

Stock Price Prediction

1. Introduction

- **Motivation:** Through this case study, we aim to explore stock price analysis using standard forecasting methods, addressing the following questions:
 - Can we predict future stock prices by analyzing historical data?
 - Can we develop an inference model that comprehends the relationship between future prices and previous price indicators?
- **Contents:**
 1. We conduct a time-series analysis on the price movements of Bitcoin (BTC) and the S&P 500 index, representing two distinct stocks with unique characteristics.
 2. We construct an AutoRegressive Integrated Moving Average (ARIMA) model, a standard autoregressive model for predicting future trends in a time series.
 3. We build a Bayesian Structural Time Series (BSTS) model, a flexible and modular explanatory model capable of handling the uncertainty associated with price volatility.

2. Background

Data Source and Scope

We utilized the `quantmod` package, designed to assist quantitative traders in exploring and building trading models effortlessly. This package provides daily open, high, low, and close (OHLC) price data for most stocks.

Stocks of Interest

Our analysis focused on the price movements of two representative stocks with distinct characteristics:

1. Standard & Poor's 500 Index (S&P 500)

- A market-capitalization-weighted index comprising 500 leading publicly traded companies in the United States.
- Regarded as one of the best indicators of prominent American equities' performance and, by extension, the overall stock market.

2. Bitcoin (BTC-USD)

- The world's largest cryptocurrency by market capitalization.
- Known for its high volatility, influenced by many factors such as supply and demand, investor and user sentiments, government regulations, and media hype.

2.1. Dataset

```
In [ ]: install.packages("quantmod")
install.packages("forecast")
install.packages("highcharter")
install.packages("bsts")
install.packages("reshape2")
install.packages("dplyr")
install.packages("lubridate")
install.packages("ggplot2")
install.packages("gridExtra")
```

```
In [ ]: library(quantmod)
library(pillar)
library(forecast)
library(highcharter)
library(bsts)
library(reshape2)
library(dplyr)
library(lubridate)
library(ggplot2)
library(gridExtra)
```

3. Data Preprocessing

The data obtained consists of daily open, high, low, and close (OHLC) prices. For the purpose of this analysis, we will focus on the closing price, which can be retrieved using the `CL()` function. The closing price is widely considered a valuable indicator for assessing stock price fluctuations over time.

It's worth noting that some investors prefer to work with adjusted prices rather than closing prices. Adjusted prices take into account corporate actions such as stock splits, dividends, and rights offerings, providing a more accurate representation of a stock's historical performance.

```
In [ ]: btc_price <- getSymbols('BTC-USD', auto.assign=FALSE, from="2014-01-01")
sp_price <- getSymbols("^GSPC", auto.assign=FALSE, from="2014-01-01")
```

```
In [ ]: summary(btc_price)
summary(sp_price)
```

Index	BTC-USD.Open	BTC-USD.High	BTC-USD.Low
Min. :2014-09-17	Min. : 176.9	Min. : 211.7	Min. : 171.5
1st Qu.:2017-02-19	1st Qu.: 1041.7	1st Qu.: 1057.7	1st Qu.: 1021.2
Median :2019-07-26	Median : 8791.7	Median : 8958.5	Median : 8598.4
Mean :2019-07-26	Mean :16487.0	Mean :16869.6	Mean :16080.0
3rd Qu.:2021-12-29	3rd Qu.:27266.2	3rd Qu.:27788.8	3rd Qu.:26824.2
Max. :2024-06-03	Max. :73079.4	Max. :73750.1	Max. :71334.1
BTC-USD.Close	BTC-USD.Volume	BTC-USD.Adjusted	
Min. : 178.1	Min. :5.915e+06	Min. : 178.1	
1st Qu.: 1042.2	1st Qu.:2.341e+08	1st Qu.: 1042.2	
Median : 8801.8	Median :1.313e+10	Median : 8801.8	
Mean :16505.1	Mean :1.725e+10	Mean :16505.1	
3rd Qu.:27282.0	3rd Qu.:2.784e+10	3rd Qu.:27282.0	
Max. :73083.5	Max. :3.510e+11	Max. :73083.5	
Index	GSPC.Open	GSPC.High	GSPC.Low
Min. :2014-01-02	Min. :1744	Min. :1756	Min. :1738
1st Qu.:2016-08-09	1st Qu.:2141	1st Qu.:2150	1st Qu.:2133
Median :2019-03-19	Median :2833	Median :2849	Median :2819
Mean :2019-03-17	Mean :3088	Mean :3104	Mean :3070
3rd Qu.:2021-10-21	3rd Qu.:3993	3rd Qu.:4018	3rd Qu.:3956
Max. :2024-05-31	Max. :5340	Max. :5342	Max. :5302
GSPC.Close	GSPC.Volume	GSPC.Adjusted	
Min. :1742	Min. :1.297e+09	Min. :1742	
1st Qu.:2140	1st Qu.:3.370e+09	1st Qu.:2140	
Median :2837	Median :3.779e+09	Median :2837	
Mean :3089	Mean :3.952e+09	Mean :3089	
3rd Qu.:3991	3rd Qu.:4.299e+09	3rd Qu.:3991	
Max. :5321	Max. :9.977e+09	Max. :5321	

Data Attributes

The dataset contains the following attributes:

- **Open:** The opening price for that day
- **High:** The highest price paid for the stock that day
- **Low:** The lowest price paid for the stock that day
- **Close:** The closing price for that day
- **Volume:** The volume of stocks traded that day
- **Adjusted:** The adjusted closing price for that day, taking into account corporate actions such as stock splits, dividends, and rights offerings.

Analysis Objective

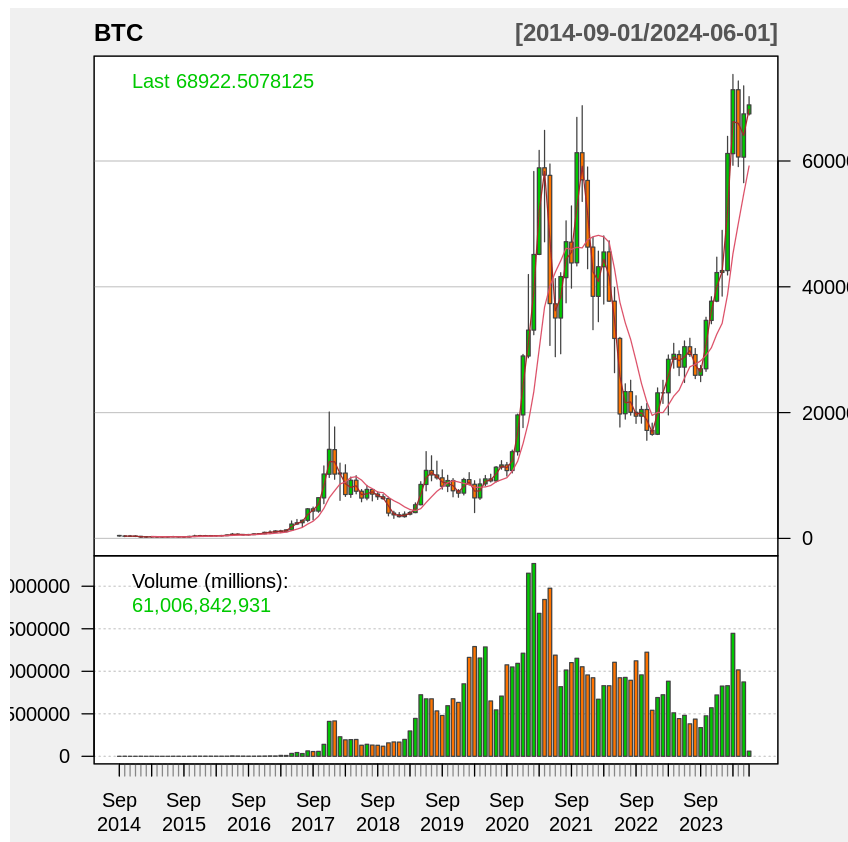
In this project, we will use the **closing price** as the **response variable** for further analysis. The closing price is considered a useful indicator to assess changes in stock prices over time. Therefore, the goal is to predict the future trend of a stock's closing price based on its previous price movements.

```
In [ ]: chartSeries(to.monthly(btc_price),
                    theme = chartTheme("white"),
```

```

name = "BTC",
TA = list("addVo()",
          "addSMA(2)",
          "addSMA(7, col = 2)")

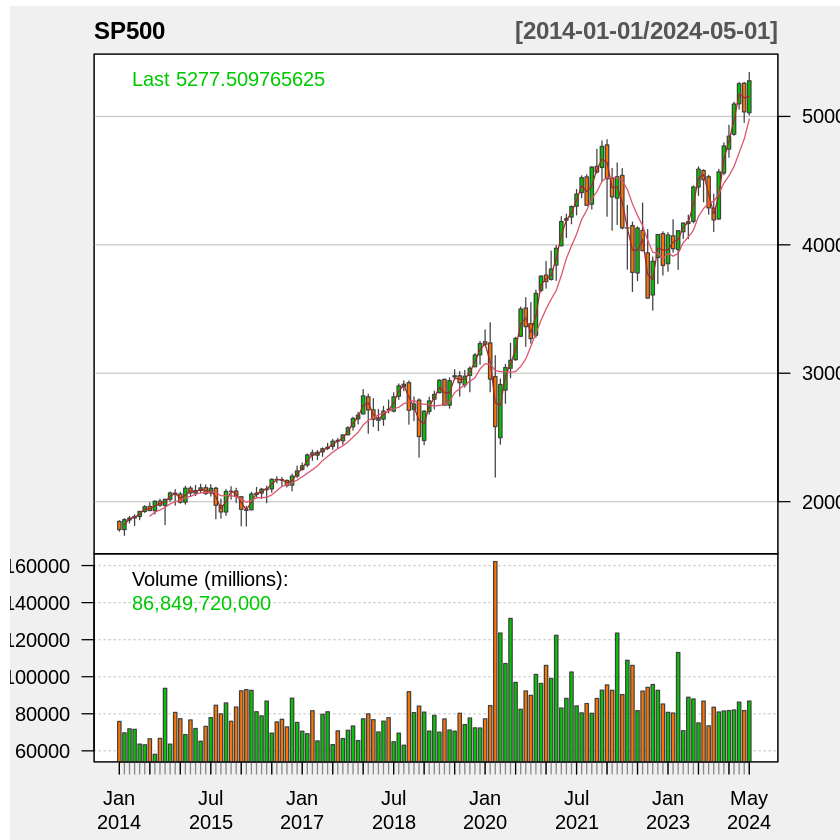
```



```

In [ ]: chartSeries(to.monthly(sp_price),
                    theme = chartTheme("white"),
                    name = "SP500",
                    TA = list("addVo()",
                              "addSMA(2)",
                              "addSMA(7, col = 2)")
                    )

