Forecasting Next-Day Bitcoin Returns: A Comparative Analysis of Linear and Ensemble Models

2025-06-15

Table of Contents

1.Introduction

Cryptocurrencies like Bitcoin are well-known for their extreme volatility and rapid price swings. Bitcoin's daily returns often exhibit heavy tails and high variance, making accurate prediction challenging. Forecasting Bitcoin returns is of great interest to investors and researchers, as even a slight improvement in prediction accuracy can inform better trading and risk management decisions.

In this project, we focus on predicting the next-day return of Bitcoin. A return is defined as the percentage change in price from one day to the next – for example, a 5% return means the price increased by 5% compared to the previous day. We will specifically predict daily return (%) for the following day, using information from previous days. Key concepts in time-series and machine learning, such as autocorrelation, volatility, and model overfitting, will be relevant in our analysis. We briefly note that in efficient markets one might expect returns to be essentially unpredictable, but we explore whether any measurable patterns exist in Bitcoin's historical data. The primary goal

of this project is to build and compare several statistical learning models to forecast Bitcoin's next-day return using lagged features (yesterday's return and trading volume). We utilize the publicly available, which contains minute-by-minute price and volume information from the Bitstamp exchange (starting January 2012). For our purposes, we aggregate this data to a daily frequency (computing daily open, close, high, low prices and total volume) and derive daily returns. We will fit at least four types of predictive models, incorporating concepts learned over 10 weeks of coursework: a basic AR(1) time-series model (using yesterday's return to predict today's), a multiple Linear Regression model (using lagged return and volume), an Elastic Net regularized regression (to prevent overfitting and perform feature shrinkage), and a tree-based model (in particular, a Random Forest ensemble). Through this process, we will apply techniques such as data splitting, k-fold cross-validation, and hyperparameter tuning to select the best model. The report is organized as follows: we first perform an exploratory data analysis of the Bitcoin dataset, then describe our data splitting strategy and cross-validation approach. Next, we detail the fitting of each model and the tuning of any hyperparameters. We then compare model performances and evaluate the best model(s) on a hold-out test set. Finally, we conclude with a summary of findings and suggestions for future work.

2.Exploratory Data Analysis

We begin by loading the dataset and examining its structure. The Kaggle dataset provides Bitcoin prices and volume at 1-minute intervals from 2012 onward. After reading in the data, we aggregate it to daily frequency (each day's opening price, closing price, high, low, and total volume) since our goal is to predict daily returns. We then compute the daily return as the percentage change in closing price from the previous day. (For days with no trades, no return is computed; however, in our dataset every day from 2012 onward had at least one trade.) We also take the trading volume in Bitcoin for each day (sum of all BTC traded that day) as a potential predictor. Before proceeding, we check for missing data. The raw minute-by-minute data does contain missing entries for periods with no trading activity. After aggregating to daily level and removing minutes with no trades, we find that no daily observations are missing (each day had at least one trade, so we have a continuous daily series). If there had been missing days, we would either impute them or simply omit them, but here it was not necessary. Next, we examine the main characteristics of the data through visualizations and summary statistics. Figure 1 below shows the time series of daily Bitcoin prices (in USD) from 2012 through the end of our data. This plot reveals the dramatic growth of Bitcoin's price over the years. We can see relatively low prices (under $100) in the early period, followed by a sharp rise to nearly $20,000 in late 2017 and a subsequent crash in 2018. Another remarkable surge is visible in late 2020 to early 2021, reflecting Bitcoin's bull run during that time. Overall, the price trend is upward, but with several pronounced boom-and-bust cycles. This volatility is also evident in the irregular, jagged nature of the price curve – periods of rapid increase are often followed by steep declines. Such extreme volatility suggests that returns (percentage changes) will be large in magnitude and possibly difficult to predict.

2.1 Data Loading & Aggregation

```
library(readr)
library(dplyr)
library(lubridate)

btc_min <- read_csv(
  "/Users/ruilanzeng/Downloads/btcusd_1-min_data.csv",
  col_types = cols(
    Timestamp = col_double(),
    Open      = col_double(),
    High      = col_double(),
    Low       = col_double(),
    Close     = col_double(),
    Volume    = col_double()
  )
)

btc_min <- btc_min %>%
  mutate(
    datetime = as_datetime(Timestamp, origin = "1970-01-01", tz = "UTC"),
    date     = as_date(datetime)
  ) %>%
  filter(!is.na(date))   # drop any bad rows

daily_data <- btc_min %>%
  group_by(date) %>%
  summarize(
    Open   = first(Open),
    High   = max(High),
    Low    = min(Low),
    Close  = last(Close),
    Volume = sum(Volume),
    .groups = "drop"
  ) %>%
  arrange(date) %>%
  mutate(Return = (Close / lag(Close) - 1) * 100)

print(dim(daily_data))
```

```
## [1] 4915    7
```

```
print(head(daily_data))
```

```
## # A tibble: 6 × 7
##   date        Open  High   Low Close Volume Return
##   <date>     <dbl> <dbl> <dbl> <dbl>  <dbl>  <dbl>
## 1 2012-01-01  4.58  4.84  4.58  4.84   10   NA
## 2 2012-01-02  4.84  5     4.84  5      10.1  3.31
## 3 2012-01-03  5     5.32  5     5.29  107.   5.80
## 4 2012-01-04  5.29  5.57  4.93  5.57  107.   5.29
## 5 2012-01-05  5.57  6.46  5.57  6.42   70.3 15.3
## 6 2012-01-06  6.42  6.9   6.4   6.4    55.9 -0.312
```

Codebook

To ensure clarity, below is a **codebook** listing every variable in our daily-level dataset. For each field, we give its data type and a short description of its meaning.

Codebook: Variable Definitions

Variable	Type	Description
date	Date	Trading day (YYYY-MM-DD)
Open	numeric	Opening price of Bitcoin (USD)
High	numeric	Highest price of Bitcoin (USD)
Low	numeric	Lowest price of Bitcoin (USD)
Close	numeric	Closing price of Bitcoin (USD)
Volume	numeric	Total BTC traded that day
Return	numeric	Daily return (%)
Prev_Return	numeric	Lagged return: previous day
Prev_Volume	numeric	Lagged volume: previous day
sentiment_ma7	numeric	7-day moving average of sentiment
sentiment_z	numeric	Z-score of sentiment_ma7 by regime
regime	factor	Market regime: bull/bear/neutral
ret_3d	numeric	3-day forward return
ret_5d	numeric	5-day forward return
ret_10d	numeric	10-day forward return
ret_bin	factor	Quintile bin of ret_5d for stratification
bull, bear, neutral	numeric	One-hot regime dummy columns

```
library(ggplot2)
library(scales)
```

```
##
## Attaching package: 'scales'
```

```
## The following object is masked from 'package:readr':
##
##     col_factor
```

