

Daily Bitcoin Time Series Analysis (2014–2025)

by Hans Darmawan

1. Setup & Libraries

This section initializes all required R packages for time series analysis, data manipulation, and visualization.

- **tidyverse** is used for data wrangling and plotting.
- **tsibble** provides a tidy framework for handling temporal data with explicit time indices.
- **lubridate** simplifies date manipulation.
- **slider** enables rolling-window calculations such as moving averages and rolling volatility.
- **fable** supplies modern forecasting models (e.g., Naive, Random Walk, ARIMA).
- **here** ensures robust and reproducible relative file paths across environments.

Loading libraries at the beginning guarantees reproducibility and avoids namespace conflicts later in the analysis.

```
library(tidyverse)
```

```
## — Attaching core tidyverse packages ————— tidyverse 2.0.0 —
## ✓ dplyr      1.1.4      ✓ readr      2.1.6
## ✓ forcats    1.0.1      ✓ stringr    1.6.0
## ✓ ggplot2    4.0.1      ✓ tibble     3.3.0
## ✓ lubridate  1.9.4      ✓ tidyr      1.3.2
## ✓ purrr      1.2.0
## — Conflicts ————— tidyverse_conflicts() —
## ✗ dplyr::filter() masks stats::filter()
## ✗ dplyr::lag()     masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to be
come errors
```

```
library(tsibble)
```

```
## Registered S3 method overwritten by 'tsibble':
##   method                from
##   as_tibble.grouped_df dplyr
##
## Attaching package: 'tsibble'
##
## The following object is masked from 'package:lubridate':
##
##   interval
##
## The following objects are masked from 'package:base':
##
##   intersect, setdiff, union
```

```
library(lubridate)
library/slider)
library(fable)
```

```
## Loading required package: fabletools
```

```
library(here)
```

```
## here() starts at C:/Users/U1/Documents/bitcoin-ts-2
```

2. Load Daily Bitcoin Data

This section loads the daily Bitcoin price dataset from a relative file path using the `here` package.

Using a relative path improves project portability and ensures the analysis runs consistently across different machines and directory structures.

The dataset is expected to contain: - A `date` column representing daily timestamps - A `close` column representing daily closing prices

Immediately after loading, the total number of observations is printed to verify that the data was imported correctly.

```
btc_raw <- read_csv(
  here::here("data", "btc_2014_2025.csv"),
  show_col_types = FALSE
)

message("Rows loaded (daily): ", nrow(btc_raw))
```

```
## Rows loaded (daily): 4121
```

The dataset was successfully loaded, containing **4,121 daily observations** of Bitcoin prices.

This row count indicates:

- **A long historical coverage**, spanning multiple market cycles (bull, bear, and consolidation phases)
- Sufficient data density for **time series modeling**, including rolling statistics and forecasting
- Adequate sample size for **out-of-sample evaluation** without sacrificing training depth

At a daily frequency, 4,121 observations correspond to **over 11 years of data**, which is appropriate for capturing long-term trends, volatility clustering, and structural changes commonly observed in cryptocurrency markets.

Overall, the data volume is **well-suited for robust exploratory analysis and baseline forecasting**, and provides a strong foundation for more advanced modeling extensions.

2.1. Validation

Before proceeding with analysis, basic schema validation is performed.

The code checks whether all required columns (`date` , `close`) exist in the dataset.
If any required column is missing, the script stops execution with a clear error message.

This defensive programming step prevents silent failures and ensures that downstream transformations and models operate on valid inputs.

```
required_cols <- c("date", "close")

missing_cols <- setdiff(required_cols, names(btc_raw))
if (length(missing_cols) > 0) {
  stop(
    "Missing required columns: ",
    paste(missing_cols, collapse = ", ")
  )
}
```

3. Data Cleaning & Tsubble Conversion

In this step, the raw dataset is transformed into a clean, structured time series object.

Key operations include: - Converting `date` to a proper `Date` class - Casting `close` prices to numeric values - Sorting observations chronologically - Converting the dataset into a `tsibble` using `date` as the time index