```



Time Series Data

A time series is a sequence of data points indexed in chronological order, typically measured at successive points in time spaced at uniform intervals. Time series data exhibits a temporal ordering and dependence, where observations closer together in time tend to be more related than observations farther apart. This inherent temporal correlation within the data enables time series analysis techniques to identify patterns and model the underlying mechanisms driving the observed behavior.

Time series analysis is widely employed for forecasting future values by leveraging the historical patterns and trends present in the data. By capturing the temporal dynamics and dependencies, time series models can extrapolate the observed patterns to make informed predictions about upcoming observations. This forecasting capability finds applications across diverse domains, including finance, economics, environmental monitoring, demand forecasting, and many others, providing valuable insights to support data-driven decision-making processes.

Additive Decomposition

The additive decomposition model is represented by the following equation:

$$y_t = S_t + T_t + R_t$$

Where:

- y_t is the observed data.
- S_t is the seasonality. Seasonality refers to the cyclical patterns or periodic fluctuations in time series data at fixed intervals. These patterns exhibit regular rises and falls driven by underlying factors that systematically influence the observed phenomenon, often associated with recurring events or cyclical phenomena. A classic example is the sales of Christmas trees, which surge in December due to the holiday season but remain low in other months, reflecting an annual seasonal cycle.
- T_t is the trend. The trend represents the overall long-term increase or decrease in the values of a time series over an extended period. It reflects the underlying direction or trajectory of the data. Identifying and modeling trends is crucial for accurate forecasting and understanding the forces shaping the observed phenomenon.
- R_t is the noise. It is just some random data that we obtain after extracting the trend and the seasonal components from the time series.

```
In [ ]: # options(repr.plot.width = 20, repr.plot.height = 10)
```

```
sp_close <- Cl(to.monthly(sp_price))
sp_dc <- decompose(as.ts(sp_close))
btc_close <- Cl(to.monthly(btc_price))
btc_dc <- decompose(as.ts(btc_close))
```

```
In [ ]: ### Extract the components
```

```
components <- cbind.data.frame(
  sp_dc$x, sp_dc$trend, sp_dc$seasonal, sp_dc$random, time(to.monthly(sp_price))
)
names(components) <- c("Observed", "Trend", "Seasonality", "Random", "Date")
components <- melt(components, id="Date")
names(components) <- c("Date", "Component", "Value")
```

```
### Plot
```

```
plot1 <- ggplot(data=components, aes(x=Date, y=Value)) + geom_line() +
  theme_bw() + theme(legend.title = element_blank()) + ylab("") + xlab("") +
  facet_grid(Component ~ ., scales="free") + guides(colour=FALSE) +
  theme(axis.text.x=element_text(angle = -90, hjust = 0)) +
  ggtitle("S&P500 Decomposition")
```

```
components <- cbind.data.frame(
  btc_dc$x, btc_dc$trend, btc_dc$seasonal, btc_dc$random, time(to.monthly(btc_price))
)
names(components) <- c("Observed", "Trend", "Seasonality", "Random", "Date")
components <- melt(components, id="Date")
names(components) <- c("Date", "Component", "Value")
```

```
### Plot
```

```
plot2 <- ggplot(data=components, aes(x=Date, y=Value)) + geom_line() +
  theme_bw() + theme(legend.title = element_blank()) + ylab("") + xlab("") +
  facet_grid(Component ~ ., scales="free") + guides(colour=FALSE) +
  theme(axis.text.x=element_text(angle = -90, hjust = 0)) +
  ggtitle("BTC Decomposition")
```

```
grid.arrange(plot1, plot2, ncol=2)
```

Warning message:

“The ``<scale>`` argument of ``guides()`` cannot be ``FALSE``. Use “none” instead as of ggplot2 3.3.4.”

Warning message:

“The ``trans`` argument of ``continuous_scale()`` is deprecated as of ggplot2 3.5.0. **i** Please use the ``transform`` argument instead.”

Don't know how to automatically pick scale for object of type `<ts>`. Defaulting to continuous.

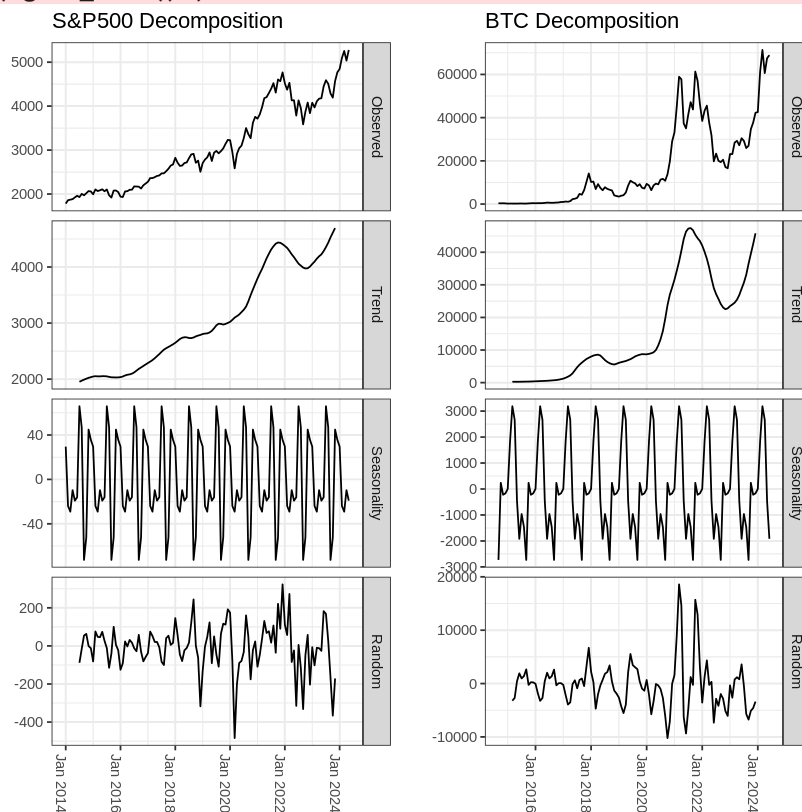
Warning message:

“Removed 6 rows containing missing values or values outside the scale range (`geom_line()`).”