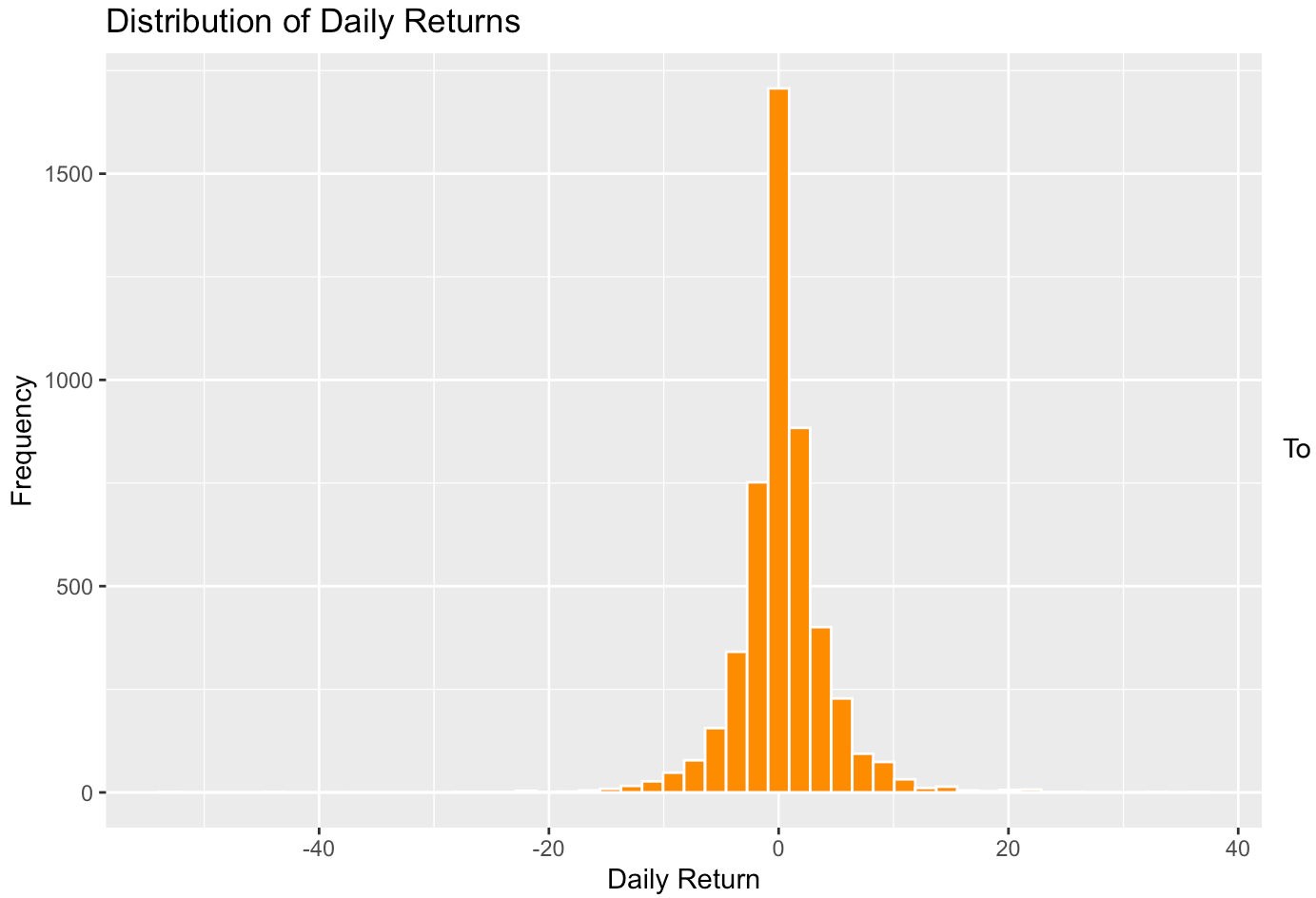
```
ggplot(daily_data, aes(x = date, y = Close)) +
  geom_line(color = "steelblue", linewidth = 0.8) +
  scale_x_date(
    date_breaks = "1 year",
    date_labels = "%Y",
    limits      = range(daily_data$date, na.rm = TRUE)
  ) +
  labs(
    title = "Daily Bitcoin Closing Price (USD)",
    x     = "Date",
    y     = "Closing Price (USD)"
  ) +
  theme_minimal()
```

Below is the interpretation of Figure 1:

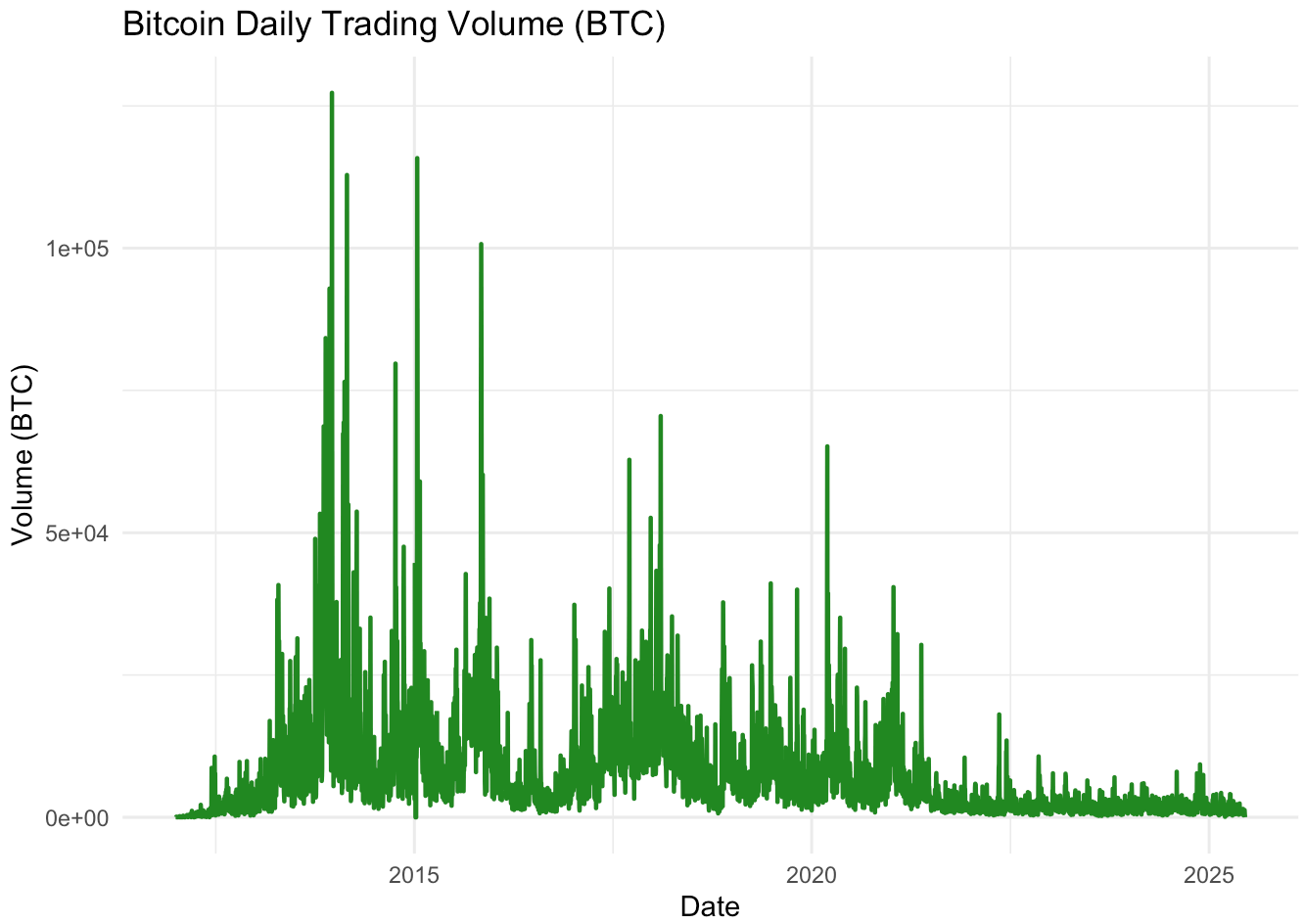
Bitcoin's price exhibits an overall exponential growth from 2012 onward, punctuated by pronounced boom-and-bust cycles. For example, the late 2017 run-up to nearly $20 k and the early 2021 surge are both visible, each followed by sharp corrections—highlighting the high volatility and cyclic bubbles in the Bitcoin market.

```
ggplot(daily_data, aes(x = Return)) +
  geom_histogram(bins = 50, fill = "darkorange", color = "white") +
  labs(x = "Daily Return", y = "Frequency", title = "Distribution of Daily Returns")
```

```
## Warning: Removed 1 row containing non-finite outside the scale range
## (`stat_bin()`).
```



