Cryptocurrency Mean Reversion Trading Strategies

Thisana

2024-06-14

Introduction

Mean reversion, a cornerstone concept in finance, posits that an asset's price tends to revert towards its historical average over time (Wikipedia). This analysis focuses on exploring mean reversion dynamics using data from two prominent cryptocurrencies, Bitcoin (BTC) and Avalanche (AVAX). *Hourly price data spanning several years was sourced from Cryptoquotes API to facilitate the analysis.* Checked dataset thoroughly for accuracy and reliability, ensuring it was clean. *This project aims to uncover patterns and trends indicative of mean reversion behavior in the cryptocurrency market.* By leveraging statistical analysis and mean reversion techniques, we seek to identify potential trading opportunities and develop effective mean reversion trading strategies tailored to BTC and AVAX.

```
#install necessary packages (cryptoquoes,dplyr,xts,ggplot2,tseries)
#install.packages(
#  pkgs = 'cryptoQuotes',
#  dependencies = TRUE
#)
```

```
library(cryptoQuotes)
```

```
##
## ── cryptoQuotes 1.3.1 ──────────────────────────────────────────────────────
## ♥ release notes ★ Browse the source code ★ Read the documentation ★ Join the
## discussion
```

#Crypto analaysis for last two years.

```
#Function to retrieve the hourly closing prices for BTC .
# Calculate the end date as today
end_date <- as.Date(Sys.Date())

# Calculate the start date as two years ago
start_date <- end_date - 365 * 2  # 365 days in a year

# Initialize an empty list to store dataframes
dataframes <- list()

# Loop to fetch data month by month
while (end_date >= start_date) {
  # Calculate the start date for the current month
  month_start <- start_date

  month_end <- start_date + 29

  # Ensure month_end does not exceed today's date
  if (month_end > end_date) {
    month_end <- end_date
  }

  # Fetch data for the current month
  data <- cryptoQuotes::get_quote(
    ticker = "BTCUSDT",
    source = "binance",
    futures = FALSE,
    interval = "1h",
    from = as.POSIXct(paste(month_start, "00:00:00")),
    to = as.POSIXct(paste(month_end, "23:59:59"))
  )

  # Append data to the list
  dataframes[[length(dataframes) + 1]] <- data

  # Move 'start_date' to the next month
  start_date <- month_end + 1
}

# Combine all dataframes into a single dataframe
BTC <- do.call(rbind, dataframes)
```

```
# Show the trend in line chart
plot( BTC$close, type = "l", main = "Line Chart")
```

The line chart visually depicts the price trend of Bitcoin (BTC) over time. The data shows a gradual increase in price initially, reaching a peak around March. Subsequently, the price stabilizes with slight fluctuations in the recent months.

```
#Function to retrieve the hourly closing prices for AVAX.
# Calculate the end date as today
end_date <- as.Date(Sys.Date())

# Calculate the start date as two years ago
start_date <- end_date - 365 * 2  # 365 days in a year

# Initialize an empty list to store dataframes
dataframes <- list()

# Loop to fetch data month by month
while (end_date >= start_date) {
  # Calculate the start date for the current month
  month_start <- start_date

  month_end <- start_date + 29

  # Ensure month_end does not exceed today's date
  if (month_end > end_date) {
    month_end <- end_date
  }

  # Fetch data for the current month
  data <- cryptoQuotes::get_quote(
    ticker = "AVAXUSDT",
    source = "binance",
    futures = FALSE,
    interval = "1h",
    from = as.POSIXct(paste(month_start, "00:00:00")),
    to = as.POSIXct(paste(month_end, "23:59:59"))
  )

  # Append data to the list
  dataframes[[length(dataframes) + 1]] <- data

  # Move 'start_date' to the next month
  start_date <- month_end + 1
}

# Combine all dataframes into a single dataframe
AVAX <- do.call(rbind, dataframes)
```

```
# Show the trend in line chart
plot( AVAX$close, type = "l", main = "Line Chart")
```

The line chart illustrates the price fluctuations of Avalanche (AVAX) over time. Initially, the price shows considerable volatility, reaching a peak around March. In recent months, there has been a slight decrease in price.

Using the dplyr library subset the dataframe to use closing prices only.

```
library(dplyr, warn.conflicts = FALSE)
```

```
BTC_close <- subset(BTC, select = "close")
AVAX_close <- subset(AVAX, select = "close")
```

#Merge the dataframes to create a single dataframe to store hourly BTC and AVAX prices

```
library(xts)
```

```
## Loading required package: zoo
```

```
##
## Attaching package: 'zoo'
```

```
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```
##
## ########################## Warning from 'xts' package ##########################
## #                                                                              #
## # The dplyr lag() function breaks how base R's lag() function is supposed to  #
## # work, which breaks lag(my_xts). Calls to lag(my_xts) that you type or       #
## # source() into this session won't work correctly.                             #
## #                                                                              #
## # Use stats::lag() to make sure you're not using dplyr::lag(), or you can add #
## # conflictRules('dplyr', exclude = 'lag') to your .Rprofile to stop           #
## # dplyr from breaking base R's lag() function.                                 #
## #                                                                              #
## # Code in packages is not affected. It's protected by R's namespace mechanism #
## # Set `options(xts.warn_dplyr_breaks_lag = FALSE)` to suppress this warning.  #
## #                                                                              #
## ################################################################################
```

```
##
## Attaching package: 'xts'
```

```
## The following objects are masked from 'package:dplyr':
##
##     first, last
```

```
# Join columns based on index
stock_df <- cbind(BTC_close,AVAX_close)
# Customize column names
colnames(stock_df) <- c("BTC", "AVAX")
head(stock_df,3)
```

```
##                          BTC  AVAX
## 2022-06-23 00:00:00 20693.98 17.03
## 2022-06-23 01:00:00 20080.10 16.56
## 2022-06-23 02:00:00 20322.73 16.71
```

Data cleaning

```
# Check the data types of each variable
sapply(stock_df, class)
```

```
##      BTC   AVAX
## [1,] "xts" "xts"
## [2,] "zoo" "zoo"
```

```
# Check for missing values
anyNA(stock_df)
```

```
## [1] FALSE
```

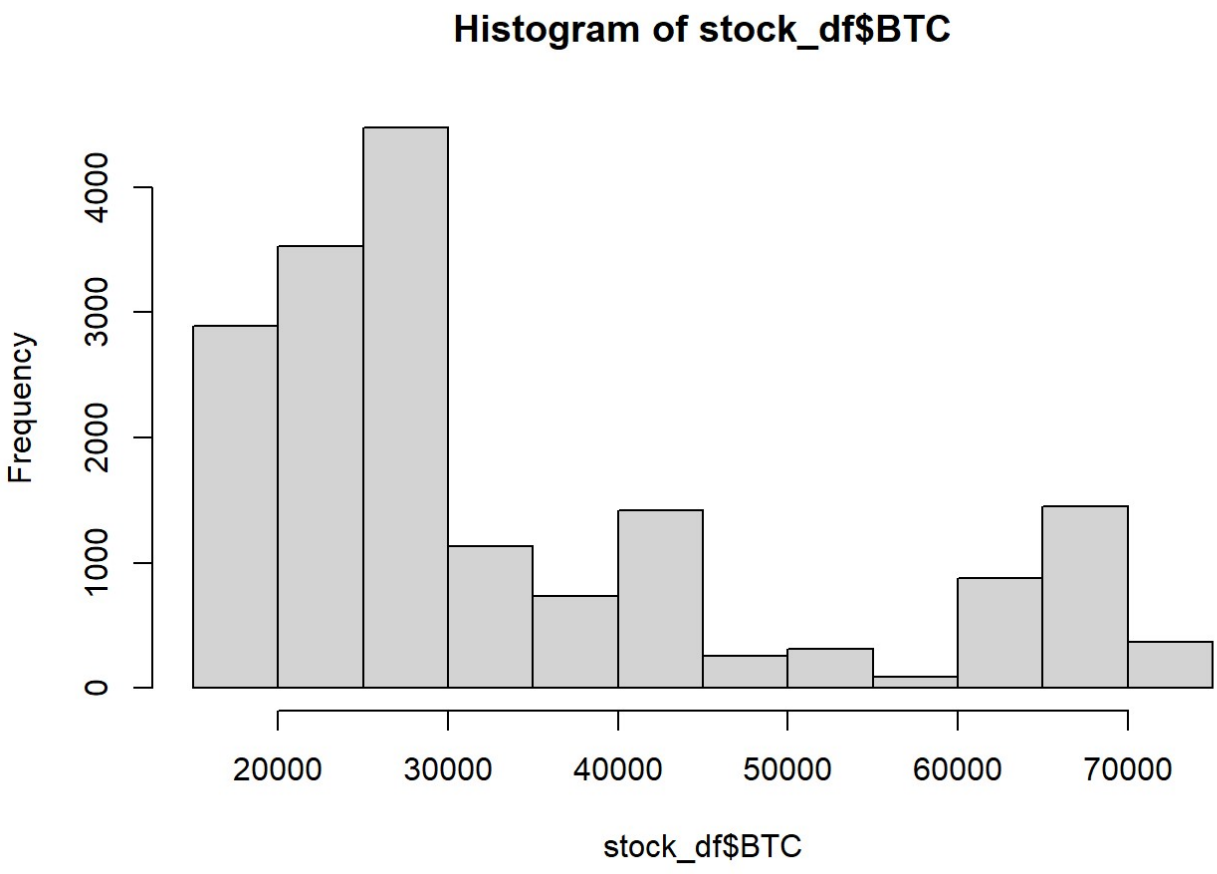
```
# Check for any impossible values
subset(stock_df, BTC <= 0)
```

```
##      BTC AVAX
```