Don't know how to automatically pick scale for object of type `<ts>`. Defaulting to continuous.

Warning message:

“Removed 6 rows containing missing values or values outside the scale range (`geom_line()`).”



The output displays four plots of our closing price data, which are:

- **Observed:** The original plot of the data.
- **Trend:** The long-term movements in the mean.
- **Seasonal:** The repetitive seasonal fluctuations in the data.
- **Random or Noise:** The irregular or random fluctuations not captured by the trend and seasonal components. When the random component is dominant in a data set, accurate forecasting becomes more challenging.

Both S&P 500 and BTC show an upward trend over the period analyzed, but the trend in BTC is more volatile and exhibits sharper peaks and troughs. Particullary, for the S&P500, we can observe a clear upward trend over time, indicating a long-term growth pattern while BTC has more erratic seasonality and higher randomness, pointing to greater unpredictability. The random component for BTC is significantly more volatile than that of the S&P 500, aligning with the general perception of BTC as a highly speculative and volatile asset compared to the relatively stable S&P 500 index.

4. Model

Data Setting

For the following analysis, we will be using weekly stock prices to reduce computational requirements and processing time. Our goal is to predict the stock prices for the next 50 weeks.

AutoRegressive Integrated Moving Average (ARIMA)

Autoregressive Model (AR)

- In a multiple regression model, we forecast the variable of interest using a linear combination of predictors. In an autoregression model, we forecast the variable of interest using a linear combination of past values of the variable itself. The term "autoregression" indicates that it is a regression of the variable against itself.
- An autoregressive model of order p is represented as:

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \varepsilon_t$$

where ε_t is white noise. This is similar to a multiple regression but with lagged values of y_t as predictors. We refer to this as an **AR**(p) model, an autoregressive model of order p .

Moving Average Model (MA)

- A moving average model uses past forecast errors in a regression-like model:

$$y_t = c + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \cdots + \theta_q \varepsilon_{t-q}$$

where ε_t is white noise. We refer to this as an **MA**(q) model, a moving average model of order q . However, we do not observe the values of ε_t , so it is not a regression in the usual sense.

ARIMA (AutoRegressive Integrated Moving Average)

- ARIMA combines differencing with autoregression and a moving average model.

- An ARIMA(p, d, q) model is represented as:

$$y'_t = c + \phi_1 y'_{t-1} + \cdots + \phi_p y'_{t-p} + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q} + \varepsilon_t$$

where y'_t is the differenced series (it may have been differenced more than once).

- p: order of the autoregressive part
- d: degree of first differencing involved
- q: order of the moving average part

```
In [ ]: # Data Splitting
btc_price <- to.weekly(btc_price)
sp_price <- to.weekly(sp_price)
n <- 50

btc_train <- head(Cl(btc_price), length(Cl(btc_price))-n)
btc_test  <- tail(Cl(btc_price), n)

sp_train <- head(Cl(sp_price), length(Cl(sp_price))-n)
sp_test  <- tail(Cl(sp_price), n)

sp_len <- length(sp_train)
btc_len <- length(btc_train)
```

```
In [ ]: arima_btc <- auto.arima(btc_train, trace=T, allowdrift = FALSE)
```

Fitting models using approximations to speed things up...

```
ARIMA(2,1,2)           : 8270.241
ARIMA(0,1,0)           : 8284.204
ARIMA(1,1,0)           : 8280.503
ARIMA(0,1,1)           : 8277.139
ARIMA(1,1,2)           : 8277.422
ARIMA(2,1,1)           : 8276.118
ARIMA(3,1,2)           : 8271.845
ARIMA(2,1,3)           : 8270.843
ARIMA(1,1,1)           : 8278.106
ARIMA(1,1,3)           : 8267.085
ARIMA(0,1,3)           : 8275.087
ARIMA(1,1,4)           : 8267.561
ARIMA(0,1,2)           : 8274.721
ARIMA(0,1,4)           : 8266.286
ARIMA(0,1,5)           : 8267.89
ARIMA(1,1,5)           : 8269.226
```

Now re-fitting the best model(s) without approximations...

```
ARIMA(0,1,4)           : 8281.744
```

Best model: ARIMA(0,1,4)

```
In [ ]: arima_sp <- auto.arima(sp_train, trace=T, allowdrift = FALSE)
```

Fitting models using approximations to speed things up...

```
ARIMA(2,1,2)           : 5630.801
ARIMA(0,1,0)           : 5628.865
ARIMA(1,1,0)           : 5629.729
ARIMA(0,1,1)           : 5628.854
ARIMA(1,1,1)           : 5631.609
ARIMA(0,1,2)           : 5630.442
ARIMA(1,1,2)           : 5633.431
```

Now re-fitting the best model(s) without approximations...

```
ARIMA(0,1,1)           : 5637.451
```

Best model: ARIMA(0,1,1)

```
In [ ]: d1 <- data.frame(c(fitted(arima_sp), # fitted and predicted
                           predict(arima_sp, n.ahead = n)$pred),
                           Cl(sp_price),
                           time(sp_price)
                           )
names(d1) <- c("Fitted", "Actual", "Date")

# MAPE <- filter(d1, year(Date)>2021) %>% summarise(MAPE=mean(abs(Actual-Fitted)/Actual))
MAPE <- tail(d1, n) %>% summarise(MAPE=mean(abs(Actual-Fitted)/Actual))

### Plot actual versus predicted
options(repr.plot.width = 20, repr.plot.height = 10)

plot1 <- ggplot(data=d1, aes(x=Date)) +
  geom_line(aes(y=Actual, colour = "Actual"), size=1.2) +
  geom_line(aes(y=Fitted, colour = "Fitted"), size=1.2, linetype=2) +
  theme_bw() + theme(legend.title = element_blank()) +
  ylab("") + xlab("") +
  geom_vline(xintercept=as.numeric(d1[sp_len,]$Date), linetype=2) +
  ggtitle(paste0("ARIMA_SP -- Holdout MAPE = ", round(100*MAPE,2), "%")) +
  theme(axis.text.x=element_text(angle = -90, hjust = 0))

d1 <- data.frame(c(fitted(arima_btc), # fitted and predicted
                   predict(arima_btc, n.ahead = n)$pred),
                   Cl(btc_price),
                   time(btc_price)
                   )
names(d1) <- c("Fitted", "Actual", "Date")

# MAPE <- filter(d1, year(Date)>2021) %>% summarise(MAPE=mean(abs(Actual-Fitted)/Actual))
MAPE <- tail(d1, n) %>% summarise(MAPE=mean(abs(Actual-Fitted)/Actual))

### Plot actual versus predicted
options(repr.plot.width = 20, repr.plot.height = 10)

plot2 <- ggplot(data=d1, aes(x=Date)) +
  geom_line(aes(y=Actual, colour = "Actual"), size=1.2) +
  geom_line(aes(y=Fitted, colour = "Fitted"), size=1.2, linetype=2) +
  theme_bw() + theme(legend.title = element_blank()) +
```

```


ylab("") + xlab("") +
geom_vline(xintercept=as.numeric(d1[btc_len,]$Date), linetype=2) +
ggtitle(paste0("ARIMA_BTC -- Holdout MAPE = ", round(100*MAPE,2), "%")) +
theme(axis.text.x=element_text(angle = -90, hjust = 0))

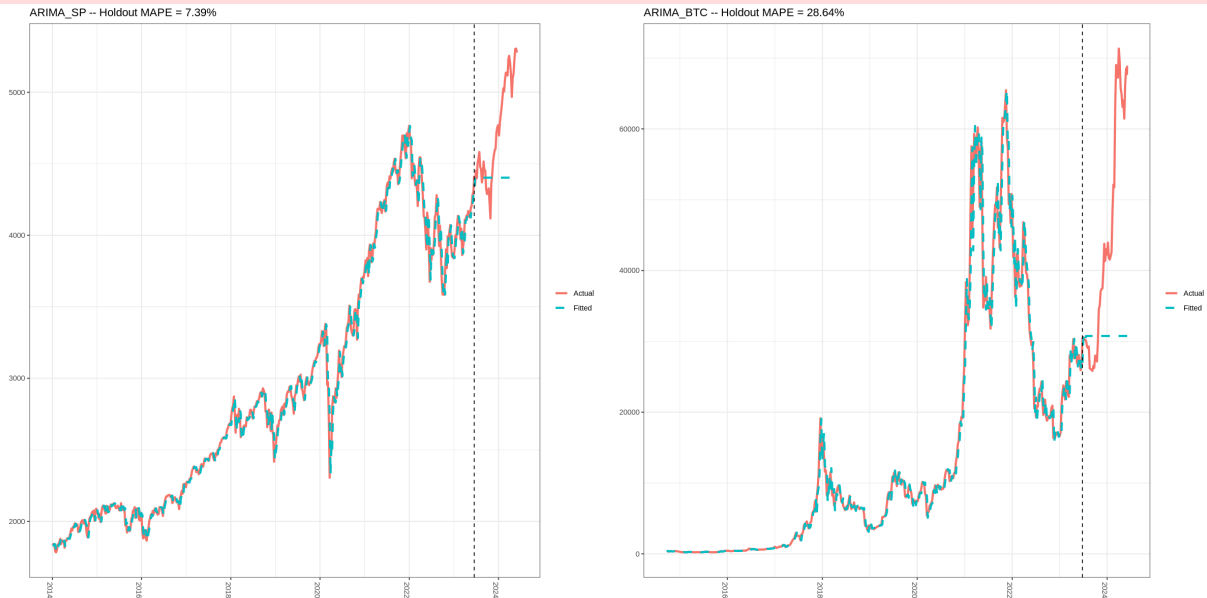
grid.arrange(plot1, plot2, ncol=2)

