SPY ETF ML Trading Strategy Analysis

Ayan Goswami

2025-07-19

Contents

Data Load

The numeric suffix for each data set represents 1. ATR multiplier 2. Trade timeout duration (minutes). View data_mining.ipynb for how this is populated.

```
data1_3 = read.csv("../csvs/train1_3.csv")
# pl_value was calculated with 1000 shares when I made it 2023. I think 100 is cleaner
data1_3$pl_value = data1_3$pl_value/10
```

Data exploration for 1_3

```
summary(data1_3)
```

```
##       X2                 Open           Close            High      
##  Length:21492       Min.   :416.3   Min.   :416.3   Min.   :416.4  
##  Class :character   1st Qu.:428.8   1st Qu.:428.8   1st Qu.:428.8  
##  Mode  :character   Median :435.8   Median :435.8   Median :435.9  
##                     Mean   :434.4   Mean   :434.4   Mean   :434.5  
##                     3rd Qu.:439.8   3rd Qu.:439.8   3rd Qu.:439.8  
##                     Max.   :450.4   Max.   :450.4   Max.   :450.4  
##       Low              Volume             ATR                PL          
##  Min.   :416.2   Min.   :      1   Min.   :0.006399   Min.   :0.0000  
##  1st Qu.:428.7   1st Qu.:   2016   1st Qu.:0.067450   1st Qu.:0.0000  
##  Median :435.7   Median :  54740   Median :0.106248   Median :0.0000  
##  Mean   :434.3   Mean   :  83707   Mean   :0.130528   Mean   :0.4414  
##  3rd Qu.:439.7   3rd Qu.: 110482   3rd Qu.:0.166756   3rd Qu.:1.0000  
##  Max.   :450.2   Max.   :4816391   Max.   :1.285800   Max.   :1.0000  
##     pl_value             SMA_k7           SMA_k20           SMA_k50      
##  Min.   :-84.50000   Min.   :0.9966   Min.   :0.9938   Min.   :0.9930  
##  1st Qu.: -9.02023   1st Qu.:0.9998   1st Qu.:0.9997   1st Qu.:0.9996  
##  Median : -2.00000   Median :1.0000   Median :1.0000   Median :1.0001  
##  Mean   :  0.05364   Mean   :1.0000   Mean   :1.0000   Mean   :1.0001  
##  3rd Qu.:  9.00000   3rd Qu.:1.0002   3rd Qu.:1.0003   3rd Qu.:1.0006  
##  Max.   :161.00000   Max.   :1.0047   Max.   :1.0048   Max.   :1.0057
```

View Summary and shape

ATR Multiplier (Lambda)

Mean Price Calculation:

$$\mu_T = \frac{\sum_{i=T-k}^{T}(OHLC_i)}{k}$$

Modified ATR Formula:

$$ATR_T = \sqrt{\frac{\sum_{i=T-k}^{T}(OHLC_i - \mu_T)^2}{k}}$$

Controls the sensitivity to market volatility:

- **Purpose**: Scales profit/loss targets based on current market conditions
- **Example**: If SPY ATR = 0.50 and $\lambda = 10$:
  - Base volatility adjustment = $10 \times 0.50 = \$5.00$
  - Sell if SPY is up 5$ from entry price

pl_value

It is determined by iterating through the dataframe and seeing if the price reaches take profit or stop loss first. It assumes a 100 share position size.

Profit-to-Loss Ratio (Chi) Asymmetric risk-reward ratio: - **Default**: 1.5 (profit targets 50% wider than loss targets)

- **Profit Target**: $Price_T + \lambda * \chi\ddot{O}ATR$
- **Loss Target**: $Price_T - \lambda * ATR$

Timeout Period (t) Maximum holding period before forced exit:

- **Purpose**: Prevents indefinite position holding
- **Logic**: If neither target hit within t periods, compare exit price to entry
- **Classification**: Profit (1) if $Price_{T+t} > Price_T$, Loss (0) otherwise

SMA

$$SMA_k = \frac{Open_T}{\frac{1}{k}\sum_{i=T-k}^{T-1} Close_i}$$

- T = current time period
- k = lookback period
- Hypothetically, values > 1.0 indicate price above historical average (bullish)
- Values < 1.0 indicate price below historical average (bearish)

Multi-Timeframe Analysis The system calculates SMA for three periods:

- **SMA_7**: Short-term momentum (1 week of 5-min bars)
- **SMA_20**: Medium-term trend (1 month of daily closes)
- **SMA_50**: Long-term trend (quarterly trend)

Signal Interpretation

- **Trend Confirmation**: Multiple SMAs above 1.0 = strong uptrend
- **Momentum Divergence**: SMA_7 > SMA_20 > SMA_50 = accelerating uptrend
- **Mean Reversion**: Extreme SMA values (>1.05 or <0.95) suggest potential reversal

See data_mining.ipynb for information

PL

Encodes 0 if `pl_value` < 0

Normalized volume

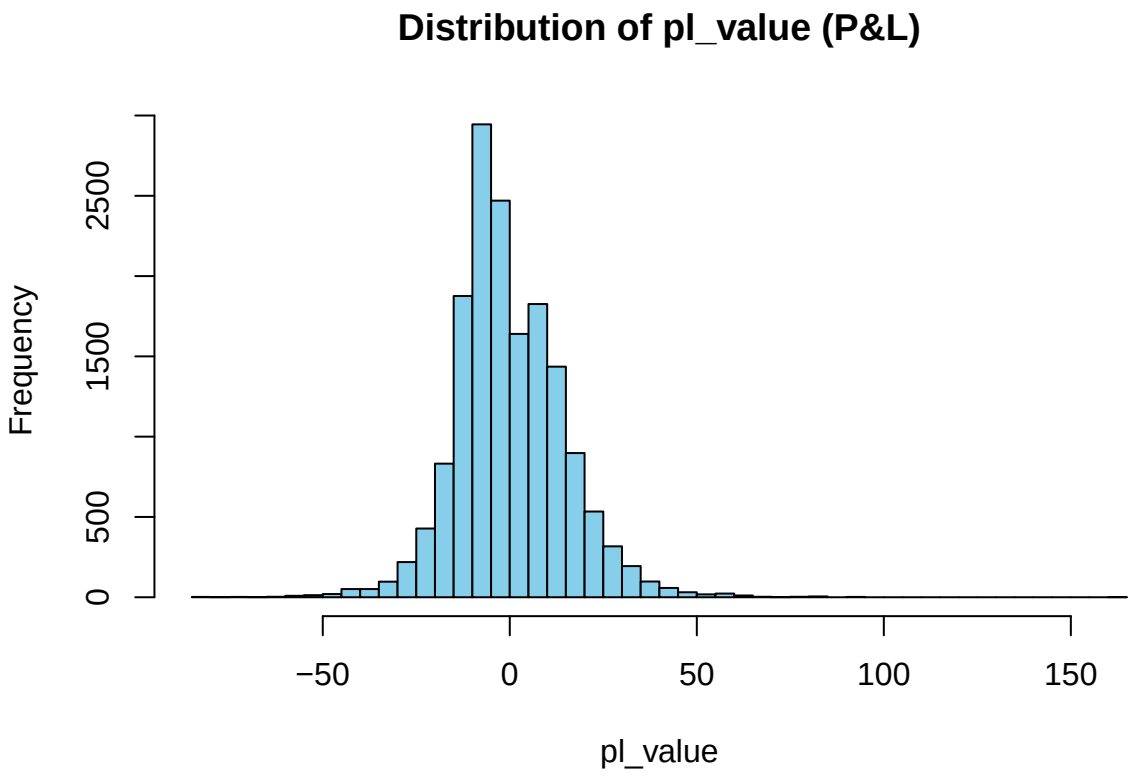
Standard normalization $norm\_vol = \frac{mean(Vol) - Vol}{sd(Vol)}$

Split train-test data

```
seed_num = 13
set.seed(seed_num) # reproducibility
data1_3$norm_volume = (data1_3$Volume - mean(data1_3$Volume))/sd(data1_3$Volume)
ind = sample(1:nrow(data1_3),0.75*nrow(data1_3))
train = data1_3[ind,]
test = data1_3[-ind,]
```

