Ene434

2025-05-14

Candidate numbers: 12 & 21

Introduction

Norway's open electricity market faces several key challenges, with price volatility being one of the most significant. As Sheybanivaziri et al. (2024, pp. 2–3) note, the market is highly sensitive to fluctuations in demand, supply, and a range of external factors. Due to the low price elasticity of electricity, prices tend to closely follow shifts in supply, which is inherently volatile (Lijesen, 2007). This volatility is largely driven by weather dependent hydropower production and further amplified by Norway's interconnectedness with the broader European power market. This has inspired our study, in which we aim to explore the use of machine learning to predict future volatility in the power market.

Our main research question is as follows: *To what extent can a classification model, using historical electricity prices, weather data, and calendar based variables, accurately predict whether price volatility in Norway's NO5 bidding zone will be high or low on a target day exactly seven days ahead?*

The Norwegian electricity market has long been recognized for its provision of cheap, clean, and reliable power. However, in recent years, structural changes in electricity supply logistics combined with increased global uncertainty have led to higher levels of price volatility (Sheybanivaziri et al., 2024, pp. 2-3). Through this project, we aim to assist producers and hedgers in planning by forecasting price spikes and high volatility periods. By using classification and machine learning techniques, we seek to predict whether the day ahead market (specifically, seven days into the future) will experience high price volatility. This will provide market participants with a practical tool for operational and financial decision making, delivered through a predictive binary classification model for the NO5 region in Norway.

Background

Volatility forecasting

Volatility forecasting plays a key role in risk management and various strategic decisions. For instance, if high volatility periods can be predicted, hedging strategies can be timed more effectively using forward or options contracts. As mentioned by Sheybanivaziri et al. (2024, pp. 2-3), high volatility leads to greater price uncertainty and risk, which can affect market margins and the value for hedgers.

This is particularly relevant in the Norwegian power market, which relies heavily on natural resources such as hydropower. Improved forecasting may help optimize the use of water reservoirs. Sheybanivaziri et al. (2024, pp. 2-3) argue further that Norway's spot prices have become increasingly dependent on imports and weather conditions, indicating greater volatility than in previous years.

Greater volatility can lead to larger forecasting errors, increasing the risk of unnecessary costs, and affecting especially low margin contracts. Moreover, large consumers and retailers can benefit from volatility forecasts, as such information helps assess price risks and adjust procurement strategies accordingly (Zareipour et al., 2011, pp. 165-166).

While forecasting exact prices may seem more intuitive, research suggests that forecasting volatility may yield more accurate and actionable insights. Sheybanivaziri et al. (2024, p. 4) indicate that although forecasts often perform well on average, machine learning and probabilistic approaches tend to outperform during highly volatile conditions.

Binary classification approach (Literature review)

Whereas traditional volatility studies often rely on time series- or GARCH-type models to estimate continuous values, this paper adopts a classification based approach to predict future volatility. Specifically, we aim to forecast whether future price volatility will exceed a predefined value threshold, turning the problem into a binary classification task.

The strength of this approach is that it reflects how decisions are made in the power market. Market participants rarely rely on exact volatility estimates, but rather on whether expected volatility crosses a certain threshold, for example, whether a day is likely to be volatile or not. Knowing whether high volatility is likely can help them make more robust strategic decisions in advance.

Zareipour et al. (2011, pp. 165-168) introduced a classification based approach to electricity price forecasting. The idea was to develop a tool that could indicate whether prices were likely to fall within a specific threshold, thereby enabling managers to base decisions on categorical outcomes rather than precise price values. The authors argue that for demand side management, it is often sufficient and more practical, to forecast categories such as "high" or "low" prices. Their results demonstrate that classification methods, particularly Support Vector Machines (SVM), can outperform traditional regression models, especially in aiding demand side management and grid optimization.

Furthermore, as previously referenced, Sheybanivaziri et al. (2024, pp. 4-6) emphasize that while traditional forecasting models may perform well on average, they often struggle during periods of heightened volatility. These are precisely the conditions where accurate predictions are most critical. The authors argue that non linear classification models, particularly tree based methods, are better suited to capturing complex relationships and handling noisy data, both of which are common in high volatility environments. The study underscores the importance of detecting and forecasting extreme price spikes. As illustrated in "Standard Deviation of the price for 2020 to 2023" later displayed, electricity prices have exhibited significantly higher and more erratic spikes after 2021 compared to earlier periods, further motivating the use of advanced classification techniques to address recent market developments.

This approach is more resilient to noisy real world data and outliers, which often compromise the accuracy of parametric models. As such, our method seeks to provide a more robust prediction of whether future days will be volatile, rather than attempting to predict precise volatility levels.

7-day prediction horizon

We have selected a 7-day prediction horizon, meaning the model predicts the volatility on the specific day that lies exactly 7 days ahead, to align with the operational and strategic planning needs of key market participants. These participants are power producers, retailers and hedging agents. While short term forecasts like day ahead or intraday tend to be more accurate, they are mainly useful for immediate trading decisions. In contrast, decisions related to hydropower dispatch, generation scheduling and maintenance planning typically require several days of lead time. Moreover, forward contracts and industry level power purchases in the Nord Pool market are often negotiated on weekly or monthly time frames, making a 7-day forecast more practically applicable to these actors.

In addition, a weekly horizon allows the model to incorporate important temporal dynamics such as weather patterns, calendar effects and hydro power storage. These seasonal and calendar based factors are particularly influential in the Nordic power system, where electricity demand surges during colder periods and hydropower production is closely linked to inflow cycles and reservoir planning (Sheybanivaziri et al., 2024, pp. 22–23). Therefore, a 7-day forecast horizon strikes a balance between forecast usefulness, strategic relevance, and data driven seasonality capture.

Data

We have collected electricity data from ENTSO-E and weather data from SeKlima. The electricity data primarily includes hourly prices from price zone N05. Prices are reported in EUR/MWh, while consumption is measured in MWh. Additionally, the dataset contains information on the potential amount of electricity stored in hydropower reservoirs each month. This variable is labeled as "storage" in the dataset. From SeKlima, we have gathered weather data related to wind and temperature conditions for Bergen area, from Florida weather station.

Besides this we have computed summary statistics for the electricity price variable. These summary statistics contains the moving average for 7 and 30 days, as well as the rolling range, standard deviation and the range of the 3qr minus the 1qr. Some of these metric is also lagged 30 and 7 days.

The classification is at a daily level so, the data is aggregated to a daily level.

Seasonality is controlled by the use of the variables month and weekend. A dummy variable for the Russian invasion of Ukraine is included because of it's severe impact on the energy market in Europe.

```
### Loading libraries --------
library(tidyverse)
library(pROC)
library(glmnet)
library(caret)
library(randomForest)
library(patchwork)
```

```
### Joining data --------
prices <- prices %>%
  rename(time = 1)

temp_wind <- temp %>%
  left_join(wind, by = "time")

data <- prices %>%
  left_join(temp_wind, by = "time") %>%
  fill(everything(), .direction = "down") %>%
  mutate(
    year = year(time),
    week = week(time)
  )

data <- data %>%
  left_join(water, by = c("year", "week"))

data <- data %>%
  arrange(time) %>%
  complete(time = seq(min(time), max(time) + hours(1), by = "hour")) %>%
  filter(time < "2025-03-01 00:01:00") %>%
  fill(everything(), .direction = "down")
```

```
### Aggregating data --------
data <- data %>%
  mutate(
    rolling_7_sd = map_dbl(row_number(), ~ rolling_sd(.x, price, 7)),
    rolling_30_sd = map_dbl(row_number(), ~ rolling_sd(.x, price, 30)),
```