```

Warning message:

“Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.

 Please use `linewidth` instead.”



Mean Absolute Percentage Error (MAPE)

MAPE measures the difference of forecast errors and divides it by the actual observation value. It is scale-independent, thus can be used to compare a model's performance across different datasets.

Model Performance

The ARIMA model performs better for the S&P 500 index with a lower MAPE, indicating higher predictive accuracy for this more stable and less volatile time series. The higher MAPE for Bitcoin highlights the challenges in modeling highly volatile and speculative assets using ARIMA.

In-sample Fit vs. Holdout Fit:

Both models perform reasonably well in-sample, meaning they can capture the historical patterns and trends. However, the holdout period reveals weaknesses, suggesting the models are not generalizing well to unseen data. This discrepancy between in-sample and holdout performance is critical, as it highlights potential overfitting to historical data without capturing underlying patterns robustly enough for future predictions.

4.2. Bayesian Structural Time Series (BSTS)

BSTS is a more transparent approach because its representation does not rely on differencing, lags, and moving averages. You can visually inspect the underlying components of the model. It handles uncertainty in a better way because you can quantify the posterior uncertainty of the individual components, control the variance of the components, and impose prior beliefs on the model. Additionally, any ARIMA model can be recast as a structural model.

Generally, we can write a Bayesian structural model as:

$$\begin{aligned} Y_t &= \mu_t + x_t\beta + e_t, e_t \sim N(0, \sigma_e^2) \\ \mu_{t+1} &= \mu_t + b_t + \nu_t, \nu_t \sim N(0, \sigma_\nu^2) \\ b_t &= b_{t-1} + w, w \sim N(0, \sigma^2) \end{aligned}$$

Where:

- x_t : a set of regressors
- μ_t : the local level term, defines how the latent state evolves over time and is often referred to as the unobserved trend.

Note that the regressor coefficients, seasonality, and trend are estimated simultaneously, which helps avoid strange coefficient estimates due to spurious relationships (similar in spirit to Granger causality). Additionally, due to the Bayesian nature of the model, we can shrink the elements of β to promote sparsity or specify outside priors for the means in case we're not able to get meaningful estimates from the historical data.

```
In [ ]: btc_price <- to.weekly(btc_price)
        sp_price <- to.weekly(sp_price)

        n <- 50
        num_data_btc <- length(btc_price)/6
        num_tr_btc <- num_data_btc - n

        # Splitting the data
        btc_price <- zoo(btc_price)
        btc_train <- btc_price[1:num_tr_btc]
        btc_test  <- btc_price[(num_tr_btc+1):num_data_btc]

        # Number of period we want to forecast
        n <- 50
        num_data_sp <- length(sp_price)/6
        num_tr_sp <- num_data_sp - n

        # Splitting the data
        sp_price <- zoo(sp_price)
        sp_train <- sp_price[1:num_tr_sp]
        sp_test  <- sp_price[(num_tr_sp+1):num_data_sp]
```

4.2.1. Fitting & Predict

```

In [ ]: ss <- AddLocalLinearTrend(list(), btc_train$btc_price.Close)
ss <- AddSeasonal(ss, btc_train$btc_price.Close, nseasons = 52)
bsts.reg.btc <- bsts(btc_train$btc_price.Close ~ ., state.specification = ss,
                    data = btc_train, niter = 1000, seed=100, expected.model.size=1, p

### Get a suggested number of burn-ins
burn <- SuggestBurn(0.1, bsts.reg.btc)

### Predict
p <- predict.bsts(bsts.reg.btc, newdata=btc_test, burn = burn, quantiles = c(.025,

### Actual versus predicted
d2 <- data.frame(
  c(as.numeric(-colMeans(bsts.reg.btc$one.step.prediction.errors[-(1:burn),])+Cl(
    as.numeric(p$mean))),
  # actual data and dates
  as.numeric(zoo(Cl(btc_price))),
  time(btc_price))
names(d2) <- c("Fitted", "Actual", "Date")

### MAPE (mean absolute percentage error)
MAPE <- tail(d2, n) %>% summarise(MAPE=mean(abs(Actual-Fitted)/Actual))

### 95% forecast credible interval
posterior.interval <- cbind.data.frame(
  as.numeric(p$interval[1,]),
  as.numeric(p$interval[2,]),
  tail(d2, 50)$Date)
names(posterior.interval) <- c("LL", "UL", "Date")

### Join intervals to the forecast
d3 <- left_join(d2, posterior.interval, by="Date")

### Plot actual versus predicted with credible intervals for the holdout period
options(repr.plot.width = 20, repr.plot.height = 10)
plot1 <- ggplot(data=d3, aes(x=Date)) +
  geom_line(aes(y=Actual, colour = "Actual"), size=1.2) +
  geom_line(aes(y=Fitted, colour = "Fitted"), size=1.2, linetype=2) +
  theme_bw() + theme(legend.title = element_blank()) + ylab("") + xlab("") +
  geom_vline(xintercept=as.numeric(d3[btc_len,]$Date), linetype=2) +
  geom_ribbon(aes(ymin=LL, ymax=UL), fill="grey", alpha=0.5) +
  ggtitle(paste0("BSTS_BTC -- Holdout MAPE = ", round(100*MAPE,2), "%")) +
  theme(axis.text.x=element_text(angle = -90, hjust = 0))

### Get the components
components.withreg <- cbind.data.frame(
  colMeans(bsts.reg.btc$state.contributions[-(1:burn),"trend",]),
  colMeans(bsts.reg.btc$state.contributions[-(1:burn),"seasonal.52.1",]),
  colMeans(bsts.reg.btc$state.contributions[-(1:burn),"regression",]),
  as.Date(time(btc_train)))
names(components.withreg) <- c("Trend", "Seasonality", "Regression", "Date")
components.withreg <- melt(components.withreg, id.vars="Date")
names(components.withreg) <- c("Date", "Component", "Value")

plot3 <- ggplot(data=components.withreg, aes(x=Date, y=Value)) + geom_line() +