Investigate dependent variable

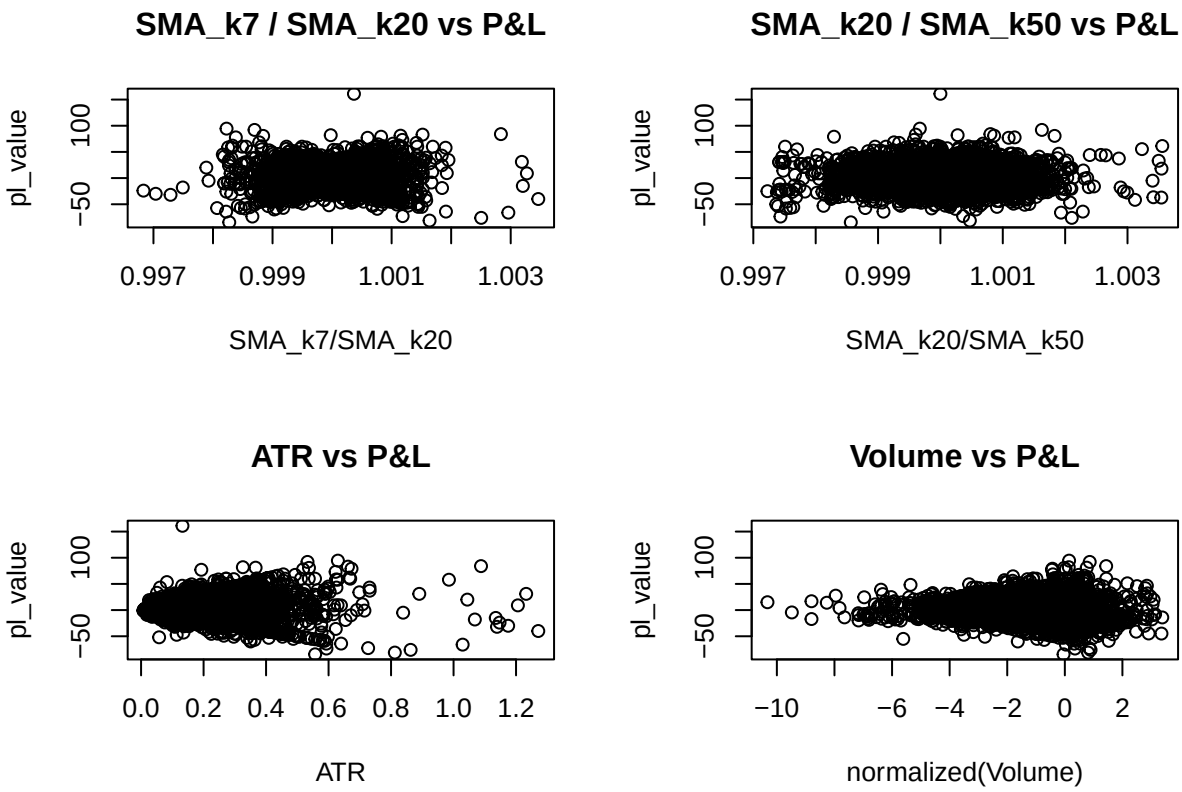
```
hist(train$pl_value, breaks = 50, col = "skyblue", main = "Distribution of pl_value (P&L)", xlab = "pl_
```



Assess correlation

```
par(mfrow=c(2,2))
plot(train$SMA_k7/train$SMA_k20, train$pl_value, main="SMA_k7 / SMA_k20 vs P&L", xlab="SMA_k7/SMA_k20",
plot(train$SMA_k20/train$SMA_k50, train$pl_value, main="SMA_k20 / SMA_k50 vs P&L", xlab="SMA_k20/SMA_k5
plot(train$ATR, train$pl_value, main="ATR vs P&L", xlab="ATR", ylab="pl_value")
plot(log(train$norm_volume), train$pl_value, main="Volume vs P&L", xlab="normalized(Volume)", ylab="pl_
```

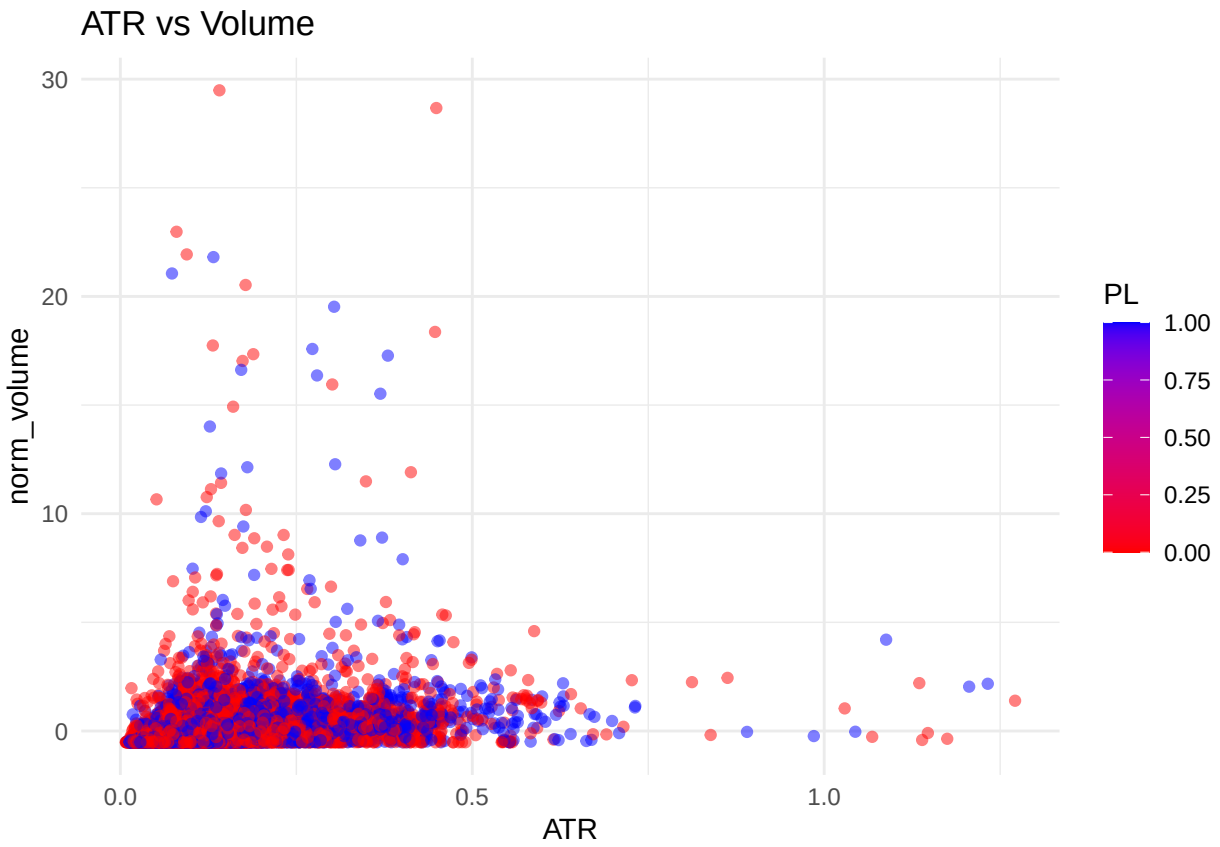
```
## Warning in log(train$norm_volume): NaNs produced
```




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

ATR shows to have a weak correlation. I think the interaction of predictors with each other will be more important. Let's see the interaction of volume and ATR

```
library(ggplot2)
ggplot(train, aes(x=ATR, y=norm_volume, color=PL)) +
  geom_point(alpha=0.5) +
  scale_color_gradient2(mid="red", high="blue") +
  theme_minimal() +
  ggtitle("ATR vs Volume")
```



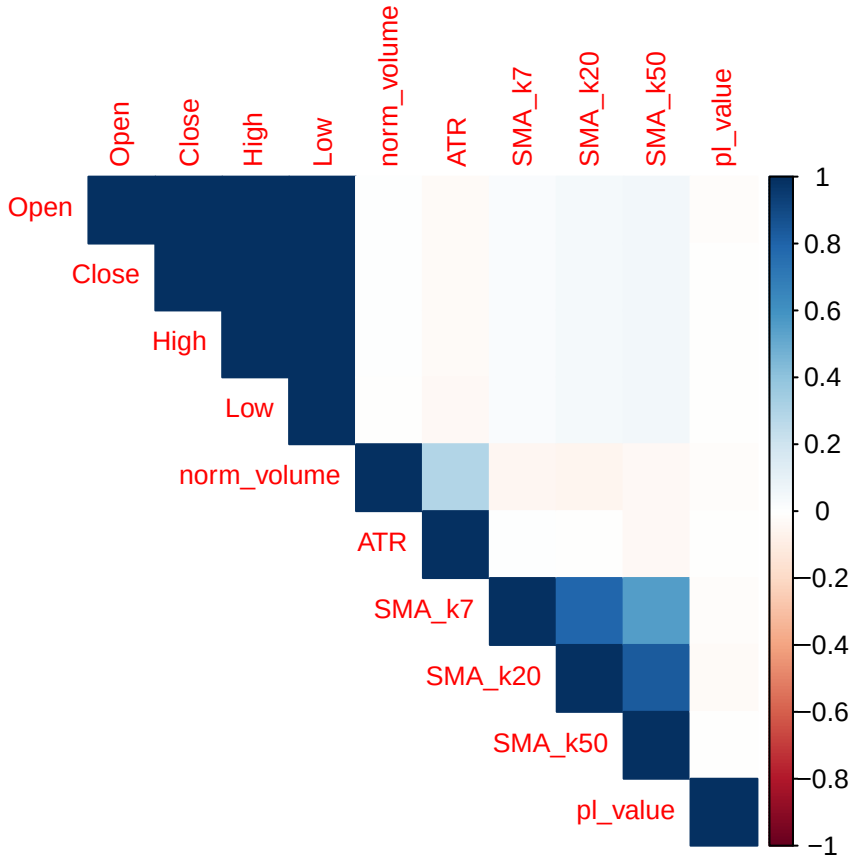
This shows us that low ATR and volume have a lower chance of being profitable. Though it tends to fluctuate quite a lot.

