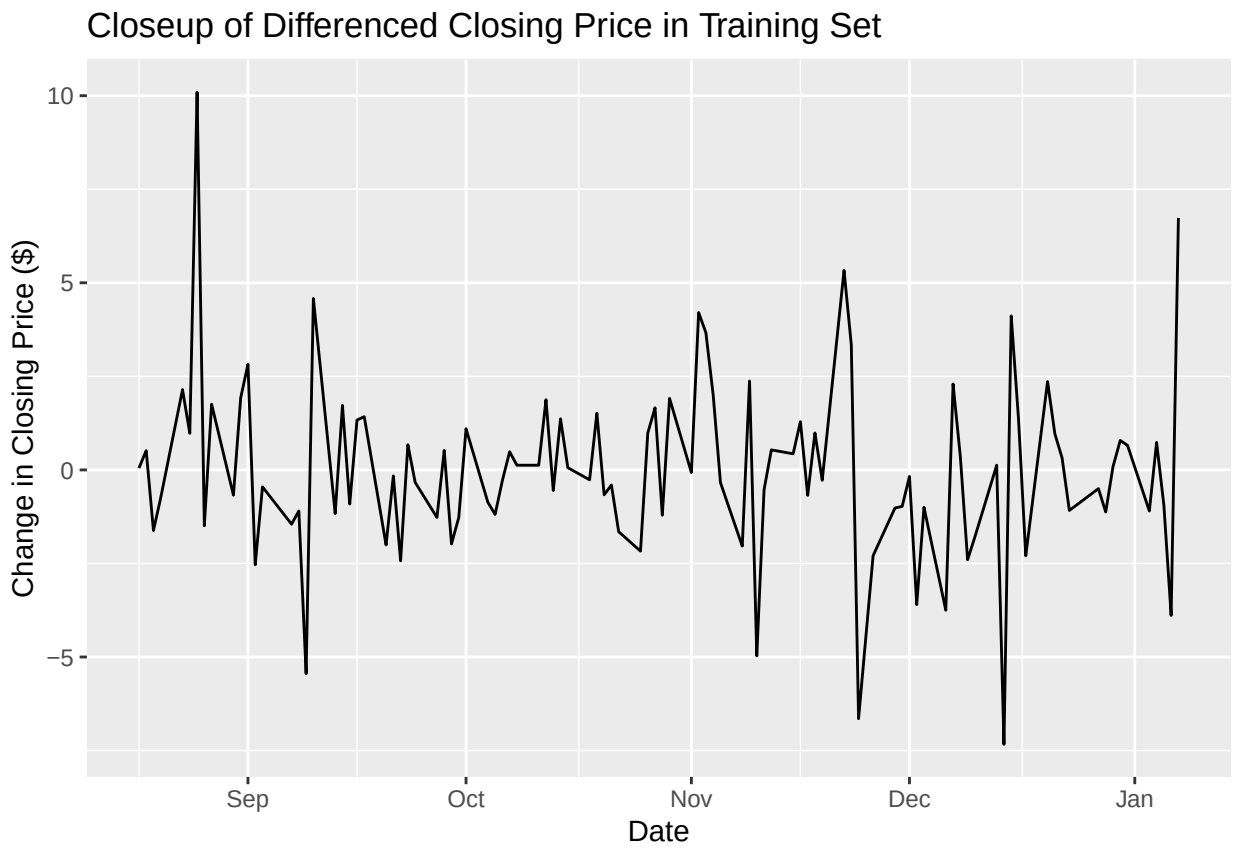
```r
# Visualize differenced training data
candleChart(as.xts(training), up.col = "blue", dn.col = "red", theme = "white",
            name = "Differenced Training Data")
```



Although the differenced data still aren't perfectly stationary, they are far more so than the non-differenced data.

The general shape of the differenced data can be better appreciated by visualizing a subset of the *GME.Close* column. The shape of this column is fairly representative of the other differenced input columns.

```
# Visualize closeup of differenced closing price data in training set
qplot(index(GME$GME.Close[151:251]), training[150:250, 1], geom = "line",
      xlab = "Date", ylab = "Change in Closing Price ($)",
      main = "Closeup of Differenced Closing Price in Training Set")
```



Visualizing the differenced *GME.Close* column is also useful because it will be used as the target during model training. This will be discussed in more detail later.

Next, the training set is standardized. This is done by centering each column before dividing the column by its standard deviation.

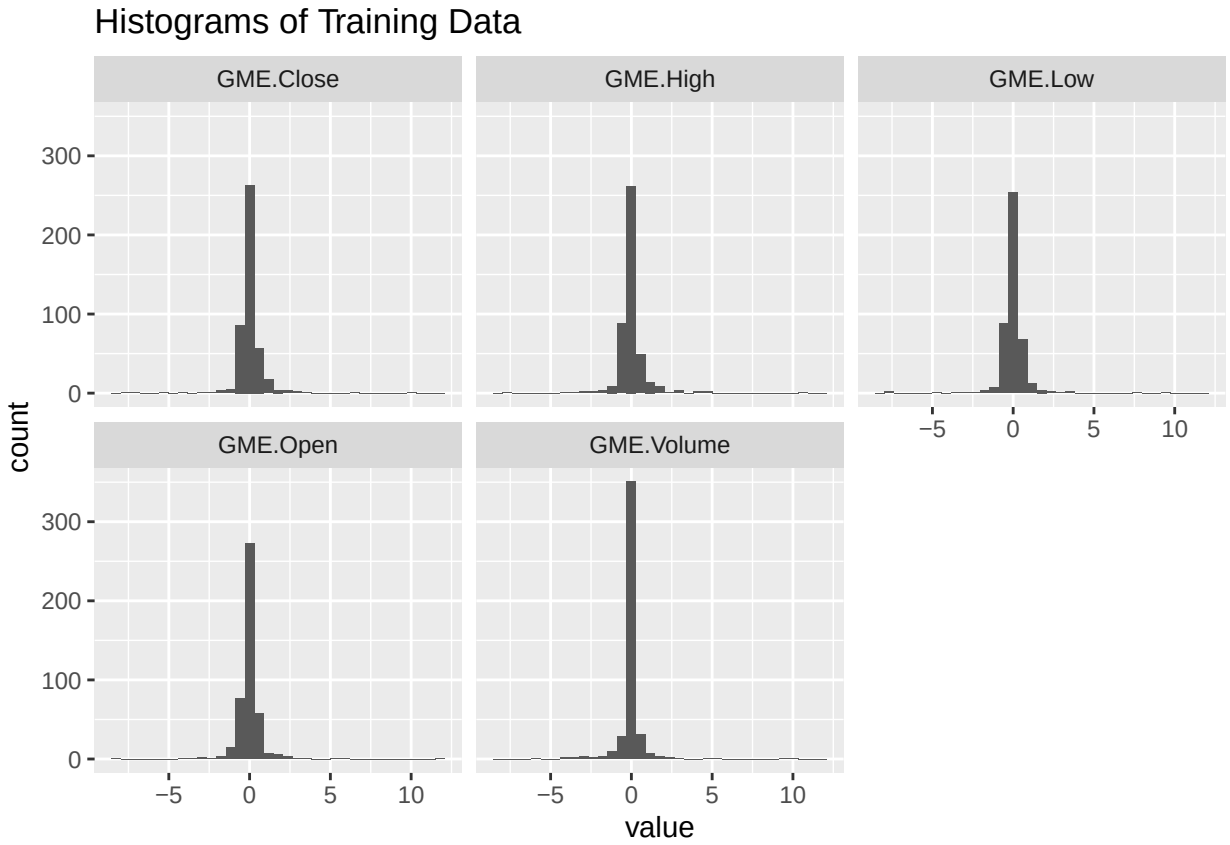
```
# Standardize training data
train_sds <- apply(training, 2, sd)
train_means <- colMeans(training)
for(i in 1:5){training[,i] <- (training[,i] - train_means[i]) / train_sds[i]}
```

The training set is subsequently converted into a "data.table" object to ease further preprocessing.

```
# Convert training data to data.table object
training <- as.data.table(training)
```

Before continuing, it is useful to visualize the distribution of values found in the training set. This can be accomplished through the use of histograms.

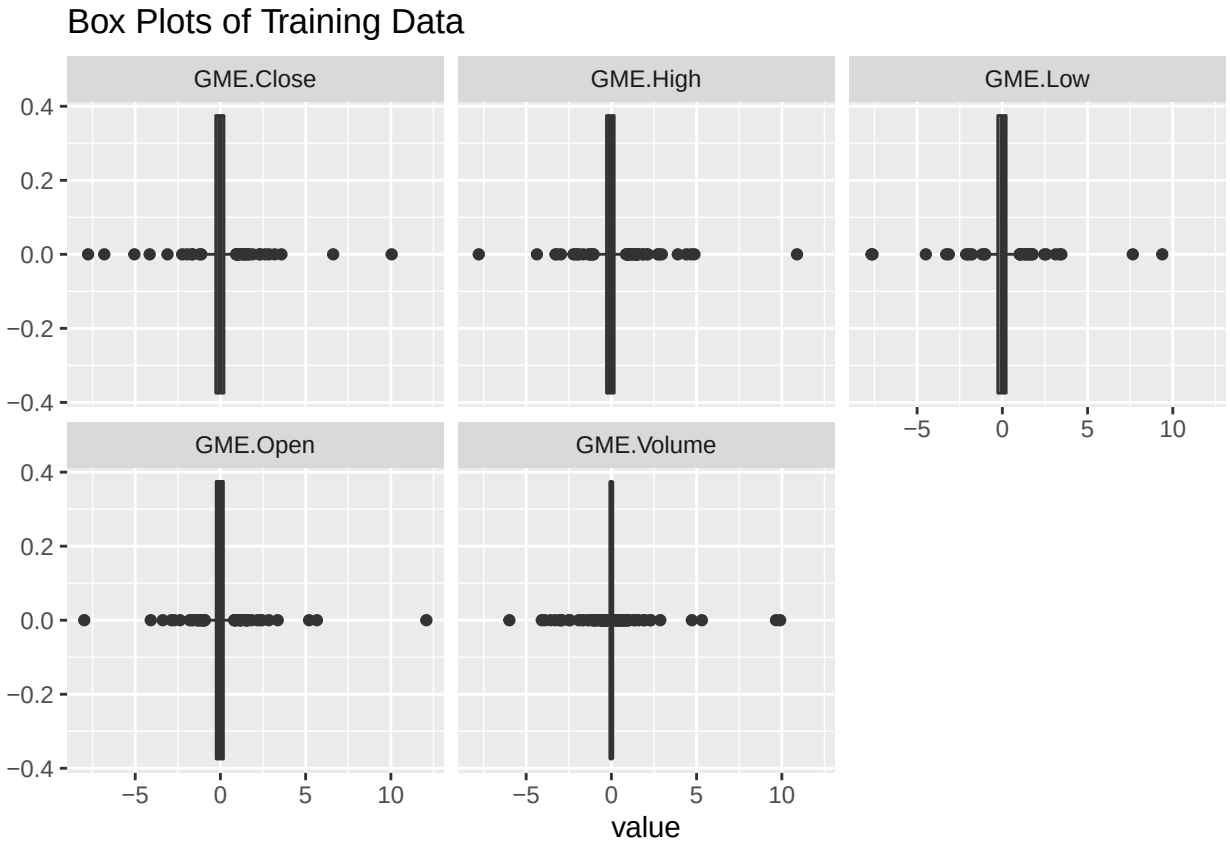
```
# Visualize distribution of values in training data with histograms
training |> pivot_longer(everything()) |> ggplot(aes(value)) +
  geom_histogram(bins = 35) + facet_wrap(vars(name)) +
  ggtitle("Histograms of Training Data")
```