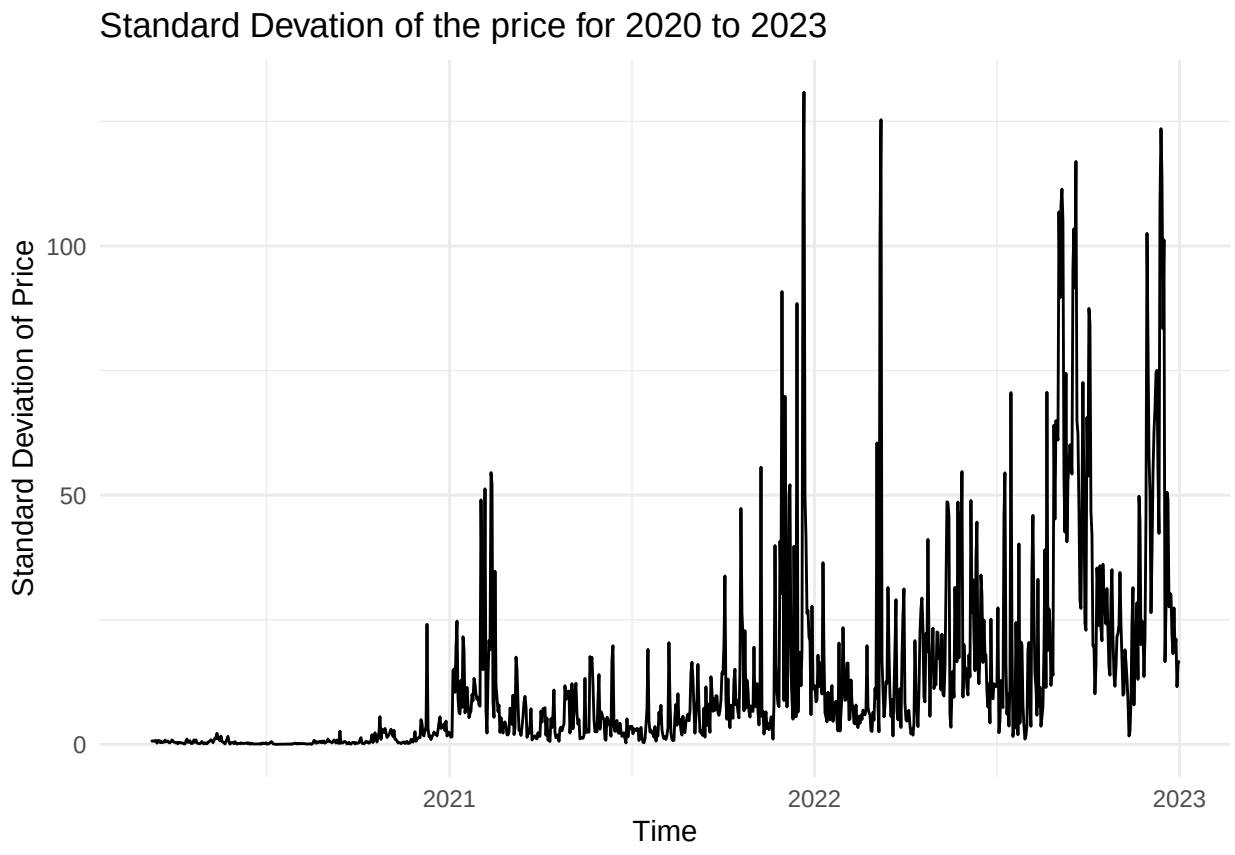
```
    ma_7 = map_dbl(row_number(), ~ moving_average(.x, price, 7)),
    ma_30 = map_dbl(row_number(), ~ moving_average(.x, price, 30)),
    rolling_7_range = map_dbl(row_number(), ~ rolling_range(.x, price, 7)),
    rolling_30_range = map_dbl(row_number(), ~ rolling_range(.x, price, 30)),
    rolling_7_iqr = map_dbl(row_number(), ~ rolling_iqr(.x, price, 7)),
    rolling_30_iqr = map_dbl(row_number(), ~ rolling_iqr(.x, price, 30)),
    rolling_7_median = map_dbl(row_number(), ~ rolling_median(.x, price, 7)),
    rolling_30_median = map_dbl(row_number(), ~ rolling_median(.x, price, 30))
  )


data <- data %>%
  mutate(
    month = month(time),
    day_of_week = as.numeric(format(time, "%u")),
    weekend = ifelse(day_of_week >= 6, 1, 0),
    day = day(time)
  ) %>%
  drop_na()


day_data <- data %>%
  group_by(year, month, day) %>%
  summarise(
    day_of_week = mean(day_of_week, na.rm = T),
    weekend = mean(weekend, na.rm = T),
    mean_price = mean(price, na.rm = T),
    sd_price = sd(price, na.rm = T),
    median_price = median(price, na.rm = T),
    range_price = max(price) - min(price),
    iqr_price = quantile(price, 0.75) - quantile(price, 0.25),
    mean_temp = mean(temp, na.rm = T),
    sd_temp = sd(temp, na.rm = T),
    mean_wind = mean(wind, na.rm = T),
    sd_wind = sd(wind, na.rm = T),
    storage = mean(storage, na.rm = T),
    time = mean(time, na.rm = T),
    rolling_7_sd = mean(rolling_7_sd, na.rm = T),
    rolling_30_sd = mean(rolling_30_sd, na.rm = T),
    ma_7 = mean(ma_7, na.rm = T),
    ma_30 = mean(ma_30, na.rm = T),
    rolling_7_range = mean(rolling_7_range, na.rm = T),
    rolling_30_range = mean(rolling_30_range, na.rm = T),
    rolling_7_iqr = mean(rolling_7_iqr, na.rm = T),
    rolling_30_iqr = mean(rolling_30_iqr, na.rm = T),
    rolling_7_median = mean(rolling_7_median, na.rm = T),
    rolling_30_median = mean(rolling_30_median, na.rm = T),
    .groups = "drop"
  )

forecast_horizon <- 7
ukraine_war_date <- ymd_hms("2022-02-24 00:00:00")
```

```r
# The prefix 'd_' indicates that these variables are shifted to exclude future information,
# preventing data leakage from the period we aim to forecast.

day_data <- day_data %>%
  mutate(
    d_price = lag(mean_price, forecast_horizon),
    d_sd_price = lag(sd_price, forecast_horizon),
    d_median_price = lag(median_price, forecast_horizon),
    d_range_price = lag(range_price, forecast_horizon),
    d_iqr_price = lag(iqr_price, forecast_horizon),
    d_rolling_7_sd = lag(rolling_7_sd, forecast_horizon),
    d_rolling_30_sd = lag(rolling_30_sd, forecast_horizon),
    d_ma_7 = lag(ma_7, forecast_horizon),
    d_ma_30 = lag(ma_30, forecast_horizon),
    d_temp = lag(mean_temp, forecast_horizon),
    d_sd_temp = lag(sd_temp, forecast_horizon),
    d_wind = lag(mean_wind, forecast_horizon),
    d_sd_wind = lag(sd_wind, forecast_horizon),
    d_storage = lag(storage, forecast_horizon),
    d_rolling_7_range = lag(rolling_7_range, forecast_horizon),
    d_rolling_30_range = lag(rolling_30_range, forecast_horizon),
    d_rolling_7_iqr = lag(rolling_7_iqr, forecast_horizon),
    d_rolling_30_iqr = lag(rolling_30_iqr, forecast_horizon),
    d_rolling_7_median = lag(rolling_7_median, forecast_horizon),
    d_rolling_30_median = lag(rolling_30_median, forecast_horizon),
    year = as.character(year),
    month = as.factor(month),
    day = as.factor(day),
    day_of_week = as.factor(day_of_week),
    weekend = as.factor(weekend),
    war = as.factor(if_else(time > ukraine_war_date, "Yes", "No"))
  ) %>%
  drop_na() %>%
  select(sd_price, mean_price, time, year, month, weekend, war, starts_with("d_"))

# Adding actual lagged variables
day_data <- day_data %>%
  mutate(
    lag7_d_price = lag(d_price, 7),
    lag30_d_price = lag(d_price, 30),
    lag7_d_rolling_7_sd = lag(d_rolling_7_sd, 7),
    lag30_d_roling_7_sd = lag(d_rolling_7_sd, 30),
    lag7_d_rolling_30_sd = lag(d_rolling_30_sd, 7),
    lag30_d_roling_30_sd = lag(d_rolling_30_sd, 30),
    lag7_d_ma_7 = lag(d_ma_7, 7),
    lag30_d_ma_7 = lag(d_ma_7, 30),
    lag7_d_ma_30 = lag(d_ma_30, 7),
    lag30_d_ma_30 = lag(d_ma_30, 30),
  ) %>%
  drop_na()

day_data <- day_data %>%
  mutate(season = case_when(
```

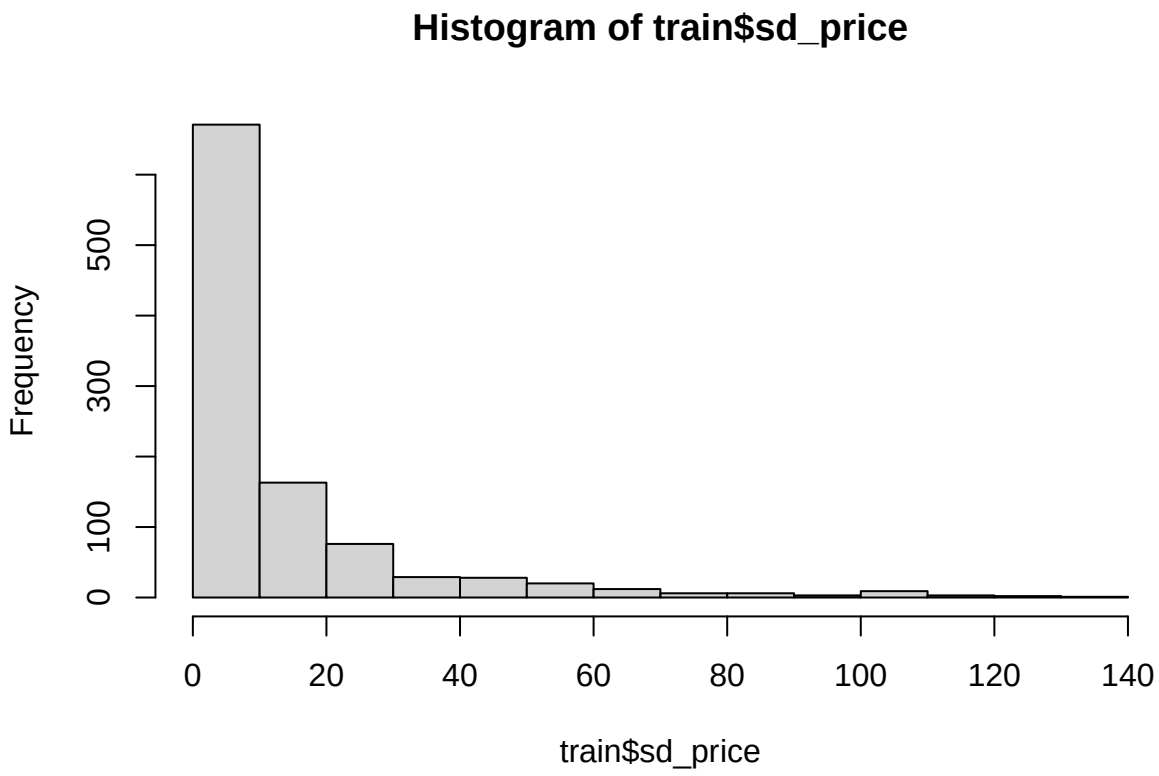
```
    month %in% c("12", "1", "2") ~ "Winter",
    month %in% c("3", "4", "5") ~ "Spring",
    month %in% c("6", "7", "8") ~ "Summer",
    month %in% c("9", "10", "11") ~ "Fall"
  ))
```

We are splitting the data, into training and test data. The training data includes information from the years 2020 to 2023, while the test data covers the period from 2023 to March 2025.

```
### Test and train split --------
set.seed(123)

day_data$year <- as.numeric(day_data$year)

train <- day_data %>%
  filter(year < 2023)

ggplot(train, aes(x = time, y = sd_price)) +
  geom_line() +
  labs(
    title = "Standard Devation of the price for 2020 to 2023",
    x = "Time",
    y = "Standard Deviation of Price"
  ) +
  theme_minimal()
```



```r
hist(train$sd_price)
```



As observed from the histogram above and the summary below, there is a wide range of standard deviations, although most of the values tend to be relatively low.

```r
summary(train$sd_price)
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
##   0.00933   1.09680   5.50464  12.85230  15.42507 130.83160
```

```r
train <- train %>%
  mutate(high_vol = as.factor(ifelse(sd_price > 10, 1, 0))) %>%
  select(-year, -mean_price, -sd_price)

test <- day_data %>%
  filter(year >= 2023) %>%
  mutate(high_vol = as.factor(ifelse(sd_price > 10, 1, 0))) %>%
  select(-year, -mean_price, -sd_price)


table(train$high_vol)
```

```
##
##   0   1
## 671 358
```

```r
table(test$high_vol)
```

```
##
##   0   1
## 496 294
```

A standard deviation threshold of 10 is used to categorize days into either high volatility or low volatility. Here, 1 indicates high volatility, and 0 indicates low volatility. As a consequence of using 10 as the standard deviation threshold, our data is somewhat imbalanced, which may pose challenges for the machine learning model. Additionally, the training and test datasets have different ratios of high and low volatility days. This issue could be addressed using techniques like SMOTE, but addressing it is beyond the scope of this paper.