Visualize predictor correlation matrix

```
library(corrplot)
```

```
## corrplot 0.95 loaded
```

```
numeric_vars <- train[, c("Open", "Close", "High", "Low", "norm_volume", "ATR", "SMA_k7", "SMA_k20", "SM
cor_matrix <- cor(numeric_vars, use="complete.obs")

corrplot(cor_matrix, method = "color", type = "upper", tl.cex = 0.8)
```



We will have to take steps to ensure we address multicollinearity, one way of doing this is looking at the VIF (Variance Inflation Factor). VIF is a diagnostic tool used to detect multicollinearity, which occurs when independent variables in a regression model are highly correlated with each other. High VIF values indicate that a variable's variance is inflated due to multicollinearity. Values over 5 may have potential issues in fitting models.

```
library(car)
```

```
## Loading required package: carData
```

```
vif_model <- lm(pl_value ~ ATR + SMA_k7 + SMA_k20+ SMA_k50+ norm_volume, data=train)
vif(vif_model)
```

```
##          ATR      SMA_k7     SMA_k20     SMA_k50 norm_volume
##     1.099763    3.044601    6.773931    3.581017    1.099538
```

This shows us that in order to reduce multicollinearity, we must omit `SMA_k20`

Log regression

Model Diagnostics and Theoretical Foundations

Logistic regression models the log-odds of a binary outcome as a linear function of predictors:

$$\log\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_k X_k$$

Where p is the probability of a profitable trade. The coefficients represent the change in log-odds associated with a one-unit increase in the predictor, holding all other variables constant. We use all parameters and all interactions initially

```
glm1 = glm(
  data = train,
  PL ~ (ATR + SMA_k7 + SMA_k50 + norm_volume)^4,
  family = "binomial"
)
```

Assess

The full model will likely have only a handful of terms with significant probabilities because of multi-colinearity

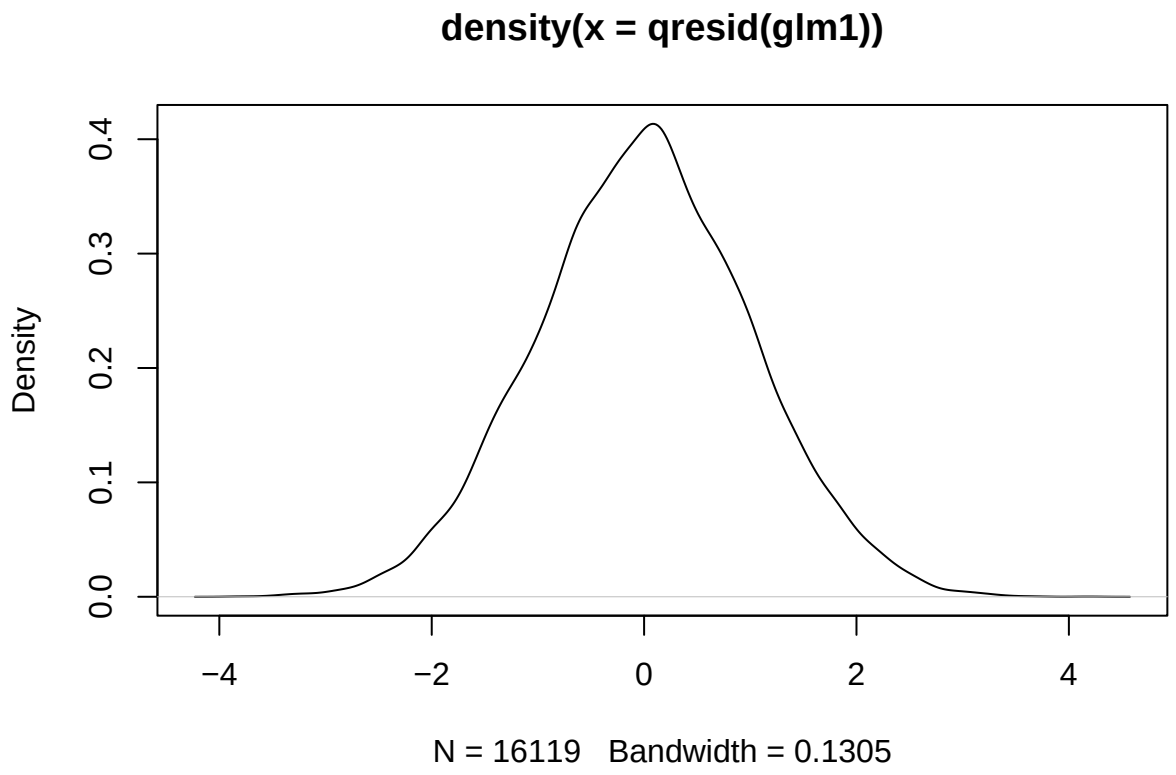
```
summary(glm1)
```

```
##
## Call:
## glm(formula = PL ~ (ATR + SMA_k7 + SMA_k50 + norm_volume)^4,
##     family = "binomial", data = train)
##
## Coefficients:
##                                 Estimate Std. Error z value Pr(>|z|)
## (Intercept)                     -111256      59065  -1.884   0.0596 .
## ATR                              204567     140044   1.461   0.1441
## SMA_k7                           111142      59062   1.882   0.0599 .
## SMA_k50                          111444      59054   1.887   0.0591 .
## norm_volume                       29720      31198   0.953   0.3408
## ATR:SMA_k7                      -204188     140034  -1.458   0.1448
## ATR:SMA_k50                     -205060     139948  -1.465   0.1429
## ATR:norm_volume                  -60923      59842  -1.018   0.3086
## SMA_k7:SMA_k50                  -111330      59051  -1.885   0.0594 .
## SMA_k7:norm_volume               -29608      31204  -0.949   0.3427
## SMA_k50:norm_volume              -30049      31203  -0.963   0.3355
## ATR:SMA_k7:SMA_k50               204682     139938   1.463   0.1436
## ATR:SMA_k7:norm_volume            60646      59861   1.013   0.3110
## ATR:SMA_k50:norm_volume           61604      59916   1.028   0.3039
## SMA_k7:SMA_k50:norm_volume        29937      31209   0.959   0.3374
## ATR:SMA_k7:SMA_k50:norm_volume   -61328      59934  -1.023   0.3062
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 22116  on 16118  degrees of freedom
## Residual deviance: 22037  on 16103  degrees of freedom
## AIC: 22069
##
## Number of Fisher Scoring iterations: 4
```

Residuals analysis

Regular residual plots don't make sense in glm. Dunn and Gordon (2018) introduce quantile residuals for discrete response variables. Their primary benefits are they do not show weird patterns (due to variable's discreteness). They are available in R via statmod::qresid().

```
library(statmod)
plot(density(qresid(glm1)))
```



This residuals do appear normally distributed. Which means there is no unexplained variance (non linearity or omitted variables) in the model.

Variable Selection

The Akaike Information Criterion (AIC) balances model fit against complexity:

