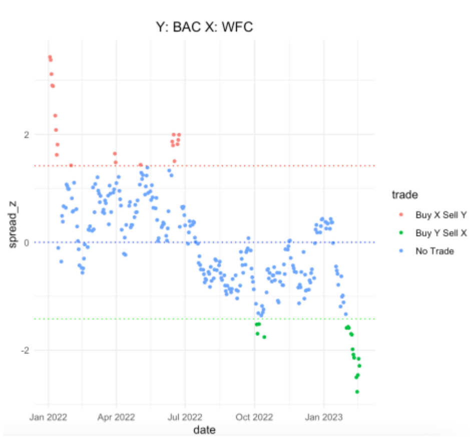
Pairs Trading + PROPHET W/ XGBOOST ERRORS

Step One: Select Spy 500 stocks from 01-01-2023 to Today
Step Two: Compute every possible 2 stock combo ex: Amazon and AMD
Step Three: Compute co integration log(y) ~ log(x) (Finding stock that move together)
Step Four: Find pairs that have p values <= .05 on their spread and correlation >= .95
Step Five: Use Prophet + XGboost for future predicts of each stock separately + injecting stochastic error of previous t to t - 14 timesteps mean(y - y_hat) of previous 14 days
Step Six: Use observed data + predict data with Kalman Filter to model the future states of the stock
Step Seven: Recompute spread: log(y, y+14) ~ log(x, x +14)
Step Eight: Standardize spread, spread – mean(spread)/sd(spread)

Decision: Spread > upper limit (1 standard deviation) **= Buy Stock X Sell Stock Y**
Decision: Spread < lower limit (-1 standard deviation) **= Buy Stock Y Sell Stock X**

Spread Vs Stock Actual Movement



Co-Integration- Code Snippets: Example

```r
vector <- list()
combs <- combn(left,2)
for(i in 1:117855){
  vector[[i]] <- combs[,i]
}

sample_list <- unique(lapply(vector , sort))

corData <- data.frame(matrix(ncol = 2, nrow = 0))
colnames(corData ) <- c('pair', "corr")

for(i in 1:length(sample_list)){
  one <- df %>%
    filter(Symbol %in% sample_list[[i]][1]) %>%
    pull(Price)
  two <- df %>%
    filter(Symbol %in% sample_list[[i]][2]) %>%
    pull(Price)

  if(length(one) !=  length(two)){
    next
  }
  model <- lm(log(one) ~ log(two) - 1)
  sprd<-residuals(model)
  corData[i, 'pair'] <- paste0(sample_list[[i]],collapse = "---")
  corData[i, 'corr'] <- cor(one, two)
  corData[i, 'beta'] <- as.numeric(coef(model)[1])
  corData[i, "pval"] <- adf.test(sprd, alternative="stationary", k=0)$p.value
}
```

```r
  filter(Symbol %in% pairs) %>%
  cbind(
    spread = one-beta*two
  ) %>%
  mutate(spread_z = (spread - mean(spread))/sd(spread),
         lower =  sd(spread)*-1,
         upper =   sd(spread),
         middle =  0,
         date = ymd(date),
         Price = as.numeric(Price),
         Sector = as.factor(Sector),
         x_y = case_when(
           pairs[1] == Symbol ~ "Y",
           TRUE ~ "X"
         )) %>%
  rowwise() %>%
  mutate(
    trade = case_when(
      spread_z  > upper ~ "Buy X Sell Y",
      spread_z  <  lower ~ "Buy Y Sell X",

      TRUE ~ "No Trade"

    )
```

Forecasting Data: Example

```r
model_spec_prophet_boost <- prophet_boost() %>%
  set_engine("prophet_xgboost", yearly.seasonality = TRUE)

workflow_fit_prophet_boost <- workflow() %>%
  add_model(model_spec_prophet_boost) %>%
  add_recipe(recipe_spec) %>%
  fit(training(splits))


model_table <- modeltime_table(
  workflow_fit_prophet_boost
)

calibration_table <- model_table %>%
  modeltime_calibrate(testing(splits))

pred_data <- calibration_table  %>%
  modeltime_refit(time) %>%
  modeltime_forecast(
    h = "2 months",
    actual_data = time
  ) %>%
  as.data.frame() %>%
  filter(.index > max(time$date)) %>%
  rename(date =  .index,
         Price = .value) %>%
  mutate(Symbol = pairs[1] ) %>%
  select(date,Price,Symbol)

pred_data
```

	date	Price	Symbol
1	2023-02-23	76.60487	AMD
2	2023-02-24	77.36558	AMD
3	2023-02-25	90.48047	AMD
4	2023-02-26	91.26076	AMD
5	2023-02-27	81.38562	AMD
6	2023-02-28	81.95862	AMD
7	2023-03-01	79.36508	AMD
8	2023-03-02	80.04180	AMD
9	2023-03-03	78.41384	AMD
10	2023-03-04	89.30924	AMD
11	2023-03-05	85.49922	AMD
12	2023-03-06	75.07702	AMD
13	2023-03-07	75.34995	AMD
14	2023-03-08	76.73025	AMD
15	2023-03-09	76.42442	AMD
16	2023-03-10	76.00981	AMD
17	2023-03-11	87.13087	AMD
18	2023-03-12	86.28829	AMD
19	2023-03-13	76.11769	AMD
20	2023-03-14	76.05553	AMD
21	2023-03-15	83.15805	AMD

We would inject error in these prices based on passed error of the stocks

Kalman Filter

```r
theta <- theta_var <- rep(NA, length(y) + 1)

# set our initial guess
theta[1] <- a0
theta_var[1] <- P0

w <- sample(seq(0.00, .4, length = 10000),6)
sigma_w <- sqrt(w[1])
sigma_v <- sqrt(w[2])
G_t <- Tt
F_t <- Zt

# iterate and make estimates
for (i in 1:length(y)) {

  # Equation 6.
  # use previous theta value for theta_hat and calculate e_t
  theta_hat <- theta[i]
  e_t <- y[i] - theta_hat * G_t * F_t

  # calculate R_t
  R_t <- G_t * theta_var[i] * G_t + sigma_w ^ 2

  # generate estimates from Equation 11
  theta[i + 1] <- G_t * theta_hat + R_t * F_t * (sigma_v ^ 2 + F_t ^ 2 * R_t) ^ (-1) * e_t
  theta_var[i + 1] <- R_t - R_t * F_t * (sigma_v ^ 2 + F_t ^ 2 * R_t) ^ (-1) * F_t * R_t
}

# adjust by one
theta <- theta[-1]
theta_var <- theta_var[-1]
```