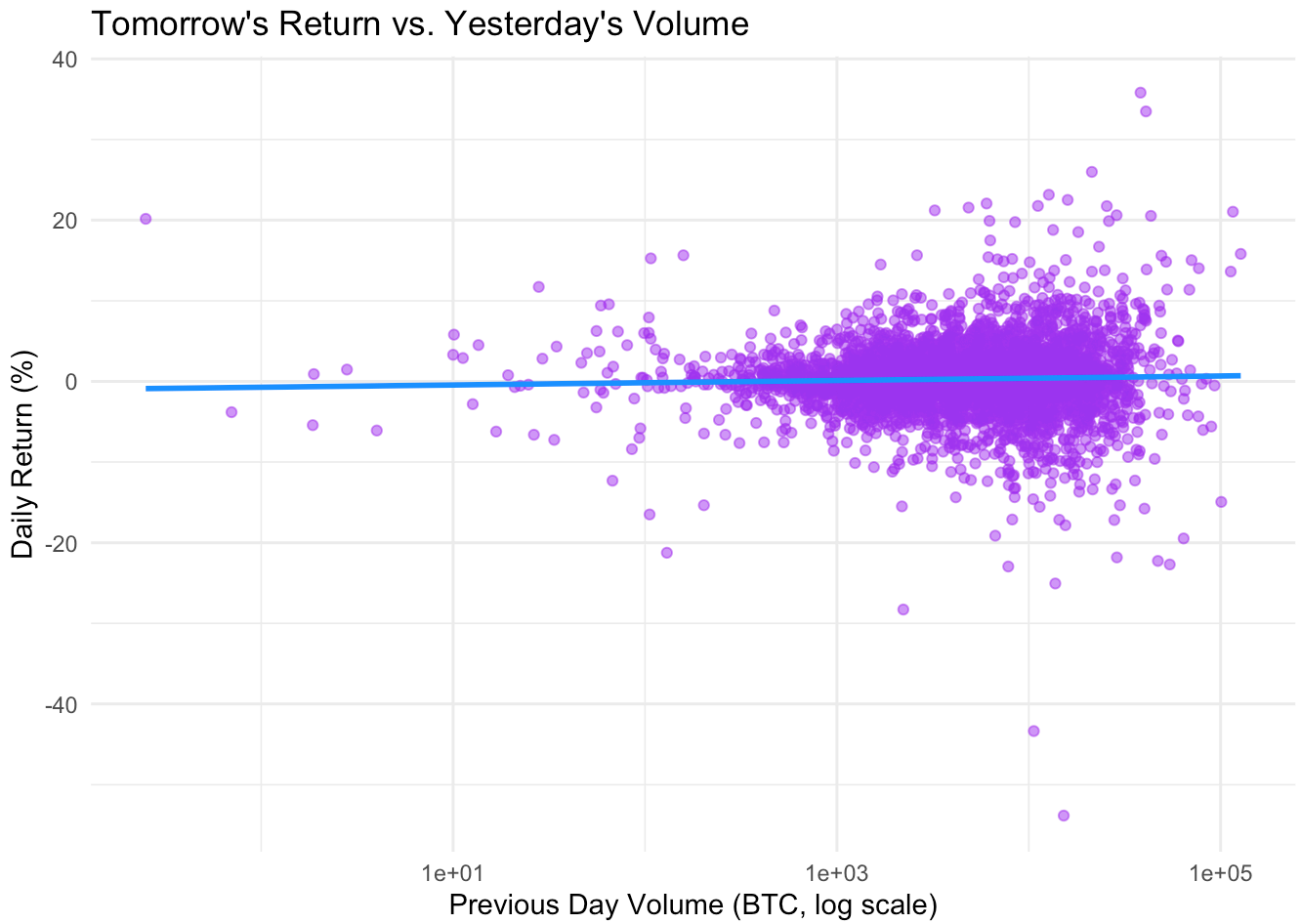
To better understand the behavior of daily returns, we plot their distribution. Figure 2 shows a histogram of the daily return percentage. The distribution of returns is centered near zero (Bitcoin does not have a consistent upward or downward daily drift), but it is wide and heavy-tailed. We observe many days with moderately large gains or losses, and a few extreme outliers on both the positive and negative side. For instance, there are days with returns above +20% and others below –20%. This heavy-tailed characteristic indicates higher kurtosis – in other words, extreme events occur more frequently than they would under a normal (Gaussian) distribution. The distribution also appears slightly asymmetric; it may have a longer tail on one side, though a formal skewness calculation would confirm this. Such a pattern (high volatility and occasional huge jumps or crashes) is consistent with other research findings that cryptocurrency returns have "fat tails" and greater skewness compared to traditional asset returns. For our modeling, this implies that error distributions might be non-normal and that very large prediction errors can occur on extreme days. We will consider performance metrics (like mean absolute error vs. root mean square error) that are sensitive to outliers when evaluating models.



Bitcoin Daily Trading Volume (BTC)

We also examine the trading volume over time. Figure 3 displays the daily volume of Bitcoin traded (in BTC) each day. It is evident that trading activity has grown dramatically over the years. In Bitcoin's early years, daily volume was very low (often just a few hundred or thousand BTC). As Bitcoin gained popularity, volume increased, reaching tens of thousands of BTC per day by 2017, and even higher in 2020-2021. The volume time series is quite spiky: volume tends to spike on days of major price moves, either upward or downward. For example, some of the highest volume days coincide with the peaks and crashes observed in the price plot. This suggests that trading volume and price volatility are related – high volume often accompanies large price changes, likely reflecting heightened market attention and participation during those times. We will consider volume as a predictor in our models, to see if unusually high (or low) trading activity today can help predict the next day's return.

```
## `geom_smooth()` using formula = 'y ~ x'
```



