Option pricing

Ayan Goswami

2025-07-20

Contents

Data load

```
file_list <- c(
  "../data/processed/option_data_with_future_prices_7_8_1.csv",
  "../data/processed/option_data_with_future_prices_7_8_2.csv"
)

# Read and bind all files
data <- do.call(rbind, lapply(file_list, read.csv))
```

```
seed_num = 1
set.seed(seed_num) # reproducibility
ind = sample(1:nrow(data),0.75*nrow(data))
train = data[ind,]
val = data[-ind,]
```

Exploratory Data Analysis

1. Data Preparation and Required Libraries

We begin by loading essential statistical and visualization libraries for options data analysis. These tools enable us to examine complex relationships between option characteristics and their behavior over time.

```
library(ggplot2)
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(tidyr)
library(corrplot)
```

```
## corrplot 0.95 loaded
```

```
library(moments)
```

2. Summary Statistics

First, we examine the basic statistical properties of our training dataset to understand the central tendencies, dispersion, and distributions of key variables in our options data.

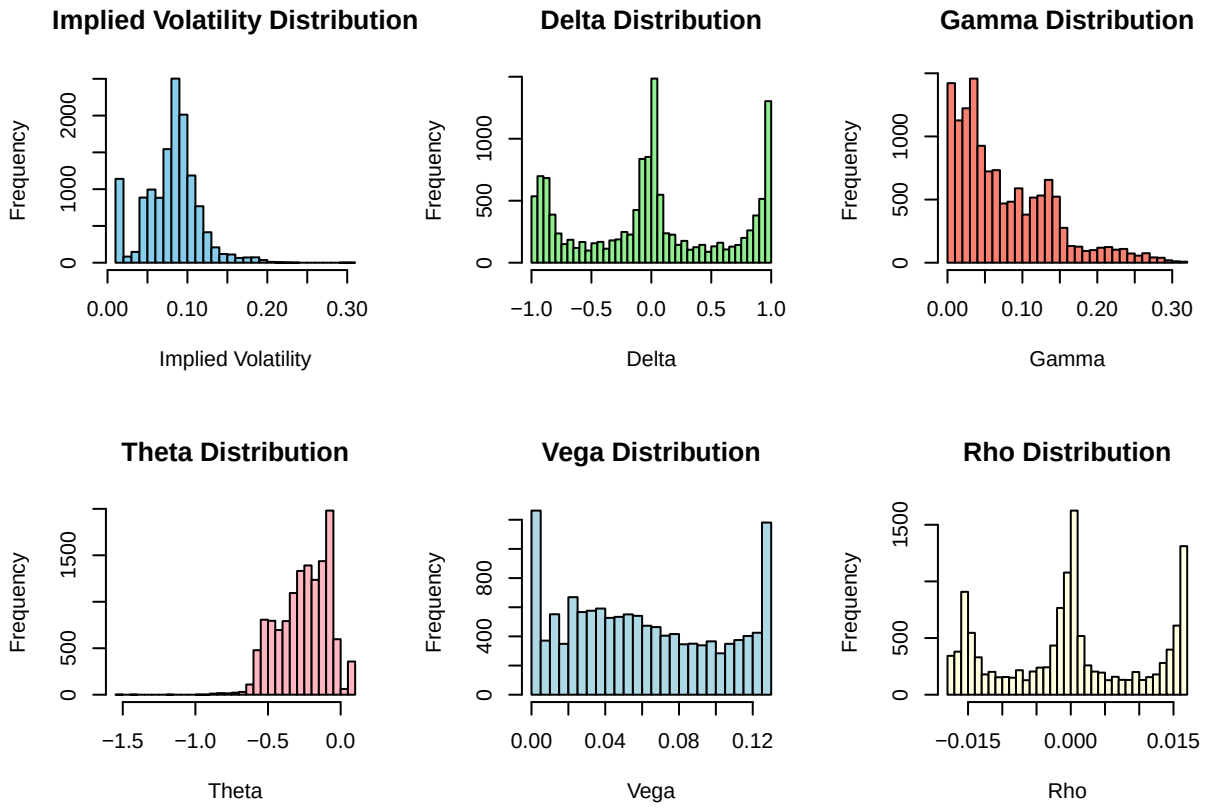
```r
summary(train)
```

```
##   timestamp            symbol          option_type            strike
##  Length:13267       Length:13267       Length:13267       Min.   :613.0
##  Class :character   Class :character   Class :character   1st Qu.:617.0
##  Mode  :character   Mode  :character   Mode  :character   Median :621.0
##                                                           Mean   :620.7
##                                                           3rd Qu.:624.0
##                                                           Max.   :628.0
##
##  latest_trade_price latest_quote_bid latest_quote_ask implied_volatility
##  Min.   :0.010      Min.   :0.000    Min.   :0.010    Min.   :0.01000
##  1st Qu.:0.100      1st Qu.:0.090    1st Qu.:0.110    1st Qu.:0.06090
##  Median :0.930      Median :0.900    Median :0.930    Median :0.08381
##  Mean   :2.014      Mean   :1.988    Mean   :2.035    Mean   :0.08034
##  3rd Qu.:3.730      3rd Qu.:3.670    3rd Qu.:3.750    3rd Qu.:0.09834
##  Max.   :7.970      Max.   :7.380    Max.   :8.380    Max.   :0.30114
##
##      delta               gamma              theta               vega
##  Min.   :-1.000000   Min.   :0.00000   Min.   :-1.52015   Min.   :0.00000
##  1st Qu.:-0.479318   1st Qu.:0.02662   1st Qu.:-0.37826   1st Qu.:0.02591
##  Median :-0.016678   Median :0.05622   Median :-0.23366   Median :0.05575
##  Mean   : 0.003187   Mean   :0.07540   Mean   :-0.25542   Mean   :0.06059
##  3rd Qu.: 0.507421   3rd Qu.:0.11816   3rd Qu.:-0.11326   3rd Qu.:0.09590
##  Max.   : 1.000000   Max.   :0.31968   Max.   : 0.08602   Max.   :0.12966
##
##       rho                 price         moneyness          standardized_price
##  Min.   :-1.720e-02   Min.   :619.6   Min.   :-1.161e-02   Min.   :0.00000
##  1st Qu.:-8.174e-03   1st Qu.:620.4   1st Qu.:-5.721e-03   1st Qu.:0.02846
##  Median :-2.843e-04   Median :620.8   Median :-3.221e-05   Median :0.30188
##  Mean   :-1.187e-06   Mean   :620.7   Mean   : 1.078e-05   Mean   :0.39325
##  3rd Qu.: 8.605e-03   3rd Qu.:621.1   3rd Qu.: 5.858e-03   3rd Qu.:0.75529
##  Max.   : 1.696e-02   Max.   :621.6   Max.   : 1.161e-02   Max.   :1.00000
##
##  price_diff_3_periods price_diff_6_periods price_diff_12_periods
##  Min.   :0.3333       Min.   :0.3333       Min.   :0.3333
##  1st Qu.:0.9707       1st Qu.:0.9583       1st Qu.:0.9480
##  Median :1.0000       Median :1.0000       Median :1.0000
##  Mean   :1.0229       Mean   :1.0250       Mean   :1.0256
##  3rd Qu.:1.0210       3rd Qu.:1.0287       3rd Qu.:1.0367
##  Max.   :3.0000       Max.   :3.1667       Max.   :3.6667
##  NA's   :49           NA's   :49           NA's   :49
##  price_diff_15_periods price_diff_30_periods
##  Min.   :0.3333        Min.   :0.3333
##  1st Qu.:0.9424        1st Qu.:0.9183
##  Median :1.0000        Median :1.0000
##  Mean   :1.0256        Mean   :1.0253
##  3rd Qu.:1.0410        3rd Qu.:1.0643
##  Max.   :3.6000        Max.   :3.3000
##  NA's   :49            NA's   :49
```

3. Distribution of Option Greeks

Option Greeks quantify different dimensions of risk in options trading. Their distributions provide insights into the risk characteristics of our dataset and potential biases in market pricing.

```
par(mfrow=c(2,3))
hist(train$implied_volatility, main="Implied Volatility Distribution", xlab="Implied Volatility", col="
hist(train$delta, main="Delta Distribution", xlab="Delta", col="lightgreen", breaks=30)
hist(train$gamma, main="Gamma Distribution", xlab="Gamma", col="salmon", breaks=30)
hist(train$theta, main="Theta Distribution", xlab="Theta", col="lightpink", breaks=30)
hist(train$vega, main="Vega Distribution", xlab="Vega", col="lightblue", breaks=30)
hist(train$rho, main="Rho Distribution", xlab="Rho", col="lightyellow", breaks=30)
```