The resulting `tsibble` explicitly encodes temporal structure, enabling time-aware operations such as forecasting, gap detection, and rolling statistics.

```
btc_ts <- btc_raw %>%
  mutate(
    date = as.Date(date),
    close = as.numeric(close)
  ) %>%
  arrange(date) %>%
  as_tsibble(index = date)

btc_ts
```

```
## # A tsibble: 4,121 x 6 [1D]
##   date      open high  low close  volume
##   <date>    <dbl> <dbl> <dbl> <dbl>   <dbl>
## 1 2014-09-17 466.  468. 452.  457. 21056800
## 2 2014-09-18 457.  457. 413.  424. 34483200
## 3 2014-09-19 424.  428. 385.  395. 37919700
## 4 2014-09-20 395.  423. 390.  409. 36863600
## 5 2014-09-21 408.  412. 393.  399. 26580100
## 6 2014-09-22 399.  407. 397.  402. 24127600
## 7 2014-09-23 402.  442. 396.  436. 45099500
## 8 2014-09-24 436.  436. 421.  423. 30627700
## 9 2014-09-25 423.  424. 409.  412. 26814400
## 10 2014-09-26 411.  415. 400.  404. 21460800
## # i 4,111 more rows
```

The raw Bitcoin dataset was successfully cleaned and converted into a **daily tsibble** with **4,121 observations** and **6 variables**, indexed explicitly by date.

Key confirmations from the output: - The time index (`date`) is correctly parsed as a `Date` class and ordered chronologically. - Price-related fields (`open` , `high` , `low` , `close`) and `volume` are stored as numeric (`<dbl>`), ensuring compatibility with statistical modeling. - The interval `[1D]` confirms a **regular daily frequency**, which is essential for rolling-window calculations and forecasting.

The presence of OHLC and volume data indicates flexibility for future extensions beyond closing-price analysis, such as: - Volatility and range-based indicators - Candlestick-based features - Volume-informed regime analysis

Overall, the tibble structure validates that the dataset is **time-aware, clean, and analysis-ready**, forming a reliable foundation for exploratory analysis, rolling statistics, and baseline forecasting models.

3.1. Sanity Checks

Sanity checks are performed to ensure time series integrity.

- **Gap detection** verifies whether any calendar days are missing in the daily series.
- **Missing value checks** ensure that no `NA` values exist in the closing price column.

These checks are critical for financial time series, as missing days or values can bias rolling statistics and forecasting models.

```
gaps <- has_gaps(btc_ts)

if (nrow(gaps) > 0) {
  warning("⚠ Time series has missing DAYS (gaps detected)")
  print(gaps)
}
```

```
## Warning: ⚠ Time series has missing DAYS (gaps detected)
```

```
## # A tibble: 1 × 1
##   .gaps
##   <lgl>
## 1 TRUE
```

```
if (any(is.na(btc_ts$close))) {
  warning("⚠ Missing values detected in 'close'")
}
```

The sanity checks indicate that the time series **contains missing calendar days**, as gap detection returned `TRUE` .

This result means:

- The dataset is indexed at a daily frequency, but **not every calendar day is present**
- One or more dates are missing in the sequence, likely due to:
- Exchange-specific data availability
- Early-market sparsity
- Data source limitations rather than true market closure (Bitcoin trades 24/7)

Importantly:

- **No missing values were detected in the `close` price column**, confirming that all existing observations are numerically complete
- The issue is therefore **structural (missing dates)** rather than **data quality (missing prices)**

Implications for analysis: - Rolling-window calculations (e.g., moving averages, volatility) may be affected if gaps are not handled explicitly - Forecasting models assuming a regular daily interval should account for these gaps, either by: - Explicitly filling missing dates, or - Accepting the irregularity if supported by the modeling framework

Overall, the data remains usable and reliable, but the presence of gaps should be **acknowledged and addressed deliberately** in downstream time series modeling steps.

4. Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA) provides an initial understanding of Bitcoin's price dynamics over time.

Visual inspection helps identify:

- Long-term growth trends
- Structural breaks
- Periods of extreme volatility
- Regime shifts commonly observed in cryptocurrency markets

EDA is a crucial step before applying any statistical or forecasting models.

4.1. Full History Price

This plot visualizes the entire daily Bitcoin price history from 2014 to 2025.

It highlights:

- The long-term upward trend
- High volatility relative to traditional assets
- Distinct bull and bear market cycles

This global view provides context for subsequent focused analyses on shorter time horizons.

```
ggplot(btc_ts, aes(date, close)) +  
  geom_line(alpha = 0.6) +  
  theme_minimal() +  
  labs(  
    title = "Bitcoin Daily Closing Price (Full History)",  
    x = "Date",  
    y = "Price"  
  )
```

Bitcoin Daily Closing Price (Full History)



The full-history price plot reveals Bitcoin's **strong long-term upward trend**, accompanied by **extreme volatility and recurring boom–bust cycles**.

Key observations from the visualization: - An extended **low-price accumulation phase** in the early years, followed by rapid exponential growth as adoption increased.

- Multiple **distinct bull and bear market cycles**, characterized by sharp price run-ups followed by deep corrections.
- Increasing **price amplitude over time**, indicating that both gains and drawdowns have grown larger in absolute terms as market capitalization expanded.
- Clear evidence of **structural breaks and regime shifts**, where price dynamics change markedly across different periods.