All of the columns seem to have roughly bell-shaped distributions, although their peaks appear significantly sharper than would be expected for normally distributed data. This is especially true for *GME.Volume*.

Although it is clear that outliers are present, the previous chart's rendering does not allow them to be easily distinguished. As such, box plots are used to better visualize the training data's outliers.

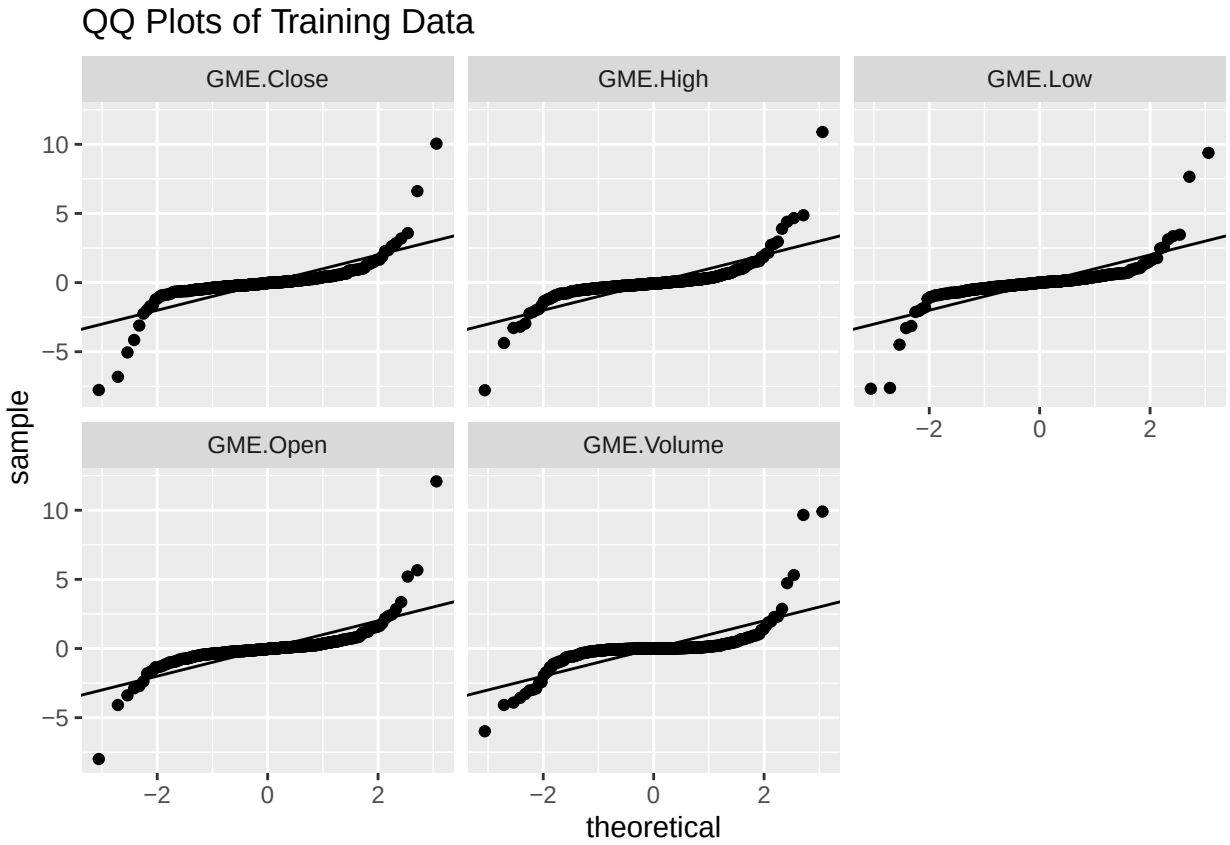
```
# Visualize outliers in training data with box plots
training |> pivot_longer(everything()) |> ggplot(aes(value)) +
  geom_boxplot() + facet_wrap(vars(name)) +
  ggtitle("Box Plots of Training Data")
```



In the above plots, outliers are defined as values which exceed a certain distance from their nearest hinge (The hinges are comprised of the 1st and the 3rd quartiles). In this case, the distance in question is equal to 1.5 * IQR, where IQR is the interquartile range. It is important to note that this method of defining outliers differs from the method which will be used during outlier removal.

Another way to visualize the distribution of training data is through the use of QQ plots. These will yield a better understanding of the degree to which the data are normally distributed.

```
# Compare distribution of values in training data to normal distribution
training |> pivot_longer(everything()) |> ggplot(aes(sample = value)) +
  geom_qq() + geom_abline() + facet_wrap(vars(name)) +
  ggtitle("QQ Plots of Training Data")
```



The above visualization demonstrates clear differences between normally distributed data and the training data. Specifically, all columns in the training set appear to have distributions with heavier tails than what would be found in a normal distribution.

Next, 14 lagged columns are created in the training dataset for each of the 5 columns currently present. This means that every row now contains 15 days worth of opening price, closing price, daily high, daily low, and trading volume data.

```
# Add lagged columns to training data
add_lagged <- function(dt, n){
  # Define new column names
  lag_names <- map_chr(1:n, ~ sprintf("lag%d", .))
  lag_names <- expand.grid(lag_names, names(dt))
  lag_names <- apply(lag_names, 1, function(v){paste(v[2], v[1], sep = "_")})
  # Create lagged columns
  dt[, (lag_names) := shift(.SD, 1:n, type = "lag"), .SDcols = names(dt)]
}

add_lagged(training, 14)
```

Once the above code is run, the training dataset contains 75 columns in total. Each row of the current dataset will serve as a single input when the k-NN and EN models are trained and given an initial evaluation.

Before model training can commence, however, one more column must be defined. The *target* column contains the correct output value for each input. In this case, the target value is the closing price recorded one day after the final date associated with the input. This column will be separated from the inputs before they are used for training or evaluation of ML models .

```
# Add target column to training data
training[, target := shift(GME.Close, 1, type = "lead")]
```

It should be noted that the process of adding lagged columns results in 14 rows with at least one empty value, while the addition of the target column adds another such row. These rows must now be removed.

```
# Drop rows with missing values from training data
training <- drop_na(training)
```

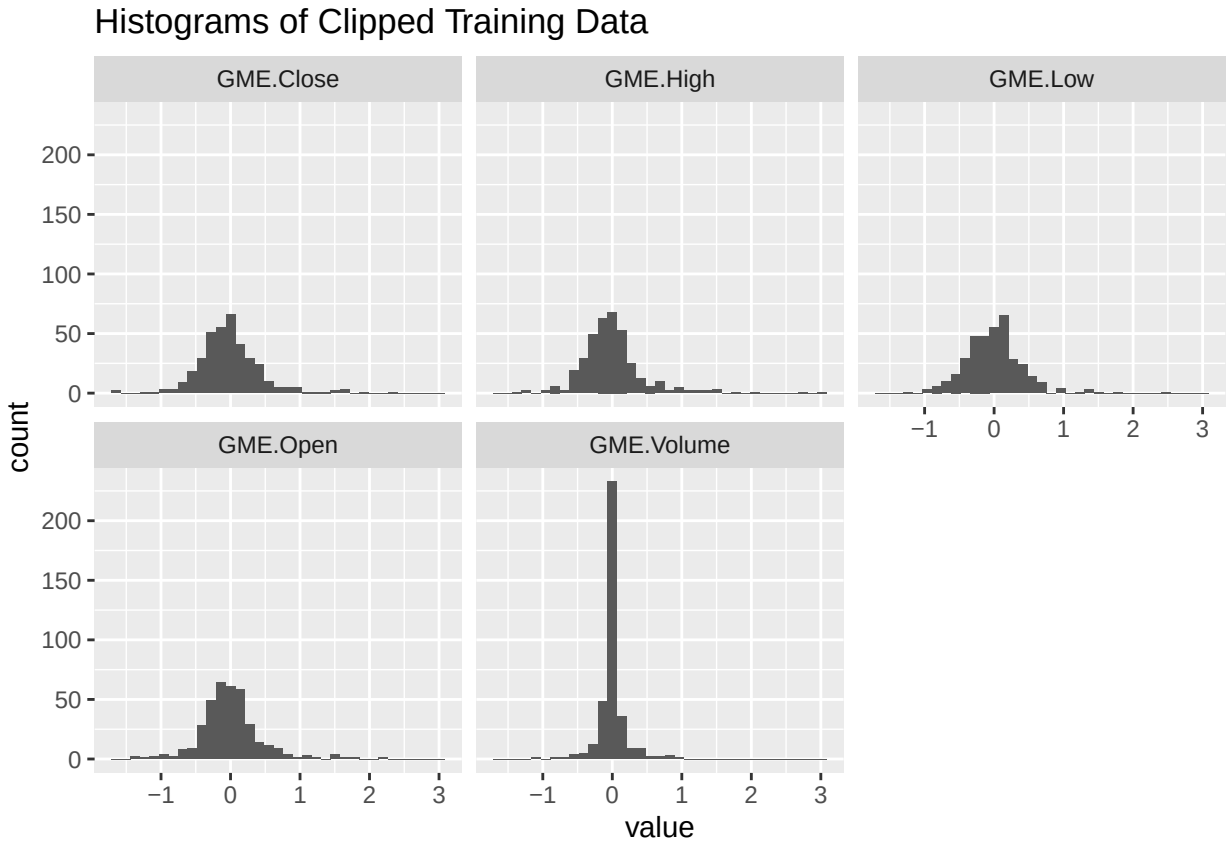
At this point, a separate training dataset is created by subjecting the current training data to an outlier removal process. Specifically, rows containing values which are more than 3 standard deviations from their respective means are removed. Because the training data are already scaled, this is easily done.

```
# Create separate training set with clipped outliers
training_clip <- training[apply(abs(training) <= 3, 1, all),]
```

Note that this operation also removes rows with a target value outside the specified range. From this point forward, the training dataset subjected to outlier removal and its associated models will be referred to as "clipped", while the dataset with no outlier removal and its associated models will be referred to as "intact". The intact training set currently contains 434 rows, while the clipped training set contains 367.