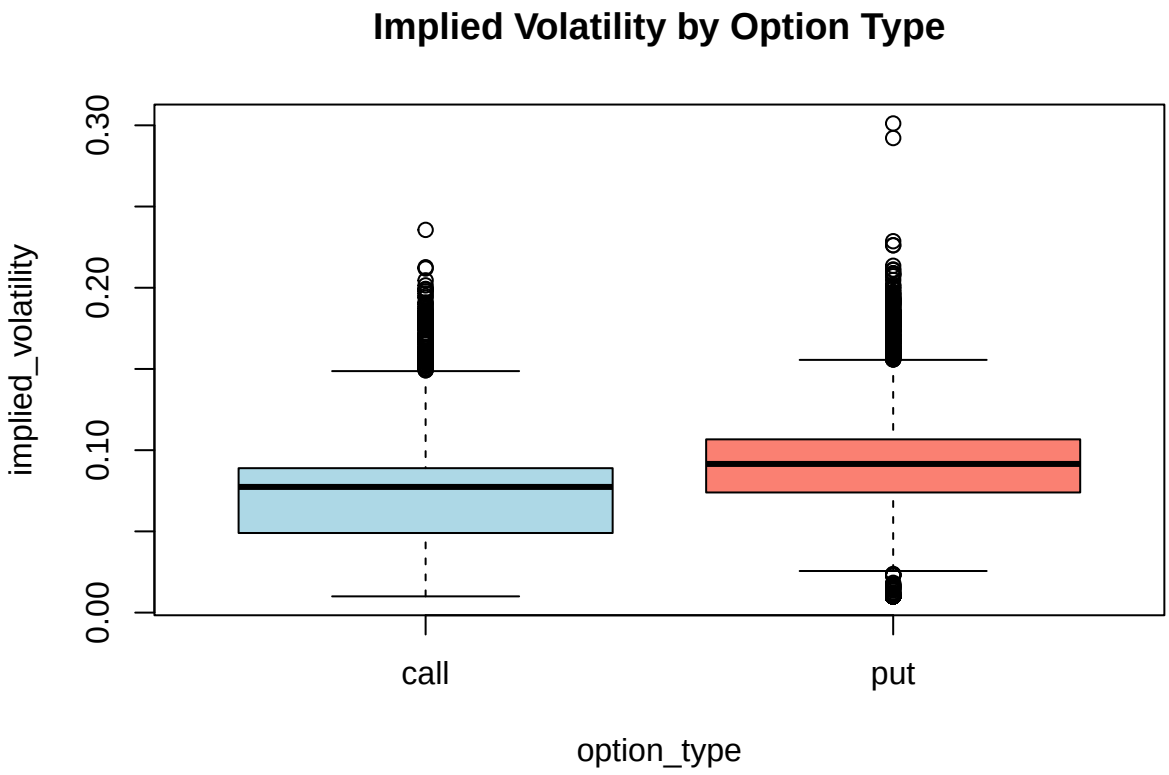

```
par(mfrow=c(1,1))
```

4. Call vs Put Option Analysis

Separating call and put options allows us to analyze their distinct characteristics. This separation is fundamental as these option types have different payoff structures and risk profiles.

```
call_data <- train[train$option_type == "call", ]
put_data <- train[train$option_type == "put", ]

boxplot(implied_volatility ~ option_type, data = train,
```

```
        main = "Implied Volatility by Option Type",
        col = c("lightblue", "salmon"))
```

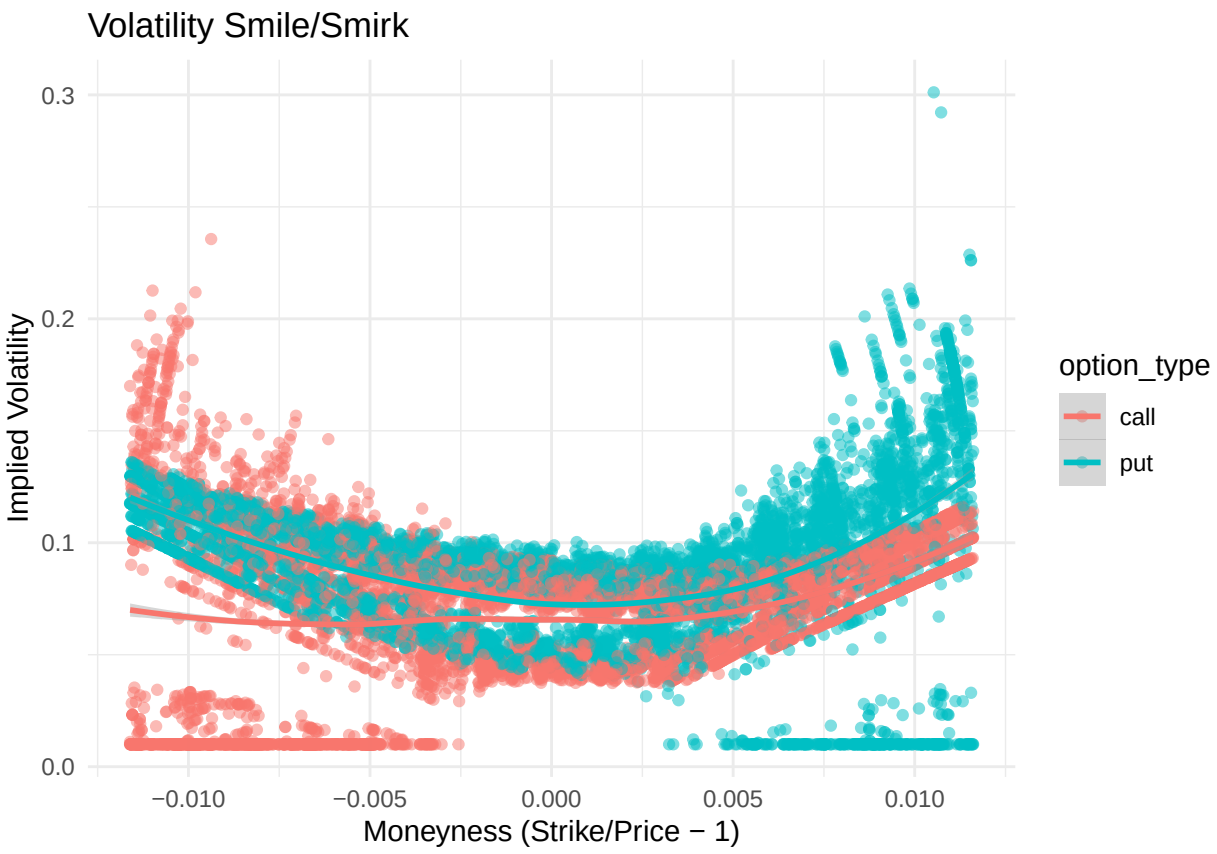


5. Volatility Smile Analysis

The volatility smile is a key phenomenon in options markets where implied volatility varies with strike price. This pattern reflects market expectations about future price movements and tail risk, contradicting the constant volatility assumption in the Black-Scholes model.

```
ggplot(train, aes(x = moneyness, y = implied_volatility, color = option_type)) +
  geom_point(alpha = 0.5) +
  geom_smooth(method = "loess", se = TRUE) +
  labs(title = "Volatility Smile/Smirk",
       x = "Moneyness (Strike/Price - 1)",
       y = "Implied Volatility") +
  theme_minimal()
```

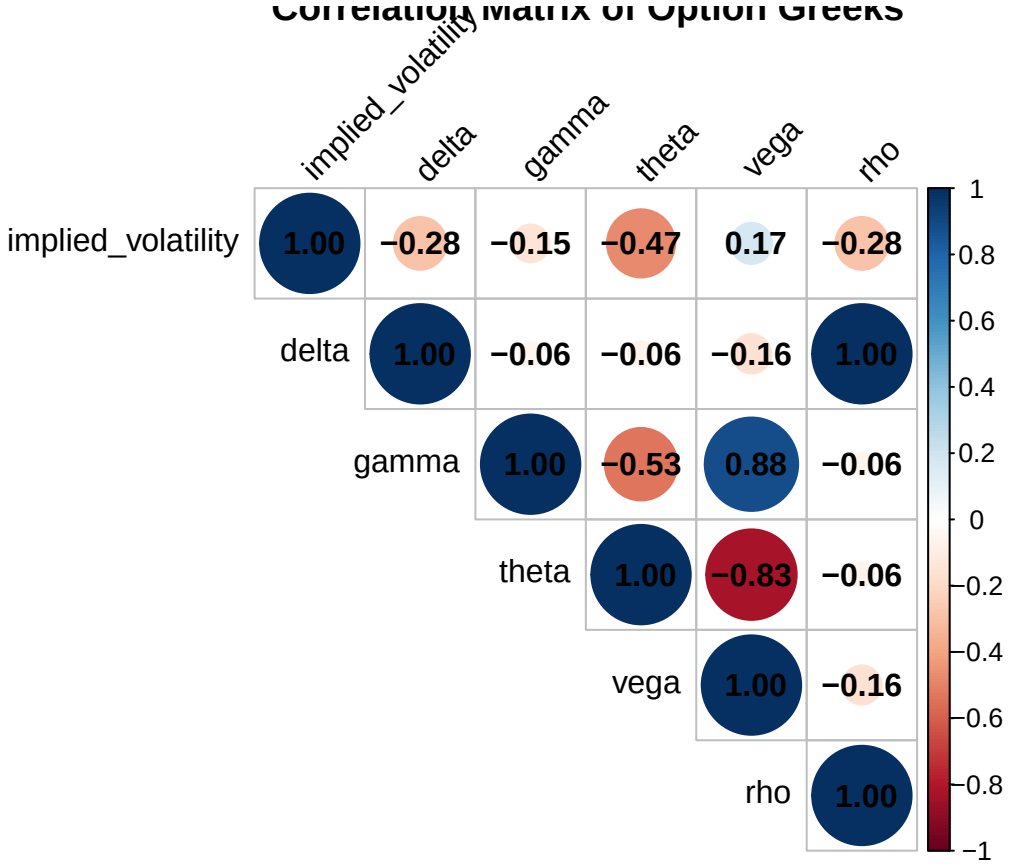
```
## `geom_smooth()` using formula = 'y ~ x'
```