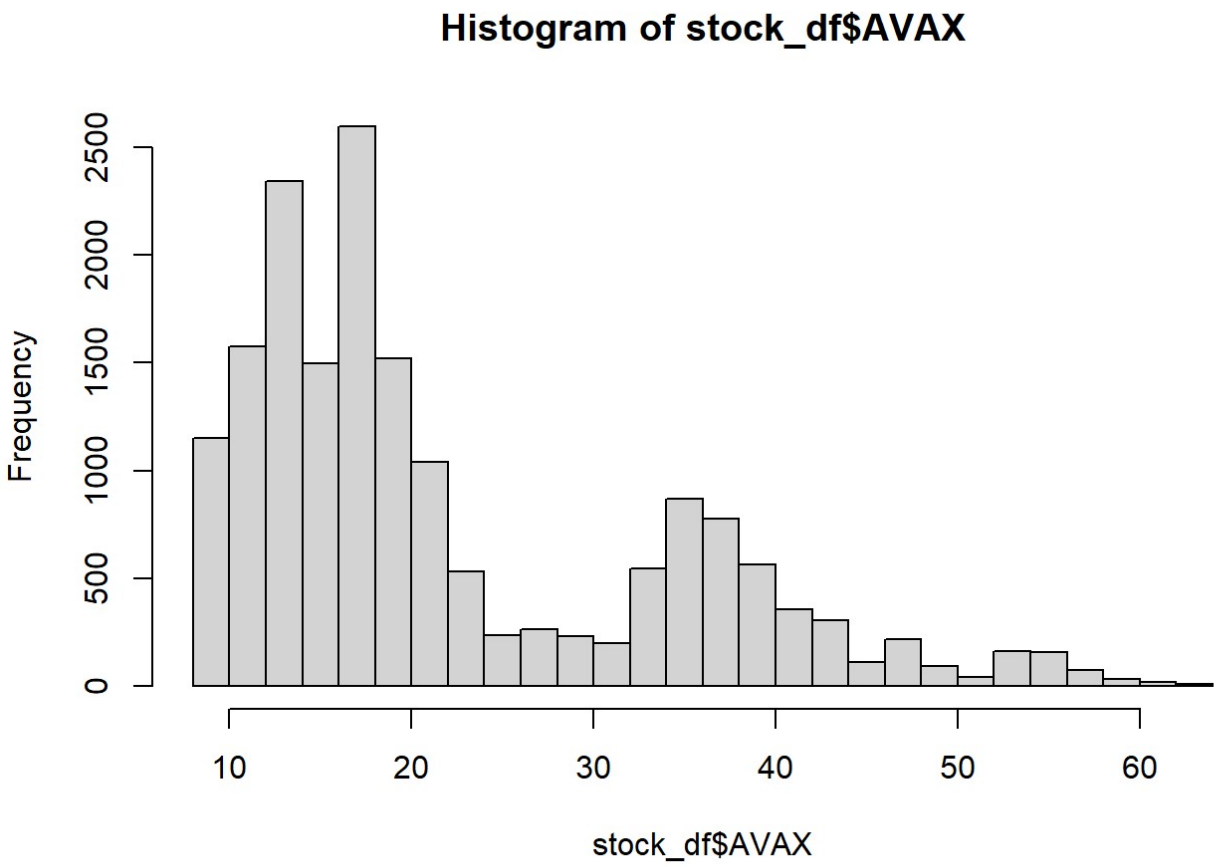
```
subset(stock_df, AVAX <= 0)
```

```
##      BTC AVAX
```

```
# Check for outliers
hist(stock_df$BTC, breaks = 20)
```




```
hist(stock_df$AVAX, breaks = 20)
```