Some insight can be gained into the result of the outlier removal process by using histograms to visualize the clipped data's distribution. This is done using the same code as before.

```
# Visualize distribution of values in clipped training data with histograms
print(training_clip[, 1:5] |> pivot_longer(everything()) |> ggplot(aes(value)) +
        geom_histogram(bins = 35) + facet_wrap(vars(name)) +
        ggtitle("Histograms of Clipped Training Data"))
```



Based on the above histograms, it would appear as though the outlier removal process was successful to a fair extent. There do appear to be remaining outliers, but they are much less extreme than many of the outliers seen previously in the intact data.

Now that the clipped training set has been created, model tuning can soon begin. As mentioned previously, both the k-NN and EN models will be utilized for this purpose. The k-NN algorithm involves first selecting a predefined number of "neighboring" input vectors which are most similar to the input being used for prediction. For continuous data, similarity is generally defined as the euclidean distance between two vectors. The number of similar inputs used for prediction is referred to as k. One the most similar inputs are identified, predictions are made by averaging the target values associated with said similar inputs. Because the model makes no underlying assumptions about the relationship between inputs and outputs (apart from the idea that similar inputs will yield similar outputs), it can prove useful in modeling data where this relationship is complex. As the relationship between inputs and outputs is not readily apparent in the GME data, this quality may be useful. Implementing the EN model means fitting a regularized linear model to the data. The penalty applied to the parameters is calculated via a mixture of L1 and L2 normalization. As such, the EN model is often thought of as a mixture between Ridge regression and Lasso regression. The extent to which one form of regularization dominates over the other is controlled by the hyperparameter *alpha*. A value close to 0 gives more weight to the L2 penalty, while a value close to 1 gives more weight to the L1 penalty. The final penalty value is scaled with the *lambda* hyperparameter, which thereby controls the extent of regularization. Regularized linear models are particularly useful when there is a high degree of colinearity among the input columns. This is because unregularized linear models are prone to overfitting when trained on such data. Because there is fairly high degree of colinearity between input columns which are associated with the same time-step, this may be a useful quality. Although the EN model ultimately entails fitting a linear model to the data, the inherent generalization involved in such a simplification may offer its own protection against overfitting.

One final step to be undertaken before model tuning begins is the creation of a function which turns raw

model outputs into closing price predictions. Recall that since the models are being trained on differenced data, their predictions correspond to changes in closing price, rather than the price itself. To create final predictions, each model output must be added to the observed closing value occurring directly before the predicted time-step. Because of the single row lost when differencing the data, the 14 rows lost when creating lagged columns, and the single row lost when adding a target column, the first 16 values of input data are not used as observations when evaluating models.

```
# Convert raw model outputs into closing price predictions during training
get_train_pred <- function(model, lambda = NULL){
  if(is.null(lambda)){
    # If labmda is null, assum k-NN model
    pred <- predict(model, as.matrix(select(training, -target)))
  }
  else{
    # If lambda is not null, assume elastic net model
    pred <- predict(model, as.matrix(select(training, -target)),
                    s = lambda)[,1]
  }
  # Unscale raw model outputs
  pred <- (pred * train_sds[1]) + train_means[1]
  # Add unscaled outputs to closing price occuring right before predicted value
  prior_vals <- as.numeric(GME$GME.Close[16:449])
  pred <- prior_vals + pred
  # Define observed values for comparison to predictions
  observed <- GME$GME.Close[17:450]

  # Baseline is equal to predictions using the baseline model
  results <- data.frame(pred = pred, observed = as.numeric(observed),
                        baseline = prior_vals,
                        ind = index(observed))
  results
}
```

The above function yields a data.frame object containing closing price predictions, the observed values corresponding to those predictions, the predictions yielded from the previously defined baseline model, and an index used for visualization purposes. It should be noted that for all models, including clipped ones, evaluation is done using the intact training set. This is done to give the best possible sense of their future testing performance.

Now that the aforementioned function has been defined, model tuning can finally commence. For all 4 models discussed in this study, tuning is conducted via K-fold cross-validation where the number of folds is equal to 10. Like the test data, all 10 validation sets used during the process are contiguous. This makes K-fold cross-validation more representative of this study's final testing conditions than techniques like bootstrapping or Monte-Carlo cross-validation. Time series cross-validation, a technique in which only the data points occurring prior to those in the validation set are used for for training, would have been the technique most representative of final testing conditions. However, the fact that this process disregards a portion of the training data during all but its final iteration, when coupled with the small quantity of training data available, makes this technique undesirable. In addition, the models explored in this study assume that the relationship between their input and output is static over time, a case in which time series cross-validation is of dubious benefit.

The intact k-NN model is the first to be tuned.

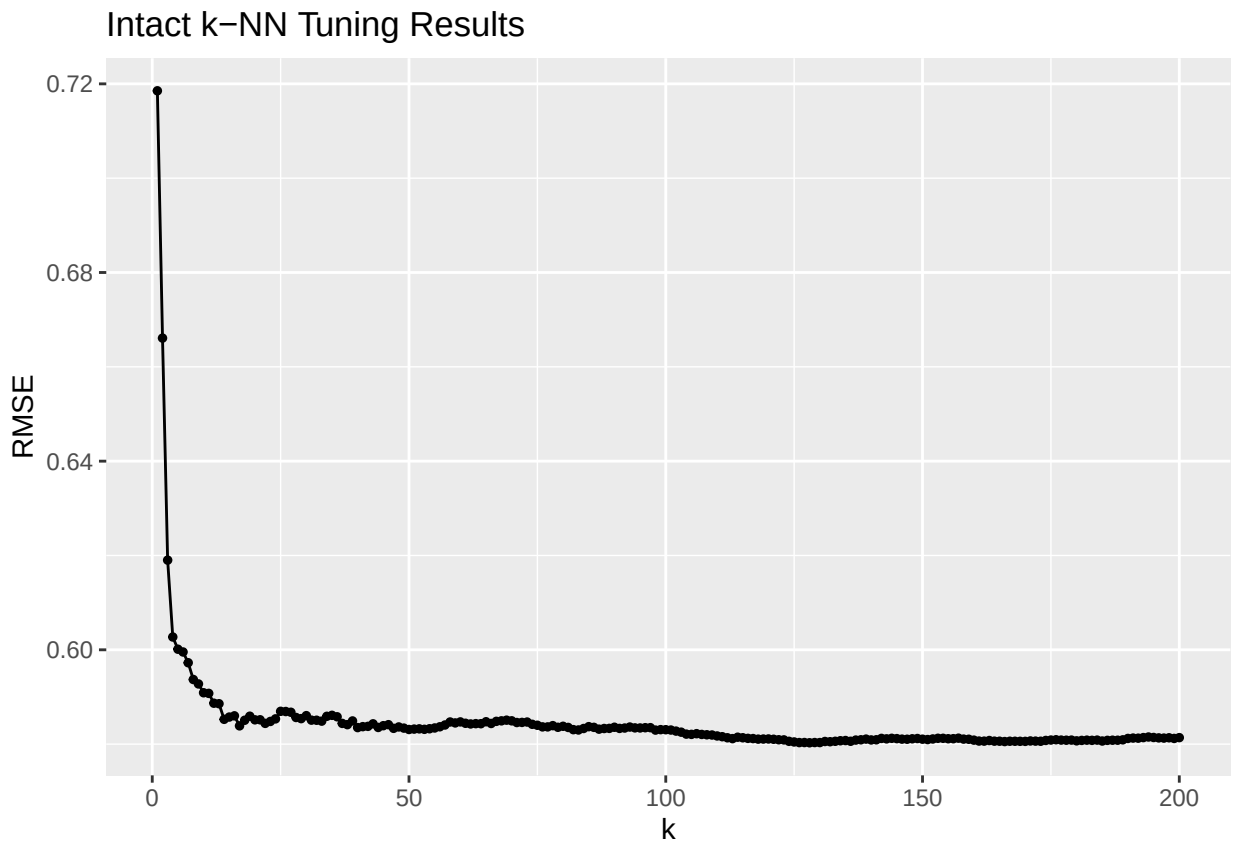
```
# Tune intact k-NN model
knn_intact <- train(select(training, -target), training[, target],
                    trControl = trainControl("cv"),
                    tuneGrid = data.frame(list(k = 1:200)),
                    method = "knn")
sprintf("Intact k-NN best tune: k = %s", knn_intact$bestTune)
```

```
## [1] "Intact k-NN best tune: k = 128"
```

The greatest value of k investigated during tuning is 200. A limit must be imposed because using too great a fraction of the training data to generate outputs will inhibit variety in the model's predictions. As shown above, the best value of k is in this case found to be 128.