6. Correlation Analysis of Option Greeks

The relationships between option Greeks provide insights into how different risk dimensions interact. Understanding these correlations is essential for constructing balanced options strategies and managing risk exposures.

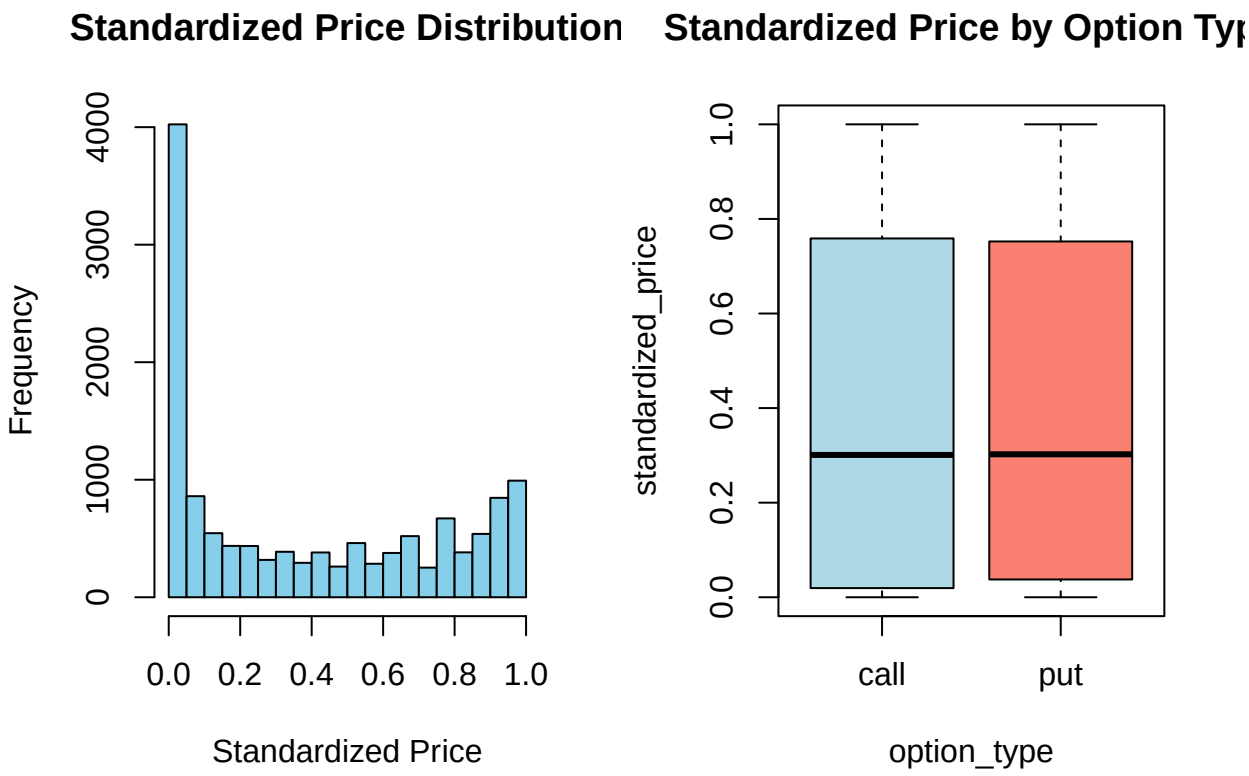
```
correlation_matrix <- cor(train[, c("implied_volatility", "delta", "gamma", "theta", "vega", "rho")],
                          use = "complete.obs")
corrplot(correlation_matrix, method = "circle", type = "upper",
         tl.col = "black", tl.srt = 45, addCoef.col = "black",
         title = "Correlation Matrix of Option Greeks")
```



7. Standardized Price Distribution Analysis

Standardized price is a key metric for comparing options across different strikes and expirations. Understanding its distribution helps identify potential pricing anomalies and opportunities.

```
par(mfrow=c(1,2))
hist(train$standardized_price, main="Standardized Price Distribution", xlab="Standardized Price", col="
boxplot(standardized_price ~ option_type, data = train,
        main = "Standardized Price by Option Type",
        col = c("lightblue", "salmon"))
```




```
par(mfrow=c(1,1))
```