Models

To predict whether a day is high or low volatility, we will use logistic regression, principal component analysis combined with logistic regression, Random Forest, and Support Vector Machines with a linear kernel. Wherever possible, time series cross validation will be used to ensure the data is appropriately prepared for testing.

Preprocessing

We optimize our models based on the Kappa statistic. While it is important to note that Kappa can encounter issues with skewed data, we find it a more reliable metric for optimization compared to simply using a threshold of 0.5 for classification (Jeni et al., 2013).

Below is the fucntion for calculating the Cohen's Kappa:

$$\kappa = \frac{p_o - p_e}{1 - p_e}$$

$p_o = Accuracy$, $p_e = Expected\ agreement\ by\ chance$

```r
### Kappa Optimization --------
calculate_best_threshold <- function(probs, true_labels) {
  calculate_kappa <- function(predictions, true_labels) {
    confusion_matrix <- table(predictions, true_labels)
    observed_accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)
    expected_accuracy <- sum(rowSums(confusion_matrix) * colSums(confusion_matrix)) /
      sum(confusion_matrix)^2
    kappa <- (observed_accuracy - expected_accuracy) / (1 - expected_accuracy)
    return(kappa)
  }

  thresholds <- seq(0, 1, by = 0.01)
  best_threshold <- 0
  best_kappa <- -Inf

  for (threshold in thresholds) {
    predictions <- ifelse(probs > threshold, 1, 0)
    kappa <- calculate_kappa(predictions, true_labels)
```

```
    if (kappa > best_kappa) {
      best_kappa <- kappa
      best_threshold <- threshold
    }
  }

  cat("Best Threshold:", best_threshold, "\n")
  return(best_threshold)
}
```

Logistic regression

Logistic regression is a commonly used and straightforward approach to applying machine learning. By encoding the response variable as either 0 or 1, logistic regression can be employed to classify whether a day is likely to experience high volatility. The predicted value Yˆ represents a probability, where outcomes above 0.5 are typically classified as 1 (high volatility) and those below as 0 (low volatility). This simple and interpretable model serves as a baseline against which more advanced machine learning models can be compared in terms of predictive performance (James et al., 2023, pp. 133-135).

$$p(X) = \frac{e^{\beta_0+\beta_1 X}}{1 + e^{\beta_0+\beta_1 X}}$$

$p(X)$ is the predicted probaility of a high volatility day

X is the explanatory variable

β_0 is the intercept term.

β_1 is the coefficient for the variable X

```
### Logistic regression --------

logic_model <- train(
  high_vol ~ d_price + d_rolling_30_sd +
    d_wind + d_temp + month + war + d_storage + weekend,
  data = train,
  method = "glm",
  family = binomial
)
```

In our model, we use a limited set of predictors to reduce the likelihood of multicollinearity. Among these variables, month and weekend are included to account for seasonality by capturing patterns across different time periods.

```
summary(logic_model)
```

```
##
## Call:
## NULL
##
## Coefficients:
##                   Estimate Std. Error z value Pr(>|z|)
## (Intercept)      1.314e-01  9.688e-01   0.136 0.892103
```

```
## d_price          6.371e-03  2.516e-03   2.532 0.011328 *
## d_rolling_30_sd  2.203e-02  6.831e-03   3.226 0.001256 **
## d_wind          -1.011e-01  5.386e-02  -1.877 0.060521 .
## d_temp          -2.587e-02  3.075e-02  -0.841 0.400341
## month2          -6.517e-01  5.004e-01  -1.302 0.192807
## month3          -2.391e+00  6.638e-01  -3.602 0.000316 ***
## month4          -2.523e+00  7.750e-01  -3.256 0.001129 **
## month5          -1.036e+00  7.388e-01  -1.402 0.160782
## month6          -1.028e+00  6.775e-01  -1.518 0.129022
## month7          -1.353e+00  6.952e-01  -1.947 0.051542 .
## month8          -8.041e-01  7.219e-01  -1.114 0.265319
## month9          -3.985e-01  6.598e-01  -0.604 0.545879
## month10          5.856e-01  5.414e-01   1.082 0.279368
## month11          4.261e-01  5.213e-01   0.817 0.413661
## month12          6.132e-01  4.670e-01   1.313 0.189149
## warYes           1.887e+00  3.685e-01   5.119 3.06e-07 ***
## d_storage       -1.430e-07  6.886e-08  -2.076 0.037873 *
## weekend1        -2.426e-01  2.078e-01  -1.168 0.242978
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1329.76  on 1028  degrees of freedom
## Residual deviance:  758.45  on 1010  degrees of freedom
## AIC: 796.45
##
## Number of Fisher Scoring iterations: 6
```

The regression from the training data reveals that the variables `d_price`, `d_rolling_30_sd`, `war` and `d_storage` and some `months` are statically significant.

```
true_labels <- as.numeric(as.character(test$high_vol))

probs <- predict(logic_model, test, type = "prob")

best_threshold <- calculate_best_threshold(probs[, 2], true_labels)
```

```
## Best Threshold: 0.55
```

```
test$high_vol <- factor(test$high_vol, levels = c(0, 1))
logic_curve <- roc(test$high_vol, probs[, 2])
print(auc(logic_curve))
```

```
## Area under the curve: 0.7051
```

The AUC value is 0.7, which is clearly better than chance.

```
predictions <- ifelse(probs[, 2] > best_threshold, 1, 0)
predictions <- as.factor(predictions)
logic_cm <- confusionMatrix(predictions, test$high_vol, positive = "1")
print(logic_cm)
```

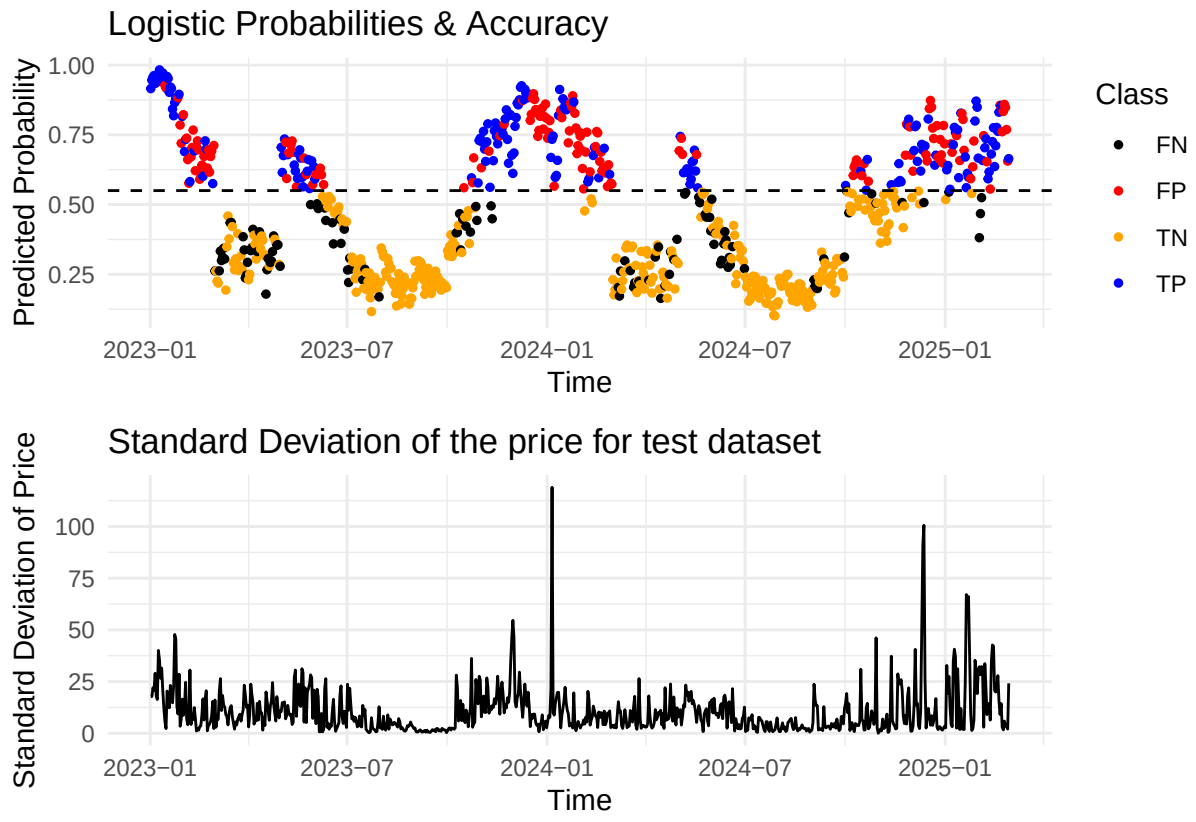
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 342 111
##          1 154 183
##
##                Accuracy : 0.6646
##                  95% CI : (0.6304, 0.6974)
##     No Information Rate : 0.6278
##     P-Value [Acc > NIR] : 0.017456
##
##                   Kappa : 0.3029
##
##  Mcnemar's Test P-Value : 0.009879
##
##             Sensitivity : 0.6224
##             Specificity : 0.6895
##          Pos Pred Value : 0.5430
##          Neg Pred Value : 0.7550
##              Prevalence : 0.3722
##          Detection Rate : 0.2316
##    Detection Prevalence : 0.4266
##       Balanced Accuracy : 0.6560
##
##        'Positive' Class : 1
##
```

The confusion matrix show us that the model has a accuracy of 66%. The TPR and TNR is 62% and 69% for the optimized threshold based on Kappa.