The k-NN tuning process can be visualized by plotting the relationship between the values of k investigated and their associated RMSEs.

```
# Confirm best value of k for intact k-NN
print(knn_intact$results |> ggplot(aes(k, RMSE)) + geom_line() +
        geom_point(size=1) +
        ggtitle("Intact k-NN Tuning Results"))
```



The above chart seems to confirm that 128 is a reasonable value for k.

Next, the tuned model is evaluated on intact training set, yielding an initial impression of its performance.

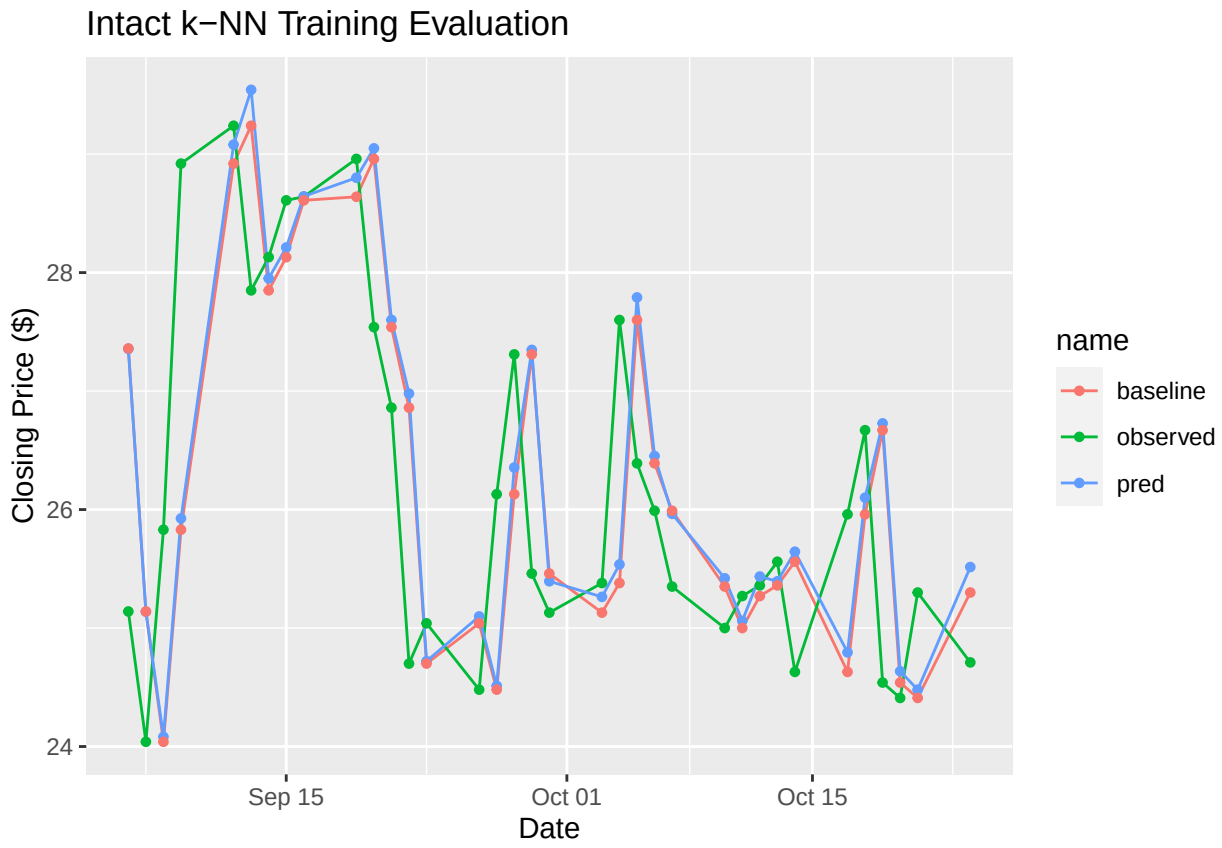
```
# Evaluate best intact k-NN model on intact training set
knn_train_intact <- get_train_pred(knn_intact$finalModel)
rmse <- sqrt(mean((knn_train_intact$pred - knn_train_intact$observed)^2))
sprintf("Intact k-NN training RMSE: %f", rmse)
```

```
## [1] "Intact k-NN training RMSE: 2.954413"
```

As shown above, the RMSE of the intact k-NN is approximately 2.954 when evaluated on the intact training set. This represents a slight improvement over the baseline training RMSE of 2.974.

A more in-depth assessment of model performance can be made by visually comparing predicted values to observed ones. The chart below represents a subset of the predictions made during the intact k-NN model's initial evaluation. Baseline predictions are also included in the chart for comparison.

```
# Visualize training set predictions yielded by intact k-NN model
print(knn_train_intact[400:434,] |>
        pivot_longer(cols = c("observed", "pred", "baseline")) |>
        ggplot(aes(ind, value, color = name)) + geom_line() +
        geom_point(size=1.2) +
        labs(title="Intact k-NN Training Evaluation", x="Date",
             y="Closing Price ($)"))
```



Based on the above visualization, it is apparent that the predicted values yielded by this model more closely follow the baseline predictions than the observations. However, there are multiple points where the intact k-NN predictions and the baseline predictions significantly differ. The similarity in training RMSE between the baseline model and the intact k-NN model appears to be supported by this visualization.

The clipped k-NN is the second model to be tuned.

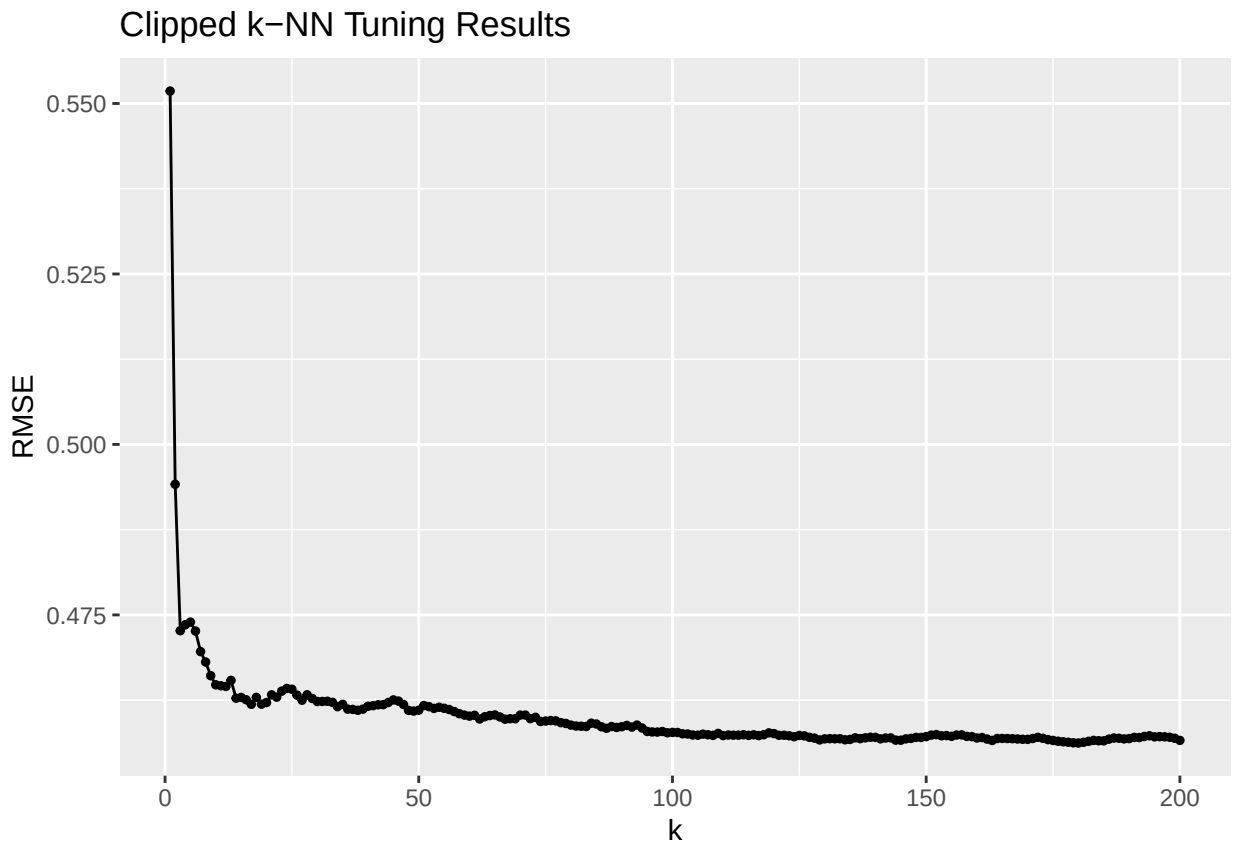
```
# Tune clipped k-NN model
knn_clipped <- train(select(training_clip, -target), training_clip[, target],
                     trControl = trainControl("cv"),
                     tuneGrid = data.frame(list(k = 1:200)),
                     method = "knn")
sprintf("Clipped k-NN best tune: k = %s", knn_clipped$bestTune)
```

```
## [1] "Clipped k-NN best tune: k = 180"
```

In this case, the best value for k is found to be 180, a fairly significant departure from the 128 neighbors used in the intact k-NN model.

As with the previous k-NN model, the relationship between the value of k and its associated RMSE is visualized.

```
# Confirm best value of k for clipped k-NN
print(knn_clipped$results |> ggplot(aes(k, RMSE)) + geom_line() +
        geom_point(size=1) +
        ggtitle("Clipped k-NN Tuning Results"))
```



It does indeed appear as though 180 is the optimal value of k out of those investigated.

The clipped k-NN model is now given an initial evaluation. As mentioned previously, both clipped and intact models are evaluated on the intact training set.