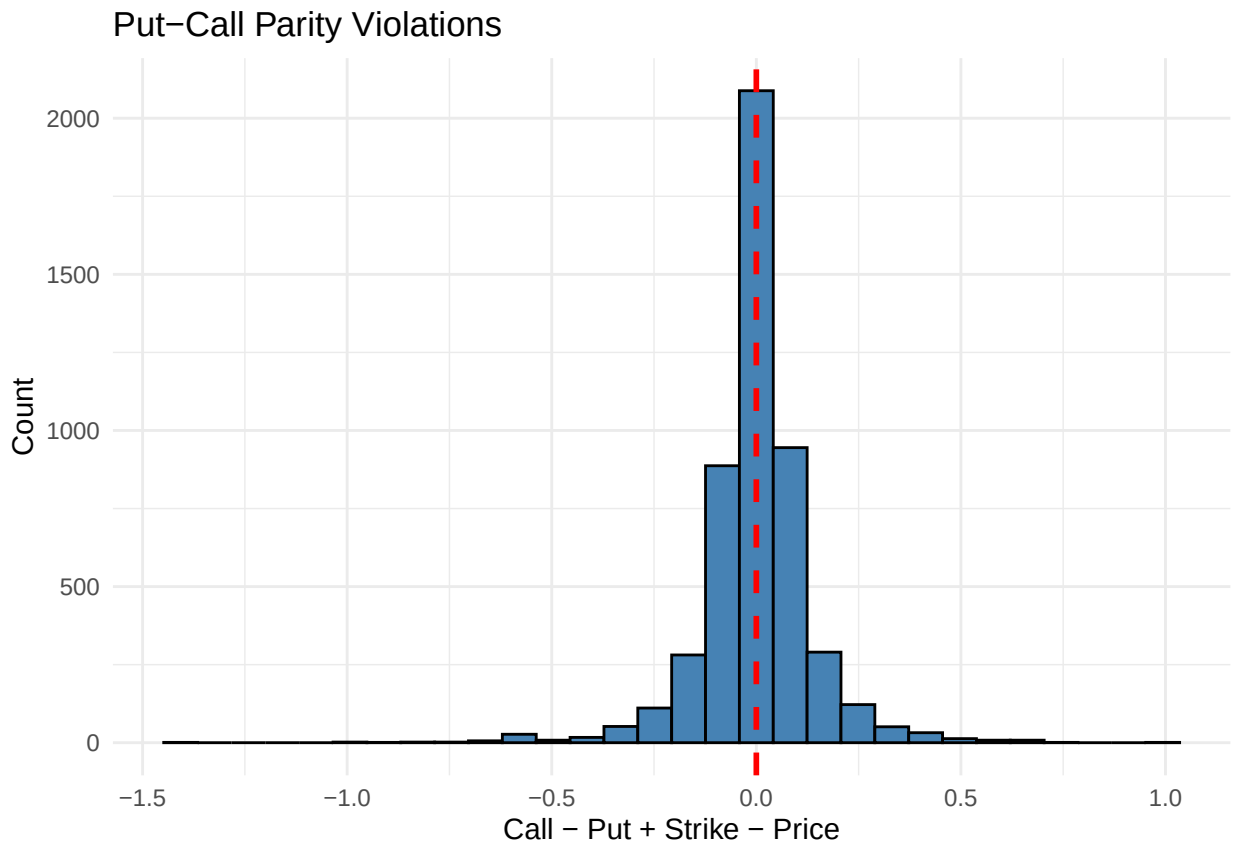
11. Put-Call Parity Analysis

Put-call parity is a fundamental principle in options pricing theory. Deviations from parity may indicate arbitrage opportunities or market inefficiencies. The formula states that C - P = S - K*e^(-rT), which we rearrange to* $C - P + K - S = K(1\text{-e}^{\wedge}(\text{-rT}))$.

```
parity_check <- train %>%
  group_by(timestamp, strike) %>%
  filter(n() == 2 & length(unique(option_type)) == 2) %>%
  pivot_wider(id_cols = c(timestamp, strike, price),
              names_from = option_type,
              values_from = latest_trade_price) %>%
  mutate(parity_diff = call - put + strike - price)

ggplot(parity_check, aes(x = parity_diff)) +
  geom_histogram(bins = 30, fill = "steelblue", color = "black") +
  geom_vline(xintercept = 0, color = "red", linetype = "dashed", size = 1) +
  labs(title = "Put-Call Parity Violations",
       x = "Call - Put + Strike - Price",
       y = "Count") +
  theme_minimal()
```

```
## Warning: Using 'size' aesthetic for lines was deprecated in ggplot2 3.4.0.
```

```
## i Please use ‘linewidth‘ instead.
## This warning is displayed once every 8 hours.
## Call ‘lifecycle::last_lifecycle_warnings()‘ to see where this warning was
## generated.
```




```
summary(parity_check$parity_diff)
```

```
##       Min.    1st Qu.     Median       Mean    3rd Qu.       Max.
## -1.4100000 -0.0500000  0.0000000  0.0005155  0.0600000  0.9900000
```

```
sd(parity_check$parity_diff, na.rm = TRUE)
```

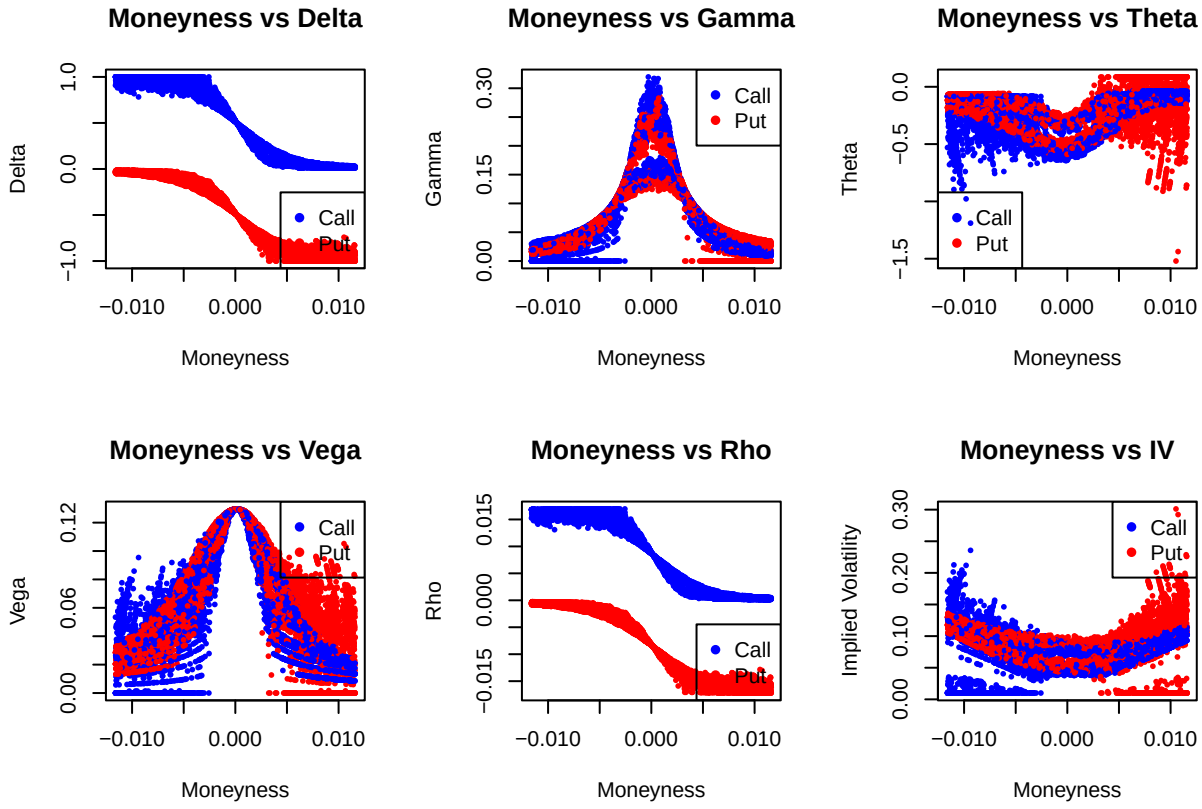
```
## [1] 0.135748
```

12. Moneyness and Option Greeks Relationships

Moneyness (the relationship between strike price and underlying price) significantly affects option behavior. These plots visualize how option Greeks vary with moneyness, providing insights into risk profiles across different strike prices.

```
par(mfrow=c(2,3))
plot(train$moneyness, train$delta, main="Moneyness vs Delta", xlab="Moneyness", ylab="Delta", col=ifels
legend("bottomright", legend=c("Call", "Put"), col=c("blue", "red"), pch=16)
```

```r
plot(train$moneyness, train$gamma, main="Moneyness vs Gamma", xlab="Moneyness", ylab="Gamma", col=ifelse
legend("topright", legend=c("Call", "Put"), col=c("blue", "red"), pch=16)

plot(train$moneyness, train$theta, main="Moneyness vs Theta", xlab="Moneyness", ylab="Theta", col=ifelse
legend("bottomleft", legend=c("Call", "Put"), col=c("blue", "red"), pch=16)

plot(train$moneyness, train$vega, main="Moneyness vs Vega", xlab="Moneyness", ylab="Vega", col=ifelse(ti
legend("topright", legend=c("Call", "Put"), col=c("blue", "red"), pch=16)

plot(train$moneyness, train$rho, main="Moneyness vs Rho", xlab="Moneyness", ylab="Rho", col=ifelse(train
legend("bottomright", legend=c("Call", "Put"), col=c("blue", "red"), pch=16)

plot(train$moneyness, train$implied_volatility, main="Moneyness vs IV", xlab="Moneyness", ylab="Implied
legend("topright", legend=c("Call", "Put"), col=c("blue", "red"), pch=16)
```