```
results <- test %>%
  mutate(
    logic_prediction = predictions,
    logic_prob = probs[, 2],
    high_vol = as.numeric(as.character(high_vol)),
    logic_correct = case_when(
      high_vol == 1 & logic_prediction == 1 ~ "TP",
      high_vol == 1 & logic_prediction == 0 ~ "FN",
      high_vol == 0 & logic_prediction == 0 ~ "TN",
      high_vol == 0 & logic_prediction == 1 ~ "FP"
    )
  )
```

```
p1 <- ggplot(results, aes(x = time)) +
  geom_point(aes(y = logic_prob, color = logic_correct), size = 1) +
  scale_color_manual(
    values = c("TP" = "blue", "TN" = "orange", "FP" = "red", "FN" = "black"),
    name = "Class"
  ) +
  geom_hline(yintercept = best_threshold, linetype = "dashed", color = "black") +
  labs(
    title = "Logistic Probabilities & Accuracy",
    x = "Time", y = "Predicted Probability"
```

```
  ) +
  theme_minimal()

p2 <- ggplot(day_data %>% filter(year >= 2023), aes(x = time, y = sd_price)) +
  geom_line() +
  labs(
    title = "Standard Deviation of the price for test dataset",
    x = "Time",
    y = "Standard Deviation of Price"
  ) +
  theme_minimal()

combined_plot <- p1 / p2

combined_plot
```



The figure "Logistic Probabilities & Accuracy" shows the predicted probabilities over time, with coloring based on the classification it got. FN: False Negative, FP: False positive, TN: True negative and TP: True positive. The figure below shows the actual standard deviation over time.

The plot above shows that the predicted probabilities change noticeably with the seasons. This is expected, as seasonal variations influence both electricity demand and supply

```
ggplot(results, aes(x = d_price, y = logic_prob, color = factor(high_vol))) +
  geom_point(alpha = 0.8) +
  labs(
```

```
    y = "Predicted Probailities",
    x = "Price",
    color = "High Vol"
  ) +
  facet_wrap(~season) +
  theme_minimal()
```

The plot above shows the correlation between the electricity price and the predicted probability and is split into four different figures for each season. There is clearly a seasonal aspect of this and this plot is shown as a contrast to similar plot later in the paper.

PCA and Logistic regression

To support further use of logistic regression, we apply Principal Component Analysis (PCA) as an additional method for addressing multicollinearity. PCA introduces transformed features into the dataset by creating new variables that capture the underlying variation in the original data. These new variables aim to provide the linear model with a set of uncorrelated components that still explain most of the variance in the data. This addresses the issue of multicollinearity, where independent variables are highly correlated, which violates a key assumption in linear regression.

From a theoretical standpoint, PCA can improve predictive performance in linear regression models, especially when many of the original variables are naturally correlated (James et al., 2023, pp. 497-498). This is the case in our dataset, where summary statistic variables from price variable naturally correltate.

Before implementing PCA, we scale the data to prevent the analysis from being biased toward variables with extreme ranges.

```r
### PCA --------

non_numeric_columns <- c("weekend", "time", "high_vol", "month", "war", "season")

scaled_train <- as.data.frame(scale(train[, !(names(train) %in% non_numeric_columns)]))

scaled_test <- as.data.frame(scale(test[, !(names(test) %in% non_numeric_columns)]))

pca_result_train <- prcomp(scaled_train, center = FALSE, scale. = FALSE)

summary(pca_result_train)
```

```
## Importance of components:
##                           PC1     PC2     PC3     PC4     PC5     PC6     PC7
## Standard deviation     4.3210 1.58399 1.31803 1.26236 1.08692 0.98369 0.84553
## Proportion of Variance 0.6224 0.08363 0.05791 0.05312 0.03938 0.03226 0.02383
## Cumulative Proportion  0.6224 0.70601 0.76392 0.81704 0.85642 0.88867 0.91251
##                           PC8     PC9    PC10    PC11    PC12    PC13    PC14
## Standard deviation     0.71135 0.64746 0.57889 0.49935 0.48650 0.44342 0.37014
## Proportion of Variance 0.01687 0.01397 0.01117 0.00831 0.00789 0.00655 0.00457
## Cumulative Proportion  0.92937 0.94335 0.95452 0.96283 0.97072 0.97727 0.98184
##                           PC15    PC16    PC17    PC18    PC19    PC20    PC21
## Standard deviation     0.33342 0.31527 0.27333 0.23553 0.22825 0.21253 0.17646
## Proportion of Variance 0.00371 0.00331 0.00249 0.00185 0.00174 0.00151 0.00104
## Cumulative Proportion  0.98554 0.98886 0.99135 0.99320 0.99493 0.99644 0.99748
##                           PC22    PC23    PC24    PC25    PC26    PC27    PC28
## Standard deviation     0.15640 0.13753 0.11456 0.08574 0.07539 0.04884 0.04460
## Proportion of Variance 0.00082 0.00063 0.00044 0.00025 0.00019 0.00008 0.00007
## Cumulative Proportion  0.99829 0.99892 0.99936 0.99961 0.99980 0.99987 0.99994
##                           PC29     PC30
## Standard deviation     0.04194 0.003789
## Proportion of Variance 0.00006 0.000000
## Cumulative Proportion  1.00000 1.000000
```

Based in the summary of the PCA results we decide that 7 principal components is enough since it explains cumulative 90 % of the variance. `PC1` explains the absolute majority of the variance of of the explanatory variables. The next component only explains 8% of the variance where as the `PC1` explains 62%.

```r
rotation <- as.data.frame(pca_result_train$rotation)
rotation <- rotation %>%
  arrange(desc(PC1))

rotation[1]
```

```
##                                PC1
## d_ma_30                0.222870017
## d_rolling_30_median    0.220541911
## d_rolling_30_sd        0.218824112
## lag7_d_ma_30           0.217820839
## lag7_d_ma_7            0.217544943
## d_rolling_30_range     0.214876109
## d_rolling_7_median     0.212946788
```

```
## lag7_d_price          0.212922293
## d_ma_7                0.212786562
## lag7_d_rolling_30_sd  0.210799584
## d_rolling_7_range     0.205372082
## d_rolling_7_sd        0.204299029
## d_rolling_30_iqr      0.202730878
## d_price               0.202449033
## d_median_price        0.201952167
## lag7_d_rolling_7_sd   0.196265994
## lag30_d_ma_7          0.193037262
## lag30_d_price         0.192972510
## d_rolling_7_iqr       0.191676027
## lag30_d_ma_30         0.185024019
## d_range_price         0.179350919
## d_sd_price            0.178103589
## d_iqr_price           0.169732686
## lag30_d_roling_30_sd  0.159500381
## lag30_d_roling_7_sd   0.151066512
## d_storage            -0.002569467
## d_sd_wind            -0.007935797
## d_sd_temp            -0.011096753
## d_wind               -0.011689385
## d_temp               -0.019917640
```

Above this, you can see what `PC1` contains in decreasing order. It is evident that `PC1` primarily represents price information. This indicates that this principal component is a blend of all the summary statistics related to price, and we expect it will be significant for both training and prediction.

```
# PCA train dataset
pca_train <- pca_result_train$x[, 1:7]

factor_train <- train[, non_numeric_columns]

pca_train <- cbind(pca_train, factor_train) %>%
  as_tibble()

# PCA test dataset

# Transforming test data with the PCA from the train data
pca_result_test <- predict(pca_result_train, newdata = scaled_test)

pca_test <- pca_result_test[, 1:7]

factor_test <- test[, non_numeric_columns]

pca_test <- cbind(pca_test, factor_test) %>%
  as_tibble()
```

We perform logistic regression using the transformed test data.

```
### Logistic regression with PCA --------
pca_model <- train(high_vol ~ . - time - season,
  data = pca_train,
```

```
  method = "glm",
  family = binomial
)
summary(pca_model)
```

```
##
## Call:
## NULL
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.450681   0.392934  -1.147 0.251397
## PC1          0.409274   0.061480   6.657 2.79e-11 ***
## PC2         -0.118813   0.089905  -1.322 0.186324
## PC3          0.055473   0.120911   0.459 0.646382
## PC4          0.084615   0.122332   0.692 0.489134
## PC5         -0.172869   0.194238  -0.890 0.373473
## PC6          0.047763   0.110755   0.431 0.666289
## PC7          0.033824   0.128944   0.262 0.793079
## weekend1    -0.283746   0.208330  -1.362 0.173197
## month2      -0.652443   0.476982  -1.368 0.171357
## month3      -1.620751   0.593132  -2.733 0.006285 **
## month4      -1.784517   0.679524  -2.626 0.008636 **
## month5      -0.326248   0.593972  -0.549 0.582823
## month6      -0.793039   0.588478  -1.348 0.177784
## month7      -1.585716   0.684922  -2.315 0.020603 *
## month8      -1.034928   0.733282  -1.411 0.158137
## month9      -0.723712   0.697432  -1.038 0.299418
## month10      0.002898   0.556378   0.005 0.995844
## month11     -0.162765   0.510179  -0.319 0.749700
## month12      0.306896   0.477758   0.642 0.520634
## warYes       1.301294   0.378616   3.437 0.000588 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1329.76  on 1028  degrees of freedom
## Residual deviance:  751.07  on 1008  degrees of freedom
## AIC: 793.07
##
## Number of Fisher Scoring iterations: 6
```

We observe that PC1, war and some months are statistically significant. This corresponds with the result from the train data for the first logistic regression. As previously noted, PC1 primarily captures summary statistic from the price variable.