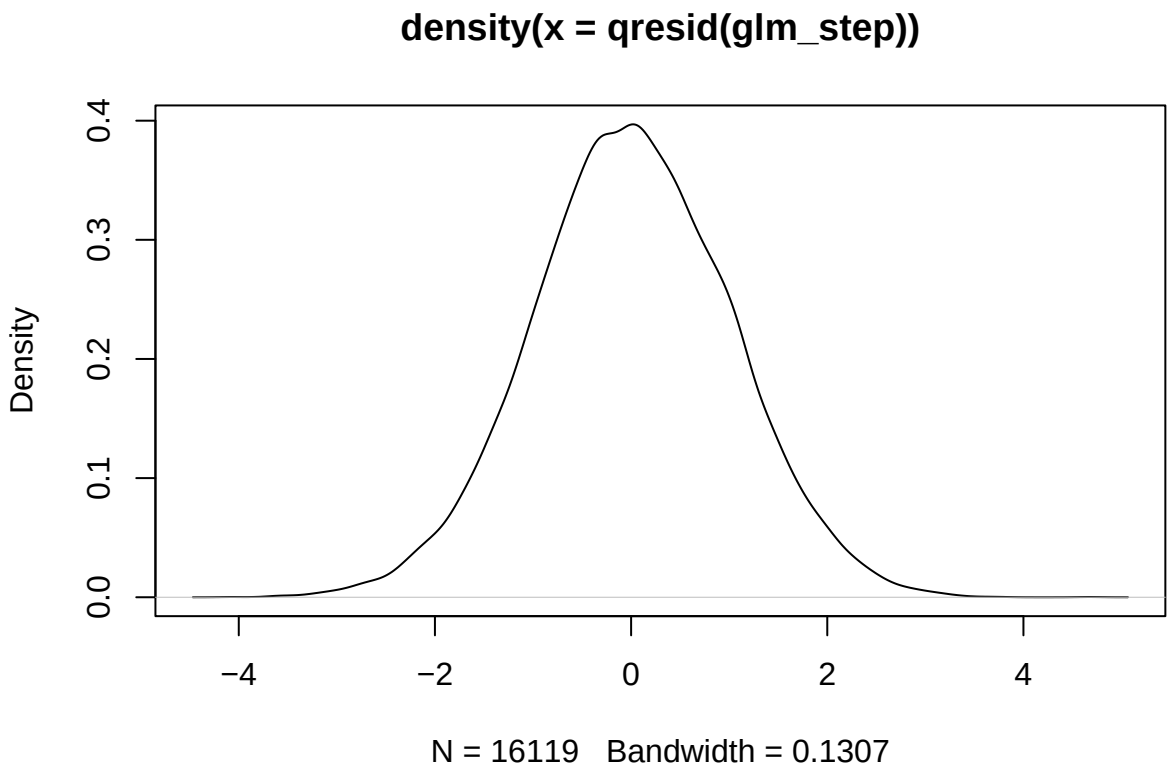
$$AIC = -2\ln(L) + 2k$$

Where L is the likelihood and k is the number of parameters. Lower AIC values indicate better models. The stepwise procedure systematically adds or removes variables to minimize AIC, providing a theoretically sound approach to model selection. Let's use forward AIC to trim some of the predictors. AIC is usually computationally expensive, however I am not dealing with a lot of predictors here.

```r
glm_step <- step(glm1, direction = "both", trace = 0)
summary(glm_step)
```

```
##
## Call:
## glm(formula = PL ~ ATR + SMA_k7 + SMA_k50 + norm_volume + ATR:SMA_k7 +
##     ATR:SMA_k50 + ATR:norm_volume + SMA_k7:norm_volume + SMA_k50:norm_volume +
##     ATR:SMA_k7:norm_volume + ATR:SMA_k50:norm_volume, family = "binomial",
##     data = train)
##
## Coefficients:
##                          Estimate Std. Error z value Pr(>|z|)
## (Intercept)                99.09      78.41   1.264 0.206311
## ATR                      -232.61     313.08  -0.743 0.457495
## SMA_k7                   -207.67      90.19  -2.303 0.021302 *
## SMA_k50                   108.17      37.94   2.851 0.004355 **
## norm_volume              -215.94      59.37  -3.637 0.000276 ***
## ATR:SMA_k7                592.95     358.75   1.653 0.098364 .
## ATR:SMA_k50              -359.05     137.63  -2.609 0.009085 **
## ATR:norm_volume           411.92     181.45   2.270 0.023196 *
## SMA_k7:norm_volume        331.90      79.38   4.181  2.9e-05 ***
## SMA_k50:norm_volume      -115.96      39.45  -2.940 0.003285 **
## ATR:SMA_k7:norm_volume   -690.88     243.31  -2.840 0.004518 **
## ATR:SMA_k50:norm_volume   278.85     113.36   2.460 0.013899 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 22116  on 16118  degrees of freedom
## Residual deviance: 22041  on 16107  degrees of freedom
## AIC: 22065
##
## Number of Fisher Scoring iterations: 4
```

```r
plot(density(qresid(glm_step)))
```



This is still a lot of predictors, let's try and do some subset selection. This method uses cross validation and fits models with the least mean squared error. By splitting the data up we are able to understand what is actually needed by minimizing MSE.

```
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1-9
```

```
# Get interaction matrix
X <- model.matrix(PL ~ (ATR + SMA_k7 + SMA_k50 + norm_volume)^4, data = train)[, -1]
y <- train$PL
cvfit <- cv.glmnet(X, y, family = "binomial", alpha = 1)

# coefs with least mse
lasso_coef <- coef(cvfit, s = "lambda.min")
selected_vars <- rownames(lasso_coef)[lasso_coef[,1] != 0][-1]  # exclude intercept
selected_vars
```

```
## [1] "ATR"                       "SMA_k50"
## [3] "norm_volume"               "ATR:SMA_k7"
## [5] "ATR:norm_volume"           "ATR:SMA_k7:SMA_k50"
## [7] "ATR:SMA_k7:norm_volume"    "ATR:SMA_k50:norm_volume"
```

These are a lot fewer than earlier, let's fit this conservative model as well.

```
glm_formula <- as.formula(paste("PL ~",
                                paste(selected_vars, collapse = " + ")))
glm_cons <- glm(glm_formula, data = train, family = "binomial")
summary(glm_cons)
```

```
##
## Call:
## glm(formula = glm_formula, family = "binomial", data = train)
##
## Coefficients:
##                             Estimate Std. Error z value Pr(>|z|)
## (Intercept)              -6.142e+01  3.119e+01  -1.970   0.0489 *
## ATR                       1.583e+02  2.076e+02   0.763   0.4457
## SMA_k50                   6.102e+01  3.118e+01   1.957   0.0504 .
## norm_volume              -2.122e-03  3.172e-02  -0.067   0.9467
## ATR:SMA_k7                1.842e+01  2.721e+02   0.068   0.9460
## ATR:norm_volume          -1.949e+02  9.273e+01  -2.102   0.0355 *
## ATR:SMA_k50:SMA_k7       -1.755e+02  1.205e+02  -1.457   0.1452
## ATR:norm_volume:SMA_k7    2.357e+02  1.201e+02   1.962   0.0498 *
## ATR:SMA_k50:norm_volume -4.088e+01  5.622e+01  -0.727   0.4671
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 22116  on 16118  degrees of freedom
## Residual deviance: 22061  on 16110  degrees of freedom
## AIC: 22079
##
## Number of Fisher Scoring iterations: 4
```

Assess performance

```
pred_glm = ifelse(predict(glm_step,test,type="response")>0.5,1,0)
winrate = mean(pred_glm == test$PL)
print(winrate)
```

```
## [1] 0.5561139
```

```
pred_glm_cons = ifelse(predict(glm_cons,test,type="response")>0.5,1,0)
winrate_cons = mean(pred_glm_cons == test$PL)
print(winrate_cons)
```

```
## [1] 0.5564861
```

```
if (winrate_cons < winrate) {
  glm2 = glm_step
} else {
  glm2 = glm_cons
}
```

The accuracy is good, but it needs to be contextualized. The worst, bare bones model would trade on a coin flip. Let's simulate $nrow(test)$ coin flips and compare it to a 55% win rate.

Simulate 1000 trading simulations with random buy indicators and see if winrate if better than it. `p = 0.05`. This is to weed out strategies that don't perform better than random chance.

```
n <- nrow(test)

# Simulate coin flips (baseline)
baseline_accuracy = c()
for (i in 1:1000) {
  set.seed(i)
  coin_flips <- rbinom(n, size = 1, prob = 0.5)
  baseline_accuracy <- c(mean(coin_flips == test$PL), baseline_accuracy)
}
set.seed(seed_num) # reproducibility
cat("Coin toss simulations that performed better than strategy :",
    length(baseline_accuracy[baseline_accuracy>winrate]) / length(baseline_accuracy))
```

```
## Coin toss simulations that performed better than strategy : 0
```

```
# Use normal approximation

cat("\nWinrate p value:",pnorm(winrate, mean(baseline_accuracy), sd(baseline_accuracy),
                              lower.tail = F))
```

```
##
## Winrate p value: 5.854805e-17
```

```
cat("\nStatistically significant lower bound on accuracy", qnorm(0.95, mean(baseline_accuracy),
                                                                 sd(baseline_accuracy)))
```

```
##
## Statistically significant lower bound on accuracy 0.5109424
```

This makes it clear that our models perform better than random chance, however we need to set a baseline for precision as well.

Random chance

It's possible that the data is looking at a biased subset of information. It's possible that the overall trend from when this data was collected was upward, and these models wouldn't hold up in turbulent market. To disprove this, A random trader was utilized, trading on a coin toss. This simulation is run 100 times to give a baseline of precision, and returns for our p-value.

```
precisions <- c()
percent_returns <- c()
sharpe_ratios <- c()
max_drawdowns <- c()

initial_capital <- (1 / 0.02) * (100 * sample(test$Open, 1))
```

```r
# Diff seeds
for (i in 1001:1100) {
  set.seed(i)
  capital <- initial_capital
  drawdown <- c(capital)
  wins <- c()

  for (j in 1:nrow(test)) {
    if (round(runif(1, 0, 1)) == 1) {
      capital <- capital + test$pl_value[j]
      drawdown <- c(drawdown, capital)
      wins <- c(wins, test$pl_value[j])
    }
  }

  returns <- diff(drawdown)
  num_wins <- sum(returns > 0)
  total_trades <- length(returns)

  precision <- if (total_trades > 0) num_wins / total_trades else NA
  percent_return <- round(((capital / initial_capital) - 1) * 100, 2)

  # Sharpe Ratio (no risk-free rate)
  sharpe <- if (length(wins) > 1 && sd(wins) > 0) {
    mean(wins) / sd(wins)
  } else {
    NA
  }

  # Max Drawdown
  equity_curve <- drawdown
  peak <- cummax(equity_curve)
  dd <- (equity_curve - peak) / peak
  max_dd <- min(dd)

  # Store metrics
  precisions <- c(precision, precisions)
  percent_returns <- c(percent_return, percent_returns)
  sharpe_ratios <- c(sharpe, sharpe_ratios)
  max_drawdowns <- c(max_dd, max_drawdowns)
}

# Baseline values at 95% confidence level
cat("\nStatistically significant lower bound on precision:",
    qnorm(0.95, mean(precisions, na.rm = TRUE), sd(precisions, na.rm = TRUE)))
```

```
##
## Statistically significant lower bound on precision: 0.4548945
```

```r
cat("\nStatistically significant lower bound on returns (%):",
    qnorm(0.95, mean(percent_returns, na.rm = TRUE), sd(percent_returns, na.rm = TRUE)))
```

```
##
```

```
## Statistically significant lower bound on returns (%): 0.04639581
```

```
cat("\nStatistically significant lower bound on Sharpe Ratio:",
    qnorm(0.95, mean(sharpe_ratios, na.rm = TRUE), sd(sharpe_ratios, na.rm = TRUE)))
```

```
##
## Statistically significant lower bound on Sharpe Ratio: 0.02511618
```

```
cat("\nStatistically significant upper bound on Max Drawdown (%):",
    qnorm(0.05, mean(max_drawdowns, na.rm = TRUE), sd(max_drawdowns, na.rm = TRUE)) * 100)
```

```
##
## Statistically significant upper bound on Max Drawdown (%): -0.05253197
```

This is a good baseline estimate for random chance precision. All models should try and perform higher than this.