```

```

theme_bw() + theme(legend.title = element_blank()) + ylab("") + xlab("") +
facet_grid(Component ~ ., scales="free") + guides(colour="none") +
theme(axis.text.x=element_text(angle = -90, hjust = 0))

###
###

ss <- AddLocalLinearTrend(list(), sp_train$sp_price.Close)
ss <- AddSeasonal(ss, sp_train$sp_price.Close, nseasons = 52)
bsts.reg.sp <- bsts(sp_train$sp_price.Close ~ ., state.specification = ss,
                  data = sp_train, niter = 1000, seed=100, expected.model.size=1, pi

### Get a suggested number of burn-ins
burn <- SuggestBurn(0.1, bsts.reg.sp)

### Predict
p <- predict.bsts(bsts.reg.sp, newdata=sp_test, burn = burn, quantiles = c(.025, .9

### Actual versus predicted
d2 <- data.frame(
  c(as.numeric(-colMeans(bsts.reg.sp$one.step.prediction.errors[-(1:burn),])+Cl(s
    as.numeric(p$mean)),
  # actual data and dates
  as.numeric(zoo(Cl(sp_price))),
  time(sp_price))
names(d2) <- c("Fitted", "Actual", "Date")

### MAPE (mean absolute percentage error)
MAPE <- tail(d2, n) %>% summarise(MAPE=mean(abs(Actual-Fitted)/Actual))

### 95% forecast credible interval
posterior.interval <- cbind.data.frame(
  as.numeric(p$interval[1,]),
  as.numeric(p$interval[2,]),
  tail(d2, 50)$Date)
names(posterior.interval) <- c("LL", "UL", "Date")

### Join intervals to the forecast
d3 <- left_join(d2, posterior.interval, by="Date")

### Plot actual versus predicted with credible intervals for the holdout period
options(repr.plot.width = 20, repr.plot.height = 10)
plot2 <- ggplot(data=d3, aes(x=Date)) +
  geom_line(aes(y=Actual, colour = "Actual"), size=1.2) +
  geom_line(aes(y=Fitted, colour = "Fitted"), size=1.2, linetype=2) +
  theme_bw() + theme(legend.title = element_blank()) + ylab("") + xlab("") +
  geom_vline(xintercept=as.numeric(d3[sp_len,]$Date), linetype=2) +
  geom_ribbon(aes(ymin=LL, ymax=UL), fill="grey", alpha=0.5) +
  ggtitle(paste0("BSTS_SP -- Holdout MAPE = ", round(100*MAPE,2), "%")) +
  theme(axis.text.x=element_text(angle = -90, hjust = 0))

### Get the components
components.withreg <- cbind.data.frame(
  colMeans(bsts.reg.sp$state.contributions[-(1:burn),"trend",]),
  colMeans(bsts.reg.sp$state.contributions[-(1:burn),"seasonal.52.1",]),
  colMeans(bsts.reg.sp$state.contributions[-(1:burn),"regression",]),

```

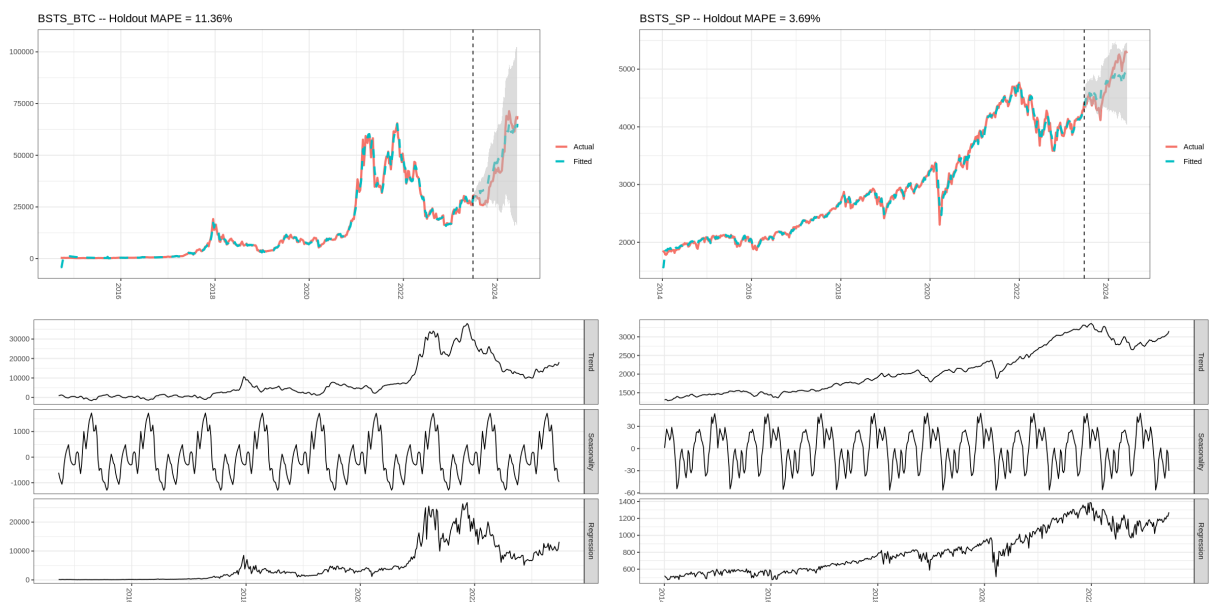
```

as.Date(time(sp_train)))
names(components.withreg) <- c("Trend", "Seasonality", "Regression", "Date")
components.withreg <- melt(components.withreg, id.vars="Date")
names(components.withreg) <- c("Date", "Component", "Value")

plot4 <- ggplot(data=components.withreg, aes(x=Date, y=Value)) + geom_line() +
  theme_bw() + theme(legend.title = element_blank()) + ylab("") + xlab("") +
  facet_grid(Component ~ ., scales="free") + guides(colour="none") +
  theme(axis.text.x=element_text(angle = -90, hjust = 0))

grid.arrange(plot1, plot2, plot3, plot4, ncol=2, nrow=2)

```



Model Performance

The BSTS model significantly outperforms the ARIMA model for both Bitcoin and the S&P 500. Specially, the lower MAPE values when compare with ARIMA for both assets indicate improved accuracy and reliability in the predictions. However, the holdout MAPE for Bitcoin (11.36%) is higher than that for the S&P 500 (3.63%), which is expected given the different nature and volatility of these assets.

In-sample Fit vs. Holdout Fit:

Both models demonstrate strong in-sample fits, indicating they effectively learn from historical data. The holdout fit shows the models' predictive power, with the S&P 500 model performing exceptionally well, while the Bitcoin model, although less precise, still shows good predictive capability considering the high volatility of the asset. The widening of confidence intervals in the Bitcoin model during the holdout period also suggests the model's recognition of increased uncertainty and volatility.

4.2.2. Bayesian Variable Selection

Spike-and-Slab Prior

- A spike-and-slab prior is a prior distribution that induces sparsity in the posterior. Unlike the Lasso prior (Laplace), which only yields MAP estimates at zero but nonzero posterior, a spike-and-slab prior is a natural way to express a prior belief that most of the regression coefficients are exactly zero by explicitly assigning positive probability of being zero on the spike part of the prior. This directly induces sparsity in the posterior due to conjugacy. Thus, the prediction is computed from the posterior predictive distribution, which is independent of the corresponding variables.
- A spike-and-slab prior can be written as:

$$p(\beta, \gamma, \sigma_\epsilon^2) = p(\beta_\gamma | \gamma, \sigma_\epsilon^2) p(\sigma_\epsilon^2 | \gamma) p(\gamma)$$

- The last factor is the "spike" part that governs the probability of a given variable being chosen for the model (i.e., having a non-zero coefficient).
- The other factors are modeled to be the "slab" - The part that shrinks the coefficients toward prior expectations (often zero).

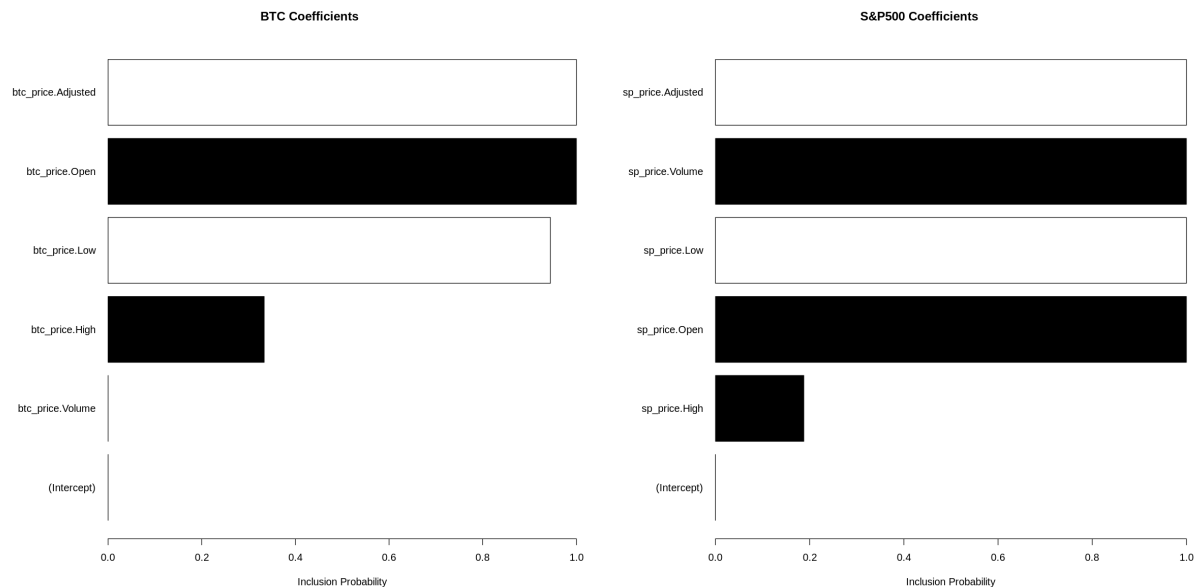
BSTS Model with Spike-and-Slab Priors

The BSTS model imposes a *spike-and-slab prior* on the regressor coefficients, which is a powerful way of reducing a large set of correlated variables. It allows the model to incorporate uncertainties of the coefficient estimates when producing the credible interval for the forecasts.