```
probs <- predict(pca_model, pca_test, type = "prob")

pca_curve <- roc(test$high_vol, probs[, 2])
print(auc(pca_curve))
```

```
## Area under the curve: 0.7203
```

```r
best_threshold <- calculate_best_threshold(probs[, 2], true_labels)
```

```
## Best Threshold: 0.52
```

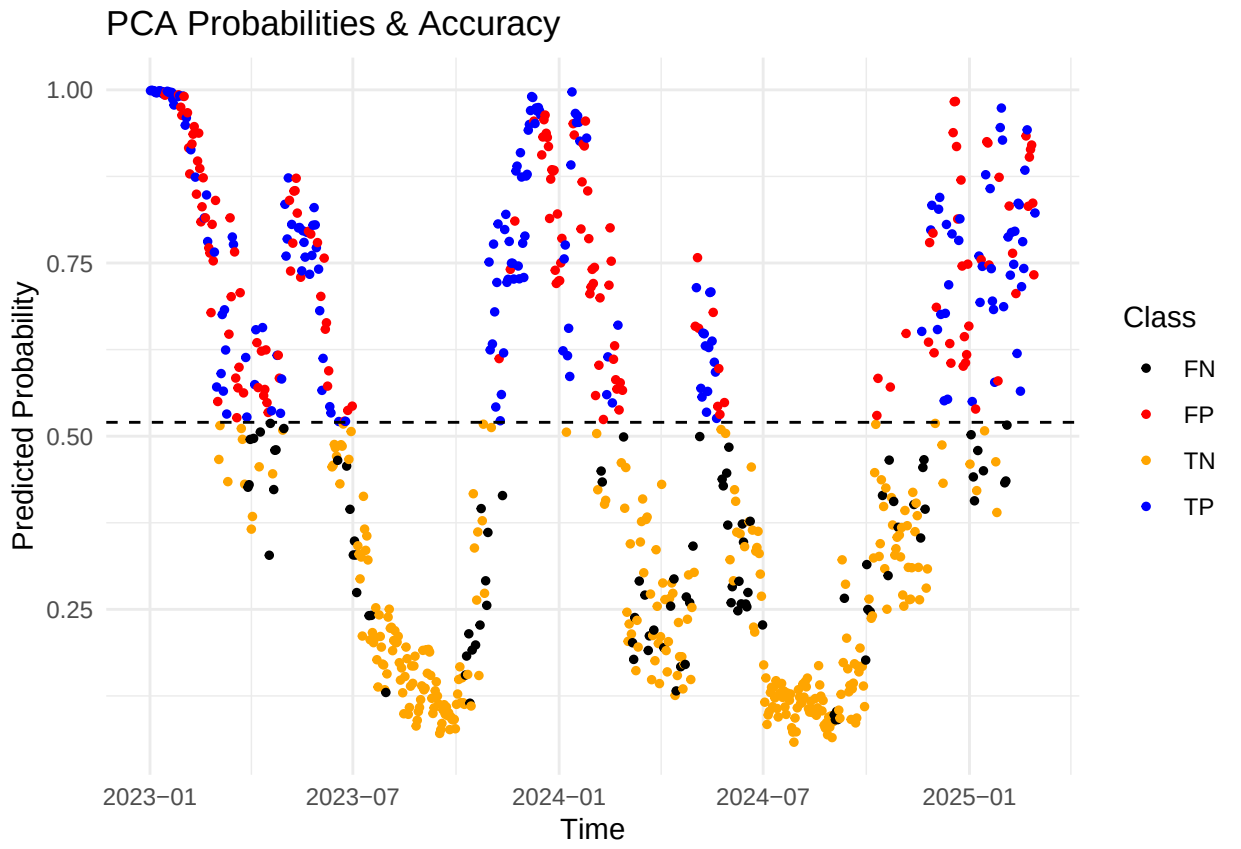
```r
predictions <- ifelse(probs[, 2] > best_threshold, 1, 0)
predictions <- as.factor(predictions)
pca_cm <- confusionMatrix(predictions, test$high_vol, positive = "1")
print(pca_cm)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 329  98
##          1 167 196
##
##                Accuracy : 0.6646
##                  95% CI : (0.6304, 0.6974)
##     No Information Rate : 0.6278
##     P-Value [Acc > NIR] : 0.01746
##
##                   Kappa : 0.3149
##
##  Mcnemar's Test P-Value : 2.951e-05
##
##             Sensitivity : 0.6667
##             Specificity : 0.6633
##          Pos Pred Value : 0.5399
##          Neg Pred Value : 0.7705
##              Prevalence : 0.3722
##          Detection Rate : 0.2481
##    Detection Prevalence : 0.4595
##       Balanced Accuracy : 0.6650
##
##        'Positive' Class : 1
##
```

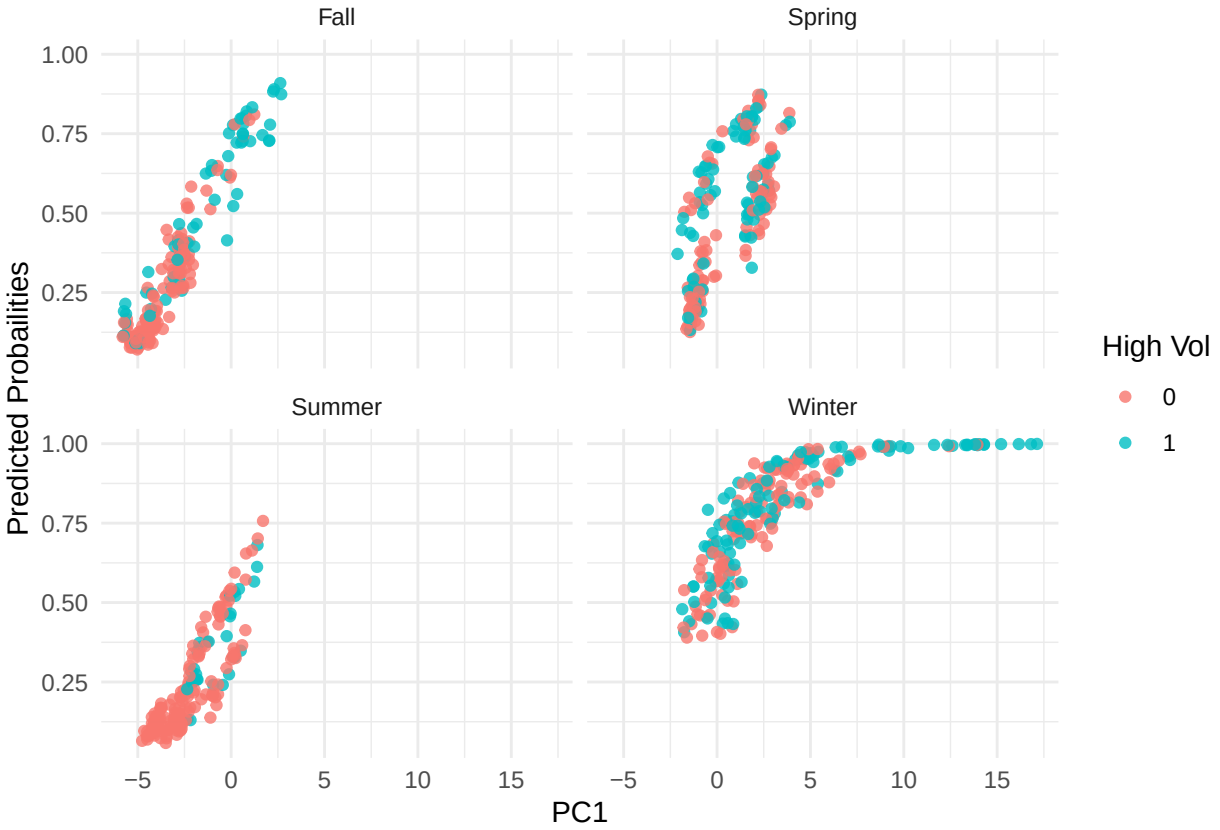
The TPR and TNR is 66% and 66% for the optimized threshold based on Kappa. The accuracy is at 66%

```r
results <- results %>%
  mutate(
    pca_prediction = predictions,
    pca_prob = probs[, 2],
    pca_correct = case_when(
      high_vol == 1 & pca_prediction == 1 ~ "TP",
      high_vol == 1 & pca_prediction == 0 ~ "FN",
      high_vol == 0 & pca_prediction == 0 ~ "TN",
      high_vol == 0 & pca_prediction == 1 ~ "FP"
    )
  )

ggplot(results, aes(x = time)) +
```

```
  geom_point(aes(y = pca_prob, color = pca_correct), size = 1) +
  scale_color_manual(
    values = c("TP" = "blue", "TN" = "orange", "FP" = "red", "FN" = "black"),
    name = "Class"
  ) +
  geom_hline(yintercept = best_threshold, linetype = "dashed", color = "black") +
  labs(
    title = "PCA Probabilities & Accuracy",
    x = "Time", y = "Predicted Probability"
  ) +
  theme_minimal()
```




```
results$PC1 <- pca_test$PC1


ggplot(results, aes(x = PC1, y = pca_prob, color = factor(high_vol))) +
  geom_point(alpha = 0.8) +
  labs(
    y = "Predicted Probabilities",
    x = "PC1",
    color = "High Vol"
  ) +
  facet_wrap(~season) +
  theme_minimal()
```



The figure above shows the correlation between the "PC1" and the predicted probability. They are split into four figures, one for each season, and the color shows us if the the given day had high volatility or not. This plot is shown to contrast with the earlier plot for the price.

We see that PC1 is more important to the prediction than price, and it has more "s" curve as expected from a logistic regression.

