Project2501

2023-04-28

## R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see <http://rmarkdown.rstudio.com>.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

library(dplyr)

##   
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':  
##   
## filter, lag

## The following objects are masked from 'package:base':  
##   
## intersect, setdiff, setequal, union

library(ggplot2)

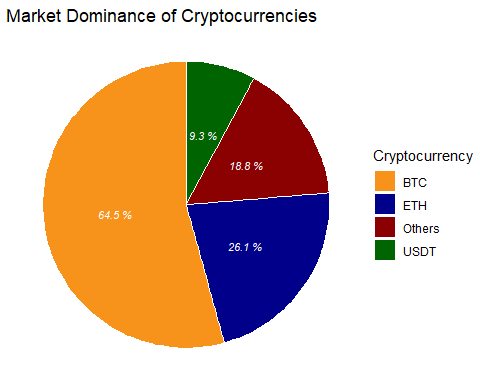
## Warning: package 'ggplot2' was built under R version 4.2.3

library(coinmarketcapr)

## Warning: package 'coinmarketcapr' was built under R version 4.2.3

library(purrr)  
library(ggthemes)  
  
# Set API key for CoinMarketCap  
apikey <- "ed1842ab-58c3-4cc1-a11c-8b3fa6e10b80"  
coinmarketcapr::setup(apikey)  
  
# Get top 20 cryptocurrencies by market cap  
crypto\_list <- get\_crypto\_listings(limit = 20)  
  
# Calculate market dominance percentages for the top 3 cryptocurrencies  
top3\_dominance <- crypto\_list %>%   
 slice(1:3) %>%   
 mutate(market\_cap\_percentage = USD\_market\_cap\_dominance / sum(USD\_market\_cap\_dominance) \* 100) %>%   
 select(symbol, market\_cap\_percentage)  
  
# Calculate market dominance percentages for all other cryptocurrencies  
others\_dominance <- crypto\_list %>%   
 slice(4:n()) %>%   
 summarize(market\_cap\_percentage = sum(USD\_market\_cap\_dominance) / sum(crypto\_list$USD\_market\_cap\_dominance) \* 100) %>%   
 mutate(symbol = "Others")  
  
# Combine the top 3 and "Others" data  
market\_dominance <- bind\_rows(top3\_dominance, others\_dominance)  
  
# Create a pie chart of the market dominance data  
ggplot(market\_dominance, aes(x = "", y = market\_cap\_percentage, fill = symbol)) +  
 geom\_bar(stat = "identity", width = 1, color = "white", size = 0.5) +  
 coord\_polar(theta = "y") +  
 theme\_void() +  
 scale\_fill\_manual(values = c("#F7931A", "darkblue", "darkred", "darkgreen")) +  
 labs(title = "Market Dominance of Cryptocurrencies",  
 fill = "Cryptocurrency",  
 x = NULL,  
 y = NULL) +  
 geom\_text(aes(label = paste(round(market\_cap\_percentage, 1), "%")),   
 position = position\_stack(vjust = 0.5),   
 color = "white", size = 3, fontface = "italic")

## 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.



## Including Plots

You can also embed plots, for example:

# Load crypto2 library  
library(crypto2)  
  
  
  
# Get historical data for BTC, ETH, DOGE and XRP  
#coins <- c("BTC", "ETH", "DOGE", "XRP")  
coins <- crypto\_list(only\_active=TRUE) |> filter(symbol %in% c("BTC", "ETH", "XRP", "DOGE"))  
coin\_hist <- crypto\_history(coins, limit=4, start\_date="20220101", end\_date="20230331", finalWait=FALSE) #15months

## ❯ Scraping historical crypto data

##

## ❯ Processing historical crypto data

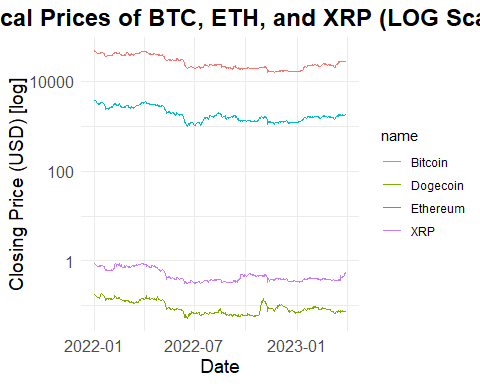
##

coin\_hist$date <- as.Date(coin\_hist$timestamp)  
  
  
  
# View data  
head(coin\_hist)

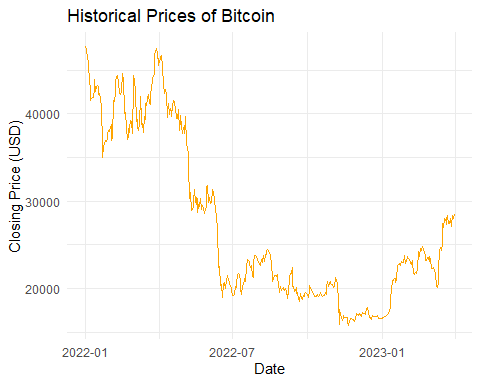
## # A tibble: 6 × 17  
## timestamp id slug name symbol ref\_cur open high low  
## <dttm> <int> <chr> <chr> <chr> <chr> <dbl> <dbl> <dbl>  
## 1 2022-01-01 23:59:59 1 bitcoin Bitcoin BTC USD 46312. 47827. 46288.  
## 2 2022-01-02 23:59:59 1 bitcoin Bitcoin BTC USD 47681. 47881. 46857.  
## 3 2022-01-03 23:59:59 1 bitcoin Bitcoin BTC USD 47344. 47511. 45836.  
## 4 2022-01-04 23:59:59 1 bitcoin Bitcoin BTC USD 46459. 47407. 45752.  
## 5 2022-01-05 23:59:59 1 bitcoin Bitcoin BTC USD 45899. 46929. 42798.  
## 6 2022-01-06 23:59:59 1 bitcoin Bitcoin BTC USD 43566. 43749. 42646.  
## # ℹ 8 more variables: close <dbl>, volume <dbl>, market\_cap <dbl>,  
## # time\_open <dttm>, time\_close <dttm>, time\_high <dttm>, time\_low <dttm>,  
## # date <date>

Note that the echo = FALSE parameter was added to the code chunk to prevent printing of the R code that generated the plot.