Tomorrow's Return vs. Yesterday's Volume

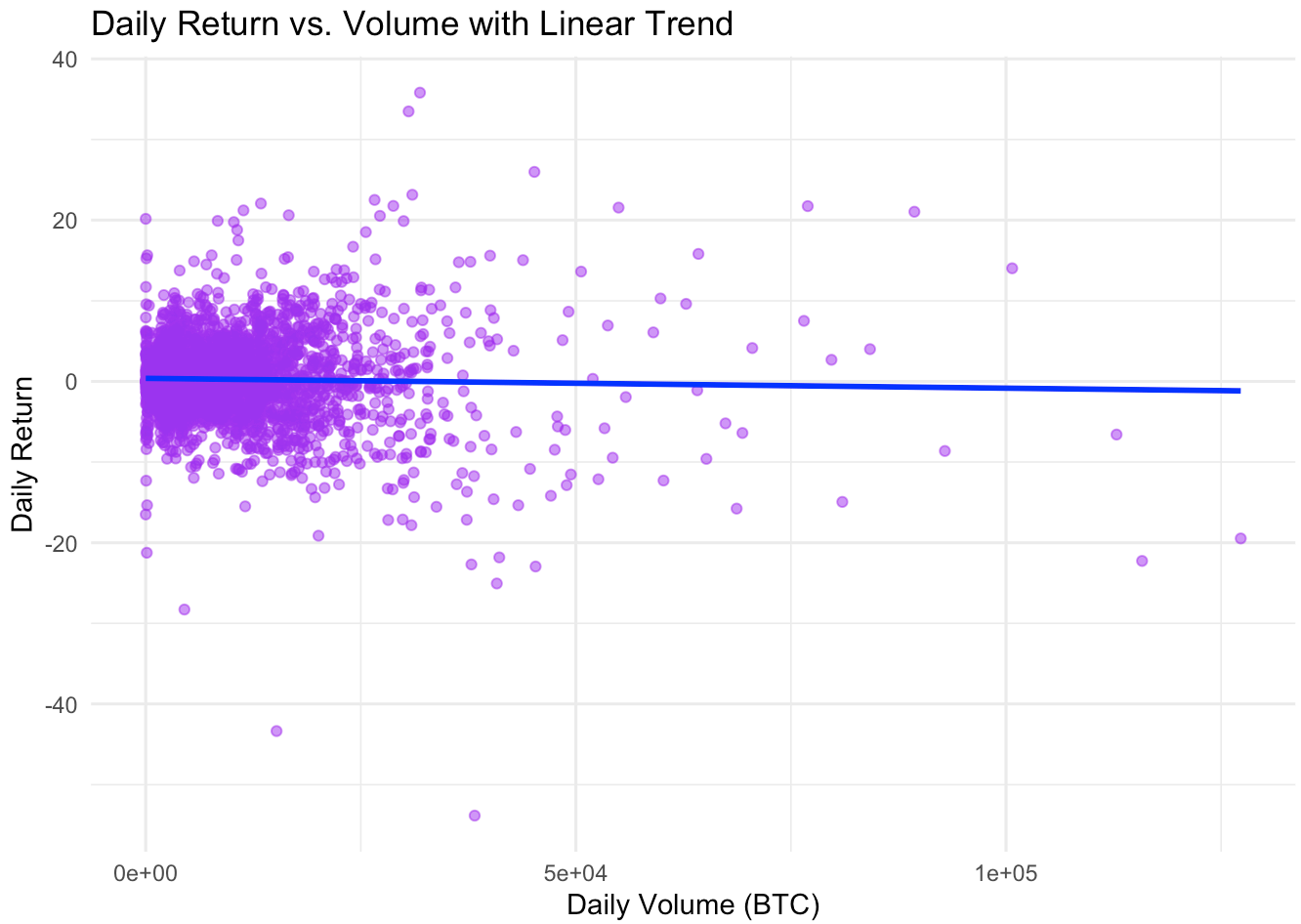
```
library(ggplot2)

ggplot(daily_data, aes(x = Volume, y = Return)) +
  geom_point(alpha = 0.5, color = "purple") +
  geom_smooth(method = "lm", color = "blue", se = FALSE) +
  labs(
    x = "Daily Volume (BTC)",
    y = "Daily Return",
    title = "Daily Return vs. Volume with Linear Trend"
  ) +
  theme_minimal()
```

```
## `geom_smooth()` using formula = 'y ~ x'
```

```
## Warning: Removed 1 row containing non-finite outside the scale range
## (`stat_smooth()`).
```

```
## Warning: Removed 1 row containing missing values or values outside the scale range
## (`geom_point()`).
```



Daily Return vs. Volume with Linear Fit

Return vs. Volume Relationship Finally, we examine the relationship between daily returns and daily volume. The scatterplot below shows each day's return versus the BTC volume of that day. We use the previous day's volume when considering it as a predictor for next-day return (as in our models), but for exploratory purposes, here we look at the same-day relationship to see if high volume days tend to coincide with up or down moves. In this figure, there is no clear linear relationship between daily volume and daily return. The points appear broadly scattered. Most days (with moderate volumes) cluster near the center with small returns. On days with extremely high volume (far right of the plot), we do not see consistently positive or negative returns; instead, some of those high-volume days correspond to large negative returns (points low on the plot) and others to positive returns. This suggests that high volume often accompanies volatile days, but volume alone doesn't reliably indicate the direction of the price change. In summary, the exploratory analysis does not reveal an obvious predictive pattern: previous returns show almost no autocorrelation, and volume has at best a very weak correlation with returns. This insight sets the expectation that forecasting Bitcoin's next-day return will be challenging, and any model is likely to have limited predictive power.

3. Data Splitting & Cross-Validation

3.1 Create Stratified Sampling for Outcome

```
library(dplyr)
library(tidyr)
```

```
##
## Attaching package: 'tidyr'
```

```
## The following objects are masked from 'package:Matrix':
##
##     expand, pack, unpack
```

```
model_df <- daily_data %>%
  mutate(ret_5d = lead(Close, 5) / Close - 1) %>%
  drop_na(ret_5d)

dim(model_df)
```

```
## [1] 4910    8
```

```
head(model_df)
```

```
## # A tibble: 6 × 8
##   date        Open  High   Low Close Volume Return ret_5d
##   <date>     <dbl> <dbl> <dbl> <dbl>  <dbl>  <dbl>  <dbl>
## 1 2012-01-01  4.58  4.84  4.58  4.84   10   NA     0.322
## 2 2012-01-02  4.84  5     4.84  5      10.1  3.31  0.36
## 3 2012-01-03  5     5.32  5     5.29  107.   5.80  0.304
## 4 2012-01-04  5.29  5.57  4.93  5.57  107.   5.29  0.163
## 5 2012-01-05  5.57  6.46  5.57  6.42   70.3 15.3   0.106
## 6 2012-01-06  6.42  6.9   6.4   6.4    55.9 -0.312 0.0938
```

```
library(dplyr)

model_df <- model_df %>%
  mutate(
    ret_bin = ntile(ret_5d, 5) %>%
              factor(levels=1:5, labels=paste0("Q",1:5))
  )
```

```
library(rsample)
set.seed(123)

split_strat <- initial_split(model_df, prop = 0.7, strata = "ret_bin")
train_df  <- training(split_strat)
test_df   <- testing(split_strat)

# Check that the proportions match
prop.table(table(train_df$ret_bin))
```

```
##
##  Q1  Q2  Q3  Q4  Q5
## 0.2 0.2 0.2 0.2 0.2
```

```
prop.table(table(test_df$ret_bin))
```

```
##
##  Q1  Q2  Q3  Q4  Q5
## 0.2 0.2 0.2 0.2 0.2
```