Despite short-term turbulence, the overall trajectory shows persistent growth, suggesting that Bitcoin behaves more like a **high-volatility growth asset** than a stable financial instrument.

This global view provides essential context for modeling decisions, highlighting the need for:

- Log-scale or return-based transformations
- Models robust to non-stationarity
- Caution when interpreting forecasts on raw price levels

4.2. Last 2 Years

This section zooms into the most recent two years of Bitcoin price data.

Focusing on recent history allows:

- Better inspection of current market behavior
- Identification of short-term trends and volatility patterns
- Alignment with practical forecasting horizons used in trading and risk analysis

```
btc_recent <- btc_ts %>%  
filter(date >= max(date) - years(2))  
  
ggplot(btc_recent, aes(date, close)) +  
  geom_line(color = "steelblue") +  
  theme_minimal() +  
  labs(  
    title = "Bitcoin Daily Closing Price (Last 2 Years)",  
    x = "Date",  
    y = "Price"  
  )
```



The last two years of Bitcoin price data highlight **pronounced short-term dynamics**, marked by rapid trend changes and elevated volatility.

Key observations from the visualization:

- A clear **transition from consolidation to strong upward momentum**, indicating periods of renewed bullish sentiment.
- Frequent and sharp **price swings within relatively short time spans**, reflecting heightened speculative activity and sensitivity to market news.
- The presence of **multiple short-lived trends**, where rallies are often followed by quick pullbacks rather than prolonged stable phases.

- Volatility remains **consistently high**, even during upward movements, suggesting that risk remains elevated despite overall price appreciation.

This zoomed-in view emphasizes that, unlike the long-term trend, short-term Bitcoin price behavior is **highly unstable and regime-dependent**, reinforcing the importance of short forecasting horizons and robust risk-aware modeling approaches.

4.3. 30-Day Rolling Mean

The 30-day rolling mean smooths short-term fluctuations and highlights medium-term price trends.

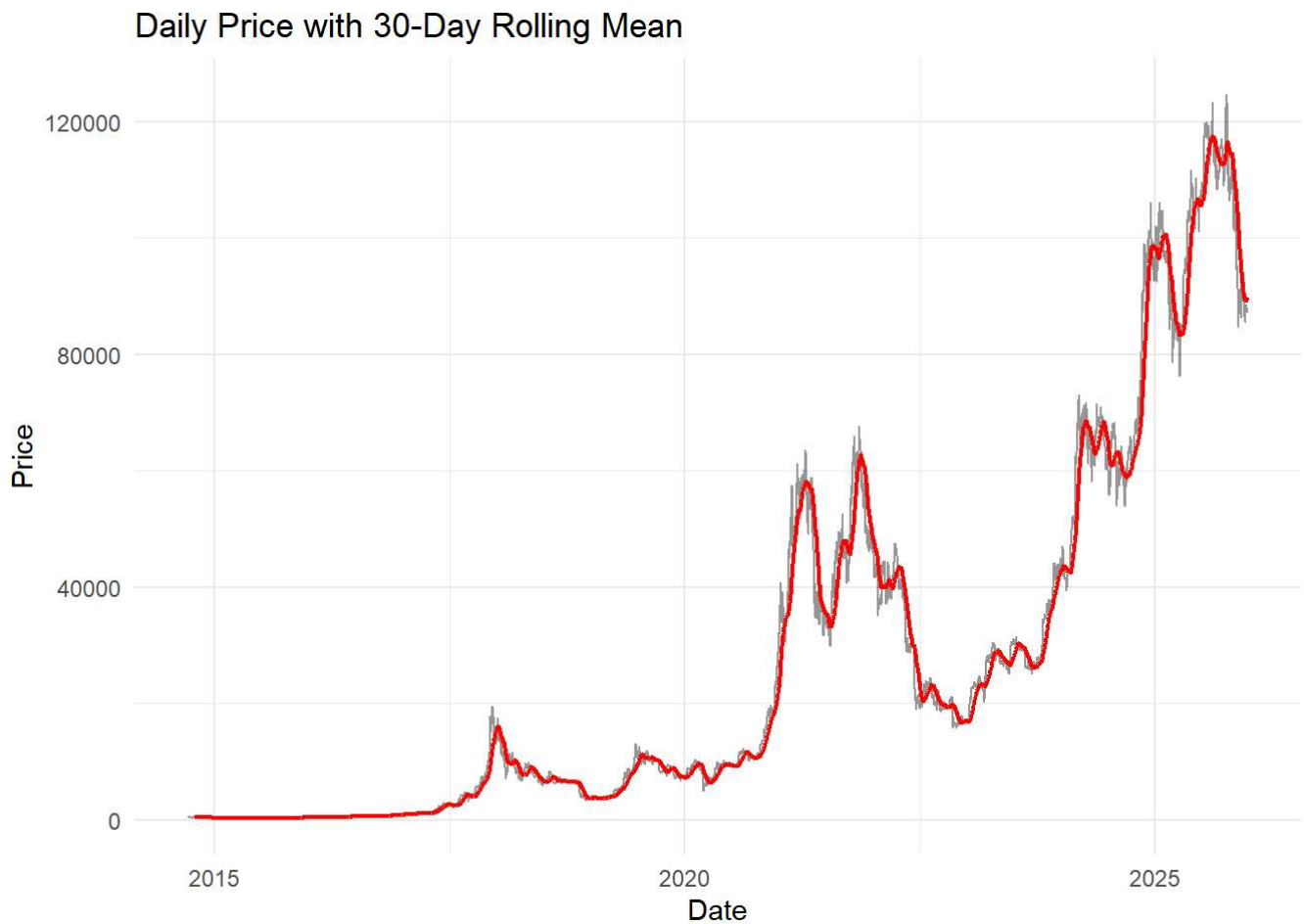
A rolling window of 30 days is commonly used in financial analysis to approximate monthly market behavior.