```
### Time Series CV ----------

folds <- list()

total_obs <- nrow(train)


fold_size <- ceiling(total_obs / 10)
counter <- 1

for (i in 5:9) {
  test_start_index <- (i * fold_size)
  test_end_index <- min((i + 1) * fold_size, total_obs)


  train_indices <- 1:(test_start_index - 1)
  test_indices <- test_start_index:test_end_index

  folds[[counter]] <- list(train = train_indices, test = test_indices)
```

```
  counter <- counter + 1
}

levels(train$high_vol) <- make.names(levels(train$high_vol))

cv_control <- trainControl(
  method = "cv",
  index = lapply(folds, function(x) x$train),
  indexOut = lapply(folds, function(x) x$test),
  classProbs = TRUE,
  summaryFunction = twoClassSummary
)
```

The time series cross validation is structured so that the first fold uses the initial 50% of the data for training and validates on the subsequent 10%. Each following fold expands the training set by 10% of the original data, while validation always performed on the following 10%, continuing this pattern until the end of the data set.

Random Forest

For further analysis, we apply Random Forest to capture high variance using a more advanced algorithm compared to logistic regression. Random Forest offers a flexible yet powerful classification method. It works by constructing an ensemble of decision trees, where the final prediction is determined by the majority vote across the trees. This ensemble approach allows the model to capture both linear and non linear relationships in the data, making it more robust than individual decision trees.

In addition to strong predictive capabilities, Random Forest models also provide insights into variable importance. This feature indicates which predictors contribute most to the model's decisions, offering valuable interpretability (James et al., 2023, pp. 343-345).

```
### Random Forest ----------

mtry_range <- seq(2, ncol(train) - 1, by = 1)

tune_grid <- expand.grid(mtry = mtry_range)

rf_model_tuned <- train(high_vol ~ . - time - season,
  data = train,
  method = "rf",
  trControl = cv_control,
  tuneGrid = tune_grid,
  ntree = 100,
  metric = "ROC"
)

print(rf_model_tuned$results)
```

```
##    mtry       ROC      Sens      Spec     ROCSD    SensSD    SpecSD
## 1     2 0.5579061 0.5102908 0.5333656 0.1835971 0.3201253 0.4130820
## 2     3 0.5760394 0.4739492 0.6259492 0.1806871 0.3362496 0.3917526
## 3     4 0.5610222 0.4197227 0.5741281 0.1607163 0.3844158 0.4552482
## 4     5 0.5323058 0.4691596 0.4934681 0.1849494 0.3830042 0.3795653
```

```
## 5     6 0.5642291 0.4751343 0.5960465 0.1901106 0.3265205 0.4022709
## 6     7 0.5508733 0.4925285 0.5827250 0.1431286 0.3083809 0.4084025
## 7     8 0.5627568 0.5778506 0.4808448 0.1594446 0.3098106 0.4119743
## 8     9 0.5357541 0.4593032 0.5786807 0.1291617 0.4361199 0.4347383
## 9    10 0.5519523 0.5462937 0.5476061 0.1374858 0.3324766 0.3781537
## 10   11 0.5311865 0.4418599 0.6242957 0.1717816 0.3507905 0.3642081
## 11   12 0.5591241 0.3449046 0.6439637 0.1970725 0.3739749 0.3664392
## 12   13 0.5480349 0.5169844 0.5762317 0.1690517 0.3333407 0.3716272
## 13   14 0.5368909 0.4054914 0.5502194 0.1684716 0.4108461 0.3579615
## 14   15 0.5363819 0.4792590 0.4861639 0.1748025 0.3385177 0.3490635
## 15   16 0.5454936 0.5397546 0.5606129 0.1719791 0.3764573 0.3898121
## 16   17 0.5524132 0.5596221 0.5691593 0.1882218 0.3765127 0.3541479
## 17   18 0.5019271 0.4494919 0.4723246 0.1834588 0.3942096 0.3840582
## 18   19 0.5234608 0.4255145 0.5506292 0.1442915 0.4007758 0.3826720
## 19   20 0.5287491 0.5185965 0.5156509 0.1976991 0.3663131 0.3234055
## 20   21 0.5045522 0.5466039 0.4091519 0.1652783 0.3161983 0.3122024
## 21   22 0.4951228 0.4414904 0.5182596 0.1592351 0.3723157 0.3792203
## 22   23 0.5339134 0.6078707 0.4761515 0.2079946 0.3208942 0.3865451
## 23   24 0.5422116 0.4620513 0.5642762 0.1806239 0.3389585 0.3472486
## 24   25 0.5104042 0.3873372 0.5857622 0.1698797 0.3446395 0.3540224
## 25   26 0.5099859 0.3828617 0.5665856 0.1646146 0.3636854 0.3557449
## 26   27 0.5285817 0.4384069 0.5228014 0.1402997 0.3651825 0.3682939
## 27   28 0.5066286 0.4590806 0.4863897 0.1765095 0.3126718 0.3260527
## 28   29 0.5326749 0.6073106 0.5272638 0.1935085 0.3766574 0.3262988
## 29   30 0.5389106 0.4308555 0.4941279 0.1679185 0.3930983 0.4088810
## 30   31 0.4872039 0.3468994 0.5634014 0.1493537 0.3909714 0.3464507
## 31   32 0.4991032 0.4163636 0.5061084 0.1030146 0.4247702 0.3818460
## 32   33 0.5206870 0.4670908 0.5537375 0.1639281 0.3360366 0.3536629
## 33   34 0.4816588 0.3240423 0.5609392 0.1651944 0.3605238 0.3142890
## 34   35 0.5035366 0.5070501 0.4285821 0.1460368 0.3732221 0.3202970
```

```r
optimal_mtry <- rf_model_tuned$bestTune %>%
  pull()

print(optimal_mtry)
```

```
## [1] 3
```

The ROC (AUC value) for the training set is quite low, falling below 60%. However, there is a considerable standard deviation in this measure. The optimal mtry is 3, meaning that three variables are selected for each split. The mtry was chosen with the use of time series cross validation.

```r
# Train the final Random Forest model with optimal mtry
rf_final_model <- randomForest(high_vol ~ . - time - season,
  data = train, ntree = 100,
  mtry = optimal_mtry, importance = TRUE
)

importance_values <- importance(rf_final_model) %>%
  as.data.frame() %>%
  rownames_to_column(var = "variable") %>%
  arrange(MeanDecreaseGini)
```

```r
importance_values %>%
  select(variable, MeanDecreaseGini)
```

```
##                variable MeanDecreaseGini
## 1               weekend         1.576988
## 2             d_storage         7.171474
## 3             d_sd_temp         7.374691
## 4             d_sd_wind         7.904932
## 5                d_wind         8.205165
## 6   lag30_d_roling_7_sd         9.098401
## 7                d_temp         9.514497
## 8           d_iqr_price         9.840490
## 9               d_price        10.415858
## 10   d_rolling_30_range        10.643283
## 11   d_rolling_7_median        10.919772
## 12        lag30_d_price        11.386219
## 13  lag7_d_rolling_7_sd        12.103390
## 14                month        12.197888
## 15 lag30_d_roling_30_sd        12.437285
## 16                  war        13.339874
## 17         lag7_d_price        13.429145
## 18    d_rolling_7_range        13.731132
## 19               d_ma_7        14.045635
## 20 lag7_d_rolling_30_sd        14.080153
## 21          lag7_d_ma_7        14.148412
## 22  d_rolling_30_median        14.819204
## 23           d_sd_price        15.234951
## 24         lag30_d_ma_7        16.103179
## 25       d_median_price        17.885064
## 26        d_range_price        18.901439
## 27              d_ma_30        19.029949
## 28       d_rolling_7_sd        19.737964
## 29      d_rolling_30_sd        21.363437
## 30     d_rolling_30_iqr        21.889550
## 31        lag30_d_ma_30        25.266032
## 32         lag7_d_ma_30        26.136164
## 33      d_rolling_7_iqr        26.311065
```

The figure above shows the importance of each variable based on the Gini index. This model faces significant challenges in accurately identifying which variables are truly the most important. This limitation arises from the presence of multiple summary statistic variables derived from the price variable, and these are correlated. Such correlations make it difficult for the Random Forest model to effectively distinguish between these variables, leading to potential inaccuracies in assessing their individual importance.

```r
probs <- predict(rf_final_model, newdata = test, type = "prob")