Prior Parameter Tuning

We analyze which regression coefficients the previous BSTS models use to predict future prices and try to change the prior parameters (from default values) based on our practical belief to improve performance.

```
In [ ]: par(mfrow = c(1, 2))
plot(bsts.reg.btc, "coef")
title("BTC Coefficients")
plot(bsts.reg.sp, "coef")
title("S&P500 Coefficients")
```

Through visualizing the inclusion probabilities of all regression coefficients in the previous BSTS models, we observe that both models utilize the Adjusted, High, and Low prices as predictors. However, these variables are highly correlated in reality. In technical analysis, volume analysis is one of the main indicators representing market strength. Therefore, we may want to include volume as a predictor in the BTC model as well, given the superior performance of the S&P 500 model. Thus, we expected to increase the weight for the volume while reduce the correlation among price-related predictors' weights.

To explicitly impose our prior belief that volume should be included, we use a spike-and-slab prior with the spike parameter set to 1 for the volume coefficient. Additionally, to reduce the correlation among price-related coefficients, we aim to use only the Adjusted price as our main predictor while restricting the others. This is achieved by setting the expected model size to 2 (Volume + Adjusted_price).

```
In [ ]: prior.spikes <- c(0.1,0.1,0.1,0.1,1.0,1.0)
prior.mean  <- c(0 ,0 ,0 ,0 ,0.0,0.5)

### Set up the priors
prior <- SpikeSlabPrior(x=model.matrix(btc_price.Close ~ ., data=btc_train),
                        y=btc_train$btc_price.Close,
                        prior.information.weight = 200,
                        prior.inclusion.proBABILITIES = prior.spikes,
                        optional.coefficient.estimate = prior.mean)

ss <- AddLocalLinearTrend(list(), btc_train$btc_price.Close)
ss <- AddSeasonal(ss, btc_train$btc_price.Close, nseasons = 52)
bsts.reg.btc <- bstS(btc_train$btc_price.Close ~ ., state.specification = ss, prior
                    data = btc_train, niter = 1000, seed=100, expected.model.size=2, p

### Get a suggested number of burn-ins
burn <- SuggestBurn(0.1, bstS.reg.btc)

### Predict
```

```

p <- predict.bsts(bsts.reg.btc, newdata=btc_test, burn = burn, quantiles = c(.025,

### Actual versus predicted
d2 <- data.frame(
  c(as.numeric(-colMeans(bsts.reg.btc$one.step.prediction.errors[-(1:burn),])+Cl(
    as.numeric(p$mean))),
  # actual data and dates
  as.numeric(zoo(Cl(btc_price))),
  time(btc_price))
names(d2) <- c("Fitted", "Actual", "Date")

### MAPE (mean absolute percentage error)
MAPE <- tail(d2, n) %>% summarise(MAPE=mean(abs(Actual-Fitted)/Actual))

### 95% forecast credible interval
posterior.interval <- cbind.data.frame(
  as.numeric(p$interval[1,]),
  as.numeric(p$interval[2,]),
  tail(d2, 50)$Date)
names(posterior.interval) <- c("LL", "UL", "Date")

### Join intervals to the forecast
d3 <- left_join(d2, posterior.interval, by="Date")

### Plot actual versus predicted with credible intervals for the holdout period
options(repr.plot.width = 20, repr.plot.height = 10)
plot1 <- ggplot(data=d3, aes(x=Date)) +
  geom_line(aes(y=Actual, colour = "Actual"), size=1.2) +
  geom_line(aes(y=Fitted, colour = "Fitted"), size=1.2, linetype=2) +
  theme_bw() + theme(legend.title = element_blank()) + ylab("") + xlab("") +
  geom_vline(xintercept=as.numeric(d3[btc_len,]$Date), linetype=2) +
  geom_ribbon(aes(ymin=LL, ymax=UL), fill="grey", alpha=0.5) +
  ggtitle(paste0("BSTS_BTC -- Holdout MAPE = ", round(100*MAPE,2), "%")) +
  theme(axis.text.x=element_text(angle = -90, hjust = 0))

### Get the components
components.withreg <- cbind.data.frame(
  colMeans(bsts.reg.btc$state.contributions[-(1:burn),"trend",]),
  colMeans(bsts.reg.btc$state.contributions[-(1:burn),"seasonal.52.1",]),
  colMeans(bsts.reg.btc$state.contributions[-(1:burn),"regression",]),
  as.Date(time(btc_train)))
names(components.withreg) <- c("Trend", "Seasonality", "Regression", "Date")
components.withreg <- melt(components.withreg, id.vars="Date")
names(components.withreg) <- c("Date", "Component", "Value")

plot3 <- ggplot(data=components.withreg, aes(x=Date, y=Value)) + geom_line() +
  theme_bw() + theme(legend.title = element_blank()) + ylab("") + xlab("") +
  facet_grid(Component ~ ., scales="free") + guides(colour="none") +
  theme(axis.text.x=element_text(angle = -90, hjust = 0))

#####
#####
#####
prior.spikes <- c(0.1,0.1,0.1,0.1,1.0,1.0)
prior.mean <- c(0 ,0 ,0 ,0 ,0.0,0.5)

```

```

#### Set up the priors
prior <- SpikeSlabPrior(x=model.matrix(sp_train$sp_price.Close ~ ., data=sp_train),
                        y=sp_train$sp_price.Close,
                        prior.information.weight = 200,
                        prior.inclusion.proBABILITIES = prior.spikes,
                        optional.coefficient.estimate = prior.mean)

ss <- AddLocalLinearTrend(list(), sp_train$sp_price.Close)
ss <- AddSeasonal(ss, sp_train$sp_price.Close, nseasons = 52)
bsts.reg.sp <- bsts(sp_train$sp_price.Close ~ ., state.specification = ss, prior=prior,
                  data = sp_train, niter = 1000, seed=100, expected.model.size=2, pi

#### Get a suggested number of burn-ins
burn <- SuggestBurn(0.1, bsts.reg.sp)

#### Predict
p <- predict.bsts(bsts.reg.sp, newdata=sp_test, burn = burn, quantiles = c(.025, .975))

#### Actual versus predicted
d2 <- data.frame(
  c(as.numeric(-colMeans(bsts.reg.sp$one.step.prediction.errors[-(1:burn),])+Cl(s
    as.numeric(p$mean)),
  # actual data and dates
  as.numeric(zoo(Cl(sp_price)))),
  time(sp_price))
names(d2) <- c("Fitted", "Actual", "Date")

#### MAPE (mean absolute percentage error)
MAPE <- tail(d2, n) %>% summarise(MAPE=mean(abs(Actual-Fitted)/Actual))

#### 95% forecast credible interval
posterior.interval <- cbind.data.frame(
  as.numeric(p$interval[1,]),
  as.numeric(p$interval[2,]),
  tail(d2, 50)$Date)
names(posterior.interval) <- c("LL", "UL", "Date")

#### Join intervals to the forecast
d3 <- left_join(d2, posterior.interval, by="Date")

#### Plot actual versus predicted with credible intervals for the holdout period
options(repr.plot.width = 20, repr.plot.height = 10)
plot2 <- ggplot(data=d3, aes(x=Date)) +
  geom_line(aes(y=Actual, colour = "Actual"), size=1.2) +
  geom_line(aes(y=Fitted, colour = "Fitted"), size=1.2, linetype=2) +
  theme_bw() + theme(legend.title = element_blank()) + ylab("") + xlab("") +
  geom_vline(xintercept=as.numeric(d3[sp_len,]$Date), linetype=2) +
  geom_ribbon(aes(ymin=LL, ymax=UL), fill="grey", alpha=0.5) +
  ggtitle(paste0("BSTS_SP -- Holdout MAPE = ", round(100*MAPE,2), "%")) +
  theme(axis.text.x=element_text(angle = -90, hjust = 0))

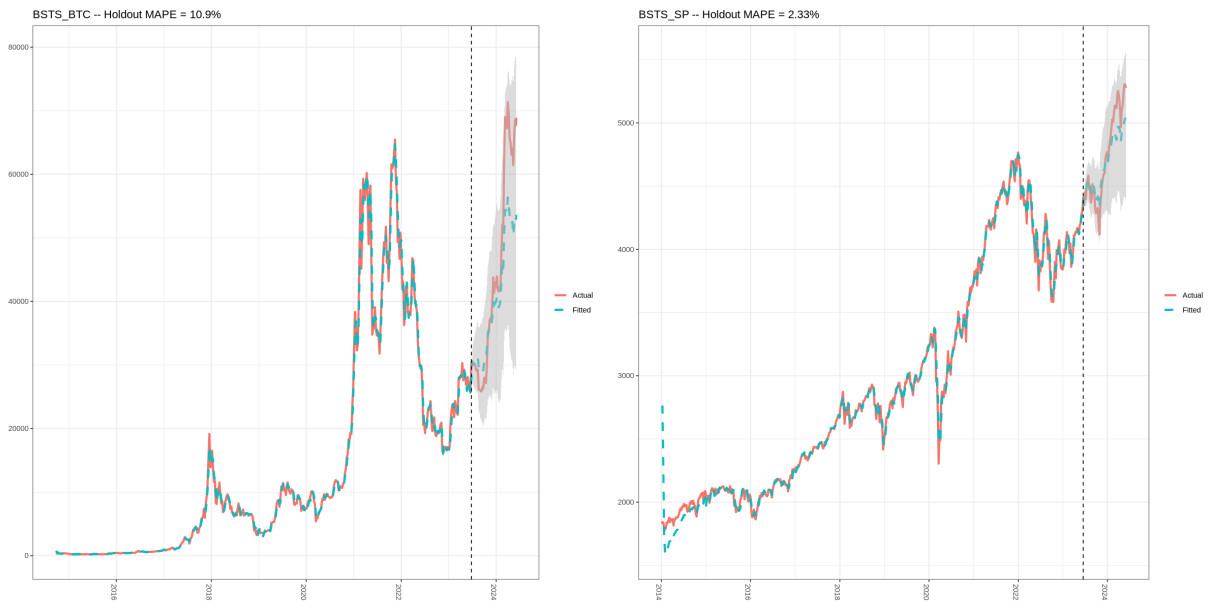
#### Get the components
components.withreg <- cbind.data.frame(
  colMeans(bsts.reg.sp$state.contributions[-(1:burn),"trend",]),
  colMeans(bsts.reg.sp$state.contributions[-(1:burn),"seasonal.52.1",]),