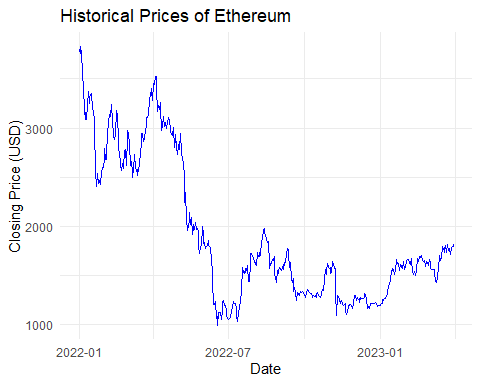
# Load crypto2 library  
  
BTC\_hist <- filter(coin\_hist, id == 1)  
ETH\_hist <- filter(coin\_hist, id == 1027)  
XRP\_hist <- filter(coin\_hist, id == 52)  
DOGE\_hist <- filter(coin\_hist, id == 74)  
  
  
  
ggplot(coin\_hist, aes(x = date, y = close, color = name)) +  
 geom\_line(size = 0.5) + # set line thickness to 1.5  
 scale\_y\_log10() + # set y-axis to log scale  
 ggtitle("Historical Prices of BTC, ETH, and XRP (LOG Scale)") +  
 labs(x = "Date", y = "Closing Price (USD) [log]") +  
 theme\_minimal() + # apply a minimalist theme  
 theme(plot.title = element\_text(size = 18, face = "bold", hjust = 0.5),  
 axis.text = element\_text(size = 12),  
 axis.title = element\_text(size = 14))



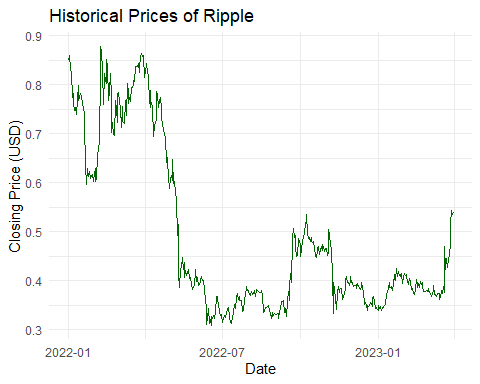
# Create line graph of closing prices over time for Bitcoin  
ggplot(BTC\_hist, aes(x = date, y = close)) +  
 geom\_line(size = 0.5, color = "orange") +  
 ggtitle("Historical Prices of Bitcoin") +  
 xlab("Date") +  
 ylab("Closing Price (USD)") +  
 theme\_minimal()



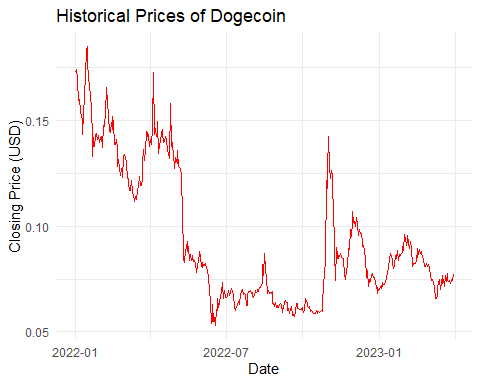
# Create line graph of closing prices over time for Ethereum  
ggplot(ETH\_hist, aes(x = date, y = close)) +  
 geom\_line(size = 0.5, color = "blue") +  
 ggtitle("Historical Prices of Ethereum") +  
 xlab("Date") +  
 ylab("Closing Price (USD)") +  
 theme\_minimal()



# Create line graph of closing prices over time for Ripple  
ggplot(XRP\_hist, aes(x = date, y = close)) +  
 geom\_line(size = 0.5, color = "darkgreen") +  
 ggtitle("Historical Prices of Ripple") +  
 xlab("Date") +  
 ylab("Closing Price (USD)") +  
 theme\_minimal()

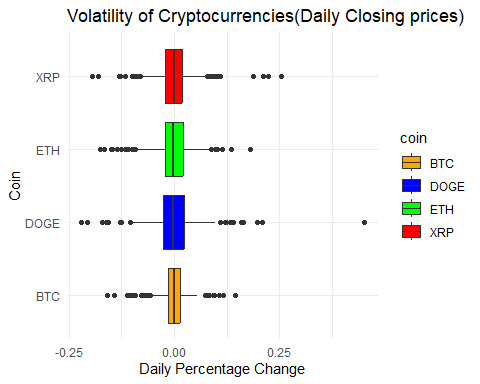


# Create line graph of closing prices over time for Dogecoin  
ggplot(DOGE\_hist, aes(x = date, y = close)) +  
 geom\_line(size = 0.5, color = "red") +  
 ggtitle("Historical Prices of Dogecoin") +  
 xlab("Date") +  
 ylab("Closing Price (USD)") +  
 theme\_minimal()