Buy threshold

We need to determine when to buy based on the threshold. This improves the win rate at the cost of fewer trades. This will miss out on a lot of winning trades too (False negatives). But to succeed at trading we need to minimize false positives. Precision, or $\frac{TP}{FP+TP}$ is much more important. "For estimating a binomial proportion, at least 10 successes and 10 failures is recommended for normal approximation intervals." Reference: Agresti, A. (2013). Categorical Data Analysis (3rd ed.)

```
# Define thresholds
probs <- 10^seq(-2, -10, length.out = 9)

fit_mean <- mean(glm2$fitted.values)
fit_sd <- sd(glm2$fitted.values)

prob_with_best_winrate = 0.05 # init
best_winrate = 0
for (prob in probs) {
  threshold <- qnorm(1 - prob, mean = fit_mean, sd = fit_sd)
  pred_glm <- ifelse(predict(glm_cons, train, type = "response") > threshold, 1, 0)

  true_positive <- sum(pred_glm == 1 & train$PL == 1)
  false_positive <- sum(pred_glm == 1 & train$PL == 0)
  total_predicted_positive <- sum(pred_glm == 1)
  if ((true_positive + false_positive) < 30) {
    break # CLT
  }
  winrate <- if (total_predicted_positive > 0) {
    true_positive / total_predicted_positive
  } else {
    NA
  }
  if (winrate > best_winrate) {
    best_winrate = winrate
    prob_with_best_winrate = prob
```

```
  }

  cat(sprintf("Prob = %e | TP = %d | FP = %d | Winrate = %.3f\n",
              prob, true_positive, false_positive, winrate))
}
```

```
## Prob = 1.000000e-02 | TP = 221 | FP = 207 | Winrate = 0.516
## Prob = 1.000000e-03 | TP = 97 | FP = 98 | Winrate = 0.497
## Prob = 1.000000e-04 | TP = 54 | FP = 55 | Winrate = 0.495
## Prob = 1.000000e-05 | TP = 39 | FP = 38 | Winrate = 0.506
## Prob = 1.000000e-06 | TP = 30 | FP = 23 | Winrate = 0.566
## Prob = 1.000000e-07 | TP = 22 | FP = 18 | Winrate = 0.550
## Prob = 1.000000e-08 | TP = 15 | FP = 18 | Winrate = 0.455
```

```
best_threshold_glm = qnorm(1 - prob_with_best_winrate,
                           mean = fit_mean, sd = fit_sd)
cat("Best winrate", best_winrate*100)
```

```
## Best winrate 56.60377
```

```
cat("\nProbability with highest precision", prob_with_best_winrate)
```

```
##
## Probability with highest precision 1e-06
```

```
cat("\nThreshold", best_threshold_glm)
```

```
##
## Threshold 0.5775394
```

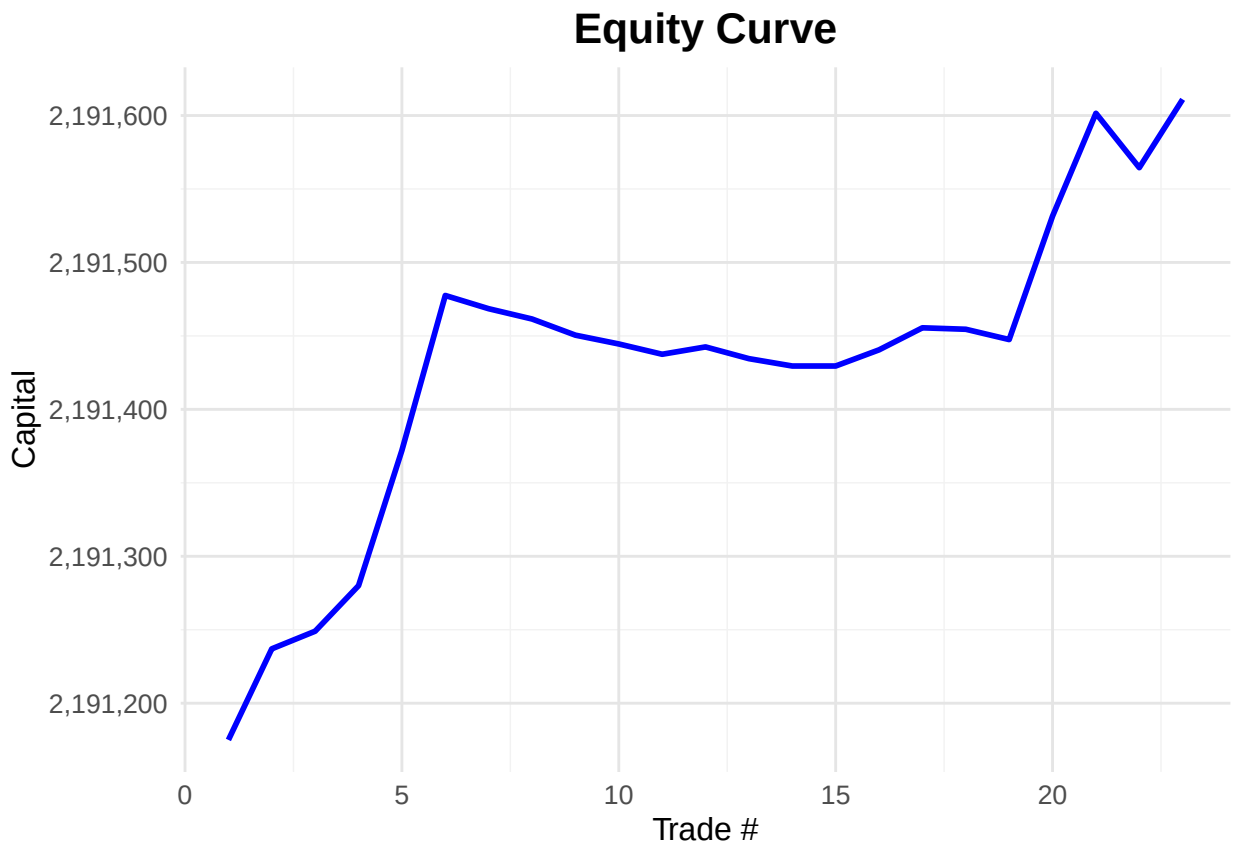
Drawdown

`pl_value` is calculated with 100 shares of SPY. In trading, using 2% of your portfolio in a trade is recommended, hence, The initial capital is set at $\frac{1}{0.02} * (100 * Price)$.

```
initial_capital = (1/(0.02))*(100 * sample(train$Open, 1)) # random price
capital <- initial_capital
drawdown <- c(capital)
wins <- c()

for (i in 1:nrow(test)) {
  pred_X <- predict(glm2, test[i, ], type = "response")
  if (pred_X > best_threshold_glm) {
    capital <- capital + test$pl_value[i]
    drawdown <- c(drawdown, capital)
    wins <- c(wins, test$pl_value[i])
  }
}

ggplot(data = data.frame(trade = 1:length(drawdown),
```

```
                       equity = drawdown),
       aes(x = trade, y = equity)) +
  geom_line(color = "blue", size = 1) +
  theme_minimal() +
  labs(title = "Equity Curve",
       x = "Trade #",
       y = "Capital") +
  theme(
    plot.title = element_text(size = 16, face = "bold", hjust = 0.5),
    axis.title = element_text(size = 12),
    axis.text = element_text(size = 10),
    panel.grid.major = element_line(color = "grey90"),
    panel.grid.minor = element_line(color = "grey95")
  ) +
  scale_y_continuous(labels = scales::comma)
```

```
## 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.
```




```
# Sharpe ratio (simplified version: mean return over SD of returns)
sharpe <- (tail(drawdown, 1) - initial_capital) / sd(wins)
```

```r
peak = max(drawdown)
trough = min(drawdown)
max_drawdown = (trough-peak)/peak


cat("Initial Capital", initial_capital)
```

```
## Initial Capital 2191175
```

```r
cat("\nFinal Capital:", tail(drawdown, 1), "\n")
```

```
##
## Final Capital: 2191611
```

```r
cat("Sharpe Ratio:", round(sharpe, 3), "\n")
```

```
## Sharpe Ratio: 11.129
```

```r
cat("Max Drawdown:", max_drawdown*100, "%\n")
```

```
## Max Drawdown: -0.01989416 %
```

```r
cat("Returns:", ((capital/initial_capital)-1)*100, "%\n")
```

```
## Returns: 0.01989812 %
```

```r
returns <- diff(drawdown)

# Count wins and total trades
num_wins <- sum(returns > 0)
num_losses <- sum(returns <= 0)
total_trades <- length(returns)

# Winrate as percentage
winrate <- num_wins / total_trades

cat("Precision:", round(winrate * 100, 2), "%\n")
```

```
## Precision: 50 %
```

XGboost

In this section, we use **XGBoost**, a gradient-boosted decision tree ensemble, to classify trade direction. Our goal is not just accuracy but **precision**, the ratio of true positives to all predicted positives. This focus reduces false positives—critical for real-world trading where bad trades are costly.