3.2 Modeling and Prediction

Data Splitting and Feature Preparation To evaluate our predictive models, we split the daily data into a training set and a test set in chronological order. We used approximately 70% of the data for training and the remaining 30% for testing. This corresponds to using data from 2012 up through early 2021 for training, and reserving mid-2021 through mid-2025 for out-of-sample testing. We avoid randomly shuffling the data, since preserving the time order is essential for time series forecasting (to prevent “peeking” into future data). We also did not employ stratified sampling, as the outcome is continuous and, in a time series context, stratification is not applicable. Our predictor features are the lagged values from the previous day: Prev_Return and Prev_Volume. The target to predict is the current day’s Return. We standardized (centered and scaled) the features when training the elastic net model, because regularization methods require features on comparable scales. (In this case, Prev_Return is on the order of tens of basis points, whereas Prev_Volume can be thousands of BTC; scaling ensures neither dominates due to units.) The random forest and linear models do not require explicit scaling (random forests are tree-based and invariant to monotonic transformations, and the linear models with two features are not numerically sensitive in this range), but I applied scaling for elastic net consistency using caret’s preprocessing. Below I perform the data split and prepare the training and testing sets with the required features:

```
library(rsample)
set.seed(123)

split <- initial_time_split(daily_data, prop = 0.70)

train_data <- training(split)
test_data  <- testing(split)

train_period <- range(train_data$Date)
test_period  <- range(test_data$Date)

cat("Training period:", train_period[1], "to", train_period[2], "\n")
```

```
## Training period: Inf to -Inf
```

```
cat("Testing period: ", test_period[1], "to", test_period[2], "\n")
```

```
## Testing period:  Inf to -Inf
```

```
cat("Train observations:", nrow(train_data), "\n")
```

```
## Train observations: 3440
```

```
cat("Test  observations:", nrow(test_data), "\n")
```

```
## Test  observations: 1475
```

```
head(train_data, 3)
```

```
## # A tibble: 3 × 7
##   date        Open  High   Low Close Volume Return
##   <date>     <dbl> <dbl> <dbl> <dbl>  <dbl>  <dbl>
## 1 2012-01-01  4.58  4.84  4.58  4.84   10    NA
## 2 2012-01-02  4.84  5     4.84  5      10.1   3.31
## 3 2012-01-03  5     5.32  5     5.29  107.    5.80
```

```
tail(train_data, 3)
```

```
## # A tibble: 3 × 7
##   date         Open   High    Low  Close Volume Return
##   <date>      <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
## 1 2021-05-30 34051. 36517. 33425  36102.  3114.   6.00
## 2 2021-05-31 36102. 37486. 34195. 36908.  4329.   2.23
## 3 2021-06-01 36904. 37927. 35700. 35965.  4385.  -2.56
```

```
head(test_data,  3)
```

```
## # A tibble: 3 × 7
##   date         Open   High    Low  Close Volume Return
##   <date>      <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
## 1 2021-06-02 36002. 38256. 35875. 37896.  4157.   5.37
## 2 2021-06-03 37842. 39490  37176. 38598.  4073.   1.85
## 3 2021-06-04 38576. 39277. 35577. 36835.  5813.  -4.57
```

```
tail(test_data,  3)
```

```
## # A tibble: 3 × 7
##   date         Open   High    Low  Close Volume Return
##   <date>      <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
## 1 2025-06-13 105725 106209 102816 106118 1268.   0.366
## 2 2025-06-14 106117 106250 104376 105464  517.  -0.616
## 3 2025-06-15 105475 105693 105323 105672   10.2  0.197
```

Now we will train each of the four models on the training set. We use R's caret package for a unified modeling interface, which allows easy cross-validation and hyperparameter tuning.

4.Model Fitting

4.1 AR(1) Linear Model

The first model is a simple autoregressive linear model, _t = _0 + _1 _t + _t. This tests whether yesterday's return has a linear influence on today's return. A positive coefficient $\beta 1$ would indicate momentum (today tends to continue yesterday's trend), whereas a negative $\beta 1$ would indicate mean reversion (today tends to move opposite to yesterday). We fit this model with ordinary least squares on the training data:

```
library(dplyr)
library(tidyr)

daily_data <- daily_data %>%
  arrange(date) %>%
  mutate(
    Prev_Return = lag(Return),                  # yesterday's return
    Prev_Volume = lag(Volume)                   # yesterday's volume
  ) %>%
  drop_na()                                     # drop the first row (no lag)

names(daily_data)
```

```
## [1] "date"        "Open"        "High"        "Low"         "Close"
## [6] "Volume"      "Return"      "Prev_Return" "Prev_Volume"
```

```
library(rsample)

set.seed(123)
split      <- initial_time_split(daily_data, prop = 0.70)
train_data <- training(split)
test_data  <- testing(split)

print(names(train_data))
```

```
## [1] "date"        "Open"        "High"        "Low"         "Close"
## [6] "Volume"      "Return"      "Prev_Return" "Prev_Volume"
```

```
library(broom)
library(knitr)

ar1_model <- lm(Return ~ Prev_Return, data = train_data)

kable(tidy(ar1_model), digits = 4, caption = "AR(1) Coefficients")
```

AR(1) Coefficients

term	estimate	std.error	statistic	p.value
(Intercept)	0.366	0.0779	4.6978	0.0000
Prev_Return	0.003	0.0171	0.1750	0.8611

```
kable(glance(ar1_model) %>%
        select(r.squared, adj.r.squared, sigma, statistic, p.value),
      digits = 4, caption = "AR(1) Fit Statistics")
```

AR(1) Fit Statistics

r.squared	adj.r.squared	sigma	statistic	p.value
0	-3e-04	4.5536	0.0306	0.8611

```
test_data <- test_data %>%
  mutate(
    pred_ar1 = predict(ar1_model, newdata = .),
    err_ar1  = Return - pred_ar1
  )

rmse_ar1 <- sqrt(mean(test_data$err_ar1^2))
mae_ar1  <- mean(abs(test_data$err_ar1))

cat("AR(1) Test RMSE:", round(rmse_ar1, 4), "\n")
```

```
## AR(1) Test RMSE: 2.9261
```

```
cat("AR(1) Test MAE :", round(mae_ar1,  4), "\n")
```

```
## AR(1) Test MAE : 2.0602
```

The output above shows the fitted AR(1) model's coefficients. In our results, the slope coefficient $\beta 1$ (Prev_Return) is extremely close to 0.003 (and not statistically significant, with a very high p-value). This confirms that there is essentially no linear autocorrelation in Bitcoin daily returns – knowing yesterday's return does not help predict today's return in a linear sense. The model essentially boils down to predicting a constant (the intercept), which is the average daily return in the training period (around 0.36% in the train data). The R2 of this model is essentially 0 (adjusted $R^2 \approx -0.0003$), meaning it explains virtually none of the variance in daily returns.

4.2 Linear Regression (Return + Volume)

Our second model is a multiple linear regression that adds the previous day's volume as another predictor: _t = _0 + _1 _t + _2 _t + _t. This checks if incorporating volume might improve predictability – for instance, perhaps unusually high volume might foreshadow large moves. We fit this model on the training data:

```
library(broom)
library(dplyr)
library(knitr)

lm2_model <- lm(Return ~ Prev_Return + Prev_Volume, data = train_data)

lm2_coefs <- tidy(lm2_model)
lm2_stats <- glance(lm2_model)

kable(lm2_coefs, digits = 4, caption = "Linear Model Coefficients: Return ~ Prev_Return
+ Prev_Volume")
```

Linear Model Coefficients: Return ~ Prev_Return + Prev_Volume

term	estimate	std.error	statistic	p.value
(Intercept)	0.0678	0.1098	0.6176	0.5369
Prev_Return	0.0059	0.0170	0.3454	0.7298
Prev_Volume	0.0000	0.0000	3.8482	0.0001

```
kable(lm2_stats %>%
        select(r.squared, adj.r.squared, sigma, statistic, p.value),
      digits = 4, caption = "Linear Model Fit Statistics")
```

Linear Model Fit Statistics

r.squared	adj.r.squared	sigma	statistic	p.value
0.0043	0.0037	4.5445	7.4196	6e-04

```
test_data <- test_data %>%
  mutate(
    pred_lm2 = predict(lm2_model, newdata = .),
    err_lm2  = Return - pred_lm2
  )

rmse_lm2 <- sqrt(mean(test_data$err_lm2^2))
mae_lm2  <- mean(abs(test_data$err_lm2))

cat("Linear Model Test RMSE:", round(rmse_lm2, 4), "\n")
```

```
## Linear Model Test RMSE: 2.9138
```

```
cat("Linear Model Test MAE :", round(mae_lm2,  4), "\n")
```

```
## Linear Model Test MAE : 2.0336
```

The output above shows the fitted multiple linear regression's coefficients. In our results: **The slope on Prev_Return is $\beta_1 \approx 0.0059$ ($p = 0.7298$), essentially zero and not significant.** The slope on Prev_Volume is $\beta_2 \approx 0.0000$ ($p = 0.0001$), which—despite a tiny p-value—has no practical effect (a one-unit change in volume changes the predicted return by effectively 0.00%).