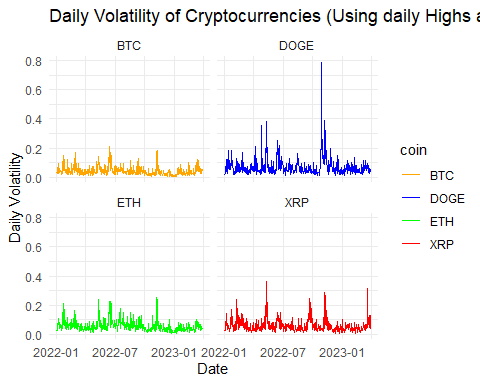


# Combine the four boxplots into one  
library(tidyr)  
# Calculate daily percentage change in closing prices for each coin  
BTC\_pct\_change <- c(NA, diff(BTC\_hist$close)/BTC\_hist$close[-nrow(BTC\_hist)])  
ETH\_pct\_change <- c(NA, diff(ETH\_hist$close)/ETH\_hist$close[-nrow(ETH\_hist)])  
XRP\_pct\_change <- c(NA, diff(XRP\_hist$close)/XRP\_hist$close[-nrow(XRP\_hist)])  
DOGE\_pct\_change <- c(NA, diff(DOGE\_hist$close)/DOGE\_hist$close[-nrow(DOGE\_hist)])  
  
# Combine percentage change data into a single data frame  
pct\_change <- data.frame(date = BTC\_hist$date,  
 BTC = BTC\_pct\_change,  
 ETH = ETH\_pct\_change,  
 XRP = XRP\_pct\_change,  
 DOGE = DOGE\_pct\_change)  
  
# Reshape the data frame from wide to long format  
pct\_change\_long <- pivot\_longer(pct\_change, cols = -date, names\_to = "coin", values\_to = "pct\_change")  
  
# Create a single boxplot of daily percentage change in closing prices for each coin  
ggplot(pct\_change\_long, aes(x = coin, y = pct\_change, fill = coin)) +  
 geom\_boxplot() +  
 ggtitle("Volatility of Cryptocurrencies(Daily Closing prices)") +  
 xlab("Coin") +  
 ylab("Daily Percentage Change") +  
 scale\_fill\_manual(values = c("orange", "blue", "green", "red")) +  
 theme\_minimal() + coord\_flip()

## Warning: Removed 4 rows containing non-finite values (`stat\_boxplot()`).

 …………………………………………………………………………………………….

# Calculate daily volatility using high and low prices for each coin  
BTC\_volatility <- (BTC\_hist$high - BTC\_hist$low) / BTC\_hist$low  
ETH\_volatility <- (ETH\_hist$high - ETH\_hist$low) / ETH\_hist$low  
XRP\_volatility <- (XRP\_hist$high - XRP\_hist$low) / XRP\_hist$low  
DOGE\_volatility <- (DOGE\_hist$high - DOGE\_hist$low) / DOGE\_hist$low  
  
# Combine volatility data into a single data frame  
volatility <- data.frame(date = BTC\_hist$date,  
 BTC = BTC\_volatility,  
 ETH = ETH\_volatility,  
 XRP = XRP\_volatility,  
 DOGE = DOGE\_volatility)  
  
# Reshape data for plotting  
volatility\_long <- pivot\_longer(volatility, cols = -date, names\_to = "coin", values\_to = "volatility")  
  
# Create a single plot with separate panels for each coin  
ggplot(volatility\_long, aes(x = date, y = volatility, color = coin)) +  
 geom\_line(size = 0.5) +  
 facet\_wrap(~coin, ncol = 2) +  
 xlab("Date") +  
 ylab("Daily Volatility") +  
 ggtitle("Daily Volatility of Cryptocurrencies (Using daily Highs and Lows)") +  
 theme\_minimal() +  
 scale\_color\_manual(values = c("orange", "blue", "green", "red"))

 ……………………………………………………………………..

# Join the data by timestamp  
coin\_join <- full\_join(BTC\_hist, ETH\_hist, by = "timestamp", suffix = c("\_BTC", "\_ETH")) %>%  
 full\_join(XRP\_hist, by = "timestamp") %>%  
 full\_join(DOGE\_hist, by = "timestamp", suffix = c("\_XRP", "\_DOGE"))  
  
# Select only the closing prices and rename the columns  
coin\_close <- coin\_join %>%  
 select(timestamp, close\_BTC, close\_ETH, close\_XRP, close\_DOGE) %>%  
 rename(BTC = close\_BTC, ETH = close\_ETH, XRP = close\_XRP, DOGE = close\_DOGE)  
  
# Calculate the correlation matrix  
coin\_corr <- cor(coin\_close[-1])  
  
# Print the correlation matrix  
print(coin\_corr)

## BTC ETH XRP DOGE  
## BTC 1.0000000 0.9756381 0.9152283 0.8592428  
## ETH 0.9756381 1.0000000 0.9206926 0.8957089  
## XRP 0.9152283 0.9206926 1.0000000 0.8739963  
## DOGE 0.8592428 0.8957089 0.8739963 1.0000000

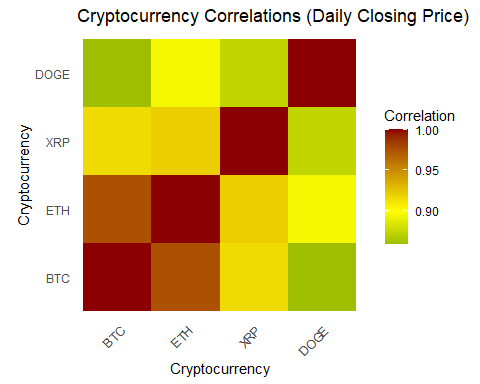
library(reshape2)