Load Data

We construct a design matrix using all two- and three-way interactions of trading indicators. The outcome variable `PL` is a binary indicator for whether a trade was profitable.

```
library(xgboost)
library(Matrix)
library(caret)
```

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

```
data = train
X = model.matrix(PL ~ (ATR + SMA_k7 + SMA_k50 + norm_volume)^4,
                 data=data)[, -1]
y = data$PL
dtrain = xgb.DMatrix(data = X, label = y)
```

XGBoost Theoretical Framework

XGBoost builds an ensemble of decision trees sequentially, with each new tree correcting errors made by previous trees. The algorithm optimizes:

$$\mathcal{L}(\phi) = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

Where: - l is a differentiable convex loss function - $\hat{y}_i$ is the prediction - Ω is a regularization term controlling model complexity - f_k represents the k-th tree in the ensemble

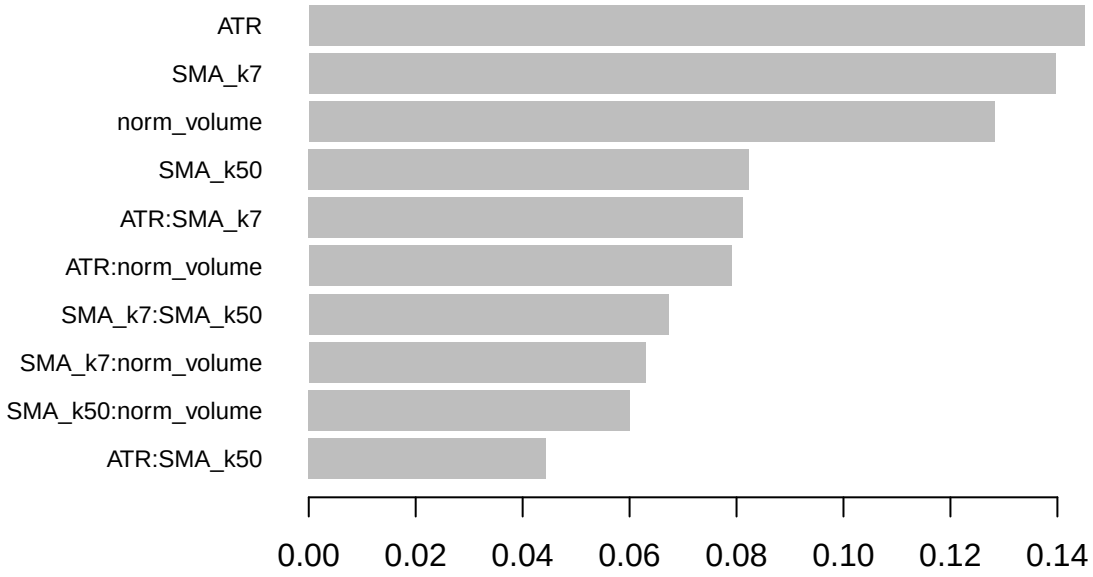
This approach allows XGBoost to capture complex non-linear relationships and interactions that linear models might miss.

Cross-Validation and Training

We optimize for **AUC-PR** (area under the precision-recall curve), which directly measures precision vs. recall trade-offs.

```
param_grid <- expand.grid(
  eta = 0.05,
  max_depth = 3,
  subsample = 0.8,
  colsample_bytree = 0.8,
  eval_metric = "aucpr",          # maximize precision-recall AUC
  objective = "binary:logistic"  # output probabilities
)

watchlist = list(train = dtrain)

model <- xgb.train(
  params = as.list(param_grid),
  data = dtrain,
  nrounds = 100,
```

```
  watchlist = watchlist,
  verbose = 0
)

importance_matrix <- xgb.importance(model = model)
xgb.plot.importance(importance_matrix[1:10,])
```



Threshold Selection Based on Precision

Rather than defaulting to a 0.5 decision threshold, we **optimize threshold** T to maximize:

$$\text{Precision} = \frac{TP}{TP + FP}$$

We assume model predictions $\hat{y} \sim N(\mu, \sigma^2)$ and scan high quantiles (e.g., 0.99999) to reduce false positives.

```
pred_probs <- predict(model, dtrain)
probs <- 10^seq(-2, -10, length.out = 9)

fit_mean = mean(pred_probs)
fit_sd = sd(pred_probs)
```

```
best_precision = 0
best_threshold_xg = 0.5

for (prob in probs){
  threshold <- qnorm(1 - prob, mean = fit_mean, sd = fit_sd)
  preds = ifelse(pred_probs > threshold, 1, 0)
  TP = sum(preds == 1 & train$PL == 1)
  FP = sum(preds == 1 & train$PL == 0)
  if ((TP + FP) < 30) {
    break # CLT
  }
  precision = ifelse((TP + FP) == 0, NA, TP / (TP + FP))
  if (!is.na(precision) && precision > best_precision){
    best_precision = precision
    best_threshold_xg = threshold
  }
  cat(sprintf("Prob = %e | TP = %d | FP = %d | Winrate = %.3f\n",
              prob, TP, FP, precision))
}
```

```
## Prob = 1.000000e-02 | TP = 128 | FP = 39 | Winrate = 0.766
```

```
cat("Best Precision:", round(best_precision * 100, 2), "% at Threshold", best_threshold_xg, "\n")
```

```
## Best Precision: 76.65 % at Threshold 0.5555247
```

Backtesting Equity Curve

Using the **test set**, we execute trades where the predicted probability $\hat{y} > T$, and track capital changes.

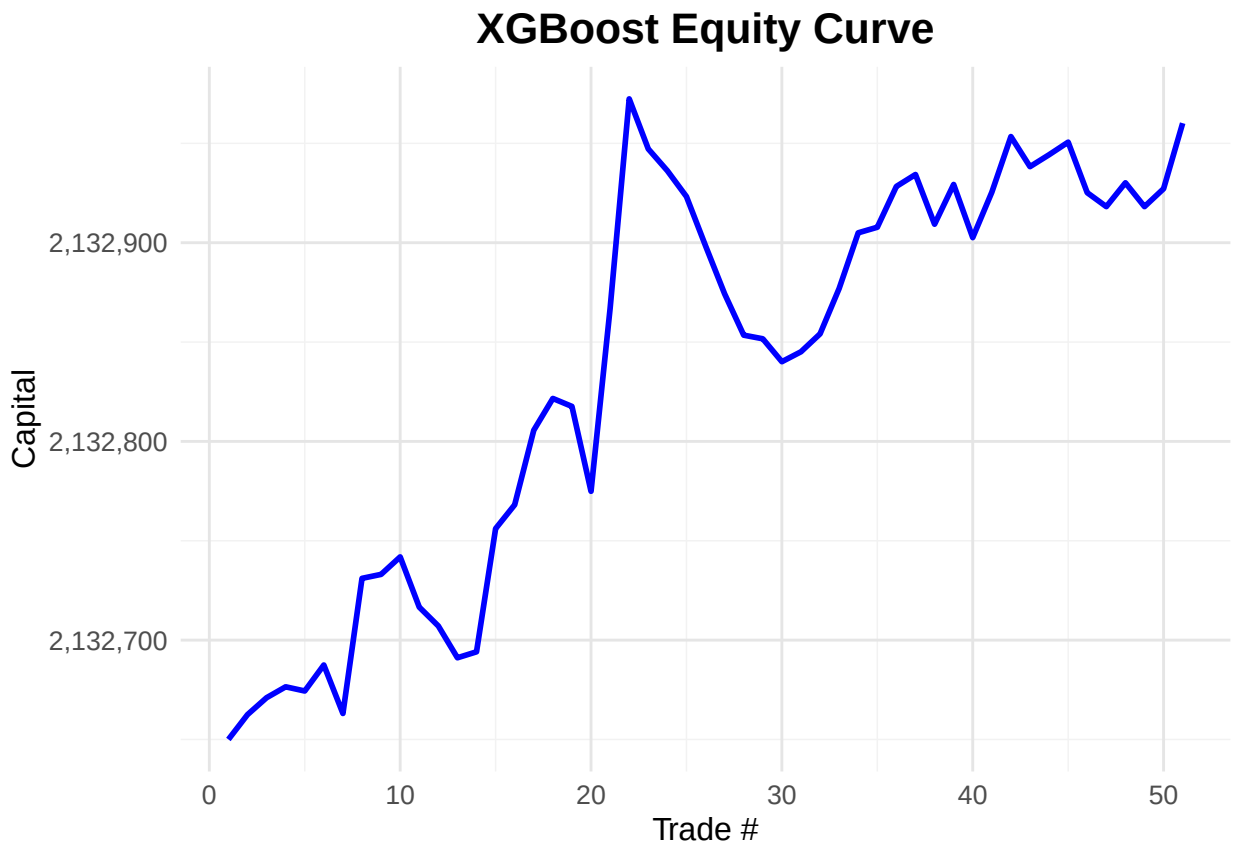
Let: - C_0: Initial capital - r_i: Profit/loss from trade i - T: Threshold for high-precision entry

Then:

$$C_{i+1} = C_i + r_i \quad \text{only if } \hat{y}_i > T$$

```
test_X = model.matrix(PL ~ (ATR + SMA_k7 + SMA_k50 + norm_volume)^4, data=test)[, -1]
dtest = xgb.DMatrix(data = test_X)

pred_probs_xg <- predict(model, dtest)

initial_capital = (1/(0.02)) * (100 * sample(train$Open, 1))
capital = initial_capital
drawdown = c(capital)
wins = c()

for (i in 1:nrow(test)) {
  pred_X = pred_probs_xg[i]
  if (pred_X > best_threshold_xg) {
    capital = capital + test$pl_value[i]
    drawdown = c(drawdown, capital)
    wins = c(wins, test$pl_value[i])
  }
```

```
}

ggplot(data = data.frame(trade = 1:length(drawdown),
                         equity = drawdown),
       aes(x = trade, y = equity)) +
  geom_line(color = "blue", size = 1) +
  theme_minimal() +
  labs(title = "XGBoost Equity Curve",
       x = "Trade #",
       y = "Capital") +
  theme(
    plot.title = element_text(size = 16, face = "bold", hjust = 0.5),
    axis.title = element_text(size = 12),
    axis.text = element_text(size = 10),
    panel.grid.major = element_line(color = "grey90"),
    panel.grid.minor = element_line(color = "grey95")
  ) +
  scale_y_continuous(labels = scales::comma)
```




```
# Performance metrics
sharpe = (tail(drawdown, 1) - drawdown[1]) / sd(wins)
max_dd = min(drawdown - cummax(drawdown)) / max(drawdown)

cat("Final Capital:", capital, "\n")
```

```
## Final Capital: 2132960
```

```
cat("Sharpe Ratio:", round(sharpe, 3), "\n")
```

```
## Sharpe Ratio: 10.598
```

```
cat("Max Drawdown:", round(max_dd * 100, 2), "%\n")
```

```
## Max Drawdown: -0.01 %
```

```
cat("Returns:", ((capital/initial_capital)-1)*100, "%\n")
```

```
## Returns: 0.01454337 %
```

```
returns <- diff(drawdown)

# Count wins and total trades
num_wins <- sum(returns > 0)
num_losses <- sum(returns <= 0)
total_trades <- length(returns)

# Winrate as percentage
winrate <- num_wins / total_trades

cat("Precision:", round(winrate * 100, 2), "%\n")
```

```
## Precision: 58 %
```

Commentary

XGBoost is particularly effective when: - Feature interactions matter - There are non-linear decision boundaries - **precision over recall**

Its ability to output probabilities makes it well-suited for **threshold tuning**, which we exploit using quantiles of the fitted distribution. By focusing on **extremely confident predictions**, we reduce bad trades and focus capital on high-reward setups.