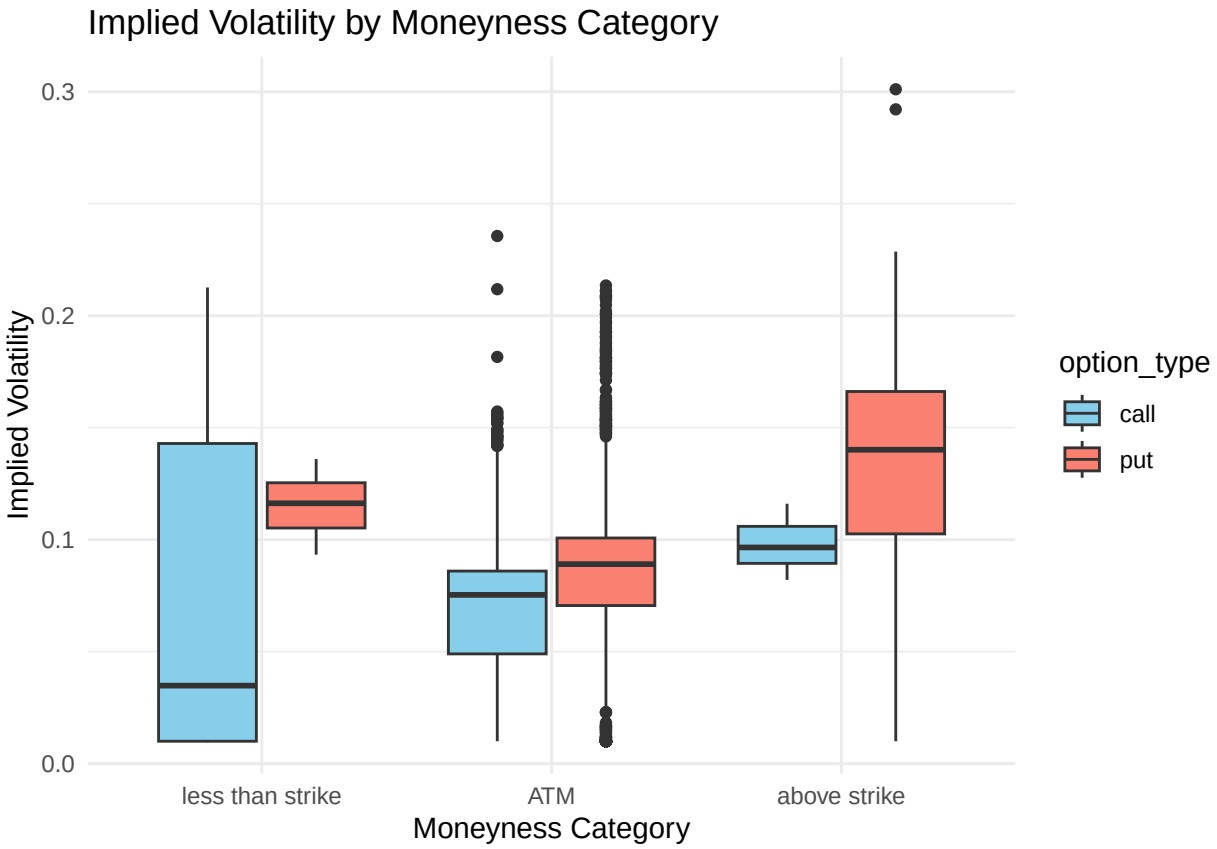



```r
par(mfrow=c(1,1))
```

13. Implied Volatility Term Structure

The term structure of implied volatility across different moneyness categories reveals how the market prices risk at different strike levels. This analysis helps identify potential mispricing and trading opportunities.

```r
train$moneyness_category <- cut(train$moneyness,
                                breaks = c(-0.05,-0.01, 0.01, 0.05),
                                labels = c("less than strike", "ATM", "above strike"))

ggplot(train, aes(x = factor(moneyness_category), y = implied_volatility, fill = option_type)) +
  geom_boxplot() +
  labs(title = "Implied Volatility by Moneyness Category",
       x = "Moneyness Category",
       y = "Implied Volatility") +
  theme_minimal() +
  scale_fill_manual(values = c("call" = "skyblue", "put" = "salmon"))
```

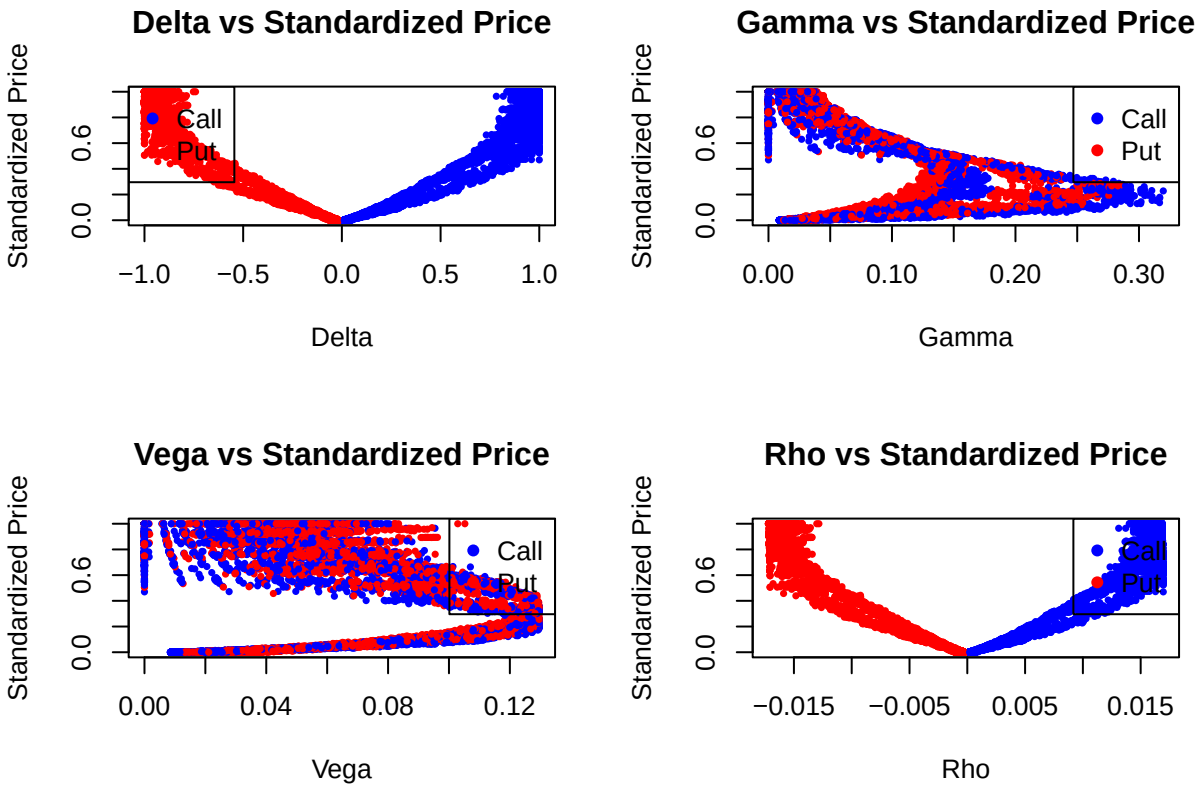


14. Relationship Between Greeks and Standardized Price

Understanding how option Greeks relate to standardized price is crucial for developing pricing models. These relationships form the foundation for our predictive modeling approach.

```r
par(mfrow=c(2,2))
plot(train$delta, train$standardized_price, main="Delta vs Standardized Price",
     xlab="Delta", ylab="Standardized Price",
     col=ifelse(train$option_type=="call", "blue", "red"), pch=16, cex=0.6)
legend("topleft", legend=c("Call", "Put"), col=c("blue", "red"), pch=16)

plot(train$gamma, train$standardized_price, main="Gamma vs Standardized Price",
```

```
     xlab="Gamma", ylab="Standardized Price",
     col=ifelse(train$option_type=="call", "blue", "red"), pch=16, cex=0.6)
legend("topright", legend=c("Call", "Put"), col=c("blue", "red"), pch=16)

plot(train$vega, train$standardized_price, main="Vega vs Standardized Price",
     xlab="Vega", ylab="Standardized Price",
     col=ifelse(train$option_type=="call", "blue", "red"), pch=16, cex=0.6)
legend("topright", legend=c("Call", "Put"), col=c("blue", "red"), pch=16)

plot(train$rho, train$standardized_price, main="Rho vs Standardized Price",
     xlab="Rho", ylab="Standardized Price",
     col=ifelse(train$option_type=="call", "blue", "red"), pch=16, cex=0.6)
legend("topright", legend=c("Call", "Put"), col=c("blue", "red"), pch=16)
```




```
par(mfrow=c(1,1))
```

KNN Modeling for Standardized Price Prediction

1. Model Preparation

We use K-Nearest Neighbors to predict standardized prices based on option Greeks (excluding theta because of 0dte), implied volatility, and moneyness. This non-parametric approach can capture complex, non-linear relationships in option pricing. Usually $k = \sqrt{N}$ where N = number of rows in the trainset. Here we will evaluate the performance with the validation data

```r
library(caret)
```

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

```r
library(class)


predict_standardized_price <- function(k_value, predictors, validation=val) {
  X <- train[, predictors]
  y <- train$standardized_price


  X_train <- scale(X)
  y_train <- y
  X_val <- scale(validation[, predictors])
  y_val <- validation$standardized_price

  knn_pred <- knn(train = X_train, test = X_val, cl = y_train, k = k_value)

  knn_pred_numeric <- as.numeric(as.character(knn_pred))

  rmse <- sqrt(mean((knn_pred_numeric - y_val)^2, na.rm = TRUE))

  sst <- sum((y_val - mean(y_val, na.rm = TRUE))^2, na.rm = TRUE)
  sse <- sum((y_val - knn_pred_numeric)^2, na.rm = TRUE)
  r_squared <- 1 - (sse / sst)

  return(list(rmse = rmse, r_squared = r_squared, predictions = knn_pred_numeric, actual = y_val))
}
```

Subset selection

To select the best subset of predictors with the lowest RMSE, we use exhaustive subset evaluation

```r
library(gtools)
baseline_k = (nrow(train)^(0.5))
all_predictors <- c("implied_volatility", "delta", "gamma", "vega", "rho", "moneyness", "theta")

# Store results
results <- data.frame(predictors = character(), rmse = numeric(),
                      r_squared = numeric(), stringsAsFactors = FALSE)

for (i in 1:length(all_predictors)) {
  subsets <- combinations(n = length(all_predictors), r = i, v = all_predictors)

  for (j in 1:nrow(subsets)) {
    current_predictors <- subsets[j, ]

    # Try to run prediction and safely catch any errors
    tryCatch({
      pred_result <- predict_standardized_price(k_value = baseline_k,
                                                predictors = current_predictors)
```

```
      results <- rbind(results, data.frame(
        predictors = paste(current_predictors, collapse = ", "),
        rmse = pred_result$rmse,
        r_squared = pred_result$r_squared
      ))
    }, error = function(e) {
      message("Skipping predictors: ", paste(current_predictors, collapse = ", "),
              " - Error: ", e$message)
    }) # This try catch prevents KNN ties
  }
}
```

```
## Skipping predictors: implied_volatility - Error: too many ties in knn
```

```
## Skipping predictors: gamma, implied_volatility - Error: too many ties in knn
```

```
## Skipping predictors: implied_volatility, vega - Error: too many ties in knn
```

```
## Skipping predictors: gamma, implied_volatility, vega - Error: too many ties in knn
```

```
# Get best predictors
optimal_predictors = results$predictors[results$rmse == min(results$rmse)]
predictor_list <- strsplit(optimal_predictors, ",\\s*")[[1]]
cat("Best predictors: ", predictor_list)
```

```
## Best predictors:  delta gamma moneyness rho
```

```
cat("\nLowest RMSE: ", min(results$rmse))
```

```
##
## Lowest RMSE:  0.02708237
```