rf_curve <- roc(test$high_vol, probs[, 2])
print(auc(rf_curve))
```

```
## Area under the curve: 0.6909
```

```r
best_threshold <- calculate_best_threshold(probs[, 2], true_labels)
```

```
## Best Threshold: 0.31
```

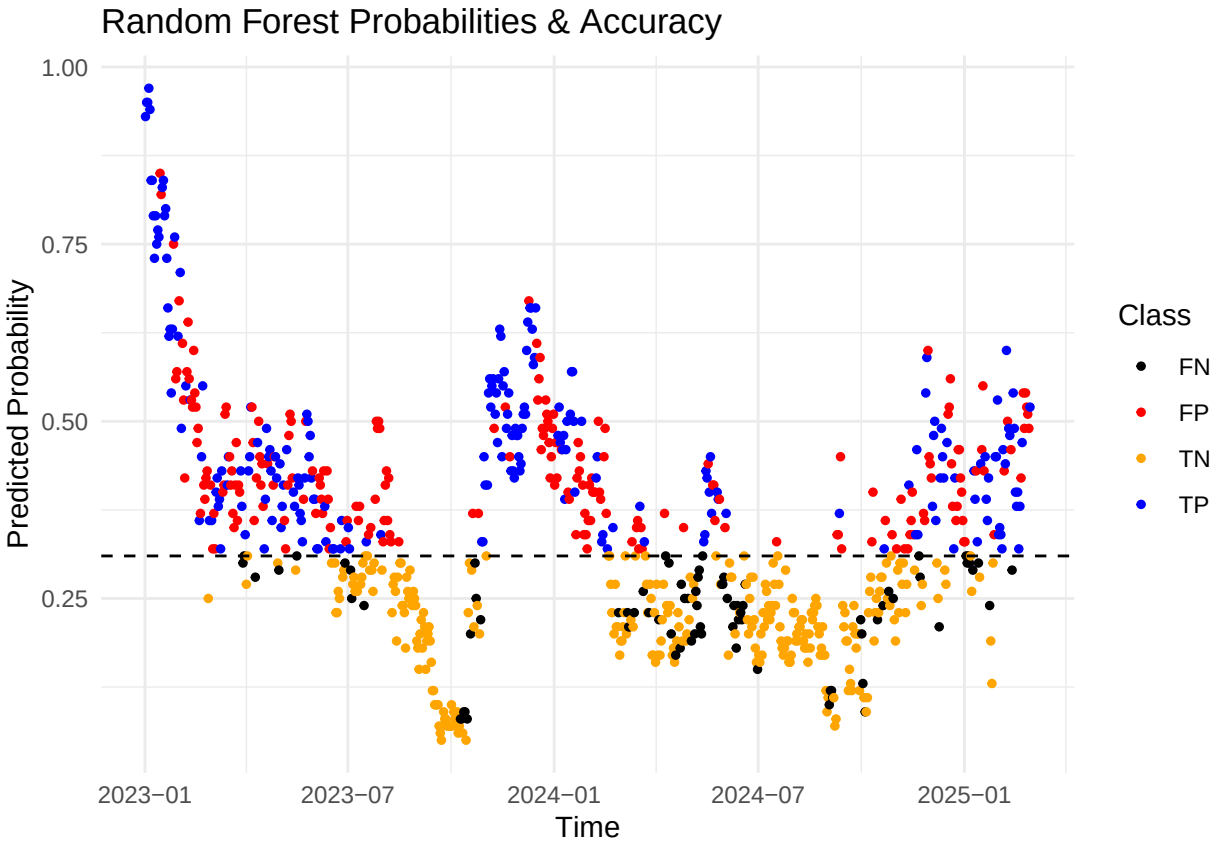
```r
predictions <- ifelse(probs[, 2] > best_threshold, 1, 0)
predictions <- as.factor(predictions)
rf_cm <- confusionMatrix(predictions, test$high_vol, positive = "1")
print(rf_cm)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 284  80
##          1 212 214
##
##                Accuracy : 0.6304
##                  95% CI : (0.5957, 0.6641)
##     No Information Rate : 0.6278
##     P-Value [Acc > NIR] : 0.4573
##
##                   Kappa : 0.2753
##
##  Mcnemar's Test P-Value : 1.772e-14
##
##             Sensitivity : 0.7279
##             Specificity : 0.5726
##          Pos Pred Value : 0.5023
##          Neg Pred Value : 0.7802
##              Prevalence : 0.3722
##          Detection Rate : 0.2709
##    Detection Prevalence : 0.5392
##       Balanced Accuracy : 0.6502
##
##        'Positive' Class : 1
##
```

The TPR and TNR is 73% and 57% for the optimized threshold based on Kappa. The accuracy is 63%

```r
results <- results %>%
  mutate(
    rf_prediction = predictions,
    rf_prob = probs[, 2],
    rf_correct = case_when(
      high_vol == 1 & rf_prediction == 1 ~ "TP",
      high_vol == 1 & rf_prediction == 0 ~ "FN",
      high_vol == 0 & rf_prediction == 0 ~ "TN",
      high_vol == 0 & rf_prediction == 1 ~ "FP"
    )
  )

ggplot(results, aes(x = time)) +
```

```
  geom_point(aes(y = rf_prob, color = rf_correct), size = 1) +
  scale_color_manual(
    values = c("TP" = "blue", "TN" = "orange", "FP" = "red", "FN" = "black"),
    name = "Class"
  ) +
  geom_hline(yintercept = best_threshold, linetype = "dashed", color = "black") +
  labs(
    title = "Random Forest Probabilities & Accuracy",
    x = "Time", y = "Predicted Probability"
  ) +
  theme_minimal()
```



As shown above the Random Forest model predict probabilities that are low and exhibit less seasonality compared to both the PCA and logistic regression models.

Support Vector Machine - with Linear Kernel

Support Vector Machines (SVM) aim to classify outcomes using a supervised learning algorithm. The core idea of SVM is to identify a hyperplane that maximizes the margin between different classes in the feature space. Maximizing the margin helps the model generalize better to new data.

SVM can be applied in both linear and non linear settings. Through the kernel functions, such as polynomial and radial basis function kernels, it is capable of capturing complex, non linear relationships. This makes SVM a powerful tool, particularly well suited for high dimensional datasets, due to its flexibility and ability to handle non linearly separable data (James et al., 2023, pp. 368-370, 380-384). In this paper, however, we

limit the analysis to the linear kernel, as the dataset is not high-dimensional and is therefore less likely to benefit from the added complexity of non-linear kernels.

```
### Support Vector Machine - with Linear Kernel ----------

scaled_train <- cbind(scaled_train, factor_train)
scaled_test <- cbind(scaled_test, factor_test)
```

The data used for training the SVM must be scaled because variables with higher values could otherwise disproportionately influence the model.

The variables for the support vector machine with linear kernel is chosen by the variables that was significant in the logistic regression.

```
levels(scaled_train$high_vol) <- make.names(levels(scaled_train$high_vol))

lsvm_model_cv <- train(
  high_vol ~ d_price + d_rolling_30_sd + month + war + d_wind + d_storage,
  data = scaled_train,
  method = "svmLinear",
  trControl = cv_control,
  metric = "ROC",
  tuneGrid = expand.grid(C = c(0.01, 0.1, 1, 10))
)
```

```
## maximum number of iterations reached 0.002263876 0.0021977
```

```
print(lsvm_model_cv$results)
```

```
##       C       ROC      Sens      Spec     ROCSD    SensSD    SpecSD
## 1  0.01 0.5911615 0.4526316 0.5494533 0.2091981 0.5110957 0.5108606
## 2  0.10 0.6134887 0.4842105 0.4949078 0.1959250 0.5012450 0.4974678
## 3  1.00 0.6144641 0.5719298 0.4577920 0.1943997 0.4337628 0.4845105
## 4 10.00 0.6148351 0.5000000 0.4700382 0.1939370 0.5000000 0.4909513
```

```
print(lsvm_model_cv$bestTune)
```

```
##    C
## 4 10
```

The optimal tuning parameter is a cost value of 10, selected through time series cross validation.

```
lsvm_probs <- predict(lsvm_model_cv, newdata = scaled_test, type = "prob")

lsvm_curve <- roc(test$high_vol, lsvm_probs[, 2])

print(auc(lsvm_curve))
```

```
## Area under the curve: 0.6966
```

```r
best_threshold <- calculate_best_threshold(lsvm_probs[, 2], true_labels)
```

```
## Best Threshold: 0.61
```

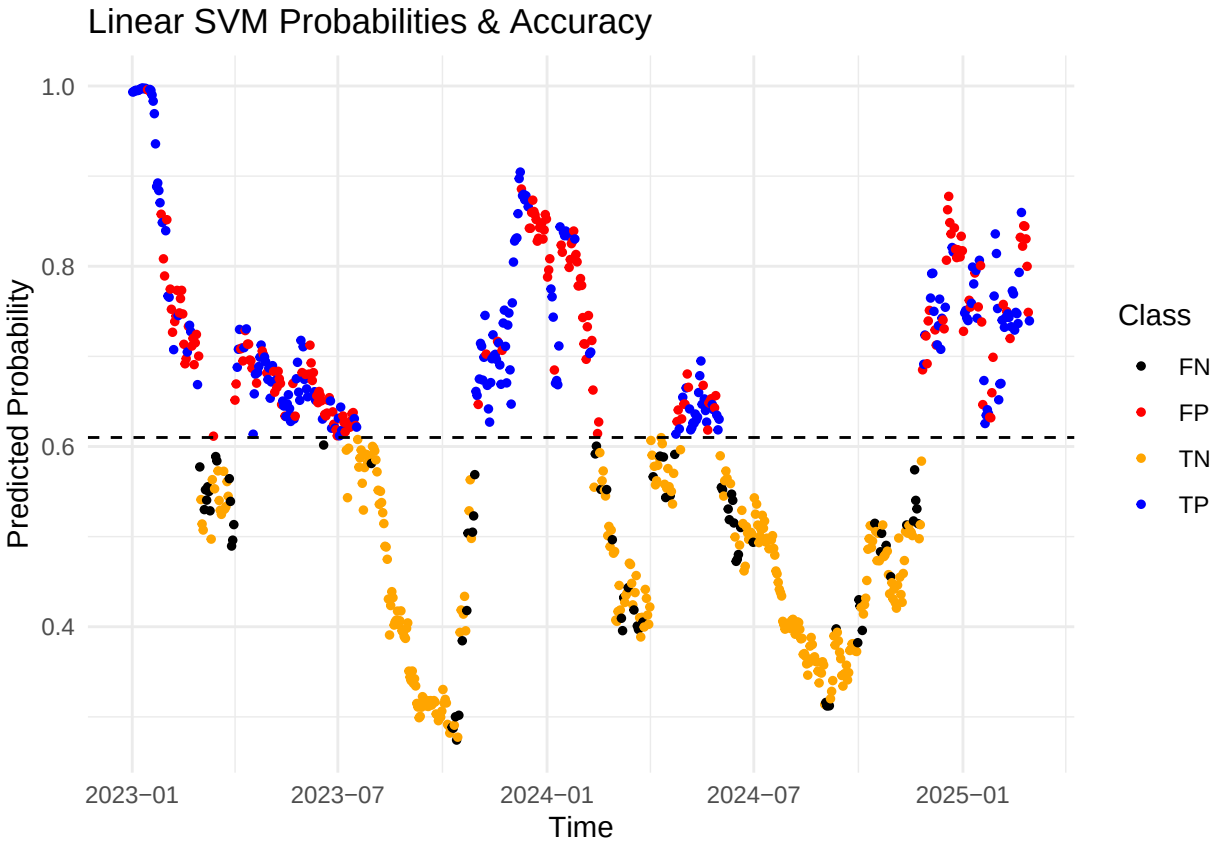
```r
lsvm_predictions <- ifelse(lsvm_probs[, 2] > best_threshold, 1, 0)
lsvm_predictions <- as.factor(lsvm_predictions)
lsvm_cm <- confusionMatrix(lsvm_predictions, test$high_vol, positive = "1")
print(lsvm_cm)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 309  78
##          1 187 216
##
##                Accuracy : 0.6646
##                  95% CI : (0.6304, 0.6974)
##     No Information Rate : 0.6278
##     P-Value [Acc > NIR] : 0.01746
##
##                   Kappa : 0.3326
##
##  Mcnemar's Test P-Value : 3.259e-11
##
##             Sensitivity : 0.7347
##             Specificity : 0.6230
##          Pos Pred Value : 0.5360
##          Neg Pred Value : 0.7984
##              Prevalence : 0.3722
##          Detection Rate : 0.2734
##    Detection Prevalence : 0.5101
##       Balanced Accuracy : 0.6788
##
##        'Positive' Class : 1
##
```

The TPR and TNR is 73% and 62% for the optimized threshold based on Kappa. The accuracy is 66%.