## Warning: package 'reshape2' was built under R version 4.2.3

##   
## Attaching package: 'reshape2'

## The following object is masked from 'package:tidyr':  
##   
## smiths

# Melt the correlation matrix for ggplot  
coin\_corr\_melt <- melt(coin\_corr)  
  
# Plot the heatmap  
ggplot(coin\_corr\_melt, aes(x = Var1, y = Var2, fill = value)) +  
 geom\_tile() +  
 scale\_fill\_gradient2(low = "darkgreen", mid = "yellow", high = "darkred", midpoint = 0.9) +  
 theme\_minimal() +  
 theme(axis.text.x = element\_text(angle = 45, vjust = 1, hjust = 1),  
 panel.grid = element\_blank()) +  
 labs(title = "Cryptocurrency Correlations (Daily Closing Price)",  
 x = "Cryptocurrency",  
 y = "Cryptocurrency",  
 fill = "Correlation") +  
 coord\_fixed()

 …………………………………………………………………………………

library(prophet)

## Warning: package 'prophet' was built under R version 4.2.3

## Loading required package: Rcpp

## Loading required package: rlang

## Warning: package 'rlang' was built under R version 4.2.3

##   
## Attaching package: 'rlang'

## The following objects are masked from 'package:purrr':  
##   
## %@%, flatten, flatten\_chr, flatten\_dbl, flatten\_int, flatten\_lgl,  
## flatten\_raw, invoke, splice

# Filter the training data from 20220101 to 20221231  
BTC\_train <- BTC\_hist %>% filter(date >= as.Date("2022-01-01") & date <= as.Date("2022-12-31"))  
  
# Filter the test data from 20230101 to 20230331  
BTC\_test <- BTC\_hist %>% filter(date >= as.Date("2023-01-01") & date <= as.Date("2023-03-31"))  
  
# Prepare the dataset for Prophet (rename columns to 'ds' and 'y')  
BTC\_prophet <- BTC\_train %>% select(date, close) %>% rename(ds = date, y = close)  
  
# Create a Prophet model  
model <- prophet(BTC\_prophet)

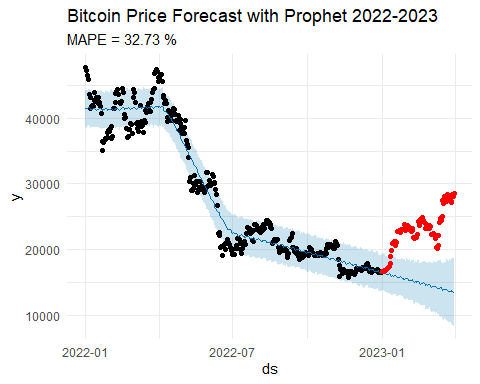
## Disabling yearly seasonality. Run prophet with yearly.seasonality=TRUE to override this.

## Disabling daily seasonality. Run prophet with daily.seasonality=TRUE to override this.

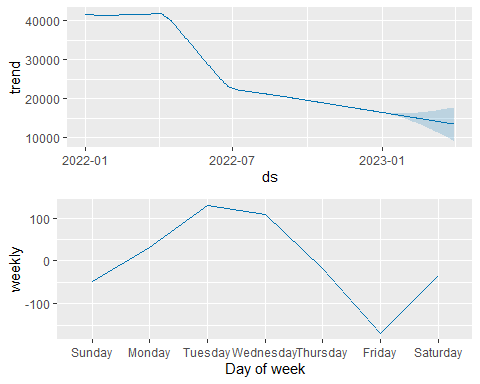
# Make predictions for the next 90 days (3 months)  
future\_dates <- make\_future\_dataframe(model, periods = 90)  
forecast <- predict(model, future\_dates)  
  
# Combine the test data with the predictions  
BTC\_test\_pred <- BTC\_test %>% select(date, close) %>% left\_join(forecast %>% select(ds, yhat), by = c("date" = "ds"))  
  
# Calculate the Mean Absolute Percentage Error (MAPE)  
MAPE <- mean(abs((BTC\_test\_pred$close - BTC\_test\_pred$yhat) / BTC\_test\_pred$close)) \* 100  
print(paste("MAPE:", MAPE))

## [1] "MAPE: 32.7284017202559"

# Convert the date column to POSIXct format  
BTC\_test\_pred$date <- as.POSIXct(BTC\_test\_pred$date)  
  
# Plot the actual data and forecast  
plot(model, forecast) +  
 geom\_point(aes(x = date, y = close), data = BTC\_test\_pred, color = "red") +  
 theme\_minimal() +  
 labs(title = "Bitcoin Price Forecast with Prophet 2022-2023",  
 subtitle = paste("MAPE =", round(MAPE, 2), "%"))



# Plot the forecast components  
prophet\_plot\_components(model, forecast)

 …………………………………..

coins2 <- crypto\_list(only\_active=TRUE) |> filter(symbol %in% c("BTC"))  
BTC\_histPro <- crypto\_history(coins2, limit=1, start\_date="20140101", end\_date="20230331", finalWait=FALSE) #15months

## ❯ Scraping historical crypto data

##

## ❯ Processing historical crypto data

##

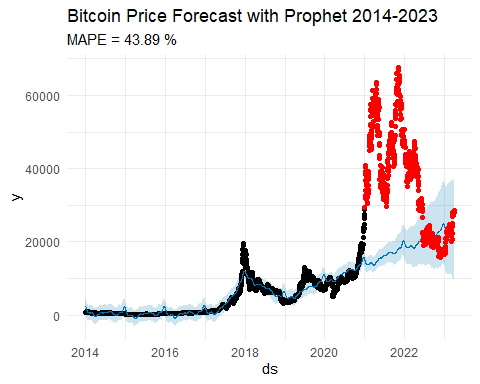
BTC\_histPro$date <- as.Date(BTC\_histPro$timestamp)  
  