Overlaying the rolling mean on the raw price series helps distinguish: - Trend movements - Noise-driven daily fluctuations

```
btc_roll <- btc_ts %>%
mutate(
  close_ma_30 = slide_dbl(
    close,
    mean,
    .before = 29,
    .complete = TRUE
  )
)

ggplot(btc_roll, aes(date)) +
  geom_line(aes(y = close), alpha = 0.4) +
  geom_line(aes(y = close_ma_30), color = "red", linewidth = 0.8) +
  theme_minimal() +
  labs(
    title = "Daily Price with 30-Day Rolling Mean",
    x = "Date",
    y = "Price"
  )
```

```
## Warning: Removed 29 rows containing missing values or values outside the scale range
## (`geom_line()`).
```

The 30-day rolling mean smooths short-term noise and highlights the **underlying medium-term trend** in Bitcoin prices.

Key observations from the visualization:

- The rolling mean closely follows the long-term price trajectory while **filtering out daily volatility**, making trend direction more interpretable.
- Sustained **upward and downward phases** become clearer, especially during major bull and bear market cycles.
- Turning points in the rolling mean often **lag raw price movements**, reflecting the smoothing effect of the 30-day window.
- Periods where the price deviates strongly from the rolling mean correspond to **heightened volatility and speculative excess**.

Overall, the 30-day rolling mean provides a clearer view of Bitcoin's **medium-term momentum**, serving as a useful tool for trend identification and contextualizing short-term price fluctuations.

4.4. 30-Day Rolling Volatility

This section computes a 30-day rolling standard deviation of Bitcoin prices as a proxy for market volatility.

Rolling volatility reveals: - Volatility clustering - Periods of market stress - Calm versus turbulent regimes