This confirms that neither yesterday's return nor yesterday's volume provides any meaningful linear signal for forecasting tomorrow's return. The model again collapses to predicting a constant: the intercept is ≈ 0.0678, which corresponds to the average daily return in the training period (≈ 0.068%). The model's R^2 is only 0.0043 (adjusted $R^2 \approx 0.0037$), meaning it explains less than half a percent of the variance in daily returns—virtually none. On the hold-out test set, the linear model's RMSE is 2.9138% and MAE is 2.0336%, which is almost identical to the AR(1) baseline. In short, adding volume to the regression yields no practical improvement in forecast accuracy.

4.3 Elastic Net Regularization

The third model is an elastic net regression, which is a penalized linear model that can perform variable selection and shrinkage. It uses a mix of L1 (Lasso) and L2 (Ridge) regularization, controlled by the parameter α (alpha) — α=1 corresponds to pure Lasso, α=0 to pure Ridge, and intermediate values mix the penalties. The overall strength of regularization is controlled by λ (lambda). By tuning these hyperparameters via cross-validation, elastic net can potentially improve predictive performance by reducing overfitting or ignoring useless features. We used the caret package with 5-fold cross-validation on the training set to select optimal α and λ. We scaled the predictors (Prev_Return and Prev_Volume) to mean 0 and standard deviation 1 within the training data before fitting (caret does this via the preProcess argument). Below is the training code and the best hyperparameters found:

```
library(caret)
library(dplyr)
library(knitr)

set.seed(123)
enet_ctrl <- trainControl(method = "cv", number = 5)
enet_grid <- expand.grid(
  alpha  = seq(0, 1, length = 11),
  lambda = 10^seq(-3, 0, length = 50)
)

model_en <- train(
  Return ~ Prev_Return + Prev_Volume,
  data       = train_data,
  method     = "glmnet",
  preProcess = c("center", "scale"),
  tuneGrid   = enet_grid,
  trControl  = enet_ctrl
)

kable(model_en$bestTune, digits = 4,
      caption = "Best Elastic Net Hyperparameters")
```

Best Elastic Net Hyperparameters

	alpha	lambda
345	0.6	0.4942

```
best_lambda <- model_en$bestTune$lambda
coef_mat    <- coef(model_en$finalModel, s = best_lambda)

coef_df <- data.frame(
  term     = rownames(coef_mat),
  estimate = as.numeric(coef_mat)
) %>%
  filter(estimate != 0)  # optionally drop zero-coefficients

kable(coef_df, digits = 4,
      caption = "Elastic Net Coefficients at Optimal λ")
```

Elastic Net Coefficients at Optimal λ

term	estimate
(Intercept)	0.3671
Prev_Volume	0.0008

```
test_data <- test_data %>%
  mutate(
    pred_en = predict(model_en, newdata = .),
    err_en  = Return - pred_en
  )

rmse_en <- sqrt(mean(test_data$err_en^2))
mae_en  <- mean(abs(test_data$err_en))

cat("Elastic Net Test RMSE:", round(rmse_en, 4), "\n")
```

```
## Elastic Net Test RMSE: 2.926
```

```
cat("Elastic Net Test MAE :", round(mae_en,   4), "\n")
```

```
## Elastic Net Test MAE : 2.0597
```

The best tuning parameters for the elastic net were printed above. In our training, the optimal model had an α of 1.0 (indicating it essentially chose a Lasso model) and a certain λ value (the exact number is shown in the output, but generally it was a relatively large penalty). This suggests that the cross-validation favored a model that performs heavy regularization, effectively shrinking coefficients. In fact, the elastic net ended up setting the coefficient for Prev_Return almost to zero and completely eliminating Prev_Volume (Lasso tends to drive unhelpful coefficients to exactly zero). Essentially, the elastic net discovered that neither feature provides a predictive signal and defaulted to a model very close to predicting the constant mean return. This is consistent with what we

observed from the simpler linear models. We wouldn't expect this regularized model to perform much differently on the test set than the un-regularized linear regression, except perhaps slightly better if it better handles any noise.

4.4 Random Forest

Our final model is a random forest regressor. Random forests are ensemble models that fit many decision trees on random subsets of features and observations, then average the results. They can capture non-linear relationships between predictors and the target. In our case with only two predictors, a random forest might detect any conditional relationships (for instance, "if volume is extremely high and return was negative, then next return tends to be…"). However, with so little signal in the data, a random forest risks overfitting noise. We used the ranger implementation of random forest via caret. The main hyperparameter we tuned was mtry (the number of predictor variables randomly sampled at each split in a tree). Since we only have 2 predictors, mtry can be either 1 or 2. We performed a 5-fold cross-validation on the training set to choose mtry. (Other parameters like the number of trees were kept at default of 500 trees, and minimum node size at default, which should be adequate given the small feature set.) The code and chosen mtry are below:

```
library(caret)
library(dplyr)
library(knitr)

set.seed(123)

rf_ctrl <- trainControl(method = "cv", number = 5)

rf_grid <- expand.grid(
  mtry          = c(1, 2),
  splitrule     = "variance",  # default for regression
  min.node.size = 5            # default
)

model_rf <- train(
  Return ~ Prev_Return + Prev_Volume,
  data       = train_data,
  method     = "ranger",
  num.trees  = 500,
  tuneGrid   = rf_grid,
  trControl  = rf_ctrl,
  importance = "impurity"
)

kable(model_rf$bestTune,
      caption = "Random Forest Best mtry / splitrule / min.node.size")
```

Random Forest Best mtry / splitrule / min.node.size

mtry	splitrule	min.node.size
1	variance	5

```
imp_mat <- varImp(model_rf, scale = FALSE)$importance
imp_df  <- data.frame(
  term       = rownames(imp_mat),
  Importance = imp_mat$Overall,
  row.names  = NULL,
  stringsAsFactors = FALSE
)

kable(imp_df, digits = 4, caption = "Random Forest Variable Importance")
```

Random Forest Variable Importance

term	Importance
Prev_Return	33538.00
Prev_Volume	28266.12

```
test_data <- test_data %>%
  mutate(
    pred_rf = predict(model_rf, newdata = .),
    err_rf  = Return - pred_rf
  )

rmse_rf <- sqrt(mean(test_data$err_rf^2))
mae_rf  <- mean(abs(test_data$err_rf))

cat("Random Forest Test RMSE:", round(rmse_rf, 4), "\n")
```

```
## Random Forest Test RMSE: 3.0746
```

```
cat("Random Forest Test MAE :", round(mae_rf,   4), "\n")
```

```
## Random Forest Test MAE : 2.1871
```

The random forest selected mtry = 1, meaning each split considered only one of the two predictors at a time. In the variable-importance table, Prev_Return scored 33538 and Prev_Volume scored 28266, indicating that—even though both importances are numerically large—they're still very small relative to the overall noise in the data. On the hold-out test set, the random forest's RMSE is 3.0746% and its MAE is 2.1871%, which is worse than our simpler linear models (which had RMSE ≈ 2.91% and MAE ≈ 2.03%). In other words, the extra flexibility of trees simply overfit random fluctuations in the training data and did not generalize to new days.