Support Vector Machines

Support Vector Machines (SVM) offer a powerful alternative approach for classification tasks in trading. Unlike linear models, SVMs can efficiently handle non-linear decision boundaries by projecting data into higher-dimensional spaces where separation becomes possible.

Theoretical Foundation

The SVM algorithm finds the optimal hyperplane that maximizes the margin between classes. For linearly separable data, this involves solving:

$$\min_{\mathbf{w},b} \frac{1}{2}||\mathbf{w}||^2$$

$$\text{subject to } y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 \text{ for all } i$$

For non-linear boundaries, SVMs employ the "kernel trick" to implicitly map data to higher dimensions without explicitly computing the transformation. The decision function becomes:

$$f(\mathbf{x}) = \text{sign}\left(\sum_{i=1}^{n} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b\right)$$

Where $K(\mathbf{x}_i, \mathbf{x})$ is the kernel function measuring similarity between points.

Model Implementation

RBF (Radial Basis Function) kernel in your SVM is a solid choice for optimizing precision

```
library(e1071)

X = model.matrix(PL ~ (ATR + SMA_k7 + SMA_k50 + norm_volume)^4,
                 data=train)[, -1]
y = as.factor(train$PL)

svm_model <- svm(x = X, y = y,
                 kernel = "radial", # RBF is renowned for precision
                 probability = TRUE,
                 cost = 10,
                 gamma = 0.1)

summary(svm_model)
```

```
##
## Call:
## svm.default(x = X, y = y, kernel = "radial", gamma = 0.1, cost = 10,
##     probability = TRUE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  10
##
## Number of Support Vectors:  13978
##
##  ( 6947 7031 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

Kernel Selection Rationale

The radial basis function (RBF) kernel is chosen for its ability to model complex non-linear decision boundaries. The RBF kernel computes:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma||\mathbf{x}_i - \mathbf{x}_j||^2)$$

Where γ controls the influence radius of each support vector. This allows the model to capture local patterns in the feature space that might be missed by linear models.

Hyperparameter Tuning

Far too computationally expensive to run, `tune.svm` will take forever to compile. This is an area of improvement.

Threshold Optimization for Precision

```
svm_probs <- attr(predict(svm_model, X, probability = TRUE),
                  "probabilities")[, "1"]

# Define threshold range
probs <- 10^seq(-2, -10, length.out = 9)
fit_mean <- mean(svm_probs)
fit_sd <- sd(svm_probs)

best_precision <- 0
best_threshold_svm <- 0.5

for (prob in probs) {
  threshold <- qnorm(1 - prob, mean = fit_mean, sd = fit_sd)
  preds <- ifelse(svm_probs > threshold, 1, 0)

  TP <- sum(preds == 1 & train$PL == 1)
  FP <- sum(preds == 1 & train$PL == 0)

  if ((TP + FP) < 30) {
    break # Ensure statistical significance
  }

  precision <- ifelse((TP + FP) == 0, NA, TP / (TP + FP))

  if (!is.na(precision) && precision > best_precision) {
    best_precision <- precision
    best_threshold_svm <- threshold
  }
  cat(sprintf("Prob = %e | TP = %d | FP = %d | Winrate = %.3f\n",
              prob, TP, FP, precision))
}
```

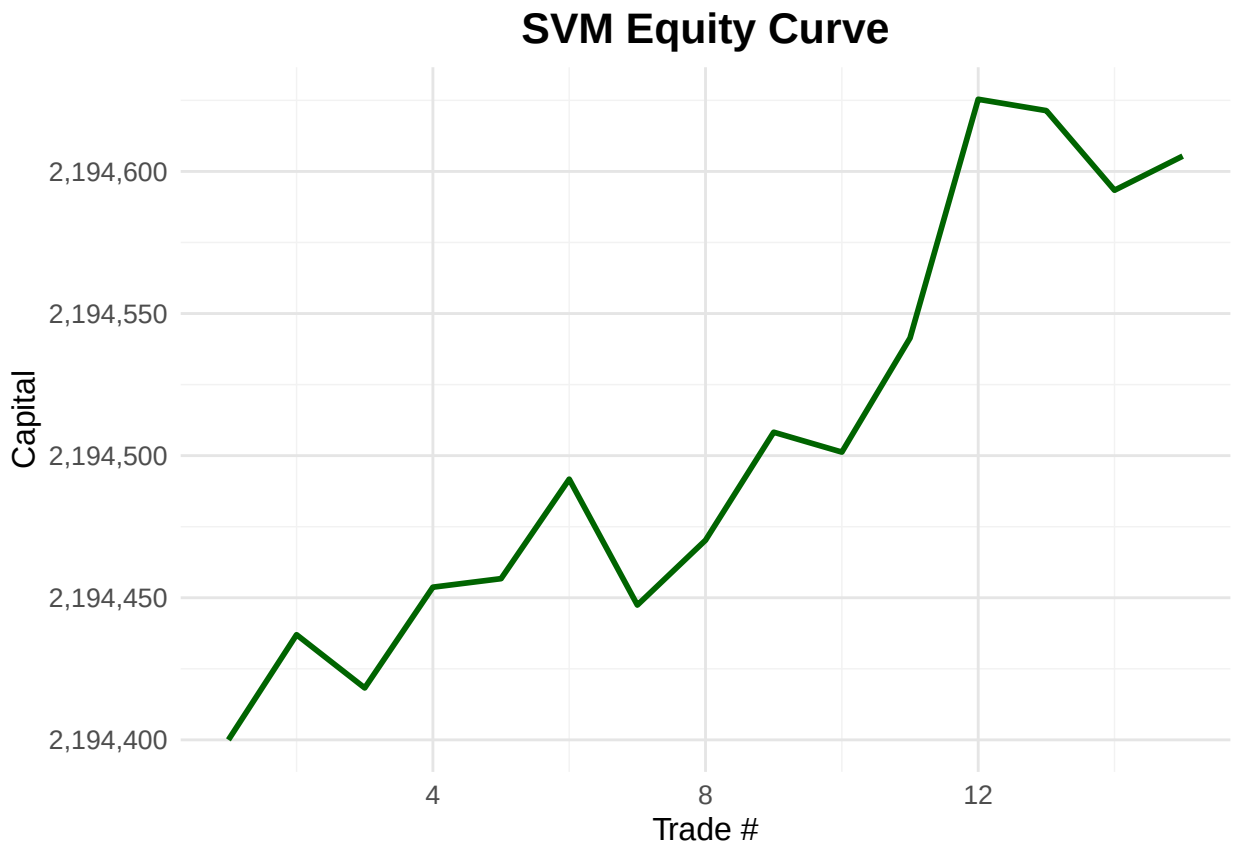
```
## Prob = 1.000000e-02 | TP = 606 | FP = 251 | Winrate = 0.707
## Prob = 1.000000e-03 | TP = 340 | FP = 93 | Winrate = 0.785
## Prob = 1.000000e-04 | TP = 71 | FP = 28 | Winrate = 0.717
## Prob = 1.000000e-05 | TP = 35 | FP = 9 | Winrate = 0.795
```

```r
cat("SVM Best Precision:", round(best_precision * 100, 2),
    "% at Threshold", best_threshold_svm, "\n")
```

```
## SVM Best Precision: 79.55 % at Threshold 0.6121074
```

Backtesting SVM Strategy

```r
#test data
test_X <- model.matrix(PL ~ (ATR + SMA_k7 + SMA_k50 + norm_volume)^4,
                       data=test)[, -1]

# Get probability predictions for test set
test_probs_svm <- attr(predict(svm_model, test_X, probability = TRUE), "probabilities")[, "1"]

initial_capital <- (1/(0.02)) * (100 * sample(train$Open, 1))
capital <- initial_capital
drawdown <- c(capital)
wins <- c()

for (i in 1:nrow(test)) {
  pred_X <- test_probs_svm[i]
  if (pred_X > best_threshold_svm) {
    capital <- capital + test$pl_value[i]
    drawdown <- c(drawdown, capital)
    wins <- c(wins, test$pl_value[i])
  }
}

ggplot(data = data.frame(trade = 1:length(drawdown),
                         equity = drawdown),
       aes(x = trade, y = equity)) +
  geom_line(color = "darkgreen", size = 1) +
  theme_minimal() +
  labs(title = "SVM Equity Curve",
       x = "Trade #",
       y = "Capital") +
  theme(
    plot.title = element_text(size = 16, face = "bold", hjust = 0.5),
    axis.title = element_text(size = 12),
    axis.text = element_text(size = 10),
    panel.grid.major = element_line(color = "grey90"),
    panel.grid.minor = element_line(color = "grey95")
  ) +
  scale_y_continuous(labels = scales::comma)
```