# Filter the training data from 20220101 to 20221231  
BTC\_train2 <- BTC\_histPro %>% filter(date >= as.Date("2014-01-01") & date <= as.Date("2020-12-31"))  
  
# Filter the test data from 20230101 to 20230331  
BTC\_test2 <- BTC\_histPro %>% filter(date >= as.Date("2021-01-01") & date <= as.Date("2023-03-31"))  
  
# Prepare the dataset for Prophet (rename columns to 'ds' and 'y')  
BTC\_prophet2 <- BTC\_train2 %>% select(date, close) %>% rename(ds = date, y = close)  
  
# Create a Prophet model  
model2 <- prophet(BTC\_prophet2)

## Disabling daily seasonality. Run prophet with daily.seasonality=TRUE to override this.

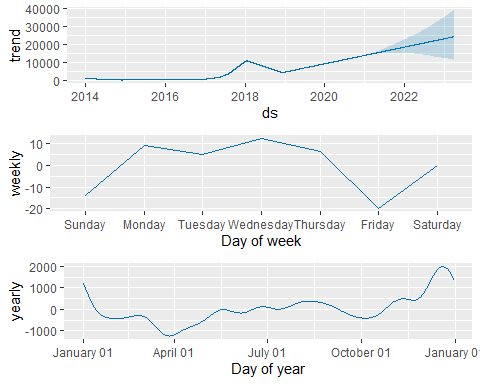
# Make predictions for the next 365 days  
future\_dates2 <- make\_future\_dataframe(model2, periods = 820)  
forecast2 <- predict(model2, future\_dates2)  
  
# Combine the test data with the predictions  
BTC\_test\_pred2 <- BTC\_test2 %>% select(date, close) %>% left\_join(forecast2 %>% select(ds, yhat), by = c("date" = "ds"))  
  
# Calculate the Mean Absolute Percentage Error (MAPE)  
MAPE2 <- mean(abs((BTC\_test\_pred2$close - BTC\_test\_pred2$yhat) / BTC\_test\_pred2$close)) \* 100  
print(paste("MAPE:", MAPE2))

## [1] "MAPE: 43.8876376453523"

# Convert the date column to POSIXct format  
BTC\_test\_pred2$date <- as.POSIXct(BTC\_test\_pred2$date)  
  
# Plot the actual data and forecast  
plot(model2, forecast2) +  
 geom\_point(aes(x = date, y = close), data = BTC\_test\_pred2, color = "red") +  
 theme\_minimal() +  
 labs(title = "Bitcoin Price Forecast with Prophet 2014-2023",  
 subtitle = paste("MAPE =", round(MAPE2, 2), "%"))



# Plot the forecast components  
prophet\_plot\_components(model2, forecast2)

 ………………………………………………..

library(prophet)  
  
# Get the Dogecoin price data  
coins3 <- crypto\_list(only\_active = TRUE) |> filter(symbol %in% c("DOGE"))  
DOGE\_histPro <- crypto\_history(coins3, limit = 1, start\_date = "20140101", end\_date = "20230331", finalWait = FALSE)

## ❯ Scraping historical crypto data

##

## ❯ Processing historical crypto data

##

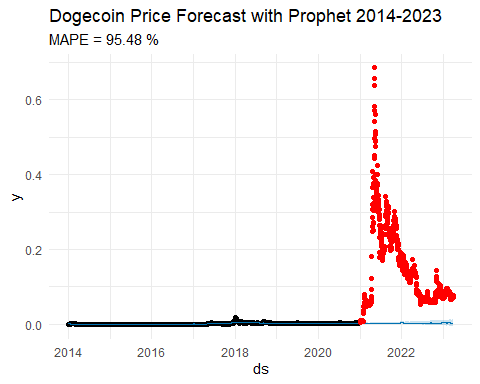
DOGE\_histPro$date <- as.Date(DOGE\_histPro$timestamp)  
  
# Filter the training data from 2014-01-01 to 2022-03-31  
DOGE\_train <- DOGE\_histPro %>% filter(date >= as.Date("2014-01-01") & date <= as.Date("2020-12-31"))  
  
# Filter the test data from 2022-04-01 to 2023-03-31  
DOGE\_test <- DOGE\_histPro %>% filter(date >= as.Date("2021-01-01") & date <= as.Date("2023-03-31"))  
  
# Prepare the dataset for Prophet (rename columns to 'ds' and 'y')  
DOGE\_prophet <- DOGE\_train %>% select(date, close) %>% rename(ds = date, y = close)  
  
# Create a Prophet model  
model <- prophet(DOGE\_prophet)

## Disabling daily seasonality. Run prophet with daily.seasonality=TRUE to override this.

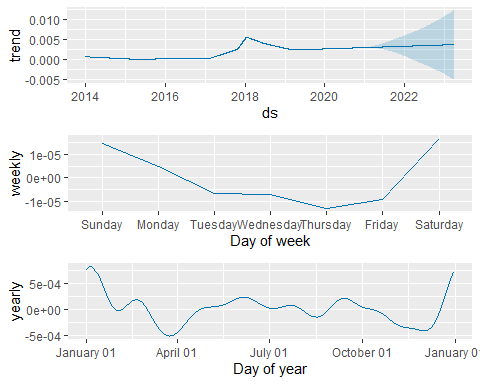
# Make predictions for the next 365 days  
future\_dates <- make\_future\_dataframe(model, periods = 820)  
forecast <- predict(model, future\_dates)  
  
# Combine the test data with the predictions  
DOGE\_test\_pred <- DOGE\_test %>% select(date, close) %>% left\_join(forecast %>% select(ds, yhat), by = c("date" = "ds"))  
  