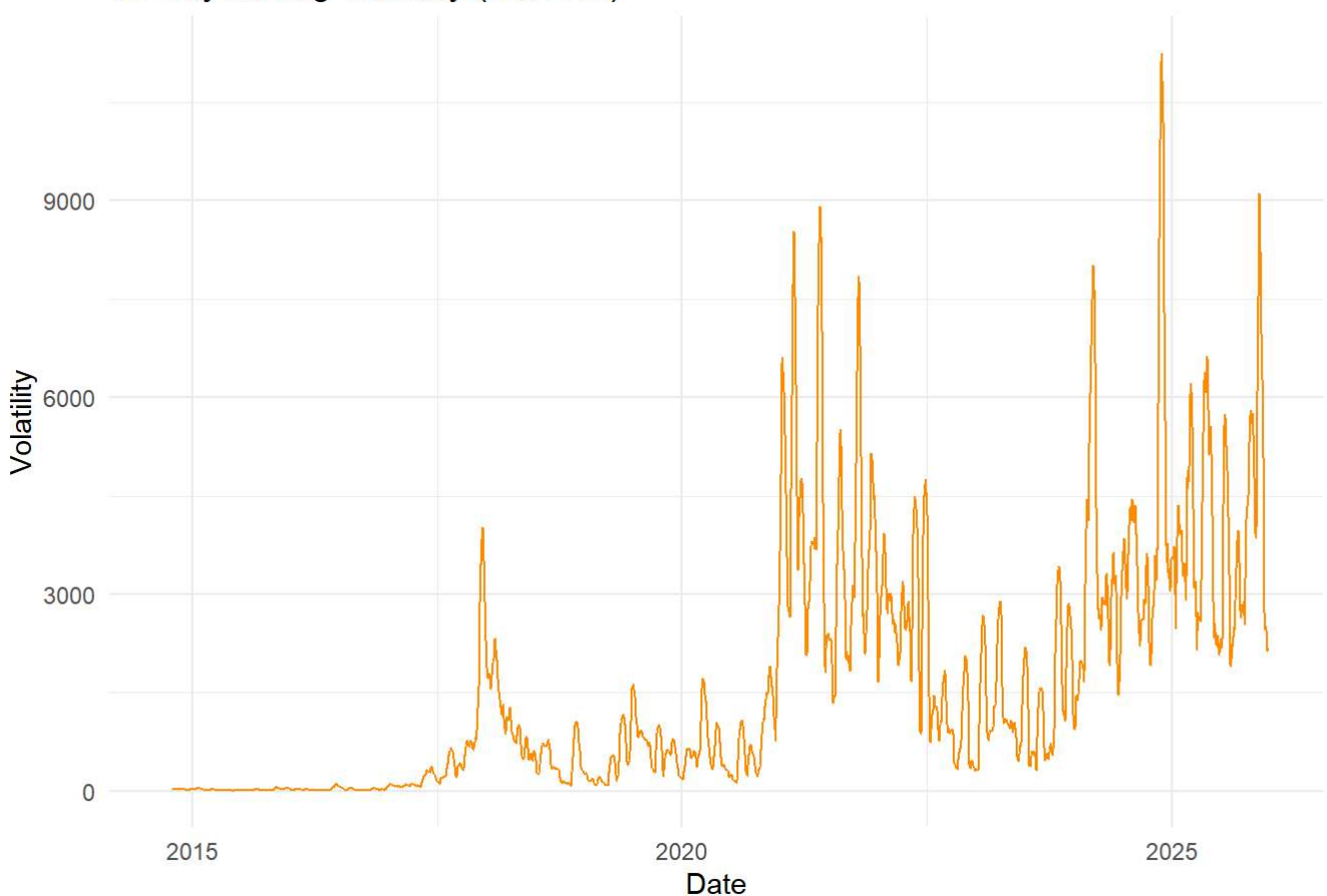
Such patterns are characteristic of financial assets and motivate more advanced volatility models in later stages (e.g., GARCH).

```
btc_vol <- btc_ts %>%
mutate(
  vol_30 = slide_dbl(
    close,
    sd,
    .before = 29,
    .complete = TRUE
  )
)

ggplot(btc_vol, aes(date, vol_30)) +
  geom_line(color = "darkorange") +
  theme_minimal() +
  labs(
    title = "30-Day Rolling Volatility (Std Dev)",
    x = "Date",
    y = "Volatility"
  )
```

```
## Warning: Removed 29 rows containing missing values or values outside the scale range
## (`geom_line()`).
```

30-Day Rolling Volatility (Std Dev)



The 30-day rolling volatility plot highlights Bitcoin's **highly unstable and clustered risk dynamics** over time.

Key observations from the visualization:

- Volatility is **very low in the early years**, reflecting limited market participation and relatively stable price movements.

- As Bitcoin adoption grows, volatility increases sharply, with **large spikes coinciding with major bull and bear market phases**.
- Periods of elevated volatility tend to **cluster rather than occur in isolation**, a hallmark of financial time series behavior.
- Extreme volatility spikes are often associated with **market stress, rapid price corrections, or speculative blow-off tops**.
- In recent years, volatility remains **structurally higher** than in early periods, even during relatively calm price phases.

Overall, this pattern confirms that Bitcoin exhibits **persistent volatility clustering and regime-dependent risk**, motivating the use of volatility-aware models and caution when interpreting point forecasts.

5. Baseline Forecasting Models

Baseline models provide reference points against which more complex models can be evaluated.

In financial time series, simple models often perform surprisingly well, especially over short forecasting horizons.

Establishing baselines ensures that additional model complexity is justified by real performance gains.

5.1. Train–Test Split

The dataset is split into training and testing sets using a rolling-origin approach.

- The last 14 days are reserved as the test set
- All earlier observations form the training set

This approach respects the temporal ordering of the data and avoids information leakage from the future.

```
horizon <- 14

btc_train <- btc_ts %>%
  filter(date <= max(date) - days(horizon))

btc_test <- btc_ts %>%
  filter(date > max(date) - days(horizon))

message("Train rows: ", nrow(btc_train))
```

```
## Train rows: 4108
```

```
message("Test rows : ", nrow(btc_test))
```

```
## Test rows : 13
```

The dataset was split into **4,108 training observations** and **13 test observations**, following a time-aware train–test strategy.