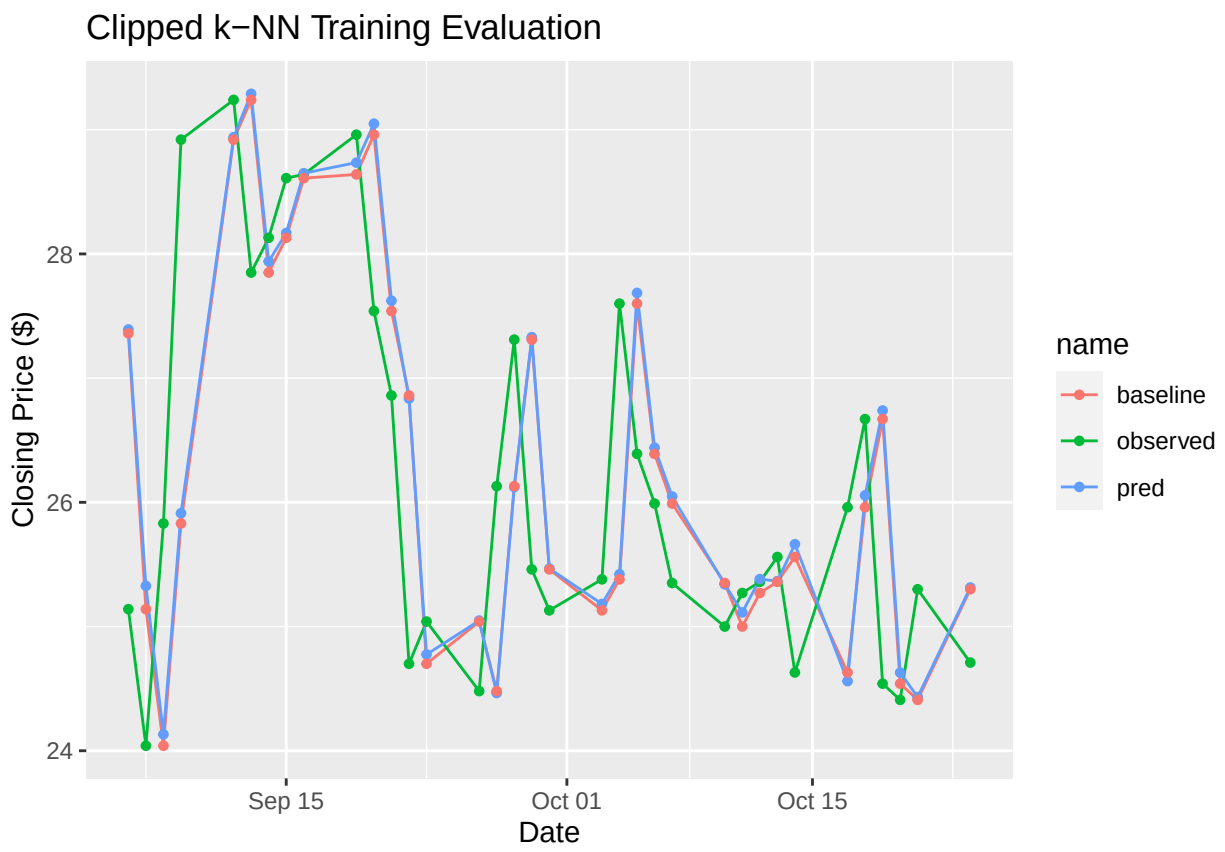
```
# Evaluate best clipped k-NN model on intact training set
knn_train_clipped <- get_train_pred(knn_clipped$finalModel)
rmse <- sqrt(mean((knn_train_clipped$pred - knn_train_clipped$observed)^2))
sprintf("Clipped k-NN training RMSE: %f", rmse)
```

```
## [1] "Clipped k-NN training RMSE: 2.969597"
```

Although the clipped k-NN's training RMSE of approximately 2.970 is still marginally better than the baseline of 2.974, it is worse than the intact k-NN's training RMSE of 2.954.

The predictions made as part of this model's initial evaluation are visualized in the same manner as the intact k-NN's training set predictions.

```
# Visualize training set predictions yielded by clipped k-NN model
print(knn_train_clipped[400:434,] |>
        pivot_longer(cols = c("observed", "pred", "baseline")) |>
        ggplot(aes(ind, value, color = name)) + geom_line() +
        geom_point(size=1.2) +
        labs(title="Clipped k-NN Training Evaluation", x="Date",
             y="Closing Price ($)"))
```



The predictions made by the clipped k-NN model follow the baseline predictions more closely than the intact k-NN model. However, some clipped k-NN predictions are still noticeably different from their associated baseline predictions. Based on the above visualization, it is easy to see why the training RMSE of the clipped k-NN model is so similar to that of the baseline model.

The intact EN model is the third to be tuned. It should be noted that the *alpha* and *lambda* hyperparameters are tuned separately: First, the value for *alpha* is found using the "caret" package. Next, the "glmnet" package is used directly to find the value of *lambda* while the value of *alpha* remains fixed. This is done because glmnet's default tuning implementation finds the optimal *lambda* by testing against an extensive, non-linear sequence of values. Although it would have been possible to reconstruct this sequence and use it with caret's "train" function, this would likely have been time-consuming. As in previous cases, the *lambda* hyperparameter is tuned using K-fold cross-validation where the number of folds is equal to 10.

```
# Tune intact elastic net model
enet_intact <- train(select(training, -target), training[, target],
                trControl = trainControl("cv"),
                method = "glmnet")
best_alpha_intact <- enet_intact$bestTune[,"alpha"]

# Use in-depth search to find best lambda for intact elastic net
lambda_cv_intact <- cv.glmnet(as.matrix(select(training, -target)),
                              training[, target], alpha = best_alpha_intact)

best_lambda_intact <- lambda_cv_intact$lambda.min

sprintf("Intact elastic net best tune: alpha = %s, lambda = %s",
              enet_intact$bestTune[1], best_lambda_intact)
```

```
## [1] "Intact elastic net best tune: alpha = 1, lambda = 0.0995151559871381"
```

In this case, the optimal value of *alpha* is found to be 1. This means that the intact EN model is functionally equivalent to a Lasso regression model. The optimal value of lambda in this instance is found to be around 0.0995.

Next, the intact EN model is given its initial evaluation.

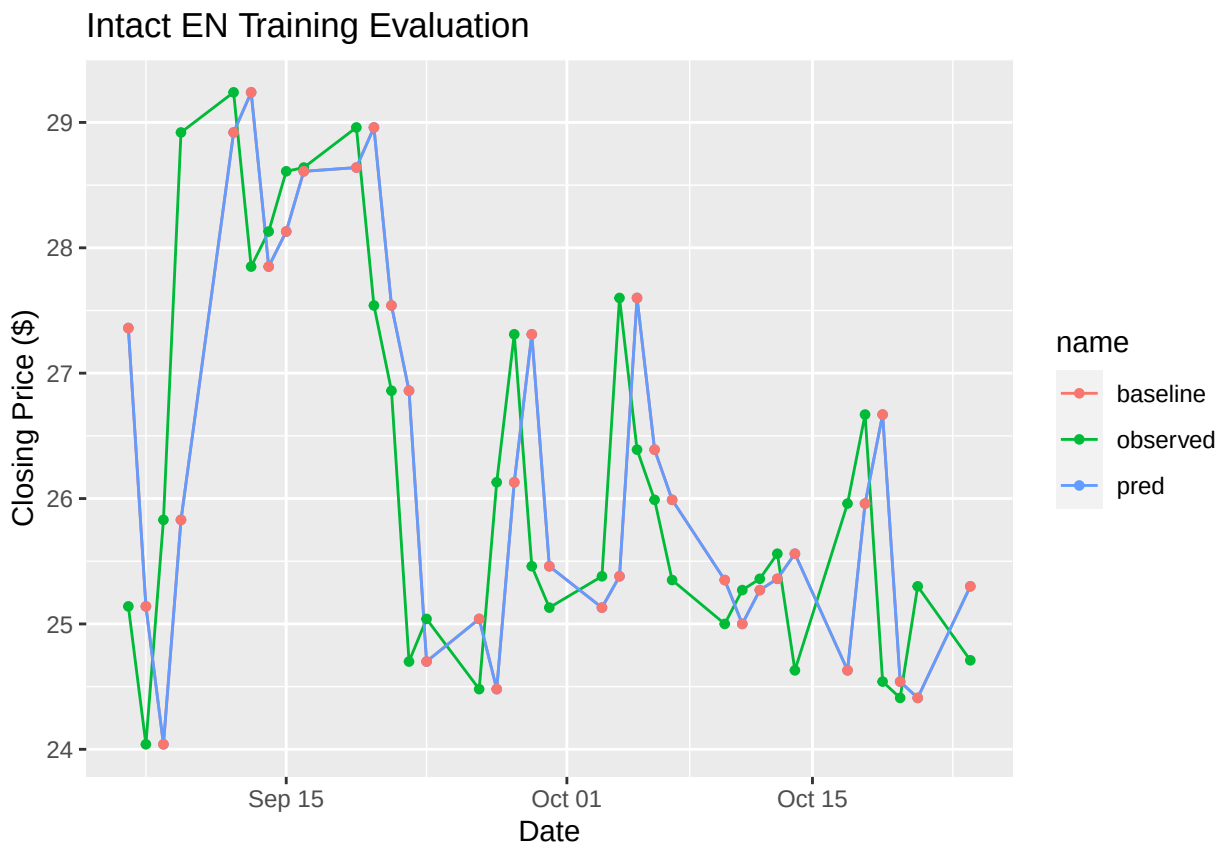
```
# Evaluate best intact elastic net model on intact training set
enet_train_intact <- get_train_pred(enet_intact$finalModel,
                                    lambda = best_lambda_intact)
rmse <- sqrt(mean((enet_train_intact$pred - enet_train_intact$observed)^2))
sprintf("Intact elastic net training RMSE: %f", rmse)
```

```
## [1] "Intact elastic net training RMSE: 2.974267"
```

The training RMSE yielded by the intact EN model is found to be identical to the baseline training RMSE of 2.974267.

As before, the intact EN model's predictions are visualized.

```
# Visualize training set predictions yielded by intact elastic net model
print(enet_train_intact[400:434,] |>
        pivot_longer(cols = c("observed", "pred", "baseline")) |>
        ggplot(aes(ind, value, color = name)) + geom_line() +
        geom_point(size=1.2) +
        labs(title="Intact EN Training Evaluation", x="Date",
             y="Closing Price ($)"))
```



The above chart demonstrates that the intact EN model's predictions appear identical to their associated baseline model predictions. This must mean that all raw outputs yielded by the model, which represent changes in closing price rather than the price itself, are close to zero. This would make the intact EN model functionally similar to the baseline model, thereby yielding a similar or identical RMSE when evaluated.