# Calculate the Mean Absolute Percentage Error (MAPE)  
MAPE <- mean(abs((DOGE\_test\_pred$close - DOGE\_test\_pred$yhat) / DOGE\_test\_pred$close)) \* 100  
print(paste("MAPE:", MAPE))

## [1] "MAPE: 95.4760913763684"

# Convert the date column to POSIXct format  
DOGE\_test\_pred$date <- as.POSIXct(DOGE\_test\_pred$date)  
  
# Plot the actual data and forecast  
plot(model, forecast) +  
 geom\_point(aes(x = date, y = close), data = DOGE\_test\_pred, color = "red") +  
 theme\_minimal() +  
 labs(title = "Dogecoin Price Forecast with Prophet 2014-2023",  
 subtitle = paste("MAPE =", round(MAPE, 2), "%"))



# Plot the forecast components  
prophet\_plot\_components(model, forecast)



library(prophet)  
  
# Get the Dogecoin price data  
coins3 <- crypto\_list(only\_active = TRUE) |> filter(symbol %in% c("DOGE"))  
DOGE\_histPro <- crypto\_history(coins3, limit = 1, start\_date = "20140101", end\_date = "20230331", finalWait = FALSE)

## ❯ Scraping historical crypto data

##

## ❯ Processing historical crypto data

##

DOGE\_histPro$date <- as.Date(DOGE\_histPro$timestamp)  
  
# Filter the training data from 2014-01-01 to 2022-03-31  
DOGE\_train <- DOGE\_histPro %>% filter(date >= as.Date("2022-01-01") & date <= as.Date("2022-12-31"))  
  
# Filter the test data from 2022-04-01 to 2023-03-31  
DOGE\_test <- DOGE\_histPro %>% filter(date >= as.Date("2023-01-01") & date <= as.Date("2023-03-31"))  
  
# Prepare the dataset for Prophet (rename columns to 'ds' and 'y')  
DOGE\_prophet <- DOGE\_train %>% select(date, close) %>% rename(ds = date, y = close)  
  
# Create a Prophet model  
model <- prophet(DOGE\_prophet)

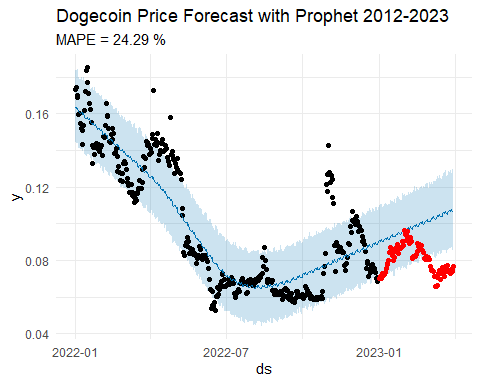
## Disabling yearly seasonality. Run prophet with yearly.seasonality=TRUE to override this.

## Disabling daily seasonality. Run prophet with daily.seasonality=TRUE to override this.

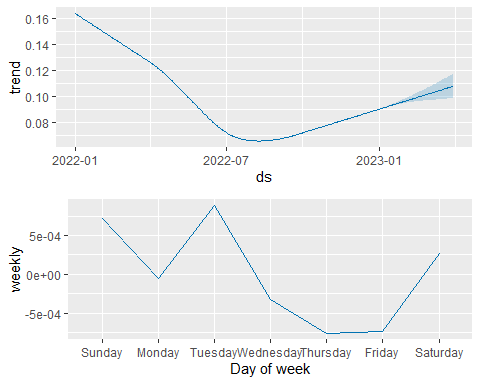
# Make predictions for the next 365 days  
future\_dates <- make\_future\_dataframe(model, periods = 90)  
forecast <- predict(model, future\_dates)  
  
# Combine the test data with the predictions  
DOGE\_test\_pred <- DOGE\_test %>% select(date, close) %>% left\_join(forecast %>% select(ds, yhat), by = c("date" = "ds"))  
  
# Calculate the Mean Absolute Percentage Error (MAPE)  
MAPE <- mean(abs((DOGE\_test\_pred$close - DOGE\_test\_pred$yhat) / DOGE\_test\_pred$close)) \* 100  
print(paste("MAPE:", MAPE))

## [1] "MAPE: 24.2884677557298"

# Convert the date column to POSIXct format  
DOGE\_test\_pred$date <- as.POSIXct(DOGE\_test\_pred$date)  
  
# Plot the actual data and forecast  
plot(model, forecast) +  
 geom\_point(aes(x = date, y = close), data = DOGE\_test\_pred, color = "red") +  
 theme\_minimal() +  
 labs(title = "Dogecoin Price Forecast with Prophet 2022-2023",  
 subtitle = paste("MAPE =", round(MAPE, 2), "%"))



# Plot the forecast components  
prophet\_plot\_components(model, forecast)



library(randomForest)

## Warning: package 'randomForest' was built under R version 4.2.3

## randomForest 4.7-1.1

## Type rfNews() to see new features/changes/bug fixes.

##   
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':  
##   
## margin

## The following object is masked from 'package:dplyr':  
##   
## combine

# Get the Bitcoin price data  
coins <- crypto\_list(only\_active = TRUE) %>% filter(symbol == "BTC")  
BTC\_hist <- crypto\_history(coins, limit = 1, start\_date = "20140101", end\_date = "20230331", finalWait = FALSE)