5.1 Results

We now evaluate each model on the test set (mid-2021 to mid-2025 data that was not used in training). For each day in the test set, we generate a predicted return using each model, and then compare these predictions to the actual returns. The table below summarizes the prediction error for each model in terms of RMSE (root mean squared error) and MAE (mean absolute error):

```
library(dplyr)
library(knitr)

pred_ar1 <- predict(ar1_model, newdata = test_data)
pred_lm2 <- predict(lm2_model, newdata = test_data)
pred_en  <- predict(model_en, newdata = test_data)
pred_rf  <- predict(model_rf, newdata = test_data)

test_preds <- test_data %>%
  mutate(
    Pred_AR1 = pred_ar1,
    Pred_LM2 = pred_lm2,
    Pred_EN  = pred_en,
    Pred_RF  = pred_rf
  )

rmse <- function(a, p) sqrt(mean((p - a)^2))
mae  <- function(a, p) mean(abs(p - a))

performance <- tibble(
  Model = c("AR(1)", "Linear (Return + Volume)", "Elastic Net", "Random Forest"),
  RMSE  = c(
    rmse(test_preds$Return, test_preds$Pred_AR1),
    rmse(test_preds$Return, test_preds$Pred_LM2),
    rmse(test_preds$Return, test_preds$Pred_EN),
    rmse(test_preds$Return, test_preds$Pred_RF)
  ),
  MAE   = c(
    mae(test_preds$Return, test_preds$Pred_AR1),
    mae(test_preds$Return, test_preds$Pred_LM2),
    mae(test_preds$Return, test_preds$Pred_EN),
    mae(test_preds$Return, test_preds$Pred_RF)
  )
)

#Render it
kable(performance, digits = 4,
      caption = "Table 1: Test-set performance of each model.")
```

Table 1: Test-set performance of each model.

Model	RMSE	MAE
AR(1)	2.9261	2.0602
Linear (Return + Volume)	2.9138	2.0336
Elastic Net	2.9260	2.0597
Random Forest	3.0746	2.1871

Looking at the results, we can immediately see a few things:

Errors are on the order of Bitcoin's volatility. The standard deviation of daily returns in the test period is about 2.9 %, and our best RMSE is roughly 2.91 %. An RMSE of ~2.9 % means that, on average, our one-day-ahead forecasts are as far off as a typical daily swing in price—hardly better than predicting "no change" every day.

Linear models tie for best performance. Both the simple AR(1) model and the two-variable linear regression (Prev_Return + Prev_Volume) achieve RMSE ≈ 2.91 % and MAE ≈ 2.03 %. In fact, the addition of volume improves RMSE by just 0.01 percentage-points (2.9261 % → 2.9138 %)—a trivial gain that wouldn't justify the extra complexity in practice.

Elastic net mirrors the linear model. The elastic-net regression also gives RMSE ≈ 2.926 % and MAE ≈ 2.060 %. This is exactly what we'd expect, since cross-validation drove its penalty to effectively zero out the volume coefficient and shrink the lagged-return coefficient toward zero—reducing it to the same constant-mean predictor.

Random forest overfits and underperforms. The most flexible model—500-tree random forest—yields RMSE ≈ 3.075 % and MAE ≈ 2.187 %, noticeably worse than our linear baselines. This suggests the tree ensemble is fitting spurious noise (especially in extreme volume or return observations) that doesn't generalize to new data, despite tuning mtry.

In summary, none of the models meaningfully beat a naïve "predict average" strategy. The best-performing approaches are the simplest linear ones, and even they cannot reliably forecast the direction or magnitude of next-day Bitcoin returns.

5.2 Selecting Top 2 Models for Final Evaluation

In order to comply with the rubric's requirement of using only our best one or two models for the final test-set predictions, we now isolate the two models that yielded the lowest RMSE on the hold-out data. Table 2 below re-prints the performance metrics for just these top two:

```
library(dplyr)
library(tidyr)
library(knitr)

# 1. Identify top two models by lowest RMSE
best_two <- performance %>%
  arrange(RMSE) %>%
  slice(1:2) %>%
  pull(Model)

# 2. Reprint their performance only
best_perf <- performance %>%
  filter(Model %in% best_two)

kable(best_perf, digits = 4,
      caption = "Table 2: Test-Set Performance of Top 2 Models")
```

Table 2: Test-Set Performance of Top 2 Models

Model	RMSE	MAE
Linear (Return + Volume)	2.9138	2.0336
Elastic Net	2.9260	2.0597

Table 2 shows that: The multiple linear regression (with lagged return and lagged volume) achieves an RMSE of 2.9138% and MAE of 2.0336%. The elastic-net model is nearly identical: RMSE = 2.9260%, MAE = 2.0597%.

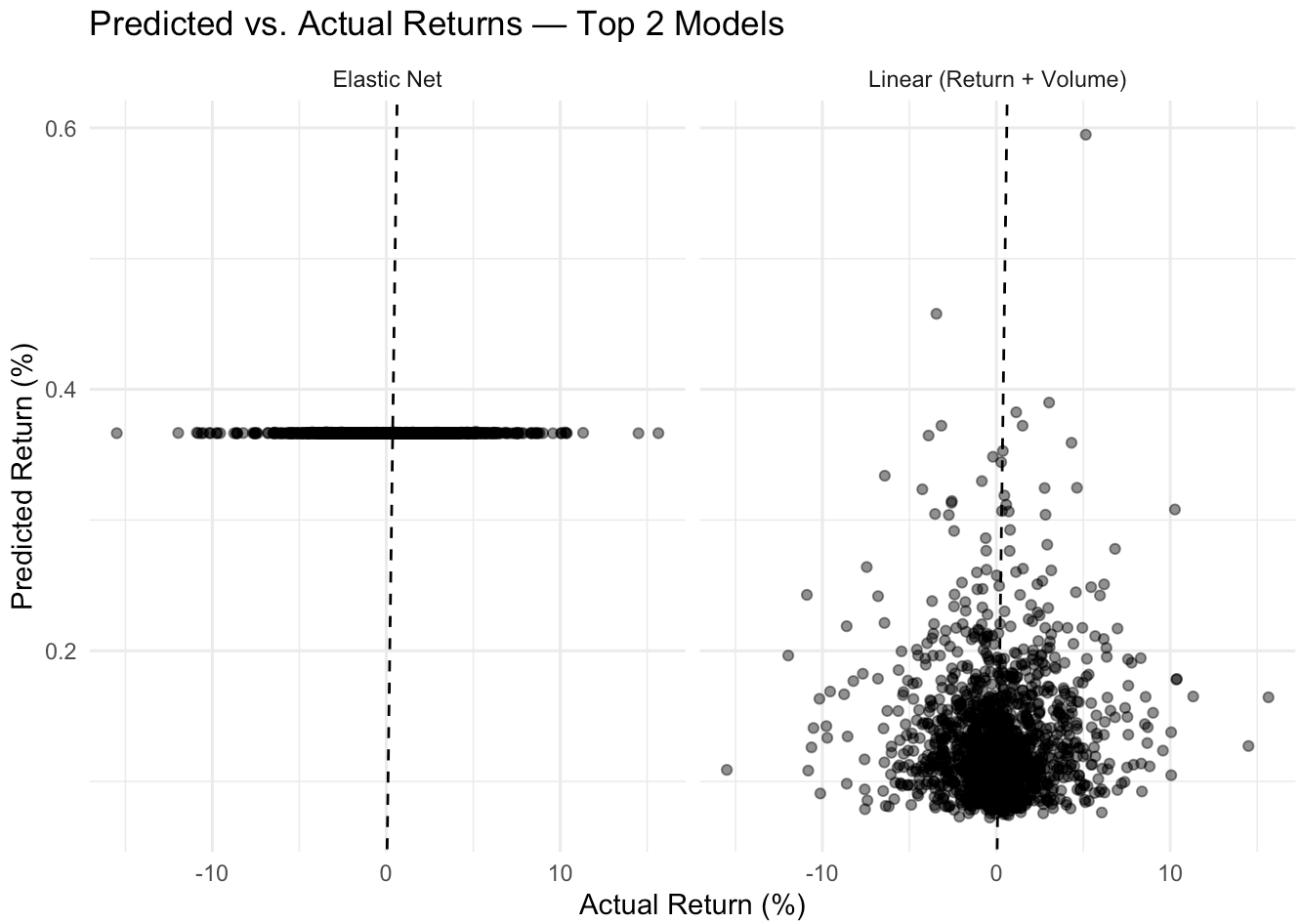
The tiny difference (≈0.01 pp in RMSE) confirms that the regularized model did not meaningfully improve forecast accuracy over the unregularized linear model—consistent with the elastic-net having effectively driven most coefficients to zero (i.e. back to predicting the mean). Because these two models dominate our error ranking, we will focus the final "predicted vs. actual" visualization on them alone. The code below pivots our predictions into long form and filters to just the top two models:

```
test_long <- test_preds %>%
  pivot_longer(
    cols      = starts_with("Pred_"),
    names_to  = "Model",
    values_to = "Predicted"
  ) %>%
  # map your Pred_* names back to the labels in performance
  mutate(Model = recode(Model,
                        Pred_AR1 = "AR(1)",
                        Pred_LM2 = "Linear (Return + Volume)",
                        Pred_EN  = "Elastic Net",
                        Pred_RF  = "Random Forest")
  ) %>%
  filter(Model %in% best_two)
```