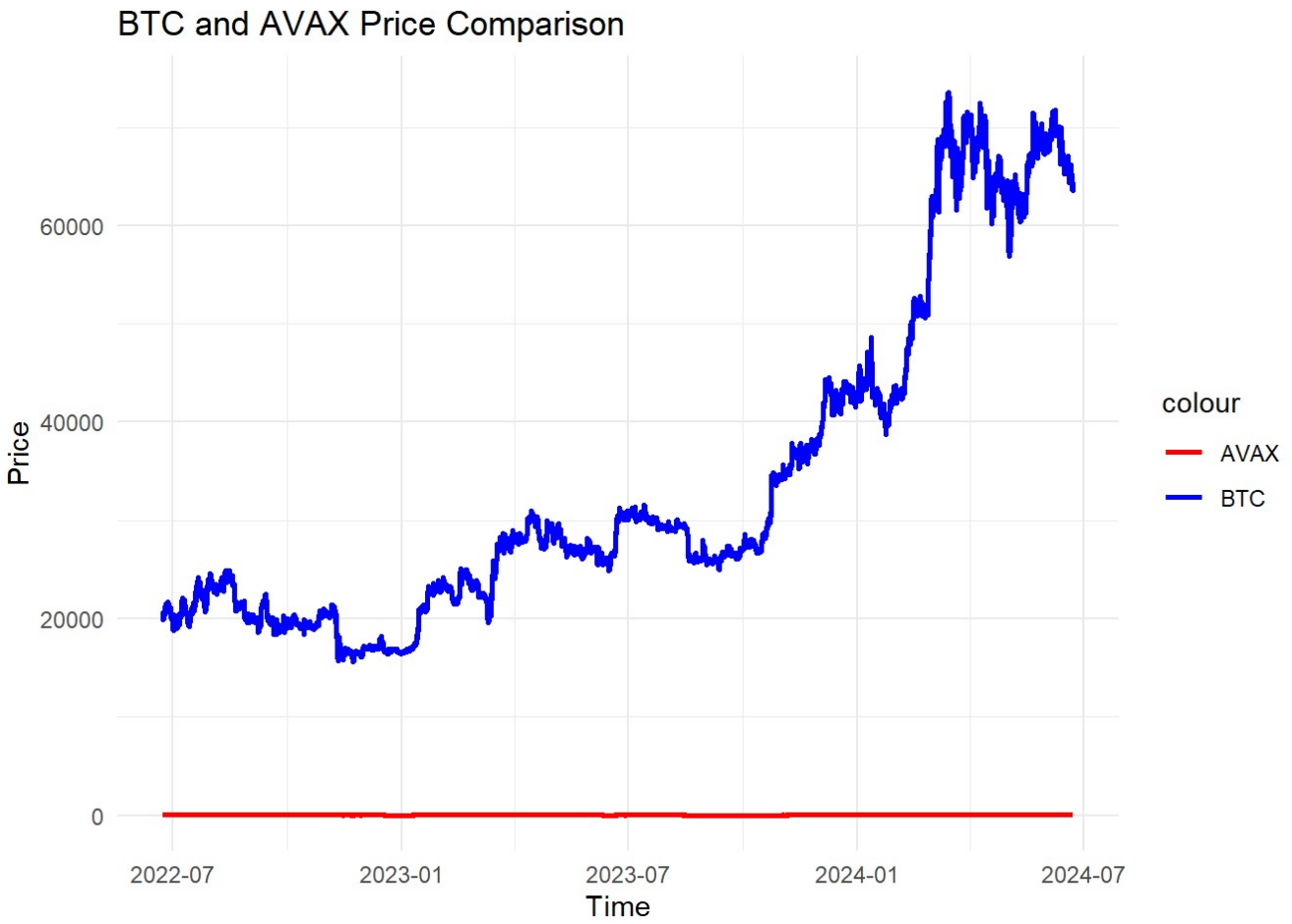


Statistical Analysis

```
# Load the ggplot2 library
library(ggplot2)
```

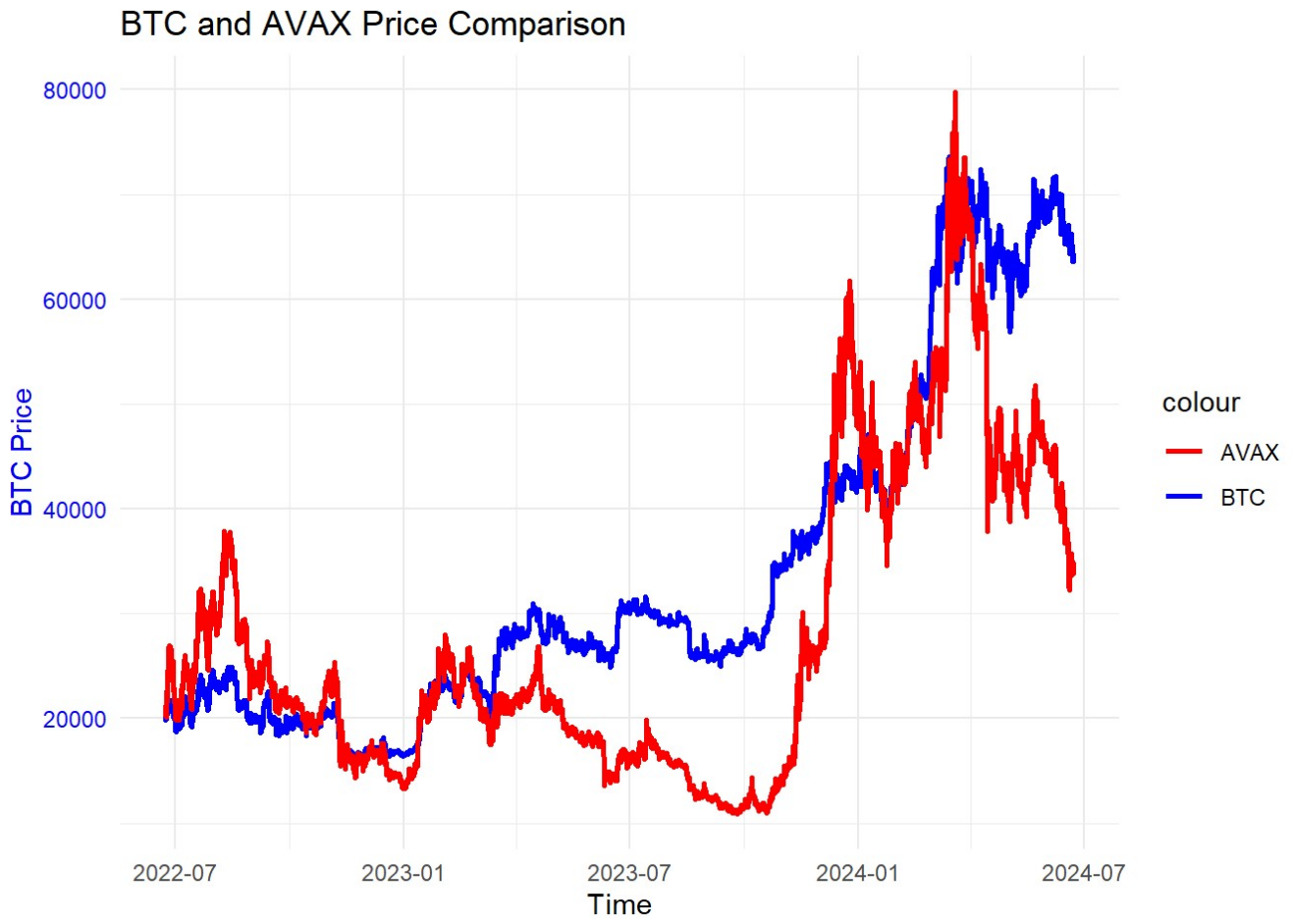
```
#  Visualize the price trend in line chart
ggplot(stock_df, aes(x = index(stock_df))) +
  geom_line(aes(y = BTC, color = "BTC"), size = 1) +
  geom_line(aes(y = AVAX, color = "AVAX"), size = 1) +
  scale_color_manual(values = c("BTC" = "blue", "AVAX" = "red")) +
  labs(x = "Time", y = "Price", title = "BTC and AVAX Price Comparison") +
  theme_minimal()
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## ℹ Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



In this case, as the price ranges differ, the trends will be displayed using dual-axis line chart.

```
# Visualize the price trend in a dual axis line chart
ggplot(stock_df, aes(x = index(stock_df))) +
  geom_line(aes(y = BTC, color = "BTC"), size = 1) +
  geom_line(aes(y = AVAX * 1250, color = "AVAX"), size = 1) +  # Scale AVAX by 2000 for be
tter visualization
  scale_color_manual(values = c("BTC" = "blue", "AVAX" = "red")) +
  labs(x = "Time", y = "BTC Price", title = "BTC and AVAX Price Comparison") +
  theme_minimal() +
  theme(axis.title.y.right = element_text(color = "red"),
        axis.text.y.right = element_text(color = "red"),
        axis.title.y.left = element_text(color = "blue"),
        axis.text.y.left = element_text(color = "blue"))
```



Upon comparing BTC and AVAX price trends in a single plot, a notable correlation is evident. Specifically, peak months and low points align closely between the two cryptocurrencies, indicating a similar trend or pattern over time.

```
#Calculate the pairwise correlation coefficients.
cc <- cor(stock_df)

print(cc)
```

```
##            BTC      AVAX
## BTC  1.0000000 0.8240209
## AVAX 0.8240209 1.0000000
```

The correlation coefficient between BTC and AVAX is 0.82. This indicates a strong positive linear relationship between the two cryptocurrencies.

```
#calculate the spread and ratios and add it into the stock dataframe
stock_stat_df <- stock_df
stock_stat_df$Spread <- stock_df$BTC - stock_df$AVAX
stock_stat_df$Ratio1 <- stock_df$BTC / stock_df$AVAX
stock_stat_df$Ratio2 <- stock_df$AVAX / stock_df$BTC
```

```
# Performing ADF test to check for the stationarity.
library(tseries)
```

```
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

```
#perform augmented Dickey-Fuller test
adf.test(stock_stat_df$BTC)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  stock_stat_df$BTC
## Dickey-Fuller = -1.7603, Lag order = 25, p-value = 0.6801
## alternative hypothesis: stationary
```

```
adf.test(stock_stat_df$AVAX)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  stock_stat_df$AVAX
## Dickey-Fuller = -1.7167, Lag order = 25, p-value = 0.6986
## alternative hypothesis: stationary
```

```
adf.test(stock_stat_df$Spread)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  stock_stat_df$Spread
## Dickey-Fuller = -1.7604, Lag order = 25, p-value = 0.6801
## alternative hypothesis: stationary
```

```
adf.test(stock_stat_df$Ratio1)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  stock_stat_df$Ratio1
## Dickey-Fuller = -1.499, Lag order = 25, p-value = 0.7909
## alternative hypothesis: stationary
```

```
adf.test(stock_stat_df$Ratio2)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  stock_stat_df$Ratio2
## Dickey-Fuller = -1.8003, Lag order = 25, p-value = 0.6631
## alternative hypothesis: stationary
```

None of the variables (BTC, AVAX, Spread, Ratio1, Ratio2) show strong evidence against the null hypothesis of non-stationarity based on the ADF test results. Therefore, we cannot conclude that any of these time series are stationary based on the ADF test at a typical significance level (e.g., 5%). Therefore further ananlysis required.

```
# Differencing
stock_stat_df$BTC_diff <- diff(stock_stat_df$BTC)
stock_stat_df$AVAX_diff <- diff(stock_stat_df$AVAX)
```

```
# Log Transformation
stock_stat_df$BTC_log <- log(stock_stat_df$BTC)
stock_stat_df$AVAX_log <- log(stock_stat_df$AVAX)
```

```
# Check for missing values
# Check for NA values column-wise
na_counts <- colSums(is.na(stock_stat_df))

# Print NA counts for each column
print("NA counts in each column:")
```

```
## [1] "NA counts in each column:"
```

```
print(na_counts)
```

```
##       BTC      AVAX    Spread    Ratio1    Ratio2  BTC_diff AVAX_diff   BTC_log
##         0         0         0         0         0         1         1         0
##  AVAX_log
##         0
```

```
# Step 1: Identify columns with missing values in the first row
cols_with_na <- colnames(stock_stat_df)[apply(stock_stat_df, 2, function(x) any(is.na(x
[1])))]

# Step 2: Calculate column means
column_means <- sapply(stock_stat_df, function(x) mean(x, na.rm = TRUE))

# Step 3: Replace NA with column means in the first row
for (col in cols_with_na) {
  if (is.na(stock_stat_df[1, col])) {
    stock_stat_df[1, col] <- column_means[col]
  }
}
```

```
#perform augmented Dickey-Fuller test
adf.test(stock_stat_df$BTC_diff)
```

```
## Warning in adf.test(stock_stat_df$BTC_diff): p-value smaller than printed
## p-value
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  stock_stat_df$BTC_diff
## Dickey-Fuller = -27.275, Lag order = 25, p-value = 0.01
## alternative hypothesis: stationary
```

```
adf.test(stock_stat_df$AVAX_diff)
```

```
## Warning in adf.test(stock_stat_df$AVAX_diff): p-value smaller than printed
## p-value
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  stock_stat_df$AVAX_diff
## Dickey-Fuller = -27.32, Lag order = 25, p-value = 0.01
## alternative hypothesis: stationary
```

```
adf.test(stock_stat_df$BTC_log)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  stock_stat_df$BTC_log
## Dickey-Fuller = -2.2124, Lag order = 25, p-value = 0.4884
## alternative hypothesis: stationary
```

```
adf.test(stock_stat_df$AVAX_log)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  stock_stat_df$AVAX_log
## Dickey-Fuller = -1.5175, Lag order = 25, p-value = 0.7831
## alternative hypothesis: stationary
```

BTC_diff and AVAX_diff is likely stationary based on the ADF test result with a very low p-value (0.01).