2. Model Evaluation

We evaluate the KNN model's performance on options, examining how well it predicts standardized prices based on the selected features.

```
results <- predict_standardized_price(k_value = baseline_k,
                                      predictors = predictor_list)

cat("KNN Model for Options:\n")
```

```
## KNN Model for Options:
```

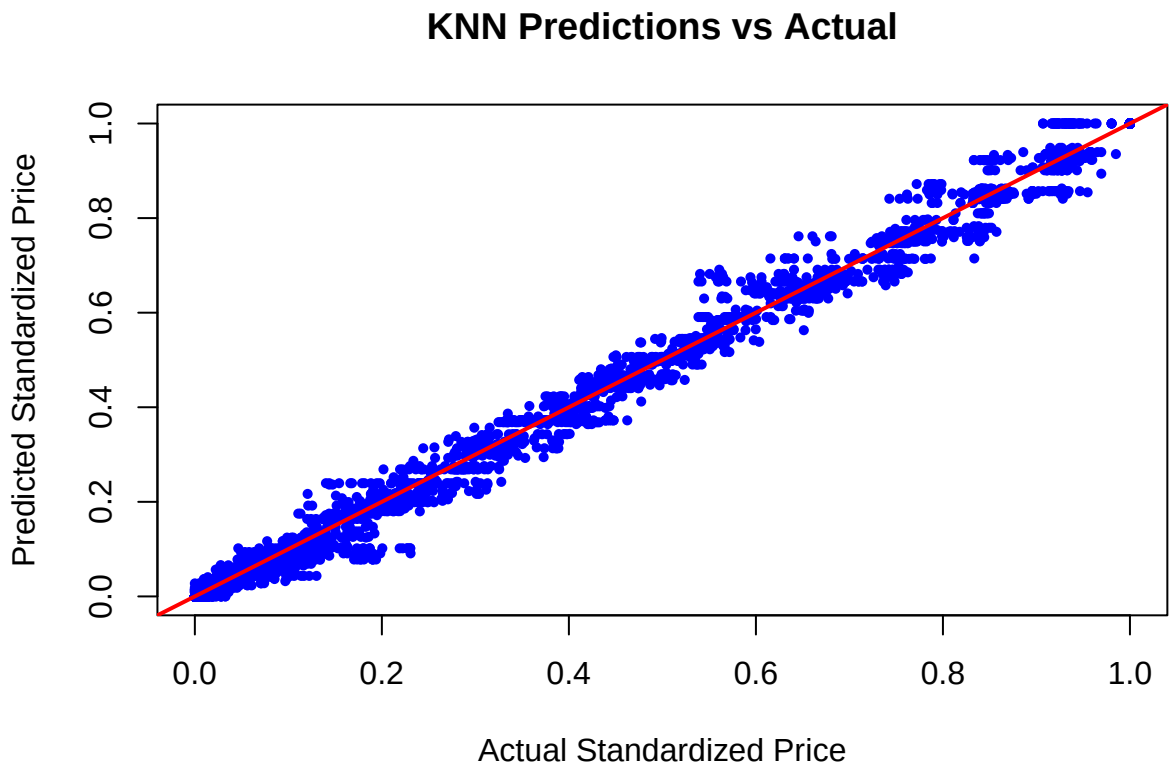
```
cat("RMSE:", round(results$rmse, 4), "\n")
```

```
## RMSE: 0.0269
```

```r
cat("R-squared:", round(results$r_squared, 4), "\n")
```

```
## R-squared: 0.9946
```

```r
plot(results$actual, results$predictions,
     main = "KNN Predictions vs Actual",
     xlab = "Actual Standardized Price",
     ylab = "Predicted Standardized Price",
     pch = 16, col = "blue", cex = 0.7)
abline(0, 1, col = "red", lwd = 2)
```



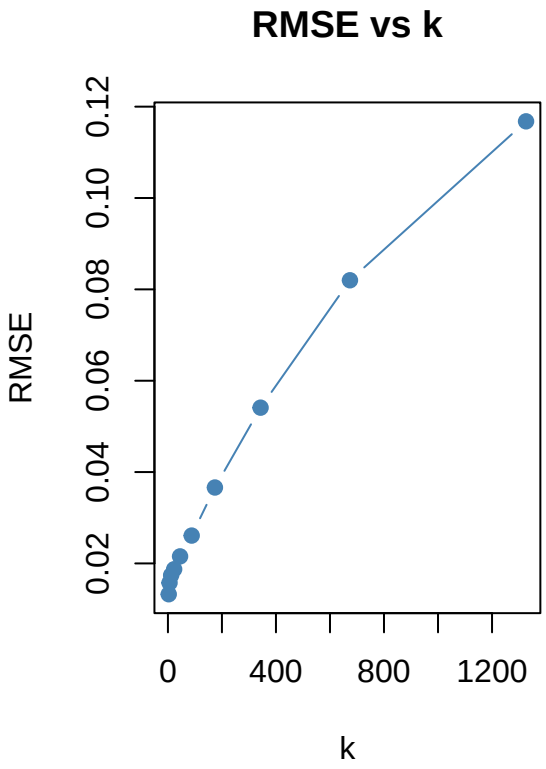
3. Optimal K Value Analysis

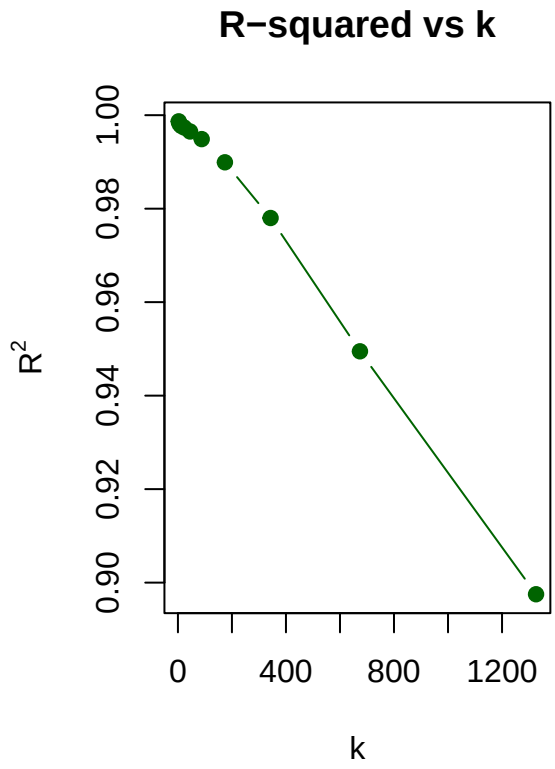
Determining the optimal number of neighbors is crucial for KNN performance. We analyze how model accuracy varies with different k values.

```r
library(FNN)
```

```
##
## Attaching package: 'FNN'

## The following objects are masked from 'package:class':
##
##     knn, knn.cv
```

```r
generate_k_range <- function(n, min_k = 3, max_frac = 0.1, steps = 10) {
  max_k <- max(min_k, floor(n * max_frac))
  k_values <- round(exp(seq(log(min_k), log(max_k), length.out = steps)))
  k_values <- unique(pmin(pmax(k_values, min_k), max_k))  # Clamp between min_k and max_k
  return(k_values)
}

evaluate_k_values <- function(data, predictors,
                              k_range = generate_k_range(nrow(data))) {
  rmse_values <- numeric(length(k_range))
  r_squared_values <- numeric(length(k_range))

  for (i in seq_along(k_range)) {
    k <- k_range[i]

    results <- tryCatch({
      predict_standardized_price(k_value = k, predictors = predictors)
    }, error = function(e) {
      message(sprintf("Skipping k = %d - Error: %s", k, e$message))
      NULL
    })

    if (!is.null(results)) {
      rmse_values[i] <- results$rmse
      r_squared_values[i] <- results$r_squared
    } else {
      rmse_values[i] <- NA
      r_squared_values[i] <- NA
    }
  }

  return(data.frame(k = k_range, rmse = rmse_values, r_squared = r_squared_values))
}

k_range <- generate_k_range(n = nrow(train))
k_results <- evaluate_k_values(data = train, predictors = predictor_list,
                               k_range = k_range)

par(mfrow = c(1, 2))

plot(k_results$k, k_results$rmse, type = "b", pch = 19,
     col = "steelblue",
     main = "RMSE vs k", xlab = "k", ylab = "RMSE")

plot(k_results$k, k_results$r_squared, type = "b", pch = 19,
     col = "darkgreen",
     main = "R-squared vs k", xlab = "k", ylab = expression(R^2))
```






```
# Select optimal k
optimal_k <- k_results$k[which.min(k_results$rmse)]
cat("Optimal k for options:", optimal_k, "\n")
```

```
## Optimal k for options: 3
```