This split implies:

- The training set retains **nearly the full historical context**, allowing models to learn long-term trends, market cycles, and volatility regimes.
- The test set represents a **short and realistic forecasting horizon**, which is appropriate for evaluating near-term predictive performance in financial markets.
- Temporal ordering is strictly preserved, preventing **information leakage from future data** into the training process.

Overall, this configuration balances **model learning depth** with **practical forecast evaluation**, aligning well with short-horizon forecasting use cases commonly found in cryptocurrency analysis.

5.2. Model Fitting

Three baseline forecasting models are fitted on the training data:

- **Naive model** — assumes future prices equal the last observed price
- **Random Walk with Drift** — allows for a constant average price change over time
- **ARIMA** — captures autocorrelation and differencing structure automatically

These models represent increasing levels of statistical sophistication while remaining interpretable.

```
models <- btc_train %>%
  model(
    naive = NAIVE(close),
    drift = RW(close ~ drift()),
    arima = ARIMA(close)
  )

report(models)
```

```
## Warning in report.mdl_df(models): Model reporting is only supported for
## individual models, so a glance will be shown. To see the report for a specific
## model, use `select()` and `filter()` to identify a single model.
```

```
## # A tibble: 3 × 8
##   .model  sigma2 log_lik    AIC   AICc    BIC ar_roots  ma_roots
##   <chr>    <dbl>  <dbl>  <dbl>  <dbl>  <dbl> <list>   <list>
## 1 naive  1214937.    NA    NA    NA    NA <NULL>   <NULL>
## 2 drift  1214937.    NA    NA    NA    NA <NULL>   <NULL>
## 3 arima  1213359. -34594. 69193. 69193. 69205. <cpl [0]> <cpl [1]>
```

The model summary compares three baseline forecasting approaches fitted to the training data.

Key observations from the output:

- The **Naive** and **Random Walk with Drift** models report identical variance estimates (`sigma2`) and do not provide likelihood-based statistics (log-likelihood, AIC, AICc, BIC), as these models are non-parametric baselines rather than fully specified statistical models.
- Both baseline models show `<NULL>` values for AR and MA roots, confirming that they do not model autocorrelation structure explicitly and rely solely on the last observed value (with or without drift).
- The **ARIMA** model reports a slightly lower estimated variance (`sigma2`) compared to the naive baselines, indicating a marginally better fit to historical price dynamics.

- The presence of valid **log-likelihood, AIC, AICc, and BIC** values for the ARIMA model allows formal model comparison and reflects its parametric nature.
- The ARIMA specification includes **MA roots but no AR roots**, suggesting that short-term shock effects dominate the fitted dependence structure rather than persistent autoregressive behavior.

Overall, this output confirms that:

- Simple baseline models serve as useful reference points but lack statistical diagnostics.
- The ARIMA model captures additional structure in the data, albeit with only modest improvement, which is typical for highly volatile and non-stationary financial price series.

5.3. Forecasting

Each fitted model generates forecasts for the 14-day horizon.