Based on these results, it seems that differencing the data has made it stationary, while the log transformation did not achieve the same result. Therefore, differenced series might be more suitable for further analysis involving mean reversion.

Mean Reversion

We calculate the mean and standard deviation as they are essential parameters for implementing a mean reversion trading strategy.

```
# Compute mean and standard deviation
mean_BTC <- mean(coredata(stock_stat_df[,"BTC"]), na.rm = TRUE)
std_BTC <- sd(coredata(stock_stat_df[,"BTC"]), na.rm = TRUE)

mean_AVAX <- mean(coredata(stock_stat_df[,"AVAX"]), na.rm = TRUE)
std_AVAX <- sd(coredata(stock_stat_df[,"AVAX"]), na.rm = TRUE)

# Print results
cat("mean BTC:", mean_BTC, "\n")
```

```
## mean BTC: 33666.7
```

```
cat("std BTC:", std_BTC, "\n")
```

```
## std BTC: 16272.11
```

```
cat("mean AVAX:", mean_AVAX, "\n")
```

```
## mean AVAX: 22.1625
```

```
cat("std AVAX:", std_AVAX, "\n")
```

```
## std AVAX: 11.64077
```

```
# Calculate rolling mean and standard deviation for BTC difference
rolling_mean_diff <- zoo::rollmean(stock_stat_df$BTC_diff, 30, na.pad = TRUE, align = "rig
ht")
```

```
## Warning in rollmean.zoo(stock_stat_df$BTC_diff, 30, na.pad = TRUE, align =
## "right"): na.pad is deprecated. Use fill.
```

```
rolling_sd_diff <- zoo::rollapply(stock_stat_df$BTC_diff, 30, sd, na.rm = TRUE, align = "r
ight")

# Calculate spread for the differences
stock_stat_df$Spread_diff <- stock_stat_df$BTC_diff - stock_stat_df$AVAX_diff

# Calculate rolling mean and standard deviation for spread of differences
rolling_mean_spread_diff <- zoo::rollmean(stock_stat_df$Spread_diff, 30, na.pad = TRUE, al
ign = "right")
```

```
## Warning in rollmean.zoo(stock_stat_df$Spread_diff, 30, na.pad = TRUE, align =
## "right"): na.pad is deprecated. Use fill.
```

```
rolling_sd_spread_diff <- zoo::rollapply(stock_stat_df$Spread_diff, 30, sd, na.rm = TRUE,
align = "right")

# Show the last few rows of data
print(tail(stock_stat_df, 3))
```

```
##                         BTC  AVAX   Spread   Ratio1       Ratio2 BTC_diff
## 2024-06-22 09:00:00 64143.56 27.48 64116.08 2334.191 0.0004284140   -11.46
## 2024-06-22 10:00:00 64056.47 27.17 64029.30 2357.618 0.0004241570   -87.09
## 2024-06-22 11:00:00 64216.11 27.31 64188.80 2351.377 0.0004252827   159.64
##                     AVAX_diff  BTC_log AVAX_log Spread_diff
## 2024-06-22 09:00:00     -0.02 11.06888 3.313458      -11.44
## 2024-06-22 10:00:00     -0.31 11.06752 3.302113      -86.78
## 2024-06-22 11:00:00      0.14 11.07001 3.307253      159.50
```

```
cat("----------------------------\n")
```

```
## ----------------------------
```

```
# Print the last few rows of rolling mean and standard deviation for the spread of differe
nces
cat("Last 3 rows of Rolling Mean for Spread of Differences:\n")
```

```
## Last 3 rows of Rolling Mean for Spread of Differences:
```

```
print(tail(rolling_mean_spread_diff, 3))
```

```
##                     Spread_diff
## 2024-06-22 09:00:00   -27.54200
## 2024-06-22 10:00:00   -35.30067
## 2024-06-22 11:00:00   -27.78000
```

```
cat("----------------------------\n")
```

```
## ----------------------------
```

```
cat("Last 3 rows of Rolling Standard Deviation for Spread of Differences:\n")
```

```
## Last 3 rows of Rolling Standard Deviation for Spread of Differences:
```

```
print(tail(rolling_sd_spread_diff, 3))
```

```
##                     Spread_diff
## 2024-06-22 09:00:00    255.1459
## 2024-06-22 10:00:00    253.2191
## 2024-06-22 11:00:00    255.6114
```

```
# Calculate Z-score of the spread difference
window_size <- 30
rolling_mean <- rollapply(stock_stat_df$Spread_diff, width = window_size, FUN = mean, alig
n = "right", fill = NA)
rolling_sd <- rollapply(stock_stat_df$Spread_diff, width = window_size, FUN = sd, align =
"right", fill = NA)

stock_stat_df$Z_Score_diff <- (stock_stat_df$Spread_diff - rolling_mean) / rolling_sd

# Define thresholds for buy and sell signals
buy_threshold <- -2  # Buy signal when Z-score is below this threshold
sell_threshold <- 2  # Sell signal when Z-score is above this threshold

# Identify potential buy and sell signals
potential_buy_signals <- stock_stat_df[stock_stat_df$Z_Score_diff < buy_threshold, ]
potential_sell_signals <- stock_stat_df[stock_stat_df$Z_Score_diff > sell_threshold, ]
```

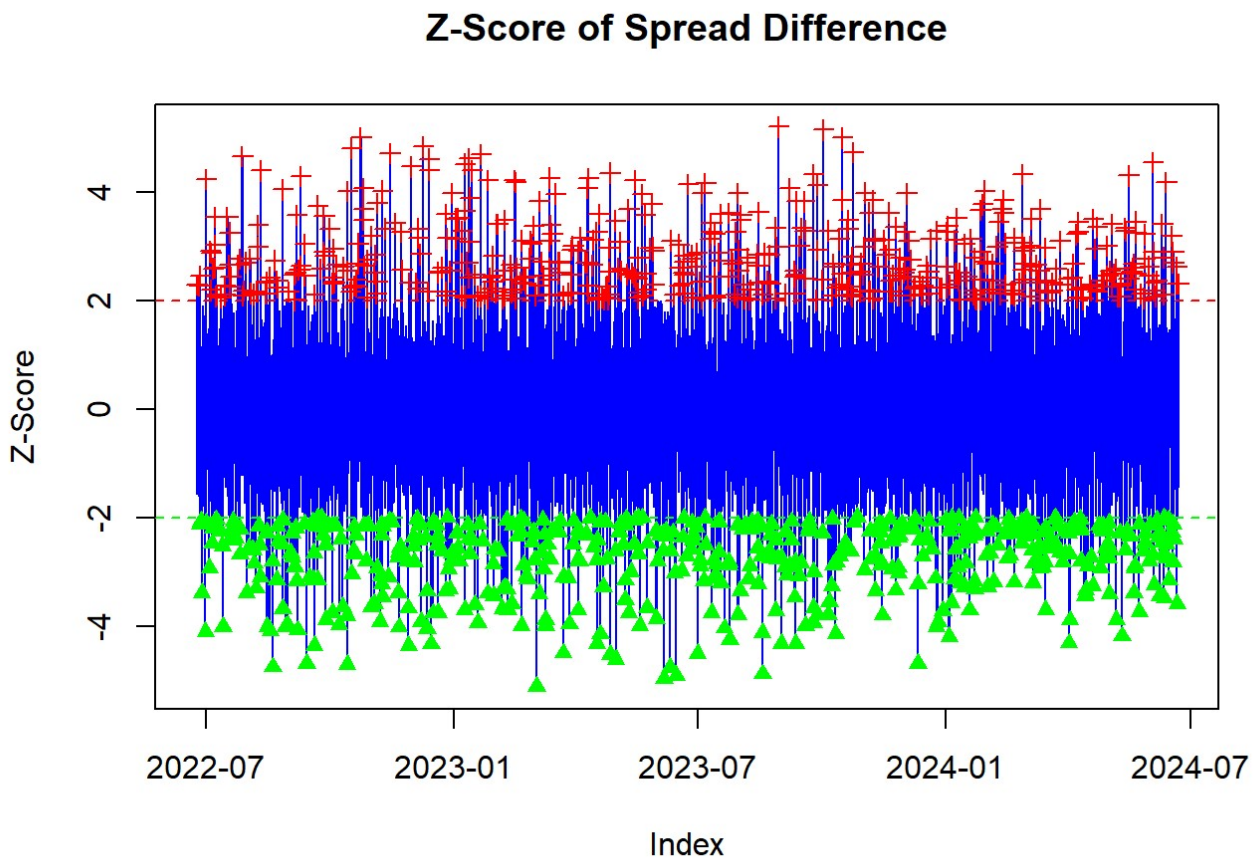
```
# Set a larger plot size and adjust margins
par(mar = c(5, 5, 4, 2) + 0.1)  # Adjust the margins as needed

plot(index(stock_stat_df), stock_stat_df$Z_Score_diff, type = "l", col = "blue", xlab = "I
ndex", ylab = "Z-Score",
     main = "Z-Score of Spread Difference")

# Adding potential buy signals as green triangles
points(index(potential_buy_signals), potential_buy_signals$Z_Score_diff, col = "green", pc
h = 17)

# Adding potential sell signals
points(index(potential_sell_signals), potential_sell_signals$Z_Score_diff,
       col = "red", pch = 3)

# Adding horizontal lines for buy and sell thresholds
abline(h = buy_threshold, col = "green", lty = 2, lwd = 1)  # Buy threshold line
abline(h = sell_threshold, col = "red", lty = 2, lwd = 1)  # Sell threshold line
```




```
# Setting x-axis label
xlabel <- "Date"  # Adjust this according to your data

# Setting y-axis label
ylabel <- "Z-Score"  # Adjust this according to your data

# Setting plot title
title <- "Z-Score of Spread Difference with Mean Reversion Signals "
```