A low optimal k means the space is very complex and higher k values underfit

4. Validation

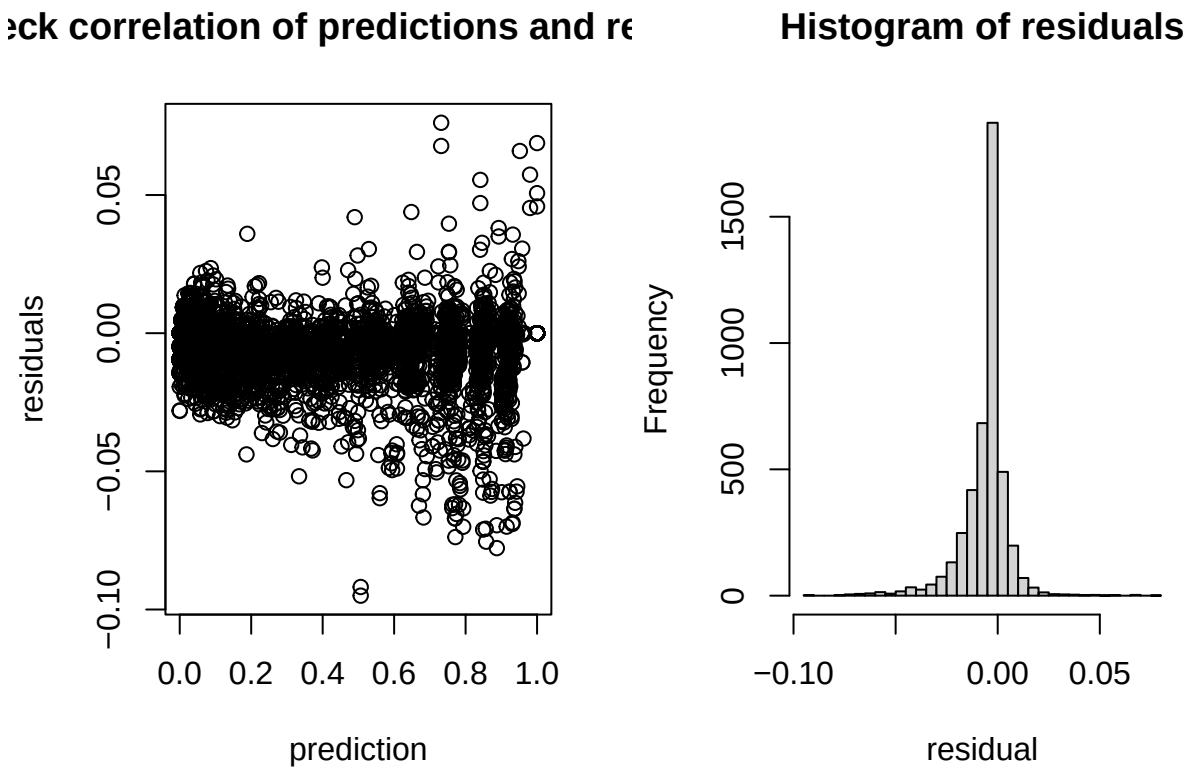
We validate our KNN model on the validation dataset and analyze instances where the model's predictions significantly ($p = 0.05$) deviate from actual values. This helps identify potential market inefficiencies or anomalies. We define significant residuals as those falling in the top 5% of absolute prediction errors. These may correspond to mispriced options or structural modeling weaknesses, potentially exploitable in a trading strategy.

1. Evaluate residuals

Check for correlation of residuals and fitted values, and if residuals are normally distributed

```
final_results <- predict_standardized_price(k_value = optimal_k,
                                            predictors = predictor_list)
residuals <- (final_results$predictions - final_results$actual)
```

```r
par(mfrow=c(1,2))
plot(final_results$predictions, residuals,
     main = "Check correlation of predictions and residuals", xlab = "prediction")
hist((final_results$predictions - final_results$actual), breaks=50, main = "Histogram of residuals", xla
```




```r
shapiro.test(residuals)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  residuals
## W = 0.83499, p-value < 2.2e-16
```

There appears to be an uptick in residuals when predictions are close to 1. However, there is enough evidence to determine residuals are normally distributed.

```r
# Assume normal distribution
significance_threshold <- qnorm(0.95, mean(residuals), sd(residuals))
cat("Significance threshold for residuals:", round(significance_threshold, 4), "\n")
```

```
## Significance threshold for residuals: 0.0146
```

```
large_residuals_geq <- (residuals) >= significance_threshold # pred > actual
large_residuals_leq <- (residuals) <= -significance_threshold # actual > pred
```

Now identify trends in price difference in these options, with respect to price_diff_x_periods, where x is the look ahead period. This is the average price movement divided by the current price:

$$price\_diff_x = \frac{\frac{\sum_{i=t}^{x+t} price_i}{x}}{price_t}$$

We test the following null hypothesis:

$$H_0 : \mu_{\text{GEQ}} = \mu_{\text{LEQ}} = \mu_{\text{All}}$$

where: - μ_{GEQ} is the mean price change for options with large positive residuals, - μ_{LEQ} is the mean price change for options with large negative residuals, - μ_{All} is the mean price change for the entire option set

Our alternative hypothesis is that options with large positive residuals (model overestimates actual) tend to **decline** in price, w hile those with large negative residuals (model underestimates actual) tend to **rise**, implying:

$$H_1 : \mu_{\text{GEQ}} > \mu_{\text{All}} > \mu_{\text{LEQ}}$$

This is based on the following interpretation of the residuals:

- If the **fitted value exceeds the actual** ($\hat{y}_t > y_t$), then the model overestimated price suggesting the option may be **undervalued**. This means the price may correct and increase.
- If the **fitted value is below the actual** ($\hat{y}_t < y_t$), then the model underestimated price suggesting the option may be **overvalued**.

The residual at time t is computed as:

$$r_t = \hat{y}_t - y_t$$

This directional hypothesis assumes that mispricings identified by residual sign carry predictive value for future returns.

2. Trim outliers

There are options that are OTM, that exponentially increase in value, causing $price\_diff_x$ values greatly above 1. Outliers can disproportionately influence non-parametric models such as KNN and distort distributional assumptions in residual analysis. To ensure robustness in our validation, we trim the 5% tails of the distribution

By applying this criterion across numeric features, we mitigate the risk of **outlier-driven distortion** in both the residual distribution and downstream performance metrics, such as the estimated significance of prediction errors.

```
remove_outliers <- function(x) {
    lower_bound <- quantile(x, 0.05, na.rm = TRUE)
    upper_bound <- quantile(x, 0.95, na.rm = TRUE)
    x <- x[x >= lower_bound & x <= upper_bound]
  }
```

3. Evaluate hypothesis

Here we use t tests to confirm that the values are greater with our p value (0.05)

```
val$predicted_price <- final_results$predictions
val$residual <- residuals
val$is_geq <- large_residuals_geq
val$is_leq <- large_residuals_leq

options_geq <- val[large_residuals_geq, ]
options_leq <- val[large_residuals_leq, ]
options_all <- val

cat("Number of options with GEQ residuals:", nrow(options_geq), "\n")
```

```
## Number of options with GEQ residuals: 75
```

```
cat("Number of options with LEQ residuals:", nrow(options_leq), "\n")
```

```
## Number of options with LEQ residuals: 651
```

```
cat("Number of total options:", nrow(options_all), "\n\n")
```

```
## Number of total options: 4423
```