The clipped EN model is the fourth and final model to be tuned.

```
# Tune clipped elastic net model
enet_clipped <- train(select(training_clip, -target), training_clip[, target],
                trControl = trainControl("cv"),
                method = "glmnet")
best_alpha_clipped <- enet_clipped$bestTune[,"alpha"]

# Use in-depth search to find best lambda for clipped elastic net
lambda_cv_clipped <- cv.glmnet(as.matrix(select(training_clip, -target)),
                               training_clip[, target], alpha = best_alpha_clipped)

best_lambda_clipped <- lambda_cv_clipped$lambda.min

sprintf("Clipped elastic net best tune: alpha = %s, lambda = %s",
              enet_clipped$bestTune[1], best_lambda_clipped)
```

```
## [1] "Clipped elastic net best tune: alpha = 1, lambda = 0.0579494499602824"
```

As in the case of the intact EN model, the value of *alpha* is chosen to be 1. This means that, like the intact version, the clipped EN model functions as a Lasso regression model. The optimal value of *lambda* in this

case is found to be around 0.0579, representing an approximately 42% decrease when compared to the intact EN model's optimal *lambda*.

The initial evaluation of the clipped EN model is now performed.

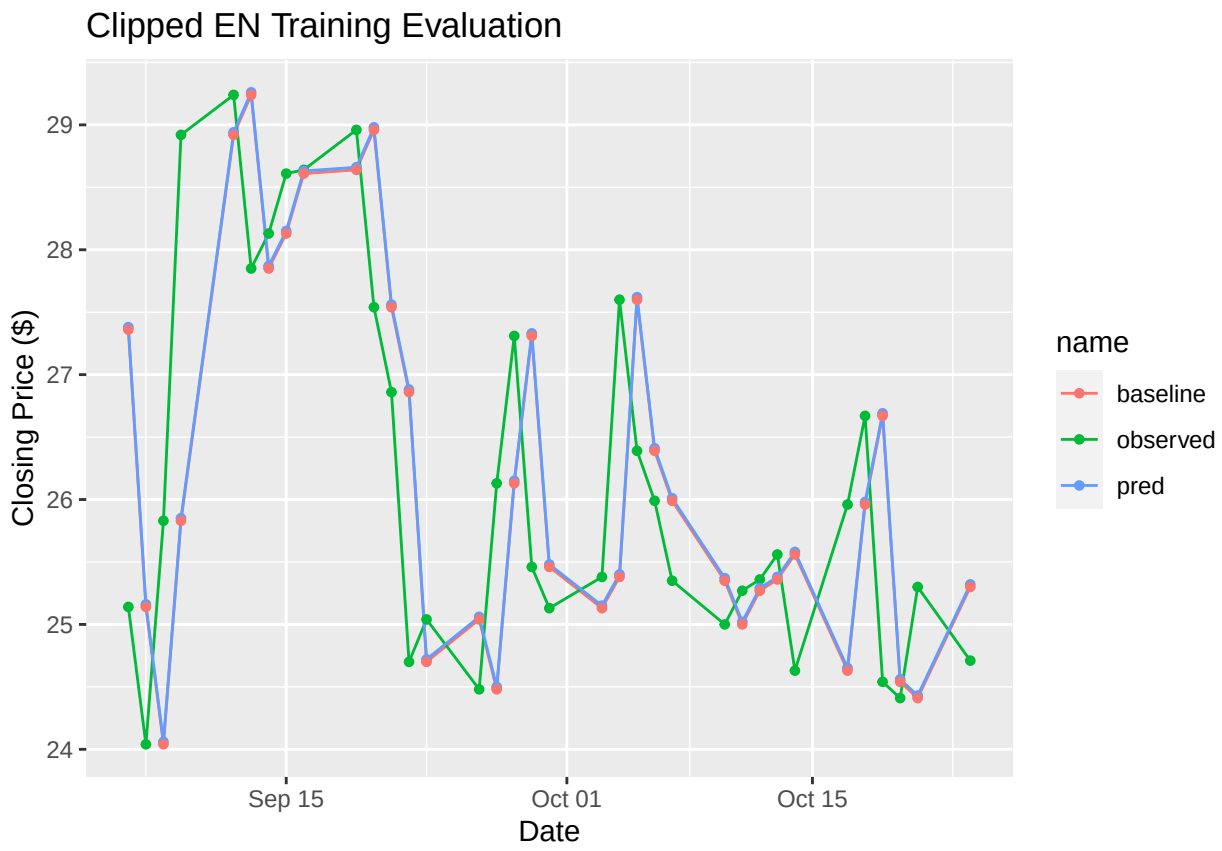
```
# Evaluate best clipped elastic net model on intact training set
enet_train_clipped <- get_train_pred(enet_clipped$finalModel,
                                     lambda = best_lambda_clipped)
rmse <- sqrt(mean((enet_train_clipped$pred - enet_train_clipped$observed)^2))
sprintf("Clipped elastic net training RMSE: %f", rmse)
```

```
## [1] "Clipped elastic net training RMSE: 2.974312"
```

The yielded training RMSE of approximately 2.974312 is the worst one out of the 4 collected, albeit by a very small margin. It is the only training RMSE found to be greater than the baseline training RMSE. As with the k-NN models, this study's outlier removal process appears to cause an increase in the training RMSE of EN models.

The clipped EN model's training data predictions are now visualized as was done for the other models.

```
# Visualize training set predictions yielded by clipped elastic net model
print(enet_train_clipped[400:434,] |>
        pivot_longer(cols = c("observed", "pred", "baseline")) |>
        ggplot(aes(ind, value, color = name)) + geom_line() +
        geom_point(size=1.2) +
        labs(title="Clipped EN Training Evaluation", x="Date",
             y="Closing Price ($)"))
```



Although the clipped EN predictions follow their associated baseline predictions almost perfectly, slight differences can be detected at all points. Unlike what was seen with the k-NN models, predictions yielded by the clipped EN model are less similar to the baseline predictions than are the intact EN predictions.

Now that that evaluation and visualization of all 4 models' performance on the training data has been completed, the models will soon be evaluated on the the test set. Although only the k-NN models outperformed the baseline model when evaluated on the intact training set, all will be investigated further for informative purposes. Before this can be done, however, the test data must be preprocessed in the same manner as the intact training data.

```
# Difference test data
test <- diff(as.matrix(test))
# Scale test data using means and SDs from training data
for(i in 1:5){test[,i] <- (test[,i] - train_means[i]) / train_sds[i]}
# Convert test data to data.table object
test <- as.data.table(test)
# Add lagged columns to test data
add_lagged(test, 14)
test <- drop_na(test)
```

It is important to note that no *target* column is added to the test data. Additionally, rather than scaling the testing columns with their own means and standard deviations as was done previously, the test data are instead scaled using the means and standard deviations calculated from the unscaled training set. It is important for new data presented to the models to be in the same scale as the data used to train them.

Next, it is worthwhile to create a testing baseline RMSE by applying the baseline model to the test data. This is done to put testing results into a more useful context.

```
# Evaluate baseline random walk model on test set
observed <- as.numeric(GME$GME.Close[467:583])
baseline_pred_test <- as.numeric(GME$GME.Close[466:582])
baseline_rmse_test <- sqrt(mean((baseline_pred_test - observed)^2))
print(sprintf("Baseline test RMSE: %f",baseline_rmse_test))
```

```
## [1] "Baseline test RMSE: 1.118241"
```

The baseline testing RMSE of approximately 1.118 is much smaller than the baseline training RMSE. This is likely because the testing data have a lower variance than the training data.

The final step before the models can be evaluated on the test set is the creation of a new function to handle raw model outputs. It is essentially the same as the function used when handling training data predictions; The only differences are that the final prediction is removed (this was not necessary during training due to the addition of the *target* column), a different set of closing values are used for addition with raw model outputs, and some minor syntactical changes stemming from the lack of a *target* column in the testing data.

```
# Convert raw model outputs into closing price predictions during testing
get_test_pred <- function(model, lambda = NULL){
  if(is.null(lambda)){
    pred <- predict(model, as.matrix(test))

  }
  else{
    pred <- predict(model, as.matrix(test),
                    s = lambda)[,1]
```

```
  }
  # Remove last prediction due to lack of corresponding observation
  pred <- pred[1: length(pred) - 1]
  pred <- (pred * train_sds[1]) + train_means[1]
  prior_vals <- as.numeric(GME$GME.Close[466:582])
  pred <- prior_vals + pred
  observed <- GME$GME.Close[467:583]

  results <- data.frame(pred = pred, observed = as.numeric(observed),
                        baseline = prior_vals,
                        ind = index(observed))
  results
}
```

Results

The models are evaluated on the test data in a very similar manner to their evaluation on the training data. First to be evaluated is the intact k-NN model.