In the chart, we observe the Z-score of the spread difference between BTC and AVAX over the past two years. The blue line tracks deviations from the historical mean spread, indicating periods where BTC may be relatively overvalued or undervalued compared to AVAX. Green triangles highlight potential buy signals, signaling opportunities when BTC is potentially undervalued relative to AVAX, while red triangles denote sell signals, suggesting times when BTC may be overvalued. The horizontal dashed lines represent buy and sell thresholds, providing clear points of reference for decision-making. This approach leverages statistical analysis to optimize trading strategies, focusing on mean reversion principles to achieve sustainable profitability in cryptocurrency markets.

#Hourly analaysis for last one year.

```
#Function to retrieve the hourly closing prices for BTC .
# Calculate the end date as today
end_date <- as.Date(Sys.Date())

# Calculate the start date as one years ago
start_date <- end_date - 365   # 365 days in a year

# Initialize an empty list to store dataframes
dataframes <- list()

# Loop to fetch data month by month
while (end_date >= start_date) {
  # Calculate the start date for the current month
  month_start <- start_date

  month_end <- start_date + 29

  # Ensure month_end does not exceed today's date
  if (month_end > end_date) {
    month_end <- end_date
  }

  # Fetch data for the current month
  data <- cryptoQuotes::get_quote(
    ticker = "BTCUSDT",
    source = "binance",
    futures = FALSE,
    interval = "1h",
    from = as.POSIXct(paste(month_start, "00:00:00")),
    to = as.POSIXct(paste(month_end, "23:59:59"))
  )

  # Append data to the list
  dataframes[[length(dataframes) + 1]] <- data

  # Move 'start_date' to the next month
  start_date <- month_end + 1
}

# Combine all dataframes into a single dataframe
BTC <- do.call(rbind, dataframes)
```

```
#Function to retrieve the hourly closing prices for AVAX.
# Calculate the end date as today
end_date <- as.Date(Sys.Date())

# Calculate the start date as one year ago
start_date <- end_date - 365   # 365 days in a year

# Initialize an empty list to store dataframes
dataframes <- list()

# Loop to fetch data month by month
while (end_date >= start_date) {
  # Calculate the start date for the current month
  month_start <- start_date

  month_end <- start_date + 29

  # Ensure month_end does not exceed today's date
  if (month_end > end_date) {
    month_end <- end_date
  }

  # Fetch data for the current month
  data <- cryptoQuotes::get_quote(
    ticker = "AVAXUSDT",
    source = "binance",
    futures = FALSE,
    interval = "1h",
    from = as.POSIXct(paste(month_start, "00:00:00")),
    to = as.POSIXct(paste(month_end, "23:59:59"))
  )

  # Append data to the list
  dataframes[[length(dataframes) + 1]] <- data

  # Move 'start_date' to the next month
  start_date <- month_end + 1
}

# Combine all dataframes into a single dataframe
AVAX <- do.call(rbind, dataframes)
```

Using the dplyr library subset the dataframe to use cloing prices only.

```
library(dplyr, warn.conflicts = FALSE)
```

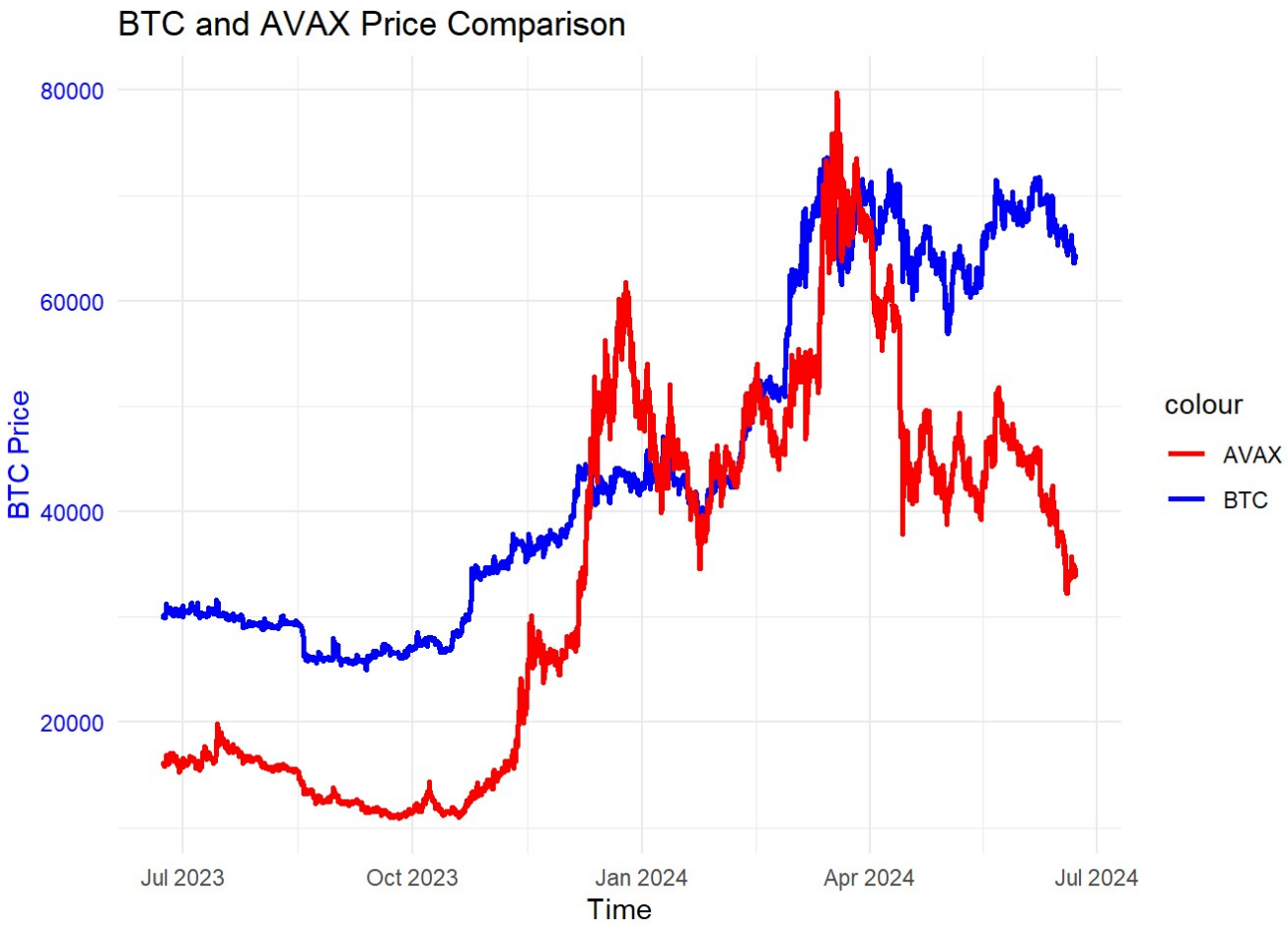
```
BTC_close <- subset(BTC, select = "close")
AVAX_close <- subset(AVAX, select = "close")
```