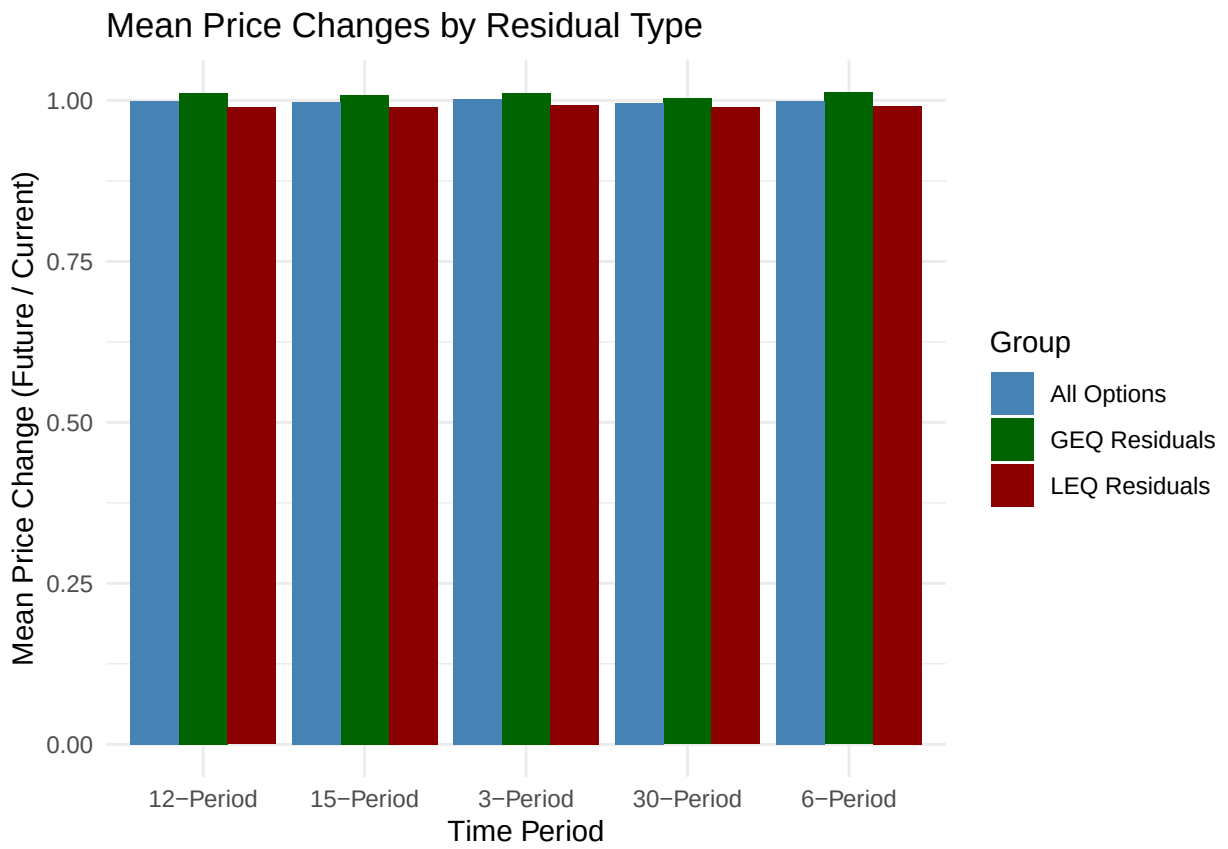
```
periods <- c(3, 6, 12, 15, 30)

cat("Mean Price Changes by Residual Type:\n")
```

```
## Mean Price Changes by Residual Type:
```

```
for (period in periods) {
  col_name <- paste0("price_diff_", period, "_periods")
  trimmed_return_all = remove_outliers(options_all[[col_name]])
  trimmed_return_geq = remove_outliers(options_geq[[col_name]])
  trimmed_return_leq = remove_outliers(options_leq[[col_name]])

  geq_mean <- mean(trimmed_return_geq, na.rm = TRUE)
  leq_mean <- mean(trimmed_return_leq, na.rm = TRUE)
  all_mean <- mean(trimmed_return_all, na.rm = TRUE)

  t_geq <- t.test(trimmed_return_geq, trimmed_return_all, alternative = "greater")
  t_leq <- t.test(trimmed_return_leq, trimmed_return_all, alternative = "less")

  cat("\n",period, "-Period | GEQ =", round(geq_mean, 4),
      "| LEQ =", round(leq_mean, 4),
      "| Overall =", round(all_mean, 4), "\n")
  cat("         | GEQ vs All p-val:", round(t_geq$p.value, 4),
      "\n         | LEQ vs All p-val:", round(t_leq$p.value, 4), "\n")
}
```

```
##
##  3 -Period | GEQ = 1.0111 | LEQ = 0.9932 | Overall = 1.0019
##          | GEQ vs All p-val: 0.0742
##          | LEQ vs All p-val: 0
##
##  6 -Period | GEQ = 1.0125 | LEQ = 0.9915 | Overall = 0.9991
##          | GEQ vs All p-val: 0.0265
##          | LEQ vs All p-val: 6e-04
##
##  12 -Period | GEQ = 1.0114 | LEQ = 0.9889 | Overall = 0.9989
##          | GEQ vs All p-val: 0.0504
##          | LEQ vs All p-val: 3e-04
##
##  15 -Period | GEQ = 1.0086 | LEQ = 0.9897 | Overall = 0.9975
##          | GEQ vs All p-val: 0.0818
##          | LEQ vs All p-val: 0.0052
##
##  30 -Period | GEQ = 1.0028 | LEQ = 0.9889 | Overall = 0.9961
##          | GEQ vs All p-val: 0.2437
##          | LEQ vs All p-val: 0.0405
```

```r
price_diff_data <- data.frame(
  Period = rep(paste0(periods, "-Period"), 3),
  Group = rep(c("GEQ Residuals", "LEQ Residuals", "All Options"), each = length(periods)),
  Mean_Price_Change = c(
    sapply(paste0("price_diff_", periods, "_periods"),
           function(col) mean(remove_outliers(options_geq[[col]]), na.rm = TRUE)),
    sapply(paste0("price_diff_", periods, "_periods"),
           function(col) mean(remove_outliers(options_leq[[col]]), na.rm = TRUE)),
    sapply(paste0("price_diff_", periods, "_periods"),
           function(col) mean(remove_outliers(options_all[[col]]), na.rm = TRUE))
  )
)

ggplot(price_diff_data, aes(x = Period, y = Mean_Price_Change, fill = Group)) +
  geom_bar(stat = "identity", position = "dodge") +
  labs(title = "Mean Price Changes by Residual Type",
       y = "Mean Price Change (Future / Current)",
       x = "Time Period") +
  theme_minimal() +
  scale_fill_manual(values = c(
    "GEQ Residuals" = "darkgreen",
    "LEQ Residuals" = "darkred",
    "All Options" = "steelblue"
  ))
```



The null hypothesis cannot be rejected for all periods except the 6 period (1 minute). There are some periods where either $\mu_{\text{GEQ}} > \mu_{\text{All}}$ or $\mu_{\text{All}} > \mu_{\text{LEQ}}$. The efficacy of these can be explored in the future.

Trading Simulation

Brand new test data to run trading simulation. Find all long/short signals and enter trade. Using the price_diff column, we calculate returns.

$$average\_price_x = (price_t * price\_diff_x)$$

Hence

$$\alpha_{\text{long}} = average\_price_x - price_t$$

$$\alpha_{\text{short}} = price_t - average\_price_x$$

These will be combined to measure α_{signal}

```
test = read.csv("../data/processed/option_data_with_future_prices_7_9.csv")
final_results <- predict_standardized_price(k_value = optimal_k, predictors = predictor_list, validatio
residuals <- (final_results$predictions - final_results$actual)

significance_threshold <- qnorm(0.95, mean(residuals), sd(residuals))

buy_signal = residuals > significance_threshold
sell_signal = -significance_threshold > residuals
```

```
returns_long = 100 * (
  (test$latest_trade_price[buy_signal]*test$price_diff_6_periods[buy_signal])-
    test$latest_trade_price[buy_signal])

returns_short = 100 * (
  (test$latest_trade_price[sell_signal]) -
    test$latest_trade_price[sell_signal]*test$price_diff_6_periods[sell_signal]
   )

final_returns = sum(returns_long, na.rm = TRUE) + sum(returns_short, na.rm = TRUE)
num_trades = length(returns_long) + length(returns_short)
cat("After", num_trades, "trades, the algorithm generates", final_returns, "$")
```

```
## After 5268 trades, the algorithm generates 1685 $
```

This shows that we can make over 1500 dollars in one trading period. Let's check if these returns are not random. Check if the returns beat the top 5% of 100 random trading simulations with the same number of trades as above.

Null hypothesis $H_0 : \alpha_{\text{random}} = \alpha_{textsignal}$

```
null_returns = c()
num_trades = length(returns_long) + length(returns_short)
for (i in 1:100){
  set.seed(i)
  sample_indices <- sample(1:nrow(test), num_trades, replace = FALSE)
  if (round(runif(1,0,1) == 1)) {
    returns = 100 * (test$latest_trade_price[sample_indices] -

    (test$latest_trade_price[sample_indices]*test$price_diff_6_periods[sample_indices]))
  } else {
    returns = 100 * (
    (test$latest_trade_price[sample_indices]*test$price_diff_6_periods[sample_indices])-test$latest_trad
  }
  null_returns = c(sum(returns, na.rm = TRUE), null_returns)
}
set.seed(seed_num)
null_threshold=qnorm(0.95,mean(null_returns),sd(null_returns))
cat("Random trade null returns lower bound threshold", null_threshold)
```

```
## Random trade null returns lower bound threshold 445.6409
```

We can hence reject the null hypothesis