```
# Calculate performance metrics
sharpe <- (tail(drawdown, 1) - drawdown[1]) / sd(wins)
max_dd <- min(drawdown - cummax(drawdown)) / max(drawdown)

cat("Initial Capital:", initial_capital, "\n")
```

```
## Initial Capital: 2194400
```

```
cat("Final Capital:", capital, "\n")
```

```
## Final Capital: 2194605
```

```
cat("Sharpe Ratio:", round(sharpe, 3), "\n")
```

```
## Sharpe Ratio: 6.09
```

```
cat("Max Drawdown:", round(max_dd * 100, 2), "%\n")
```

```
## Max Drawdown: 0 %
```

```
cat("Returns:", round(((capital/initial_capital)-1)*100, 2),
    "%\n")
```

```
## Returns: 0.01 %
```

```
returns <- diff(drawdown)

# Count wins and total trades
num_wins <- sum(returns > 0)
num_losses <- sum(returns <= 0)
total_trades <- length(returns)

# Winrate as percentage
winrate <- num_wins / total_trades

cat("Precision:", round(winrate * 100, 2), "%\n")
```

```
## Precision: 64.29 %
```

SVM Advantages for Trading

SVMs offer several advantages for financial prediction:

1. **Robustness to Outliers**: The support vector focus makes SVMs less sensitive to outliers compared to regression models.
2. **Effective in High-Dimensional Spaces**: SVMs perform well with many features relative to sample size, ideal for complex market data.
3. **Flexibility**: Through kernel selection, SVMs can model various decision boundaries without explicit feature engineering.
4. **Theoretical Guarantees**: SVMs are founded on statistical learning theory, providing formal bounds on generalization error.

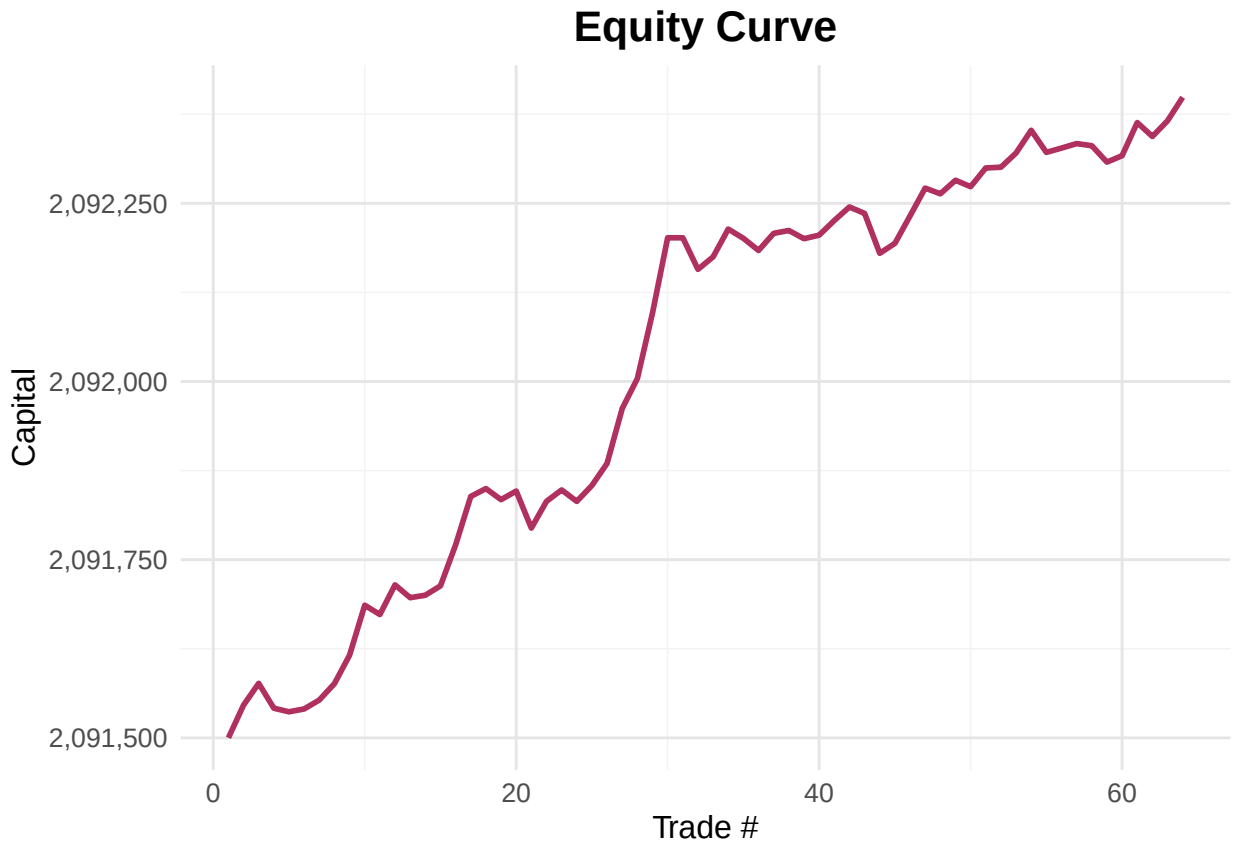
The non-linear mapping capability is particularly valuable for capturing market regimes where relationships between indicators and outcomes change dynamically.

Ensemble prediction

For production trading systems, an ensemble approach combining these models might offer the most robust performance across varying market conditions. Each model captures different aspects of market behavior, and their combined signals could provide more reliable trading decisions than any single model alone.

```
initial_capital = (1/(0.02))*(100 * sample(train$Open, 1)) # random price
capital <- initial_capital
drawdown <- c(capital)
wins <- c()

for (i in 1:nrow(test)) {
  pred_X <- predict(glm2, test[i, ], type = "response")
  pred_xg = pred_probs_xg[i]
  pred_svm = test_probs_svm[i]
  # Best threshold not used because all 3 never aligned in dataset
```

```
  if (pred_X > 0.5 & pred_xg > 0.5 & pred_svm > 0.5) {
    capital <- capital + test$pl_value[i]
    drawdown <- c(drawdown, capital)
    wins <- c(wins, test$pl_value[i])
  }
}

ggplot(data = data.frame(trade = 1:length(drawdown),
                         equity = drawdown),
       aes(x = trade, y = equity)) +
  geom_line(color = "maroon", size = 1) +
  theme_minimal() +
  labs(title = "Equity Curve",
       x = "Trade #",
       y = "Capital") +
  theme(
    plot.title = element_text(size = 16, face = "bold", hjust = 0.5),
    axis.title = element_text(size = 12),
    axis.text = element_text(size = 10),
    panel.grid.major = element_line(color = "grey90"),
    panel.grid.minor = element_line(color = "grey95")
  ) +
  scale_y_continuous(labels = scales::comma)
```



```r
# Sharpe ratio (simplified version: mean return over SD of returns)
sharpe <- (tail(drawdown, 1) - initial_capital) / sd(wins)


peak = max(drawdown)
trough = min(drawdown)
max_drawdown = (trough-peak)/peak


cat("Initial Capital", initial_capital)
```

```
## Initial Capital 2091500
```

```r
cat("\nFinal Capital:", tail(drawdown, 1), "\n")
```

```
##
## Final Capital: 2092399
```

```r
cat("Sharpe Ratio:", round(sharpe, 3), "\n")
```

```
## Sharpe Ratio: 28.095
```

```r
cat("Max Drawdown:", max_drawdown*100, "%\n")
```

```
## Max Drawdown: -0.04295543 %
```

```r
cat("Returns:", ((capital/initial_capital)-1)*100, "%\n")
```

```
## Returns: 0.04297389 %
```

```r
returns <- diff(drawdown)

# Count wins and total trades
num_wins <- sum(returns > 0)
num_losses <- sum(returns <= 0)
total_trades <- length(returns)

# Winrate as percentage
winrate <- num_wins / total_trades

cat("Precision:", round(winrate * 100, 2), "%\n")
```

```
## Precision: 68.25 %
```

‘‘‘

Final Thoughts

We have now explored **three models** in depth:

Model	Output Type	Thresholding	Precision Tuning	Purpose
Logistic Regression (GLM)	Probabilities (binary)	Yes	qnorm thresholds	Predict trade direction
XGBoost	Probabilities (binary)	Yes	precision-optimized threshold	Precision-focused execution
Support Vector Machines	Probabilities (binary)	Yes	precision-optimized threshold	Non-linear boundary detection

Each model offers distinct advantages for trading strategy development:

GLM provides interpretable coefficients that directly quantify the impact of each feature on trade outcomes. This interpretability is valuable for understanding market dynamics and regulatory compliance.

XGBoost excels at capturing complex feature interactions and non-linear patterns without requiring explicit specification. Its ensemble nature provides robustness against overfitting, particularly important in noisy financial data.

SVM offers strong theoretical guarantees and effectively handles non-linear decision boundaries through kernel methods. Its focus on maximizing the margin between classes helps identify the most reliable trading signals.

In trading, minimizing **false positives** is often more critical than maximizing true positives due to the **asymmetric cost of bad trades**. All three models allow for threshold optimization to achieve precision-focused execution. These models were put into production, and we found that glm models performed the best in real-time trading.