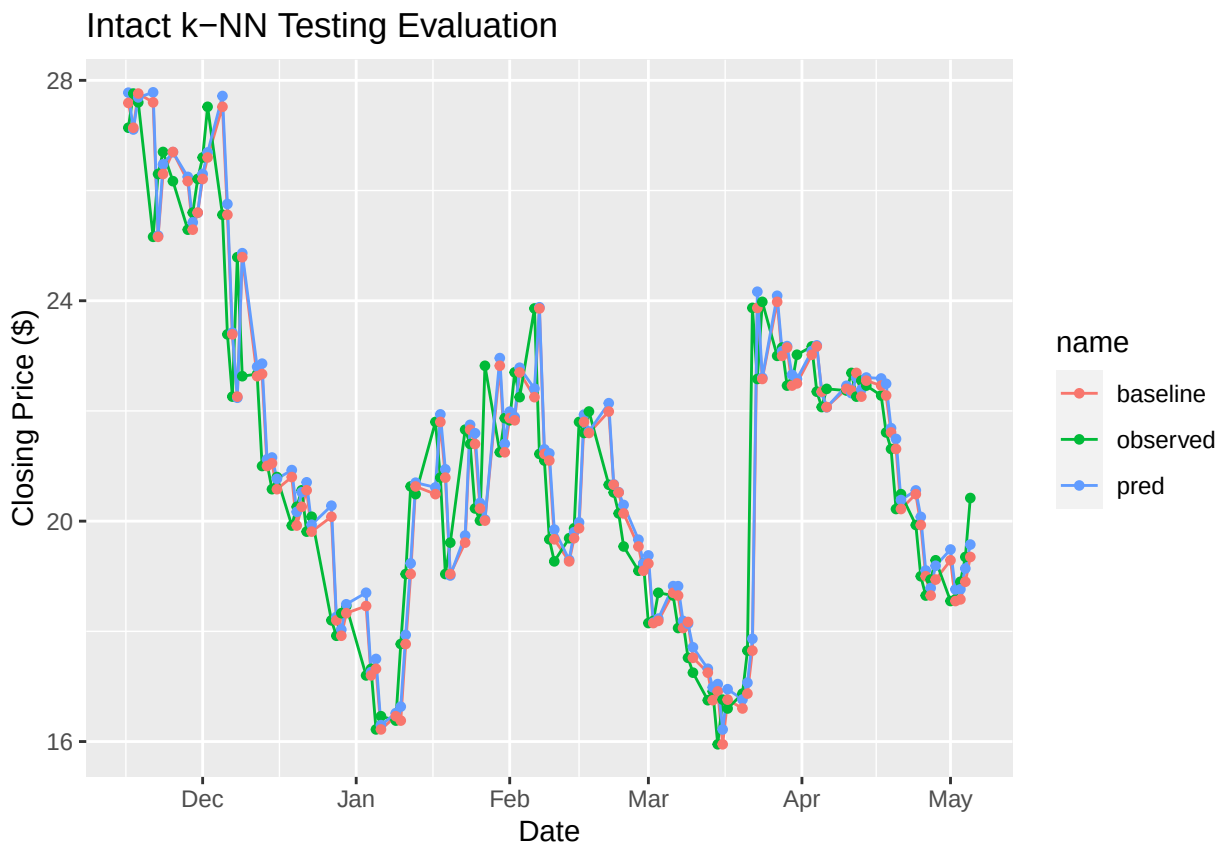
```
# Evaluate intact k-NN model on testing data
knn_test_intact <- get_test_pred(knn_intact$finalModel)
rmse <- sqrt(mean((knn_test_intact$pred - knn_test_intact$observed)^2))
sprintf("Intact k-NN test RMSE: %f", rmse)
```

```
## [1] "Intact k-NN test RMSE: 1.137575"
```

The yielded RMSE of approximately 1.138 is worse than the baseline testing RMSE.

The intact k-NN model's performance is further examined with a visualization as was done before. The below chart encompasses all of the testing data.

```
# Visualize intact k-NN model on testing data
print(knn_test_intact |> pivot_longer(cols = c("observed", "pred", "baseline"))
      |> ggplot(aes(ind, value, color = name)) + geom_line() +
        geom_point(size=1.2) +
        labs(title="Intact k-NN Testing Evaluation", x="Date",
             y="Closing Price ($)"))
```



As seen in the intact k-NN model's initial evaluation, model predictions follow baseline predictions more than observed values, although noticeable differences between model predictions and baseline predictions still exist. It appears as though their is less of a difference between the intact k-NN model's predictions and baseline predictions than what was seen in the initial evaluation.

Next to be evaluated on the test set is the clipped k-NN model.

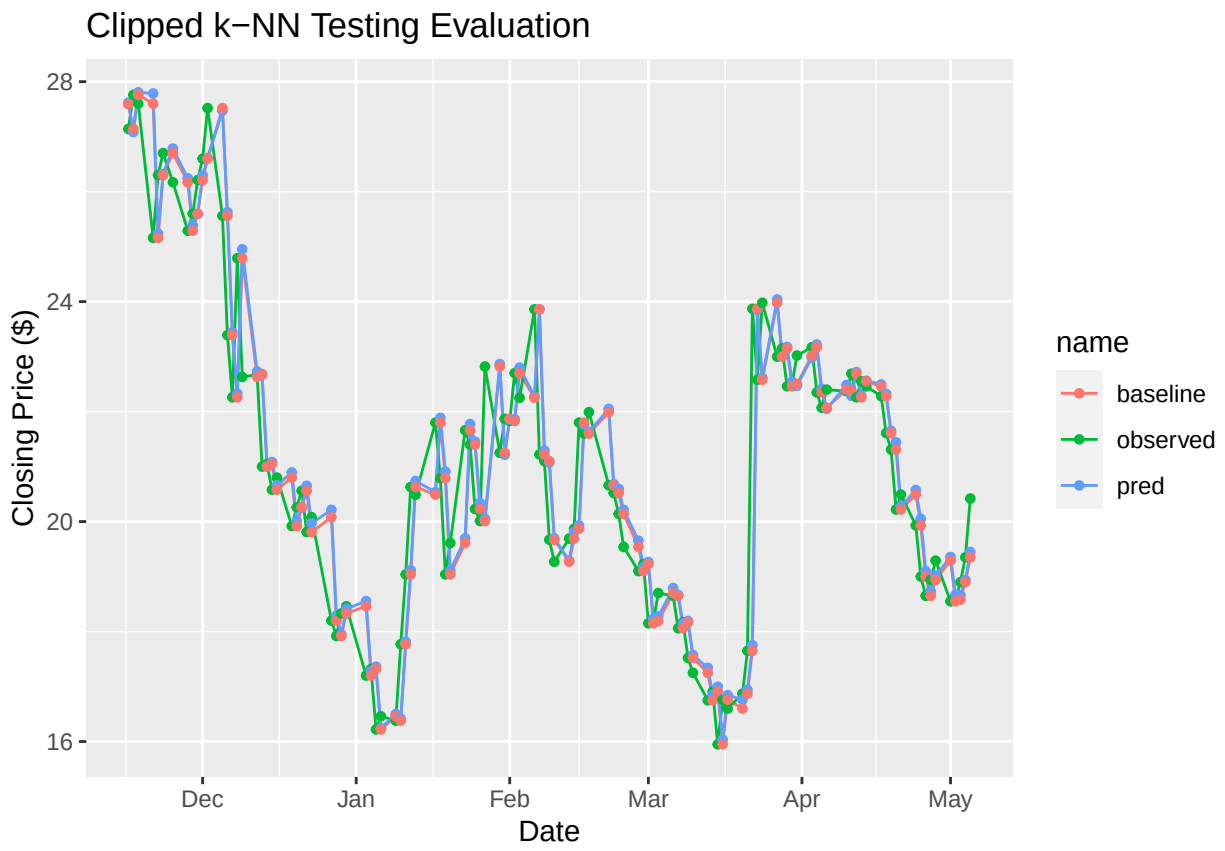
```
# Evaluate clipped k-NN model on testing data
knn_test_clipped <- get_test_pred(knn_clipped$finalModel)
rmse <- sqrt(mean((knn_test_clipped$pred - knn_test_clipped$observed)^2))
sprintf("Clipped k-NN test RMSE: %f", rmse)
```

```
## [1] "Clipped k-NN test RMSE: 1.126971"
```

As in the previous case, the yielded RMSE of around 1.1267 is worse than the baseline testing RMSE. Unlike what was seen during initial evaluations, however, the clipped k-NN model yields a smaller error than the intact version.

The clipped k-NN model's testing predictions are visualized below.

```
# Visualize clipped k-NN model on testing data
print(knn_test_clipped |> pivot_longer(cols = c("observed", "pred", "baseline"))
      |> ggplot(aes(ind, value, color = name)) + geom_line() +
        geom_point(size=1.2) +
        labs(title="Clipped k-NN Testing Evaluation", x="Date",
             y="Closing Price ($)"))
```



As was noted during initial evaluations, the clipped k-NN model's predictions follow the baseline prediction more closely than predictions from the intact k-NN model. Like the intact k-NN model, the clipped version seems to follow the baseline model more closely during testing than it did during initial evaluations.

The intact EN is the third model to be tested.

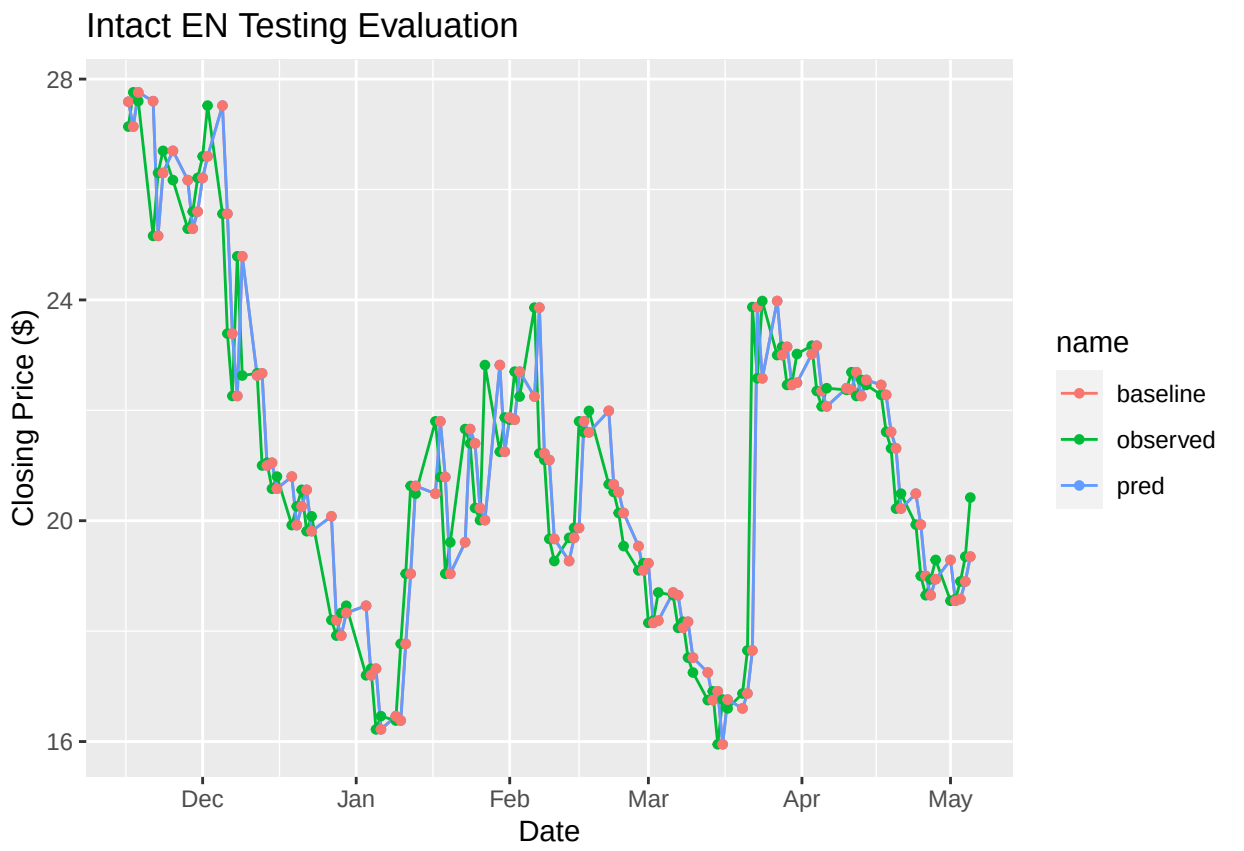
```
# Evaluate intact elastic net model on testing data
enet_test_intact <- get_test_pred(enet_intact$finalModel,
                                  lambda = best_lambda_intact)
rmse <- sqrt(mean((enet_test_intact$pred - enet_test_intact$observed)^2))
sprintf("Intact elastic net test RMSE: %f", rmse)
```

```
## [1] "Intact elastic net test RMSE: 1.118236"
```