```

```
colMeans(bsts.reg.sp$state.contributions[-(1:burn),"regression",]),
as.Date(time(sp_train)))
names(components.withreg) <- c("Trend", "Seasonality", "Regression", "Date")
components.withreg <- melt(components.withreg, id.vars="Date")
names(components.withreg) <- c("Date", "Component", "Value")

plot4 <- ggplot(data=components.withreg, aes(x=Date, y=Value)) + geom_line() +
  theme_bw() + theme(legend.title = element_blank()) + ylab("") + xlab("") +
  facet_grid(Component ~ ., scales="free") + guides(colour="none") +
  theme(axis.text.x=element_text(angle = -90, hjust = 0))

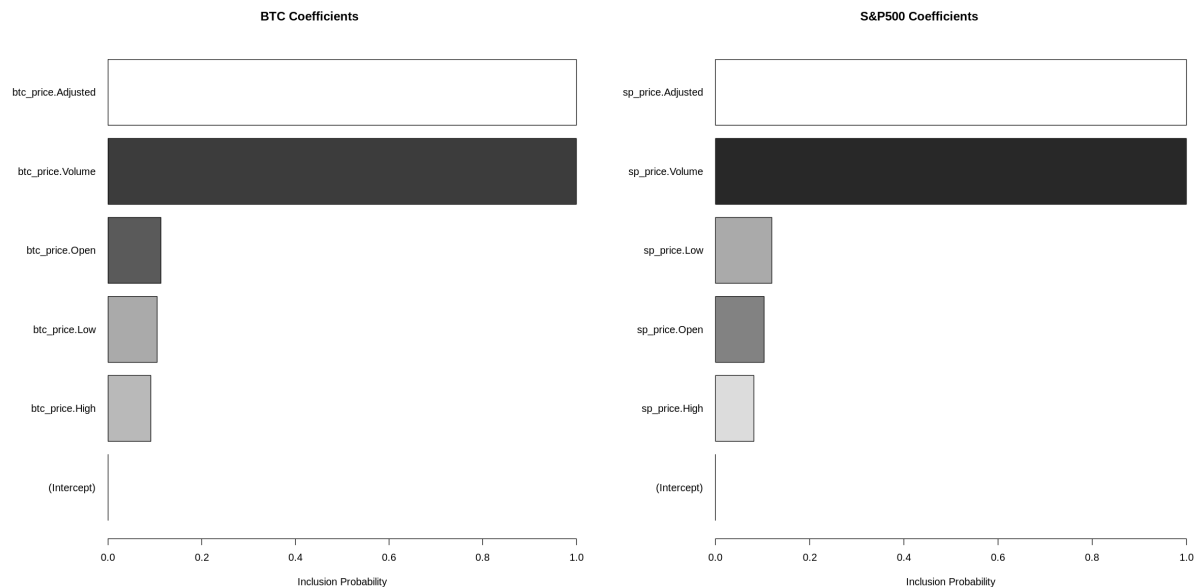
grid.arrange(plot1, plot2, ncol=2)
```



Model Performance

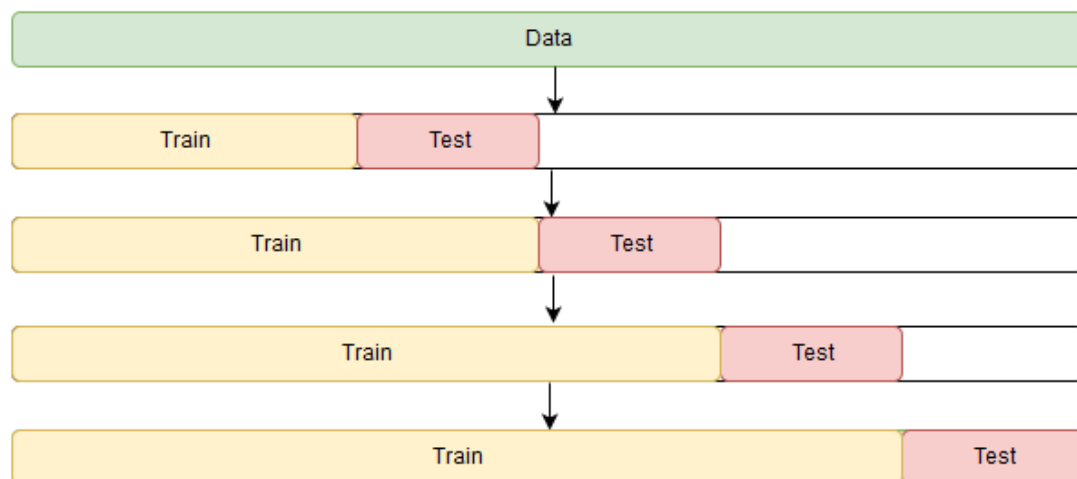
The BSTS model with prior outperforms previous BSTS model for both Bitcoin and the S&P 500. Specially, the BSTS model with our prior performs slightly better for both BTC and S&P500 with MAPE is 10.9% and 2.33% comparing to 11.26% and 3.69%, respectively. These improvements, though modest, demonstrate the effectiveness of the prior assumptions in enhancing the model's forecasting accuracy. The priors, which aimed to increase the weight of trading volume as a predictor variable while reducing the correlation among price-related predictors, appear to have positively impacted the model's ability to capture the underlying dynamics of these time series.

```
In [ ]: par(mfrow = c(1, 2))
plot(bsts.reg.btc, "coef") #, x="BTC Coefficients")
title("BTC Coefficients")
plot(bsts.reg.sp, "coef")#, xlab="S&P Coefficients")
title("S&P500 Coefficients")
```



4.2.3. Time series cross-validation

Forward chaining involves cross-validation on a rolling basis. The process starts with a small subset of data for training purposes, followed by forecasting for the later data points and then checking the accuracy for the forecasted data points. The same forecasted data points are then included as part of the next training dataset, and subsequent data points are forecasted.



```

In [ ]: bsts.cv.loop <- function(data,
                                number_of_folds,
                                nseasons, model_size,
                                horizon=50,
                                niter=1000,
                                seed=100,
                                verbose=FALSE,
                                debug=FALSE) {

  mape_v <- c()