#Merge the dataframes to create a single dataframe to store hourly BTC and AVAX prices

```
library(xts)
```

```
# Join columns based on index
stock_df <- cbind(BTC_close,AVAX_close)
# Customize column names
colnames(stock_df) <- c("BTC", "AVAX")
```

```
# Visualize the price trend in a dual axis line chart
ggplot(stock_df, aes(x = index(stock_df))) +
  geom_line(aes(y = BTC, color = "BTC"), size = 1) +
  geom_line(aes(y = AVAX * 1250, color = "AVAX"), size = 1) +  # Scale AVAX by 2000 for be
tter visualization
  scale_color_manual(values = c("BTC" = "blue", "AVAX" = "red")) +
  labs(x = "Time", y = "BTC Price", title = "BTC and AVAX Price Comparison") +
  theme_minimal() +
  theme(axis.title.y.right = element_text(color = "red"),
        axis.text.y.right = element_text(color = "red"),
        axis.title.y.left = element_text(color = "blue"),
        axis.text.y.left = element_text(color = "blue"))
```




```
#Calculate the pairwise correlation coefficients.
cc <- cor(stock_df)

print(cc)
```

```
##            BTC      AVAX
## BTC  1.0000000 0.8500973
## AVAX 0.8500973 1.0000000
```

```
#calculate the spread and ratios and add it into the stock dataframe
stock_stat_df <- stock_df
stock_stat_df$Spread <- stock_df$BTC - stock_df$AVAX
stock_stat_df$Ratio1 <- stock_df$BTC / stock_df$AVAX
stock_stat_df$Ratio2 <- stock_df$AVAX / stock_df$BTC
```

```
# Performing ADF test to check for the stationarity.
library(tseries)

#perform augmented Dickey-Fuller test
adf.test(stock_stat_df$BTC)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  stock_stat_df$BTC
## Dickey-Fuller = -2.2122, Lag order = 20, p-value = 0.4885
## alternative hypothesis: stationary
```

```
adf.test(stock_stat_df$AVAX)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  stock_stat_df$AVAX
## Dickey-Fuller = -1.1131, Lag order = 20, p-value = 0.9206
## alternative hypothesis: stationary
```

```
adf.test(stock_stat_df$Spread)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  stock_stat_df$Spread
## Dickey-Fuller = -2.2131, Lag order = 20, p-value = 0.4881
## alternative hypothesis: stationary
```

```
adf.test(stock_stat_df$Ratio1)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  stock_stat_df$Ratio1
## Dickey-Fuller = -0.71567, Lag order = 20, p-value = 0.9692
## alternative hypothesis: stationary
```

```
adf.test(stock_stat_df$Ratio2)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  stock_stat_df$Ratio2
## Dickey-Fuller = -0.94155, Lag order = 20, p-value = 0.9483
## alternative hypothesis: stationary
```

None of the variables (BTC, AVAX, Spread, Ratio1, Ratio2) show strong evidence against the null hypothesis of non-stationarity based on the ADF test results. Therefore, we cannot conclude that any of these time series are stationary based on the ADF test at a typical significance level (e.g., 5%). Therefore further ananlysis required.

```
# Differencing
stock_stat_df$BTC_diff <- diff(stock_stat_df$BTC)
stock_stat_df$AVAX_diff <- diff(stock_stat_df$AVAX)
```

```
# Log Transformation
stock_stat_df$BTC_log <- log(stock_stat_df$BTC)
stock_stat_df$AVAX_log <- log(stock_stat_df$AVAX)
```

```
# Check for missing values
# Check for NA values column-wise
na_counts <- colSums(is.na(stock_stat_df))

# Print NA counts for each column
print("NA counts in each column:")
```

```
## [1] "NA counts in each column:"
```

```
print(na_counts)
```

```
##       BTC      AVAX    Spread    Ratio1    Ratio2  BTC_diff AVAX_diff   BTC_log
##         0         0         0         0         0         1         1         0
##  AVAX_log
##         0
```

```
# Step 1: Identify columns with missing values in the first row
cols_with_na <- colnames(stock_stat_df)[apply(stock_stat_df, 2, function(x) any(is.na(x
[1])))]

# Step 2: Calculate column means
column_means <- sapply(stock_stat_df, function(x) mean(x, na.rm = TRUE))

# Step 3: Replace NA with column means in the first row
for (col in cols_with_na) {
  if (is.na(stock_stat_df[1, col])) {
    stock_stat_df[1, col] <- column_means[col]
  }
}
```

```
#perform augmented Dickey-Fuller test
adf.test(stock_stat_df$BTC_diff)
```

```
## Warning in adf.test(stock_stat_df$BTC_diff): p-value smaller than printed
## p-value
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  stock_stat_df$BTC_diff
## Dickey-Fuller = -20.477, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
```

```
adf.test(stock_stat_df$AVAX_diff)
```

```
## Warning in adf.test(stock_stat_df$AVAX_diff): p-value smaller than printed
## p-value
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  stock_stat_df$AVAX_diff
## Dickey-Fuller = -20.53, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
```

```
adf.test(stock_stat_df$BTC_log)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  stock_stat_df$BTC_log
## Dickey-Fuller = -2.0552, Lag order = 20, p-value = 0.555
## alternative hypothesis: stationary
```

```
adf.test(stock_stat_df$AVAX_log)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  stock_stat_df$AVAX_log
## Dickey-Fuller = -0.66499, Lag order = 20, p-value = 0.9737
## alternative hypothesis: stationary
```

BTC_diff and AVAX_diff is likely stationary based on the ADF test result with a very low p-value (0.01).

Based on these results, it seems that differencing the data has made it stationary, while the log transformation did not achieve the same result. Therefore, differenced series might be more suitable for further analysis involving mean reversion.

Mean Reversion

We calculate the mean and standard deviation as they are essential parameters for implementing a mean reversion trading strategy.

```
# Compute mean and standard deviation
mean_BTC <- mean(coredata(stock_stat_df[,"BTC"]), na.rm = TRUE)
std_BTC <- sd(coredata(stock_stat_df[,"BTC"]), na.rm = TRUE)

mean_AVAX <- mean(coredata(stock_stat_df[,"AVAX"]), na.rm = TRUE)
std_AVAX <- sd(coredata(stock_stat_df[,"AVAX"]), na.rm = TRUE)

# Print results
cat("mean BTC:", mean_BTC, "\n")
```

```
## mean BTC: 45024.1
```

```
cat("std BTC:", std_BTC, "\n")
```

```
## std BTC: 15978.44
```

```
cat("mean AVAX:", mean_AVAX, "\n")
```

```
## mean AVAX: 27.09945
```

```
cat("std AVAX:", std_AVAX, "\n")
```

```
## std AVAX: 14.41406
```

```
# Calculate rolling mean and standard deviation for BTC difference
rolling_mean_diff <- zoo::rollmean(stock_stat_df$BTC_diff, 30, na.pad = TRUE, align = "rig
ht")
```

```
## Warning in rollmean.zoo(stock_stat_df$BTC_diff, 30, na.pad = TRUE, align =
## "right"): na.pad is deprecated. Use fill.
```

```
rolling_sd_diff <- zoo::rollapply(stock_stat_df$BTC_diff, 30, sd, na.rm = TRUE, align = "r
ight")

# Calculate spread for the differences
stock_stat_df$Spread_diff <- stock_stat_df$BTC_diff - stock_stat_df$AVAX_diff

# Calculate rolling mean and standard deviation for spread of differences
rolling_mean_spread_diff <- zoo::rollmean(stock_stat_df$Spread_diff, 30, na.pad = TRUE, al
ign = "right")
```

```
## Warning in rollmean.zoo(stock_stat_df$Spread_diff, 30, na.pad = TRUE, align =
## "right"): na.pad is deprecated. Use fill.
```

```
rolling_sd_spread_diff <- zoo::rollapply(stock_stat_df$Spread_diff, 30, sd, na.rm = TRUE,
align = "right")

# Show the last few rows of data
print(tail(stock_stat_df, 3))
```

```
##                         BTC  AVAX   Spread   Ratio1       Ratio2 BTC_diff
## 2024-06-22 09:00:00 64143.56 27.48 64116.08 2334.191 0.0004284140   -11.46
## 2024-06-22 10:00:00 64056.47 27.17 64029.30 2357.618 0.0004241570   -87.09
## 2024-06-22 11:00:00 64216.12 27.31 64188.81 2351.378 0.0004252826   159.65
##                     AVAX_diff  BTC_log AVAX_log Spread_diff
## 2024-06-22 09:00:00     -0.02 11.06888 3.313458      -11.44
## 2024-06-22 10:00:00     -0.31 11.06752 3.302113      -86.78
## 2024-06-22 11:00:00      0.14 11.07001 3.307253      159.51
```

```
cat("----------------------------\n")
```

```
## ----------------------------
```

```
# Print the last few rows of rolling mean and standard deviation for the spread of differe
nces
cat("Last 3 rows of Rolling Mean for Spread of Differences:\n")
```

```
## Last 3 rows of Rolling Mean for Spread of Differences:
```

```
print(tail(rolling_mean_spread_diff, 3))
```

```
##                     Spread_diff
## 2024-06-22 09:00:00   -27.54200
## 2024-06-22 10:00:00   -35.30067
## 2024-06-22 11:00:00   -27.77967
```

```
cat("----------------------------\n")
```

```
## ----------------------------
```

```
cat("Last 3 rows of Rolling Standard Deviation for Spread of Differences:\n")
```

```
## Last 3 rows of Rolling Standard Deviation for Spread of Differences:
```

```
print(tail(rolling_sd_spread_diff, 3))
```

```
##                     Spread_diff
## 2024-06-22 09:00:00    255.1459
## 2024-06-22 10:00:00    253.2191
## 2024-06-22 11:00:00    255.6116
```

```
# Calculate Z-score of the spread difference
window_size <- 30
rolling_mean <- rollapply(stock_stat_df$Spread_diff, width = window_size, FUN = mean, alig
n = "right", fill = NA)
rolling_sd <- rollapply(stock_stat_df$Spread_diff, width = window_size, FUN = sd, align =
"right", fill = NA)

stock_stat_df$Z_Score_diff <- (stock_stat_df$Spread_diff - rolling_mean) / rolling_sd

# Define thresholds for buy and sell signals
buy_threshold <- -2  # Buy signal when Z-score is below this threshold
sell_threshold <- 2  # Sell signal when Z-score is above this threshold

# Identify potential buy and sell signals
potential_buy_signals <- stock_stat_df[stock_stat_df$Z_Score_diff < buy_threshold, ]
potential_sell_signals <- stock_stat_df[stock_stat_df$Z_Score_diff > sell_threshold, ]
```