## ❯ Scraping historical crypto data

##

## ❯ Processing historical crypto data

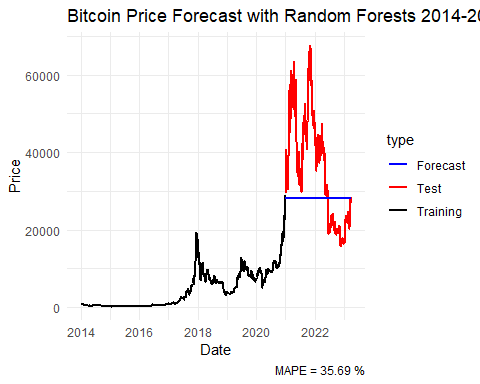
##

BTC\_hist$date <- as.Date(BTC\_hist$timestamp)  
  
# Filter the training data from 2014-01-01 to 2020-12-31  
BTC\_train <- BTC\_hist %>% filter(date >= as.Date("2014-01-01") & date <= as.Date("2020-12-31"))  
  
# Filter the test data from 2021-01-01 to 2023-03-31  
BTC\_test <- BTC\_hist %>% filter(date >= as.Date("2021-01-01") & date <= as.Date("2023-03-31"))  
  
# Prepare the training data  
BTC\_train\_rf <- BTC\_train %>% select(date, close)  
BTC\_train\_rf$date <- as.numeric(BTC\_train\_rf$date)  
  
# Train the random forest model  
model\_rf <- randomForest(close ~ date, data = BTC\_train\_rf)  
  
# Prepare the test data  
BTC\_test\_rf <- BTC\_test %>% select(date, close)  
BTC\_test\_rf$date <- as.numeric(BTC\_test\_rf$date)  
  
# Make predictions for the test data  
BTC\_test\_pred\_rf <- predict(model\_rf, newdata = BTC\_test\_rf)  
  
# Calculate the Mean Absolute Percentage Error (MAPE)  
MAPE\_rf <- mean(abs((BTC\_test\_rf$close - BTC\_test\_pred\_rf) / BTC\_test\_rf$close)) \* 100  
print(paste("MAPE:", MAPE\_rf))

## [1] "MAPE: 35.6854563528149"

# Combine the training and test data  
BTC\_all <- rbind(BTC\_train\_rf, BTC\_test\_rf)  
BTC\_all$date <- as.Date(BTC\_all$date, origin = "1970-01-01")  
  
# Add a column to indicate whether the data is training or test data  
BTC\_all$type <- ifelse(BTC\_all$date %in% BTC\_train\_rf$date, "Training", "Test")  
  
# Add a column for the predicted values  
BTC\_all$forecast <- ifelse(BTC\_all$type == "Training", NA, BTC\_test\_pred\_rf)  
  
# Plot the actual data and forecast  
forecast\_plot\_rf <- ggplot(BTC\_all, aes(x = date, y = close)) +  
 geom\_line(aes(color = type), size = 1) +  
 geom\_line(aes(x = date, y = forecast, color = "Forecast"), size = 1) +  
 xlab("Date") +  
 ylab("Price") +  
 ggtitle("Bitcoin Price Forecast with Random Forests 2014-2023") +  
 labs(caption = paste("MAPE =", round(MAPE\_rf, 2), "%")) +  
 scale\_color\_manual(values = c("Training" = "black", "Test" = "red", "Forecast" = "blue")) + theme\_minimal()  
  
print(forecast\_plot\_rf)

## Warning: Removed 2557 rows containing missing values (`geom\_line()`).



# Load the forecast package  
library(forecast)

## Warning: package 'forecast' was built under R version 4.2.3

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

library(xts)

## Warning: package 'xts' was built under R version 4.2.3

## Loading required package: zoo

## Warning: package 'zoo' was built under R version 4.2.3

##   
## 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

# Obtain the Bitcoin price data  
coins <- crypto\_list(only\_active = TRUE) %>% filter(symbol == "BTC")  
BTC\_hist <- crypto\_history(coins, limit = 1, start\_date = "20140101", end\_date = "20230331", finalWait = FALSE)

## ❯ Scraping historical crypto data

##

## ❯ Processing historical crypto data

##

BTC\_hist$date <- as.Date(BTC\_hist$timestamp)  
  
# Split the data into training and test sets  
BTC\_train <- BTC\_hist %>% filter(date >= as.Date("2014-01-01") & date <= as.Date("2020-12-31"))  
BTC\_test <- BTC\_hist %>% filter(date >= as.Date("2021-01-01") & date <= as.Date("2023-03-31"))  
  
# Extract the closing prices for the training set  
BTC\_train\_ets <- BTC\_train %>% select(date, close)  
BTC\_train\_ets <- xts(BTC\_train\_ets$close, order.by = BTC\_train\_ets$date)  
  
# Build an ETS model on the training set  
model\_ets <- ets(BTC\_train\_ets)  
  
# Predict the closing prices for the test set  
BTC\_test\_ets <- BTC\_test %>% select(date, close)  
BTC\_test\_ets <- xts(BTC\_test\_ets$close, order.by = BTC\_test\_ets$date)  
BTC\_test\_pred\_ets <- forecast(model\_ets, h = nrow(BTC\_test\_ets))  
  
# Compute the Mean Absolute Percentage Error (MAPE)  
MAPE\_ets <- mean(abs((BTC\_test\_ets - BTC\_test\_pred\_ets$mean) / BTC\_test\_ets)) \* 100

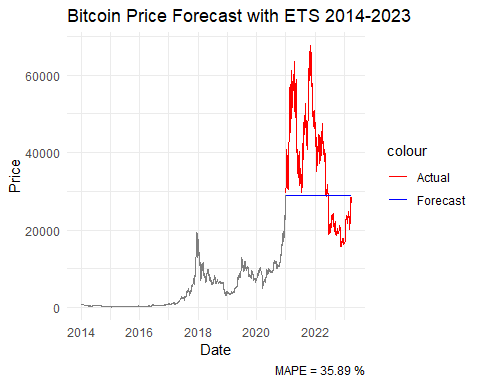
## Warning in mean(abs((BTC\_test\_ets - BTC\_test\_pred\_ets$mean)/BTC\_test\_ets)):  
## Incompatible methods ("Ops.xts", "Ops.ts") for "-"

