Project 1: Bitcoin Price Prediction

Hoda Abokhadra[*], Philip Myint[†], Bryan Reynolds[‡], Taha Wasiq[§]
CS 598: Practical Statistical Learning (Fall 2025)
University of Illinois Urbana-Champaign

September 28, 2025

1 Introduction

Our goal in this project is to analyze and predict Bitcoin prices at two distinct time horizons, next day vs. next week, with the expectation that the latter will be more challenging to predict accurately. To achieve this purpose, we have cleaned a raw data file (e.g., by removing missing or invalid values) and produced many additional features based on common finance/banking industry metrics, selected and implemented regression models with varying degrees of sophistication, and applied those models to make price predictions and analyze/interpret our results. These are the subjects of Sections 2, 3, and 4 of this report, respectively. We conclude in Section 5 with a brief summary of our findings. Our code is developed in `Python` and hosted on Github at `https://github.com/TahaWasiq/CS_598_Project_1/`, with the following directory structure:

- `scripts_cleaning/`: Contains the data cleaning and configuration scripts described in Section 2.
- `data/`: The raw, original csv Bitcoin prices data file and the various cleaned/reorganized data files.
- `scripts_models/`: Scripts for analyzing the data, fitting the models, and performing model selection. Our modeling methodology is explained in Section 3, and the results are presented in Section 4.
- `report/`: Contains the figures and LaTeX files used to generate this report.

2 Data Cleaning and Configuration

2.1 Overview of pipeline

We divide our data cleaning and configuration operations over five scripts that we run in the following order:

1. `clean.py`: Takes the raw **bitcoin.csv** file and produces a cleaned version **bitcoin_aligned.csv**.
2. `build-features.py`: Reads **bitcoin_aligned.csv** and produces dozens of additional features to produce two csv files: **btc_features_h1_full.csv** and **btc_features_h7_full.csv**, which may be used for predicting the Bitcoin price in the next day (i.e., with a horizon of $h = 1$ days) and the next week ($h = 7$ days), respectively.
3. `split.py`: Standardizes (produces z-scores) on the features in **btc_features_h1_full.csv** and **btc_features_h7_full.csv**, and breaks the data in each of these two files into training, validation, and test sets, for a total of six sets. Each set is stored in its own csv file.

[*]hoda2@illinois.edu
[†]pcmyint2@illinois.edu
[‡]bryanmr3@illinois.edu
[§]twasiq2@illinois.edu

4. `remove_features_vif.py`: Calculates the variance inflation factor (VIF) for each feature and removes the ones that have a high VIF to improve the interpretability of the subsequent regression models.
5. `checks.py`: Outputs summary statistics of the training, validation, and test sets, and verifies that none of the sets contain `NA` values.

More details on each of these steps in our pipeline are described in the following subsections.

2.2 Initial cleaning

`clean.py` renames some of the columns in the raw **bitcoin.csv** file (e.g., **btc_trade_volume** is renamed to **btc_volume**). Values in three critical columns (**btc_close** [renamed from **btc_market_price**], **btc_mktcap**, and **btc_volume**) are converted/coerced to numeric values. `NA` values in these columns are forward-filled by nearby non-`NA` ones. If there is a duplicate date, the second one is kept. Finally, a column for the % daily change is added, with all the aforementioned changes saved to a new file, **bitcoin_aligned.csv**.

2.3 Additional features

Using `build-features.py`, we have created a large number of additional features (a total of 69), some of which are inspired by metrics commonly used in the finance/banking industries. These features include:

- **btc_rsi14**: The Relative Strength Index (RSI), a common finance metric used to measure momentum. Based on a rolling window of daily price changes averaged over the past 14 days, RSI captures recent buying vs. selling pressure (i.e., whether recent price changes have been mostly upward or downward).
- **btc_ema12** and **btc_ema26**: The average closing price over the last 12 days and 26 days, respectively, with more weight placed on recent days (an exponentially weighted moving average [EMA]).
- **btc_macd**: Difference between a short-term trend (12-day EMA) and a longer-term trend (26-day EMA). If it's positive, short-term momentum is stronger, if negative, momentum is weaker.
- **btc_atr14**: The average size of daily price moves over the past 14 days, indicating price volatility.