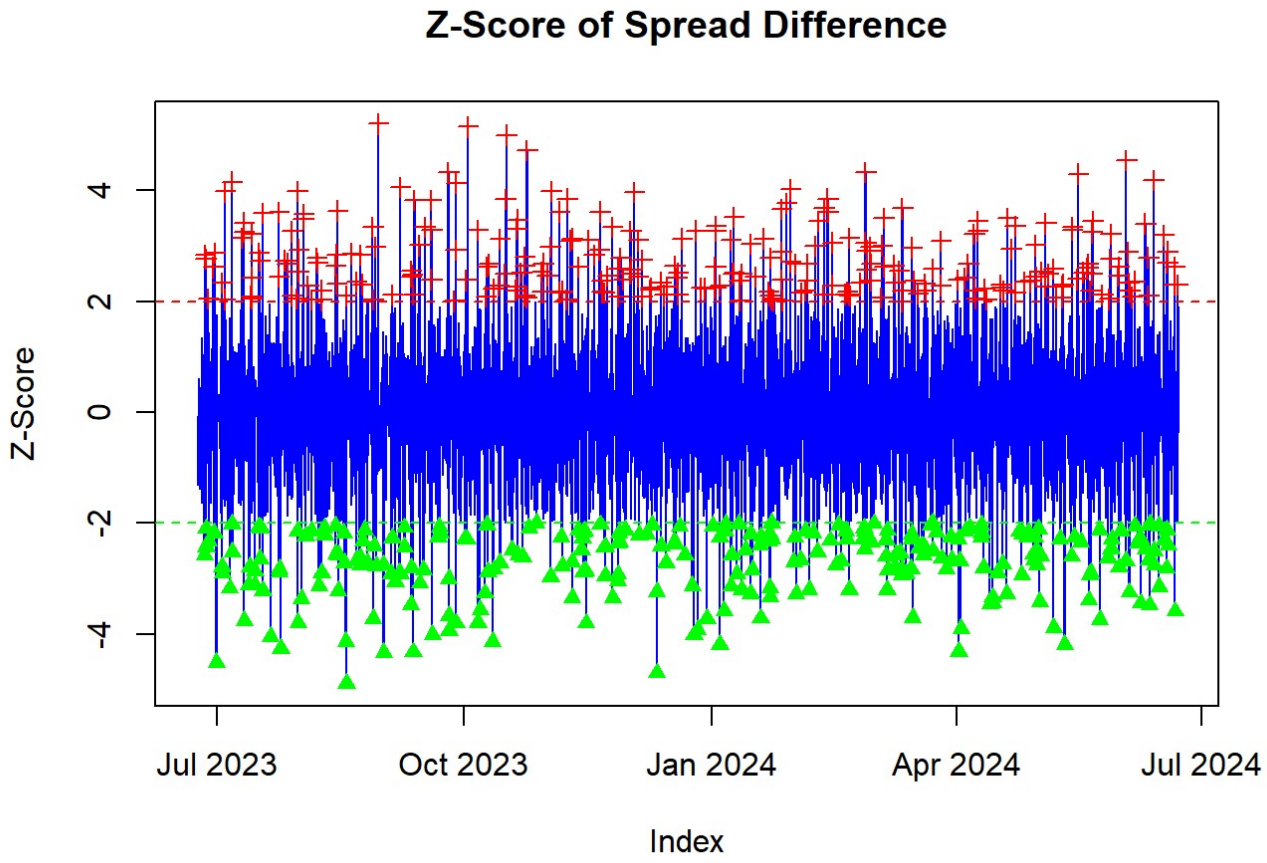
```
# Set a larger plot size and adjust margins
par(mar = c(5, 5, 4, 2) + 0.1)  # Adjust the margins as needed

plot(index(stock_stat_df), stock_stat_df$Z_Score_diff, type = "l", col = "blue", xlab = "I
ndex", ylab = "Z-Score",
     main = "Z-Score of Spread Difference")

# Adding potential buy signals as green triangles
points(index(potential_buy_signals), potential_buy_signals$Z_Score_diff, col = "green", pc
h = 17)

# Adding potential sell signals
points(index(potential_sell_signals), potential_sell_signals$Z_Score_diff,
       col = "red", pch = 3)

# Adding horizontal lines for buy and sell thresholds
abline(h = buy_threshold, col = "green", lty = 2, lwd = 1)  # Buy threshold line
abline(h = sell_threshold, col = "red", lty = 2, lwd = 1)  # Sell threshold line
```




```
# Setting x-axis label
xlabel <- "Date"  # Adjust this according to your data

# Setting y-axis label
ylabel <- "Z-Score"  # Adjust this according to your data

# Setting plot title
title <- "Z-Score of Spread Difference with Mean Reversion Signals"
```

When the Z-score is low, it indicates that the current spread is below its historical average, suggesting that it may be undervalued. Conversely, when the Z-score is high, it indicates that the current spread is above its historical average, suggesting that it may be overvalued.

However it's essential to thoroughly backtest and validate any trading strategy before implementing it in a live trading environment. Additionally, the effectiveness of such strategies may vary depending on the market conditions and the specific assets being traded.