## Warning in mean(abs((BTC\_test\_ets - BTC\_test\_pred\_ets$mean)/BTC\_test\_ets)):  
## Incompatible methods ("Ops.ts", "Ops.xts") for "/"

print(paste("MAPE:", MAPE\_ets))

## [1] "MAPE: 35.88835785846"

# Plot the actual and predicted prices  
BTC\_test\_pred\_ets\_df <- data.frame(date = index(BTC\_test\_ets), close = coredata(BTC\_test\_ets), forecast = coredata(BTC\_test\_pred\_ets$mean))  
  
forecast\_plot\_ets <- ggplot() +  
 geom\_line(aes(x = date, y = close, color = "Actual"), data = BTC\_test\_pred\_ets\_df) +  
 geom\_line(aes(x = date, y = forecast, color = "Forecast"), data = BTC\_test\_pred\_ets\_df) +  
 xlab("Date") +  
 ylab("Price") +  
 ggtitle("Bitcoin Price Forecast with ETS 2014-2023") +  
 labs(caption = paste("MAPE =", round(MAPE\_ets, 2), "%")) +  
 scale\_color\_manual(values = c("Actual" = "red", "Forecast" = "blue"))+ theme\_minimal()  
  
# Add the training data to the plot  
forecast\_plot\_ets <- forecast\_plot\_ets +  
 geom\_line(aes(x = date, y = close, color = "Training"), data = BTC\_train)  
  
print(forecast\_plot\_ets)



library(forecast)  
library(tidyverse)

## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──  
## ✔ forcats 1.0.0 ✔ stringr 1.5.0  
## ✔ lubridate 1.9.2 ✔ tibble 3.1.8  
## ✔ readr 2.1.4   
## ── Conflicts ────────────────────────────────────────── tidyverse\_conflicts() ──  
## ✖ rlang::%@%() masks purrr::%@%()  
## ✖ randomForest::combine() masks dplyr::combine()  
## ✖ dplyr::filter() masks stats::filter()  
## ✖ xts::first() masks dplyr::first()  
## ✖ rlang::flatten() masks purrr::flatten()  
## ✖ rlang::flatten\_chr() masks purrr::flatten\_chr()  
## ✖ rlang::flatten\_dbl() masks purrr::flatten\_dbl()  
## ✖ rlang::flatten\_int() masks purrr::flatten\_int()  
## ✖ rlang::flatten\_lgl() masks purrr::flatten\_lgl()  
## ✖ rlang::flatten\_raw() masks purrr::flatten\_raw()  
## ✖ rlang::invoke() masks purrr::invoke()  
## ✖ dplyr::lag() masks stats::lag()  
## ✖ xts::last() masks dplyr::last()  
## ✖ randomForest::margin() masks ggplot2::margin()  
## ✖ rlang::splice() masks purrr::splice()  
## ℹ Use the ]8;;http://conflicted.r-lib.org/conflicted package]8;; to force all conflicts to become errors

#Get the Bitcoin price data  
coins2 <- crypto\_list(only\_active=TRUE) %>%  
filter(symbol %in% c("BTC"))  
BTC\_hist <- crypto\_history(coins2, limit=1, start\_date="20140101", end\_date="20230331", finalWait=FALSE) #15months

## ❯ Scraping historical crypto data

##

## ❯ Processing historical crypto data

##

BTC\_hist$date <- as.Date(BTC\_hist$timestamp)  
  
#Filter the training data from 2014-01-01 to 2020-12-31  
BTC\_train <- BTC\_hist %>% filter(date >= as.Date("2014-01-01") & date <= as.Date("2020-12-31"))  
  
#Filter the test data from 2021-01-01 to 2023-03-31  
BTC\_test <- BTC\_hist %>% filter(date >= as.Date("2021-01-01") & date <= as.Date("2023-03-31"))  
  
#Prepare the training dataset for ARIMA (select 'close' column and convert to time series)  
BTC\_train\_ts <- ts(BTC\_train$close)  
  
#Fit an ARIMA model to the training data  
model <- auto.arima(BTC\_train\_ts)  
  
#Make predictions for the next 820 days  
forecast <- forecast(model, h = 820)  
  
#Combine the test data with the predictions  
BTC\_test\_pred <- BTC\_test %>% select(date, close) %>% mutate(forecast = forecast$mean)  
BTC\_test\_pred <- as.data.frame(BTC\_test\_pred) # convert to data frame  
  
#Calculate the Mean Absolute Percentage Error (MAPE)  
MAPE <- mean(abs((BTC\_test\_pred$close - BTC\_test\_pred$forecast) / BTC\_test\_pred$close)) \* 100  
print(paste("MAPE:", MAPE))

## [1] "MAPE: 279.459614892151"

#Plot the actual data, training data and forecast  
forecast\_plot <- ggplot() +  
geom\_line(aes(x = date, y = close, color = "Training"), data = BTC\_train) +  
geom\_line(aes(x = date, y = close, color = "Actual"), data = BTC\_test\_pred) +  
geom\_line(aes(x = date, y = forecast, color = "Forecast"), data = BTC\_test\_pred) +  
xlab("Date") +  
ylab("Price") +  
ggtitle("Bitcoin Price Forecast with ARIMA 2014-2023") +  
labs(caption = paste("MAPE =", round(MAPE, 2), "%")) +  
scale\_color\_manual(values = c("Training" = "black", "Actual" = "red", "Forecast" = "blue")) + theme\_minimal()  
  
print(forecast\_plot)