```r
results <- results %>%
  mutate(
    lsvm_prediction = lsvm_predictions,
    lsvm_prob = lsvm_probs[, 2],
    lsvm_correct = case_when(
      high_vol == 1 & lsvm_prediction == 1 ~ "TP",
      high_vol == 1 & lsvm_prediction == 0 ~ "FN",
      high_vol == 0 & lsvm_prediction == 0 ~ "TN",
      high_vol == 0 & lsvm_prediction == 1 ~ "FP"
    )
  )

ggplot(results, aes(x = time)) +
```

```
  geom_point(aes(y = lsvm_prob, color = lsvm_correct), size = 1) +
  scale_color_manual(
    values = c("TP" = "blue", "TN" = "orange", "FP" = "red", "FN" = "black"),
    name = "Class"
  ) +
  geom_hline(yintercept = best_threshold, linetype = "dashed", color = "black") +
  labs(
    title = "Linear SVM Probabilities & Accuracy",
    x = "Time", y = "Predicted Probability"
  ) +
  theme_minimal()
```

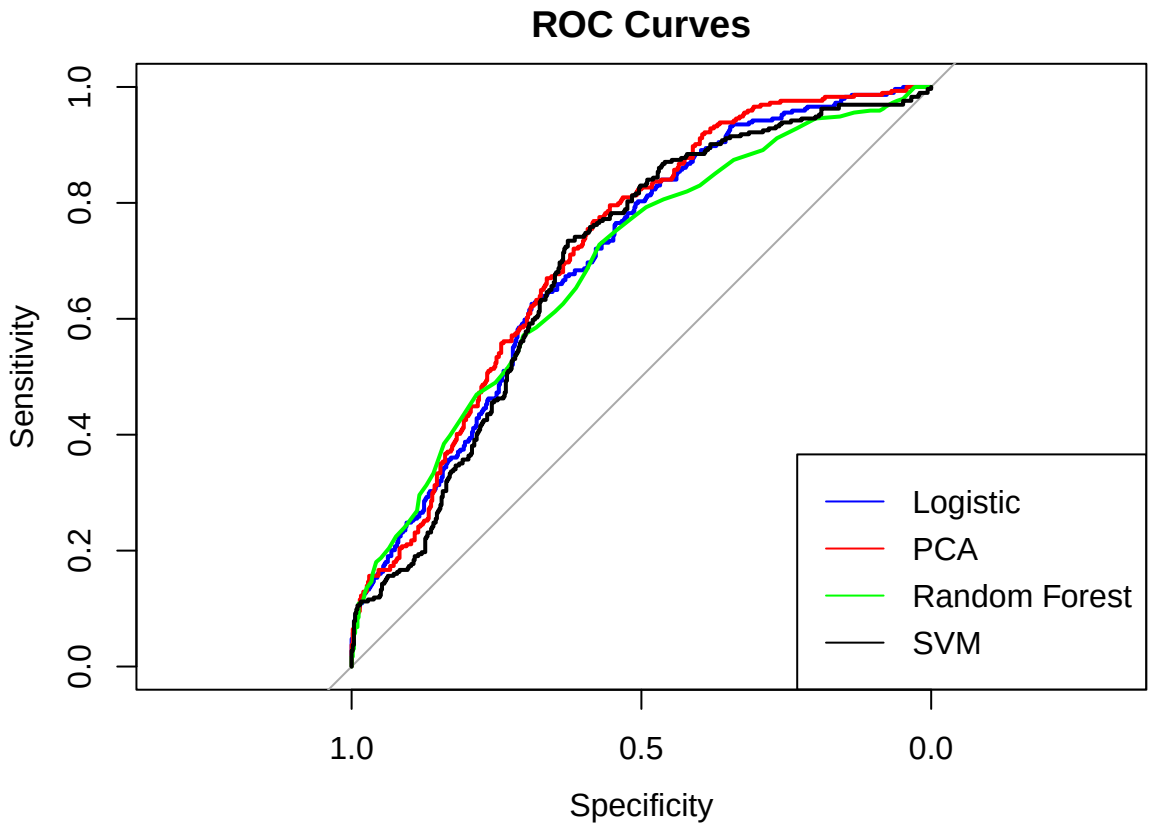


SVM has a greater range in its predictions and the seasonality is clearly shows.

Results

```
plot(logic_curve, col = "blue", main = "ROC Curves")

lines(pca_curve, col = "red")

lines(rf_curve, col = "green")

lines(lsvm_curve, col = "black")
```

```
legend("bottomright",
  legend = c("Logistic", "PCA", "Random Forest", "SVM"),
  col = c("blue", "red", "green", "black"), lty = 1
)
```




```
result_table %>% arrange(desc(AUC))
```

```
## # A tibble: 4 x 5
##   model          AUC Kappa   TPR   TNR
##   <chr>        <dbl> <dbl> <dbl> <dbl>
## 1 PCA_Logistic 0.720 0.315 0.667 0.663
## 2 Logistic     0.705 0.303 0.622 0.690
## 3 LSVM         0.697 0.333 0.735 0.623
## 4 RF           0.691 0.275 0.728 0.573
```

As seen in descending order of performance based on AUC, the models are ranked as follows – PCA, logistic regression, SVM and Random Forest. According to these results, PCA appears to be the most accurate model for predicting high volatility, as reflected in its ROC curve and the highest accumulated AUC value. However, the visual ROC curves for all models show substantial overlap, suggesting no clear difference between the models.

While PCA achieved the highest AUC score, the SVM model obtained the highest Cohen's Kappa value, 0.333 vs 0.315. Unlike AUC, which evaluates a model's ability to rank predictions regardless of classification threshold, Kappa reflects the agreement between predicted and actual classes at a specific threshold, adjusted

for chance. This suggests that, despite similar overall discriminative performance, the SVM may produce slightly more reliable class predictions under the chosen threshold. Random Forest, by contrast, achieved the lowest Kappa score of 0.275, indicating weaker agreement.As noted earlier in the paper, Kappa is more sensitive to class imbalance, and this should be considered when interpreting the relative performance of the models.

Overall, the models achieve Kappa scores between 0.275 and 0.333, indicating a modest but meaningful ability to predict high volatility days beyond random chance. Given the limited variables, primarily price metrics, weather data, and water storage levels this level of performance suggests that the models capture some useful signal for seven day ahead volatility classification.

Discussion

Although both Random Forest and Support Vector Machine are theoretically better suited to capture complex, non linear relationships, the dataset used in this study may lack the necessary feature complexity for these models to fully realize their potential. Most variables consist of summary statistics derived from the electricity price, supplemented by a limited selection of weather and hydro storage indicators. Since we only use a few different sources of data, the models have a limited foundation for detecting more complex patterns and relationships. As a result, the more advanced algorithms like Random Forest and Support Vector Machines may be underperforming in this setting.

By contrast, Principal Component Analysis enhances logistic regression not by increasing complexity but by addressing the problem of multicollinearity. Given the limited diversity in variables, it is consistent with theoretical expectations that PCA enhanced logistic regression performs better than the more complex models in this case. Its relative simplicity, along with its ability to combine highly correlated variables into a smaller set of independent components, makes it well suited for the structure of the current dataset.

To better support and evaluate models such as RF and SVM, it would be essential to broaden the dataset. Future research should aim to incorporate more diverse and relevant features. This could include data from all bidding zones in the Nord Pool electricity market, along with gas, oil, and CO2 prices, which are key inputs to electricity pricing. Moreover, the inclusion of historical weather forecasts and predicted hydro reservoir levels could further enrich the data foundation. Such additions would likely allow more complex models to better capture the underlying drivers of volatility and improve both predictive performance across model types.

Conclusion

This paper explores how machine learning can help predict price volatility in the NO5 bidding zone in Norway. Understanding future volatility is important for making good decisions in the power market. Instead of predicting exact prices, we use a classification method to find out if future price volatility will be high or low. This is more useful for real world decisions, where companies often act when volatility passes certain levels.

We compare different machine learning models. Although complex models like Random Forest and Support Vector Machines are often strong in theory, our results show that a simpler model, logistic regression with Principal Component Analysis, gives better results for our dataset. This is likely because the dataset has limited information, and simpler models help reduce problems like multicollinearity. In this case, reducing the number of features is more helpful than using more advanced models.

Our results show that machine learning can be a useful tool for predicting price volatility. If companies know when high volatility is likely, they can plan better, utilize resources more efficiently, and mitigate risk. This is important for producers, retailers, and companies that manage financial risk. Future research should look into improving the input data and testing new methods to make predictions even more accurate in a changing electricity market.

Sources

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2023). *An introduction to statistical learning with applications in R*(2nd ed., corrected reprint). Springer. https://www.statlearning.com/

Jeni, L. A., Cohn, J. F., & De La Torre, F. (2013). Facing imbalanced data: Recommendations for the use of performance metrics. *2013 Humaine Association Conference on Affective Computing and Intelligent Interaction*, 245–251. https://doi.org/10.1109/ACII.2013.47

Lijesen, M. G. (2007). The real-time price elasticity of electricity. *Energy Economics*, 29(2), 249–258. https://doi.org/10.1016/j.eneco.2006.08.008

Sheybanivaziri, S., Le Dréau, J., & Kazmi, H. (2024). *Forecasting price spikes in day-ahead electricity markets: Techniques, challenges, and the road ahead* (Discussion Paper FOR 1/2024). Norwegian School of Economics (NHH). https://openaccess.nhh.no/nhh-xmlui/bitstream/handle/11250/3112109/0124.pdf

Zareipour, H., Bhattacharya, K., & Canizares, C. A. (2011). Electricity market price volatility: The case of Ontario. *Energy Policy*, 39(3), 165–177. https://doi.org/10.1016/j.enpol.2010.10.006