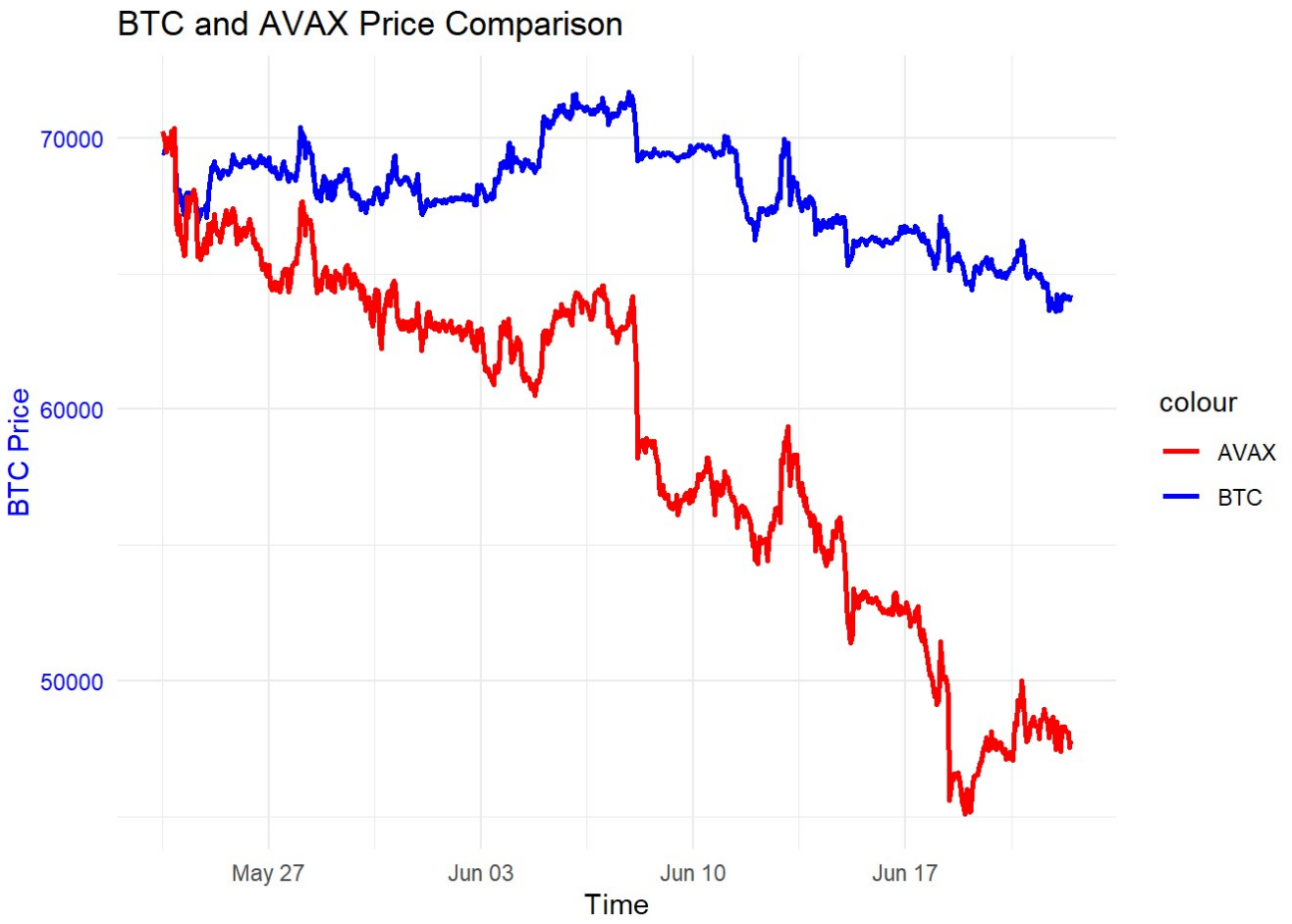
##Machine learning Algorithms

I suggest machine learning algorithms for this mean reversion strategy. Beacuse it is more safe and we can continuously monitor the performance of your classification model and trading strategy. Iterate on feature selection, model training, and hyperparameter tuning to improve the accuracy and robustness of your predictions.

BTC, AVAX price comparision

```
# Calculate correlation for last month (approximately 30 days)
last_month_df <- tail(stock_df, 30*24)
```

```
# Visualize the price trend in a dual axis line chart
ggplot(last_month_df, aes(x = index(last_month_df))) +
  geom_line(aes(y = BTC, color = "BTC"), size = 1) +
  geom_line(aes(y = AVAX * 1750, color = "AVAX"), size = 1) +  # Scale AVAX for better vis
ualization
  scale_color_manual(values = c("BTC" = "blue", "AVAX" = "red")) +
  labs(x = "Time", y = "BTC Price", title = "BTC and AVAX Price Comparison") +
  theme_minimal() +
  theme(axis.title.y.right = element_text(color = "red"),
        axis.text.y.right = element_text(color = "red"),
        axis.title.y.left = element_text(color = "blue"),
        axis.text.y.left = element_text(color = "blue"))
```



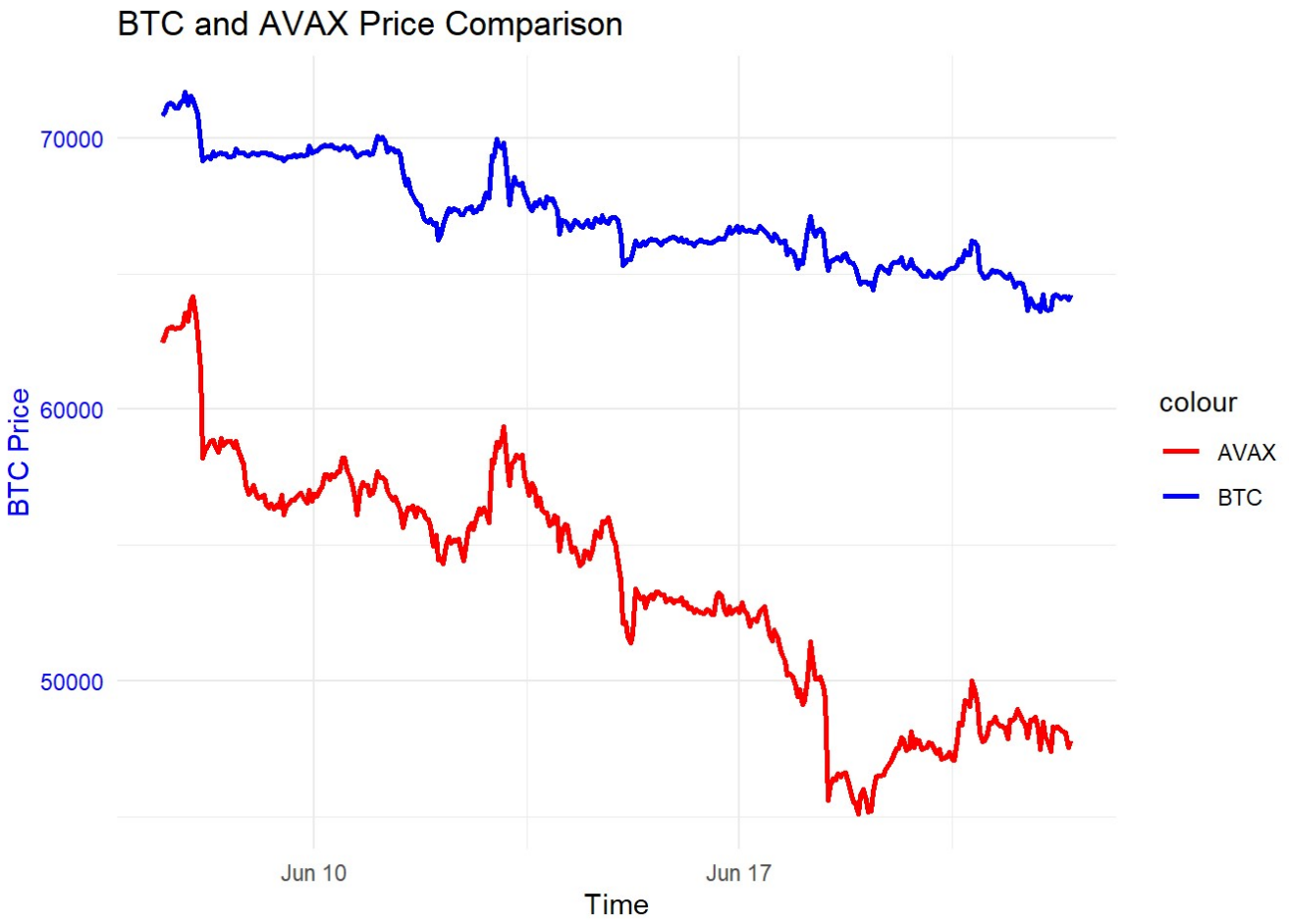

```
#Calculate the pairwise correlation coefficients for last month.
cc <- cor(stock_df)

print(cc)
```

```
##            BTC      AVAX
## BTC  1.0000000 0.8500973
## AVAX 0.8500973 1.0000000
```

```
# Calculate correlation for last month (approximately last 15 days)
last_15_df <- tail(stock_df, 15*24)
```

```
# Visualize the price trend in a dual axis line chart
ggplot(last_15_df, aes(x = index(last_15_df))) +
  geom_line(aes(y = BTC, color = "BTC"), size = 1) +
  geom_line(aes(y = AVAX * 1750, color = "AVAX"), size = 1) +  # Scale AVAX for better vis
ualization
  scale_color_manual(values = c("BTC" = "blue", "AVAX" = "red")) +
  labs(x = "Time", y = "BTC Price", title = "BTC and AVAX Price Comparison") +
  theme_minimal() +
  theme(axis.title.y.right = element_text(color = "red"),
        axis.text.y.right = element_text(color = "red"),
        axis.title.y.left = element_text(color = "blue"),
        axis.text.y.left = element_text(color = "blue"))
```




```
#Calculate the pairwise correlation coefficients for last month.
cc <- cor(last_15_df)

print(cc)
```

```
##            BTC      AVAX
## BTC  1.0000000 0.9230487
## AVAX 0.9230487 1.0000000
```