```

```

for (fold in 1:number_of_folds) {
  # construct data_train/data_test
  l <- length(data)/6 - fold*horizon
  # if (debug) print(l)
  data <- zoo(data)
  data_train <- data[1:l]
  data_test <- data[(l+1):(l+horizon)]

  prior.spikes <- c(0.1,0.1,0.1,0.1,1.0,1.0)
  prior.mean <- c(0 ,0 ,0 ,0 ,0.0,0.5)

  ### Set up the priors
  prior <- SpikeSlabPrior(x=model.matrix(Close ~ ., data=data_train),
                          y=Cl(data_train),
                          prior.information.weight = 200,
                          prior.inclusion.proBABILITIES = prior.spikes,
                          optional.coefficient.estimate = prior.mean)

  ss <- AddLocalLinearTrend(list(), Cl(data_train))
  ss <- AddSeasonal(ss, Cl(data_train), nseasons=nseasons)
  model <- bsts(Close~., data=data_train,
                state.specification=ss, expected.model.size=model_size,
                niter=niter,
                seed=seed, prior=prior,
                ping=0)
  burn <- SuggestBurn(0.1, model)

  ### Predict
  pred <- predict.bsts(model, newdata=data_test, burn = burn, quantiles = c(.025,
  # print(length(data_test))
  # print(length(p$mean))

  ### Actual versus predicted
  d2 <- data.frame(
    c(as.numeric(-colMeans(model$one.step.prediction.errors[-(1:burn),])+Cl(dat
    as.numeric(pred$mean)),
    # actual data and dates
    as.numeric(zoo(Cl(data[1:(l+horizon)]))),
    time(data[1:(l+horizon)]))
  names(d2) <- c("Fitted", "Actual", "Date")

  ### MAPE (mean absolute percentage error)
  MAPE <- tail(d2, n) %>% summarise(MAPE=mean(abs(Actual-Fitted)/Actual))

  # evaluation
  mape <- round(MAPE[1,] * 100, 2)
  mape_v <- c(mape_v, mape)
  if (verbose) print(paste0("fold ", fold, ": mape ", mape))
}
return(data.frame(mape=mape_v))
}

```

```

In [ ]: btc_price <- getSymbols('BTC-USD', auto.assign=FALSE, from="2014-01-01")
btc_price <- to.weekly(btc_price)
names(btc_price) <- c('Open', 'High', 'Low', 'Close', 'Volume', 'Adjusted')

```

```

res_btc <- c()
for (ns in c(5, 10, 20)) {
  for (sz in c(1,2,3)) {
    res <- bst.cv.loop(btc_price, 5, ns, sz)
    res_row <- data.frame(ns=ns,
                          sz=sz,
                          mean_mape=mean(res$mape))
    res_btc <- rbind(res_btc, res_row)
  }
}
print("BTC CV scores:")
print(res_btc)
b <- res_btc[which.min(res_btc$mean_mape),]
cat(paste0("Best model for BTC has ", b$ns, " seasons and ", b$sz, " regression coe

sp_price <- getSymbols('^GSPC', auto.assign=FALSE, from="2014-01-01")
sp_price <- to.weekly(sp_price)
names(sp_price) <- c('Open', 'High', 'Low', 'Close', 'Volume', 'Adjusted')

res_sp <- c()
for (ns in c(5, 10, 20)) {
  for (sz in c(1,2,3)) {
    res <- bst.cv.loop(sp_price, 5, ns, sz)
    res_row <- data.frame(ns=ns,
                          sz=sz,
                          mean_mape=mean(res$mape))
    res_sp <- rbind(res_sp, res_row)
  }
}
cat('\n')
cat("S&P500 CV scores:")
print(res_sp)
b <- res_sp[which.min(res_sp$mean_mape),]
cat(paste0("Best model for S&P500 has ", b$ns, " seasons and ", b$sz, " regression

```

Warning message:

“BTC-USD contains missing values. Some functions will not work if objects contain missing values in the middle of the series. Consider using na.omit(), na.approx(), na.fill(), etc to remove or replace them.”

Warning message in to.period(x, "weeks", name = name, ...):

“missing values removed from data”

[1] "BTC CV scores:"

	ns	sz	mean_mape
1	5	1	24.336
2	5	2	24.344
3	5	3	24.050
4	10	1	22.240
5	10	2	22.416
6	10	3	21.794
7	20	1	21.798
8	20	2	22.196
9	20	3	21.714

Best model for BTC has 20 seasons and 3 regression coefficients with MAPE = 21.714

5. Discussion

5.1. Limitations

Both ARIMA and BSTS models offer unique advantages and face distinct limitations when applied to financial time series forecasting. The ARIMA model is valued for its simplicity and ease of implementation, making it a popular choice for analysts. Its ability to model the linear relationships in stable datasets, such as the S&P 500, to some extent, providing a solid baseline for predictive performance. However, ARIMA models are constrained by their assumptions of linearity and stationarity, which limit their effectiveness in modeling complex and volatile markets like Bitcoin. Moreover, the model's performance in the holdout period is poorer, reducing its predictive power when faced with new data. Furthermore, ARIMA models lack the capability to provide confidence intervals, which are crucial for understanding prediction uncertainty. On the other hand, the BSTS model excels in handling the intricate patterns inherent in financial data by incorporating trend, seasonality, and regression components. This adaptability allows BSTS models to maintain higher predictive accuracy, as evidenced by lower MAPE values, particularly in volatile markets. The incorporation of prior information, such as trading volume, enhances the model's robustness and flexibility. Moreover, BSTS models offer confidence intervals, providing a measure of prediction uncertainty essential for risk management. However, the complexity of BSTS models requires significant computational resources and expertise in Bayesian methods. Their performance is highly dependent on the quality of priors, and poor priors can bias the results.

5.2. Future Direction

The performance of forecasting models is fundamentally correlated with a deep understanding of the corresponding stock's dynamics. Notably, the more volatile the price action, the greater the uncertainty in the model's future predictions. This uncertainty amplifies significantly as the forecast horizon extends further into the future. There exist numerous technical indicators, either independent or derived from price data, that can be incorporated into the current model to enhance its predictive capabilities. One such approach is the CausalImpact R package, which implements a method for estimating the causal effect of a designed intervention on a time series. This package builds a Bayesian structural time series model based on multiple comparable control groups (or markets) and utilizes this model to project (or forecast) a series of baseline values for the time period after the event of interest. By incorporating this approach, the model's ability to account for causal factors and measure their impact on the target time series could be augmented, potentially improving its overall performance.

6. Conclusion

In this project, we aim to learn and address time-series-related problems through a case study on stock price analysis for Bitcoin (BTC) and the S&P 500 index. First, we conduct a time-series analysis on the price actions of each stock, allowing us to determine their representative characteristics. Subsequently, we construct an ARIMA (Autoregressive Integrated Moving Average) model as a baseline to predict future trends of the stock based on its historical price moving average and lagged values. We then build an explanatory model for each time series using the Bayesian Structural Time Series (BSTS) method. These models provide an actual understanding of the underlying generative distribution of the time series and effectively handle the uncertainty of price volatility. Finally, we significantly improve the BSTS models through Bayesian variable selection and forward-chaining cross-validation.

The results suggest that the output predictions and confidence intervals from these models can serve as informative indicators of the corresponding stock's future price movements. Specifically, ARIMA models are straightforward and effective for stable, linear datasets but fall short in capturing non-linear and volatile patterns, often leading to overfitting in the in-sample period and underperformance in the holdout period. BSTS models provide greater accuracy and adaptability at the cost of higher complexity and computational demands. The choice between these models should consider the specific data characteristics, computational resources, and the need for interpretability and uncertainty quantification.

A. References

- https://jasonlian.github.io/Rmarkdown/Tutorial_for_BSTS.html
- <https://otexts.com/fpp2/>
- <https://www.investopedia.com/>
- <https://sisifospage.tech/2017-10-30-forecasting-bsts.html>
- <https://www.rpubs.com/AurelliaChristie/time-series-and-stock-analysis>
- <https://www.unofficialgoogledatascience.com/2017/07/fitting-bayesian-structural-time-series.html>
- <https://towardsdatascience.com/forecasting-time-series-data-stock-price-analysis-324bcc520af5>
- <https://claude.ai>