The yielded RMSE of 1.118236 is marginally better than the baseline testing RMSE 1.118241, although this is unlikely to be significant. Unlike what was seen during initial evaluations, the intact EN model performs better than the intact k-NN model.

As in the previous cases, the intact EN model's predictions are shown below.

```
# Visualize intact elastic net model on testing data
print(enet_test_intact |> pivot_longer(cols = c("observed", "pred", "baseline"))
      |> ggplot(aes(ind, value, color = name)) + geom_line() +
        geom_point(size=1.2) +
        labs(title="Intact EN Testing Evaluation", x="Date",
             y="Closing Price ($)"))
```



As was seen during initial evaluations, the intact EN model's predictions appear indistinguishable baseline predictions, suggesting that the model's raw outputs are close to zero. Although the slight discrepancy between the intact EN model's RMSE and the baseline testing RMSE indicates that the two sets of predictions are not completely identical, this cannot be visually confirmed.

The final model to be evaluated on the test set is the clipped EN model.

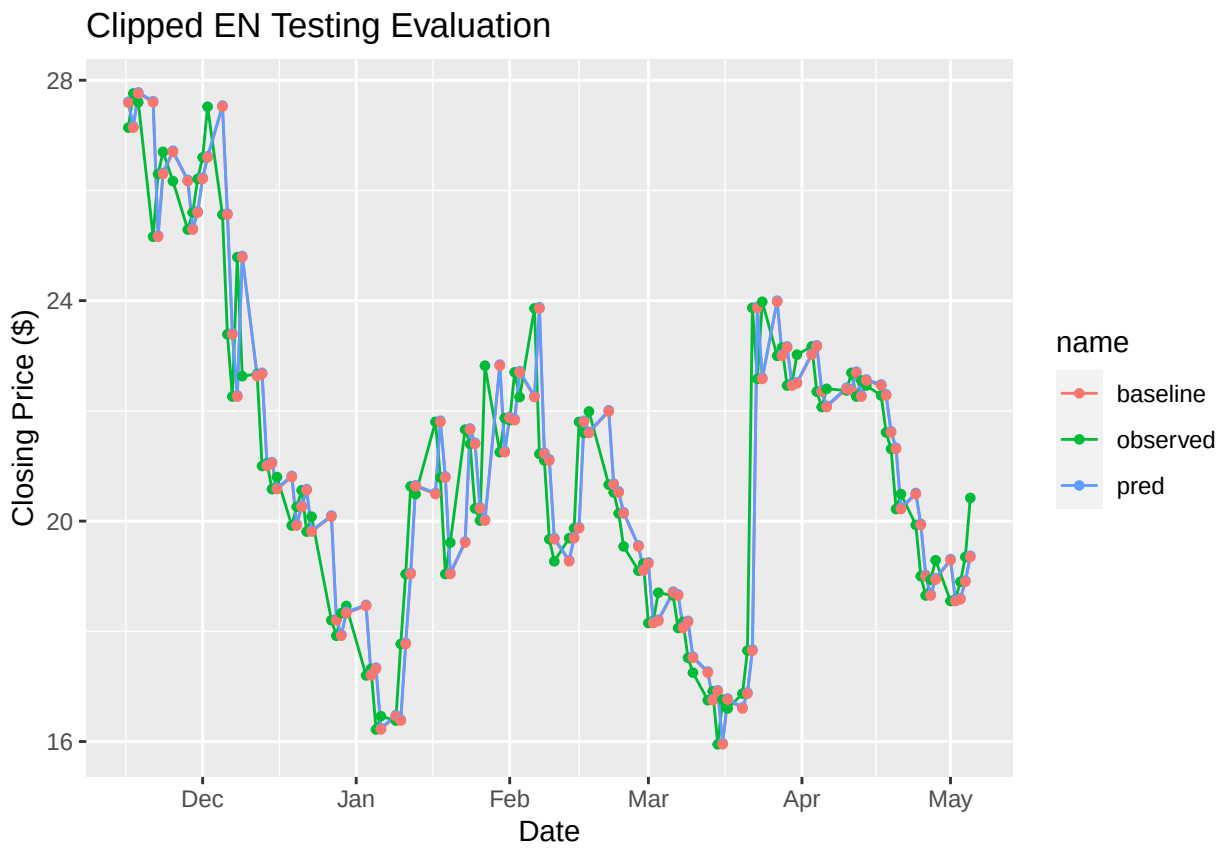
```
# Evaluate clipped elastic net model on testing data
enet_test_clipped <- get_test_pred(enet_clipped$finalModel,
                                   lambda = best_lambda_clipped)
rmse <- sqrt(mean((enet_test_clipped$pred - enet_test_clipped$observed)^2))
sprintf("Clipped elastic net test RMSE: %f", rmse)
```

```
## [1] "Clipped elastic net test RMSE: 1.119552"
```

The yielded RMSE of approximately 1.120 is larger than the baseline testing RMSE. As was seen during initial evaluations, the clipped EN model demonstrated slightly worse performance than the intact version.

The clipped EN model's predictions are visualized below.

```
# Visualize clipped elastic net model on testing data
print(enet_test_clipped |> pivot_longer(cols = c("observed", "pred", "baseline"))
      |> ggplot(aes(ind, value, color = name)) + geom_line() +
        geom_point(size=1.2) +
        labs(title="Clipped EN Testing Evaluation", x="Date",
             y="Closing Price ($)"))
```



As was noted during initial evaluations, the clipped EN model's predictions follow the baseline predictions extremely closely, but not perfectly. Also like what was seen earlier, the clipped EN model's predictions differ more from the baseline predictions than do the intact EN model's predictions, again running counter to what is observed when the k-NN models are compared.

Conclusion

Numerous steps were taken over the course of this study to train and evaluate ML models capable of predicting GME's closing price. After the raw stock data were loaded, they were split into training and test sets. A random walk model was then chosen as the baseline model. After this, the training set was both differenced and standardized. Lagged columns were then added to the training set, causing each row to contain 15 time-steps worth of data. Subsequently, a clipped version of the training set was created by subjecting the training set to outlier removal. The clipped and intact versions of the training set were each used to train both a k-NN model and an EN model, yielding a total of 4 models to be evaluated. These models were initially evaluated on the training set, then finally evaluated on the test set.

Before the final performance of the ML models is discussed, there are are a few observations regarding their behavior worth mentioning. First, all 4 models appeared to be severely underfitted. This is most prominent when considering the intact EN model, where all raw outputs were extremely close to zero. Although visualizations where final predictions of closing price closely follow baseline predictions may have appeared to be instances of overfitting, it is important to remember that the raw model outputs used to calculate final predictions correspond to changes in closing price, rather than the closing price itself. Although it was thought that colinearity among model inputs would make the models prone to overfitting, this seemed to be outweighed by the complex relationship between model inputs and their associated target values. Another observation worth noting is the effect of outlier removal on k-NN model performance. During initial evaluations, it appeared as though outlier removal hindered model performance. During testing, however, a

small improvement in RMSE was seen when comparing the clipped k-NN model to the intact version. This is likely because the the clipped training set is more representative of the relatively low-variance test set than is the intact training set. The fact that clipped k-NN predictions appeared to follow baseline predictions more closely than do intact k-NN predictions suggests that, in this case, outlier removal served as a regularizing force on k-NN models. It is interesting to note that this relationship did not seem to be present in the case of EN models. The intact EN model made predictions which were more in accordance with baseline predictions than those made by the clipped EN model. This is likely because of the EN model's linear nature; The greater presence of both positive and negative outliers in the intact training set forced the model to shrink its predictions toward zero.

Regarding the final performance of the ML models, none of the 4 models explored in this study were capable of significantly outperforming the baseline random walk model when evaluated on the test data. Although the intact EN model testing RMSE is technically smaller than the baseline testing RMSE, the difference between them is likely insignificant. The potential effectiveness of all models was likely reduced due to the large difference in variance between the training data and the test data. The non-stationary relationship between variance and time within the training data likely interfered with successful modeling as well. Another factor contributing to modeling difficulty was the high degree of noise present in the differenced training data. This is a known drawback of making the data more stationary via differencing. Another clear limitation of the present study is the relatively small quantity of data used. This may be addressed in the future when more data are available. Finding GME data with a higher frequency could also potentially mitigate this issue. It should also be noted that the outlier removal method used in this study is very basic. The use of a more advanced outlier removal method such as local outlier factor (LOF) may serve to improve results in future research. Another limitation in this study is that only 2 types of ML models were explored. Future research should consider a wider array of model types.