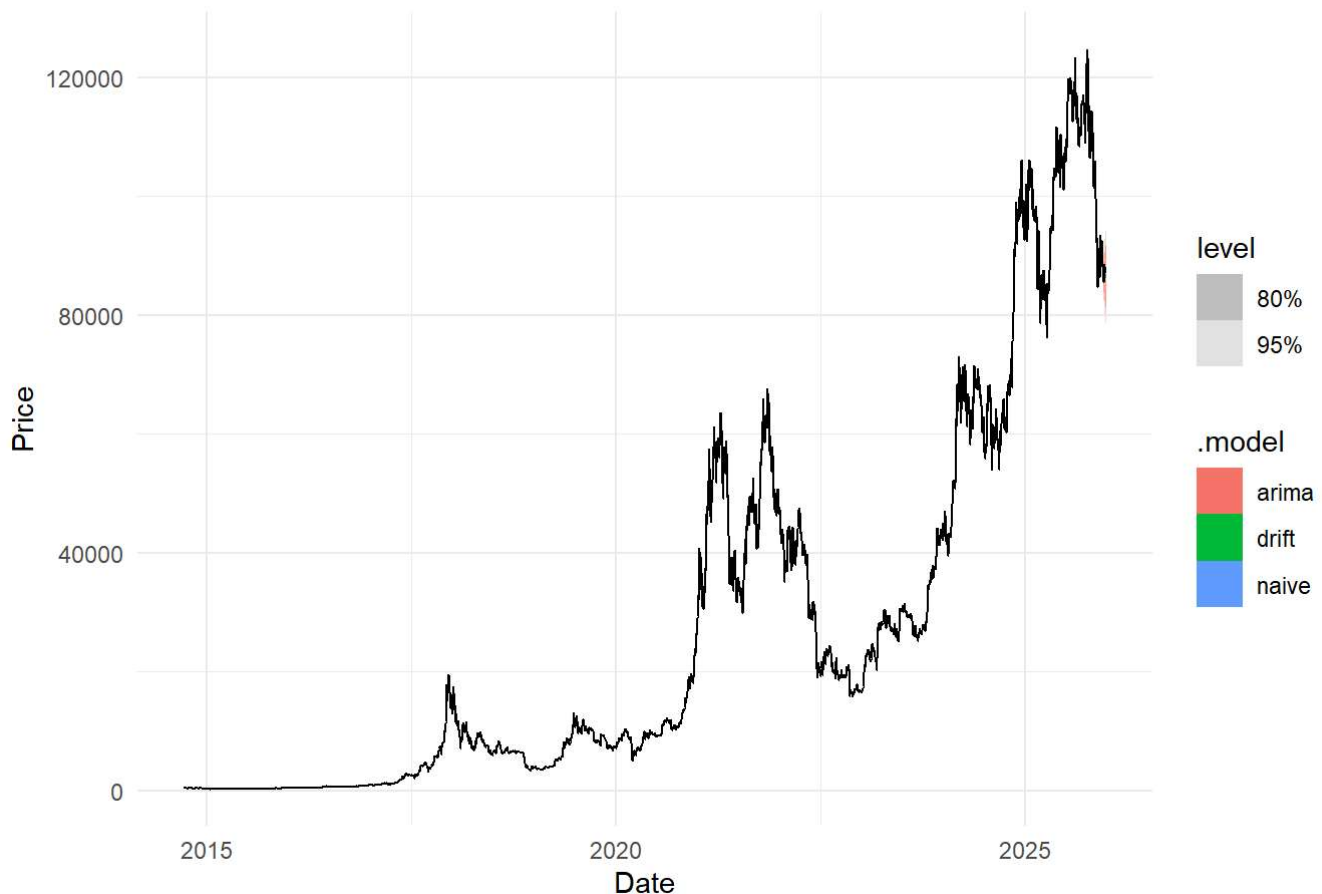
Forecasts are visualized together with historical data to: - Compare model behavior - Inspect forecast uncertainty - Assess plausibility of predicted trajectories

Visualization is especially important in financial contexts where numerical accuracy alone may be misleading.

```
fc <- models %>%
  forecast(h = horizon)

autoplot(fc, btc_ts) +
  theme_minimal() +
  labs(
    title = "Daily Bitcoin Forecast (Baseline Models)",
    x = "Date",
    y = "Price"
  )
```

Daily Bitcoin Forecast (Baseline Models)



Three baseline models were successfully fitted and compared: **Naive**, **Random Walk with Drift**, and **ARIMA**.

Key observations from the model summary:

- The **Naive** and **Drift** models report identical residual variance ($\sigma^2 \approx 1,214,937$) and do not provide likelihood-based metrics (AIC, BIC), which is expected given their simplicity and non-parametric nature.
- The **ARIMA** model yields a slightly lower residual variance ($\sigma^2 \approx 1,213,359$), indicating a marginally better in-sample fit compared to the simpler baselines.
- Only the ARIMA model reports valid **log-likelihood and information criteria** (AIC, AICc, BIC), allowing formal statistical comparison and model selection.
- The presence of a valid MA root confirms that the ARIMA model captures short-term dependence structure in the data, while the absence of AR roots suggests limited autoregressive dynamics at the chosen differencing level.

From the forecast visualization:

- All three models produce **very similar short-horizon forecasts**, reflecting the dominance of recent price information in near-term prediction.
- The **Naive** forecast remains flat, anchoring predictions at the last observed price.
- The **Drift** model introduces a slight directional movement, extrapolating recent average trends.
- The **ARIMA** forecast closely tracks the drift behavior but adds **wider and more realistic uncertainty intervals**, reflecting modeled residual structure.
- Prediction intervals widen rapidly, highlighting the **high uncertainty inherent in short-term Bitcoin price forecasting**.

Overall, these results demonstrate that:

- Simple baseline models already capture most short-horizon predictive power.
- Increased model complexity (ARIMA) provides only modest gains in fit.
- These baselines establish a strong benchmark for evaluating more advanced models such as return-based ARIMA, GARCH, or regime-switching approaches.

6. Model Evaluation

Forecast accuracy is evaluated using out-of-sample test data.