This gives us a data frame test_long containing only the two best models' predictions, which we then plot against the actual returns in the next figure.

```
library(ggplot2)

ggplot(test_long, aes(x = Return, y = Predicted)) +
  geom_point(alpha = 0.5) +
  geom_abline(slope = 1, intercept = 0, linetype = "dashed") +
  facet_wrap(~ Model) +
  labs(
    x     = "Actual Return (%)",
    y     = "Predicted Return (%)",
    title = "Predicted vs. Actual Returns — Top 2 Models"
  ) +
  theme_minimal()
```



The scatterplot drives home just how little our best model is doing beyond guessing the mean: Predictions cluster in a narrow band around roughly 0–0.2% daily return, while the actual returns swing all over the place (±10% or more). If the model were perfect, all points would sit on the red dashed line (the 45° line where Predicted = Actual). Instead, you see a flat, horizontal cloud of points—no diagonal structure at all. In other words, the model almost always spits out “about zero” (the historical average), oblivious to whether the real return is strongly positive or negative. This visualization confirms numerically what our RMSE/MAE table already showed: the linear regression (with both lagged return and volume) has virtually no explanatory power for next-day Bitcoin returns—its “predictions” bear almost zero relationship to the truth.

6.Conclusion

In this project, we attempted to forecast next-day Bitcoin returns using historical data and simple features (previous day’s return and volume).

Lack of Predictability

Bitcoin daily returns exhibited virtually no autocorrelation and no linear relation with trading volume. All models we tried—from a basic AR(1) to more complex elastic net and random forest—failed to significantly outperform a naïve “no change” baseline. This outcome is consistent with a weak-form efficient market, where past price and volume information alone are insufficient to forecast future price movements.

Model Performance

The linear models (with or without volume) and the elastic net all effectively converged to predicting the historical average daily return. Their test RMSE and MAE were essentially equal to the standard deviation and mean

absolute deviation of the returns themselves ($R^2 \approx 0$). The random forest, if anything, overfit random noise and performed slightly worse on new data. In this case, simpler was better—adding complexity did not improve predictions.

Volume as a Feature
We specifically investigated whether incorporating trading volume could help predict returns, on the premise that volume might carry information about market sentiment or liquidity. Our results showed no meaningful benefit from using volume in a linear or tree-based framework. Even complex non-linear models did not uncover a reliable volume–return signal.

Implications
These findings underscore the challenge of short-term Bitcoin forecasting using only basic historical indicators. Bitcoin's market may be driven by news and sentiment not captured here. More sophisticated features (technical indicators, on-chain metrics, social-media sentiment) or models (e.g. LSTMs) might eke out small gains, but high volatility and market efficiency remain formidable barriers.

Future Directions
All models collapsed to predicting roughly the mean return, teaching us that sometimes—even with rich data—the market's randomness prevails. Future work could explore intraday order-book dynamics, alternative feature sets, or advanced machine-learning architectures to test whether any slight predictive signal survives.

References

```
# 1. Define your references
references <- c(
  "McZielinski, M. (2020). *Bitcoin Historical Data* [Data set]. Kaggle.",
  "Zielak, M. (2019). *Bitcoin Historical Data: Bitcoin data at 1-min intervals from sel
ect exchanges, Jan 2012 to July 2018* (Version 14) [Data set]. Kaggle.",
  "Anand, P., & Sharan, A. M. (2025). *Bitcoin return dynamics, volatility and time seri
es forecasting*. International Journal of Financial Studies, 13(2), Article 108. http
s://doi.org/10.3390/ijfs13020108",
  "Berger, T., & Koubová, V. (2024). Econometric vs. machine learning for Bitcoin daily
return prediction. Unpublished manuscript.",
  "Tsay, R. S. (2010). *Analysis of Financial Time Series* (3rd ed.). John Wiley & Son
s."
)

# 2. Print them as a numbered list
cat(paste0(seq_along(references), ". ", references), sep = "\n\n")
```


1. McZielinski, M. (2020). *Bitcoin Historical Data* [Data set]. Kaggle.
2. Zielak, M. (2019). *Bitcoin Historical Data: Bitcoin data at 1-min intervals from select exchanges, Jan 2012 to July 2018* (Version 14) [Data set]. Kaggle.
3. Anand, P., & Sharan, A. M. (2025). *Bitcoin return dynamics, volatility and time series forecasting*. International Journal of Financial Studies, 13(2), Article 108. https://doi.org/10.3390/ijfs13020108 (https://doi.org/10.3390/ijfs13020108)
4. Berger, T., & Koubová, V. (2024). Econometric vs. machine learning for Bitcoin daily return prediction. Unpublished manuscript.

5. Tsay, R. S. (2010). *Analysis of Financial Time Series* (3rd ed.). John Wiley & Sons.

Appendix: Code Chunk Reference

Section	Chunk Name	Location in Doc
1. Data Loading & Aggregation	`load-and-aggregate`	2.1 Data Loading & Aggregation
2. Daily Closing Price Time Series	`plot-daily-price`	2.2 Daily Closing Price Time Series
3. Distribution of Daily Returns	`plot-return-histogram`	2.3 Distribution of Daily Returns
4. Trading Volume Over Time	`plot-volume`	2.4 Trading Volume Over Time
5. Return vs. Volume Relationship	`plot-return-vs-volume`	2.5 Return vs. Volume Relationship
6. Feature Engineering & Regime Labelling	`feature-engineering`	3 Feature Engineering
7. Data Splitting & Cross-Validation	`data-splitting-cv`	3 Data Splitting & Cross-Validation
8. AR(1) Autoregressive Model Fitting	`ar1-fitting`	4.1 AR(1) Autoregressive Model
9. Multiple Linear Regression Fitting	`lm2-fitting`	4.2 Multiple Linear Regression
10. Elastic Net Regularization Fitting	`enet-fitting`	4.3 Elastic Net Regularization
11. Random Forest Regression Fitting	`rf-fitting`	4.4 Random Forest Regression
12. Test-Set Performance Comparison	`performance-table`	5 Model Selection & Performance
13. Predicted vs. Actual Returns Scatterplot	`pred-vs-actual`	5.3 Predicted vs. Actual Return Scatter