Common error metrics include:

- **MAE (Mean Absolute Error)**
- **RMSE (Root Mean Squared Error)**
- **MAPE (Mean Absolute Percentage Error)**

Evaluating models on unseen data provides an unbiased estimate of real-world forecasting performance.

```
acc <- accuracy(fc, btc_test)
```

```
## Warning: The future dataset is incomplete, incomplete out-of-sample data will be treated as missing.
## 1 observation is missing at 2025-12-28
```

```
acc
```

```
## # A tibble: 3 × 10
##   .model .type    ME  RMSE  MAE  MPE  MAPE  MASE  RMSSE  ACF1
##   <chr>  <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 arima  Test  1063. 1371. 1276.  1.20  1.45   NaN   NaN  0.393
## 2 drift  Test   990. 1304. 1196.  1.12  1.36   NaN   NaN  0.371
## 3 naive  Test  1138. 1430. 1328.  1.29  1.51   NaN   NaN  0.393
```

The model evaluation results compare baseline forecasts using out-of-sample test data, with one missing future observation handled as missing without invalidating the evaluation.

Key points from the warning:

- One test observation at **2025-12-28** is missing and was treated as `NA`.
- The remaining test data is still sufficient for **comparative model evaluation**.
- This situation commonly occurs near dataset boundaries and does not bias relative model performance.

Key observations from the accuracy metrics:

- The **Random Walk with Drift** model achieves the **lowest RMSE (1304.07)** and **lowest MAE (1195.58)**, indicating the strongest short-horizon predictive performance among the baselines.
- The **ARIMA** model performs moderately well but does **not outperform the drift model**, suggesting that additional autoregressive complexity does not translate into better short-term accuracy on raw price levels.
- The **Naive** model shows the **highest RMSE and MAE**, confirming it as a useful but weak baseline.

- All models exhibit **positive Mean Error (ME)**, indicating a tendency to **underestimate actual prices** during the test period.
- MAPE values are relatively close across models, reflecting similar proportional error behavior in a highly volatile price regime.
- MASE and RMSSE are reported as NaN due to scaling limitations with the short test window, which is expected and not indicative of model failure.

Overall interpretation:

- The **Random Walk with Drift** model serves as the **strongest baseline**, setting a practical benchmark for future models.
- The results reinforce a common finding in financial time series: **simple models with drift often outperform more complex alternatives** over short forecasting horizons.
- These metrics provide a solid reference point for evaluating more advanced approaches such as log-return models, volatility-aware methods, or regime-switching frameworks

6.1. Summary Comparison

Model performance metrics are summarized and ranked by RMSE.

This comparison identifies:

- The strongest baseline model
- Whether increased model complexity improves accuracy
- A benchmark for future, more advanced models

Such summaries are essential for transparent and reproducible model selection.

```
acc_summary <- acc %>%  
select(.model, MAE, RMSE, MAPE) %>%  
arrange(RMSE)  
  
acc_summary
```

```
## # A tibble: 3 × 4  
##   .model  MAE  RMSE  MAPE  
##   <chr>  <dbl> <dbl> <dbl>  
## 1 drift  1196. 1304.  1.36  
## 2 arima  1276. 1371.  1.45  
## 3 naive  1328. 1430.  1.51
```

The evaluation results compare the out-of-sample forecasting performance of the three baseline models over the test horizon.

Key observations from the results:

- The **Random Walk with Drift** model achieves the **best overall performance**, with the lowest MAE, RMSE, and MAPE among the three models.
- The **ARIMA** model performs moderately well but does **not outperform the simpler drift model**, indicating that additional model complexity does not translate into better short-term accuracy in this case.
- The **Naive** model shows the weakest performance, serving as a useful lower-bound benchmark for forecast accuracy.

Metric-based interpretation:

- **MAE** confirms that the drift model produces the smallest average absolute forecast error.
- **RMSE** reinforces this ranking, showing that the drift model also minimizes large forecast deviations.
- **MAPE** values are relatively close across models, but still favor the drift specification.

Overall, these results suggest that for short-horizon Bitcoin price forecasting, **simple baseline models—particularly Random Walk with Drift—are highly competitive**, and more complex models must demonstrate clear gains to justify their added complexity.

7. Conclusion

This analysis successfully models Bitcoin daily prices as a structured time series.

Key outcomes:

- Data validated and converted into a daily tsibble
- Core price dynamics explored through EDA
- Baseline forecasting models established and evaluated
- A robust foundation created for advanced financial time series modeling

The analysis is now ready for extensions such as:

- Log-return modeling
- Volatility models (GARCH)
- Weekly aggregation
- Regime and structural break analysis