In addition, the features include rolling means and standard deviations over the past 7, 14, and 30 days of the closing price, returns (% change), and trading volume. We also include values of the closing price, returns, and volume from the previous day (1-day lag) and two weeks ago (15-day lag). Rows that have missing or invalid (`NA`) values are dropped; this is the cleaning step where the first 5–6 months of data collected in 2010 were dropped, since the reported Bitcoin market price in the raw csv file remained flat at $0 during that early period so that the % daily change and features based on that metric were undefined for those rows. As mentioned above, we save our additional features in two new files: **btc_features_h1_full.csv** and **btc_features_h7_full.csv**. The target columns in these files represent the Bitcoin price at the close of the next day ($h = 1$) and the next week ($h = 7$), respectively. Furthermore, these files contain a column for the date and a column for each of the 69 features, for a total of 71 columns in each file.

2.4 Data splitting and standardization

As its name implies, `split.py` splits **btc_features_h1_full.csv** and **btc_features_h7_full.csv** into training, validation, and test sets, producing a total of six new csv files (three for $h = 1$ and another three for $h = 7$). These sets are split chronologically, so that the first 70% of entries are put into the training set, the next 15% in the validation set, and the last 15% in the test set. The idea is to develop models that can predict more recent trends after being trained on older trends. That is, we will train our models on historical data, tune/validate hyperparameters with more near-term data, and then finally test our models on the most recent data. Each of the 69 features are standardized so that they are transformed to z-scores based on means and standard deviations computed from the training set.

2.5 Analyzing collinearity and removing high VIF features

We check for collinearity by running a VIF analysis with **`remove_features_vif.py`**. The heart of this script is a function that recursively removes the feature with the highest VIF, until none of the remaining features have a VIF that lies above a specified threshold. By default, we set the threshold to a rather high value of 1000 so that only features that are strongly linearly correlated with at least one other feature are removed. This ends up removing 11 of our features so that we end up with a total of $69 - 11 = 58$. We could have chosen a more typical VIF threshold of 5 or 10, but our approach at this stage is to be cautious and avoid prematurely removing features that could end up being important. The regularization that we apply in Section 3 provides another opportunity to remove highly collinear features in a more systematic manner.

2.6 Data checks and verification

The last step in our data cleaning and configuration pipeline is to run **`checks.py`**, which outputs summary statistics of the training, validation, and test sets, and verifies that none of the sets contain `NA` values. Table 1 provides a summary of the final, cleaned version of our data sets.

Table 1: Number of days (i.e., number of rows in each file), start date, and end date for each of the six data sets after applying the cleaning and configuration operations described in Section 2. Each file contains 58 features. For reference, the raw csv file contains 2920 rows spanning from 2010-02-23 to 2018-02-20, with 22 features. The vast majority of the ∼300 rows that were dropped are in the first several months of 2010, where the Bitcoin price is listed at $0 in the raw file. Other reasons for dropping include duplicate dates or missing/invalid entries for the target or features.

Data set	Number of days	Start date	End date
Next-day prediction ($h = 1$) training set	1877	2010-09-16	2015-12-07
Next-day prediction ($h = 1$) validation set	402	2015-12-08	2017-01-12
Next-day prediction ($h = 1$) test set	403	2017-01-13	2018-02-19
Next-week prediction ($h = 7$) training set	1873	2010-09-16	2015-12-03
Next-week prediction ($h = 7$) validation set	401	2015-12-04	2017-01-07
Next-week prediction ($h = 7$) test set	402	2017-01-08	2018-02-13

3 Methodology

We apply the following four regression models to each of the $h = 1$ and $h = 7$ data sets:

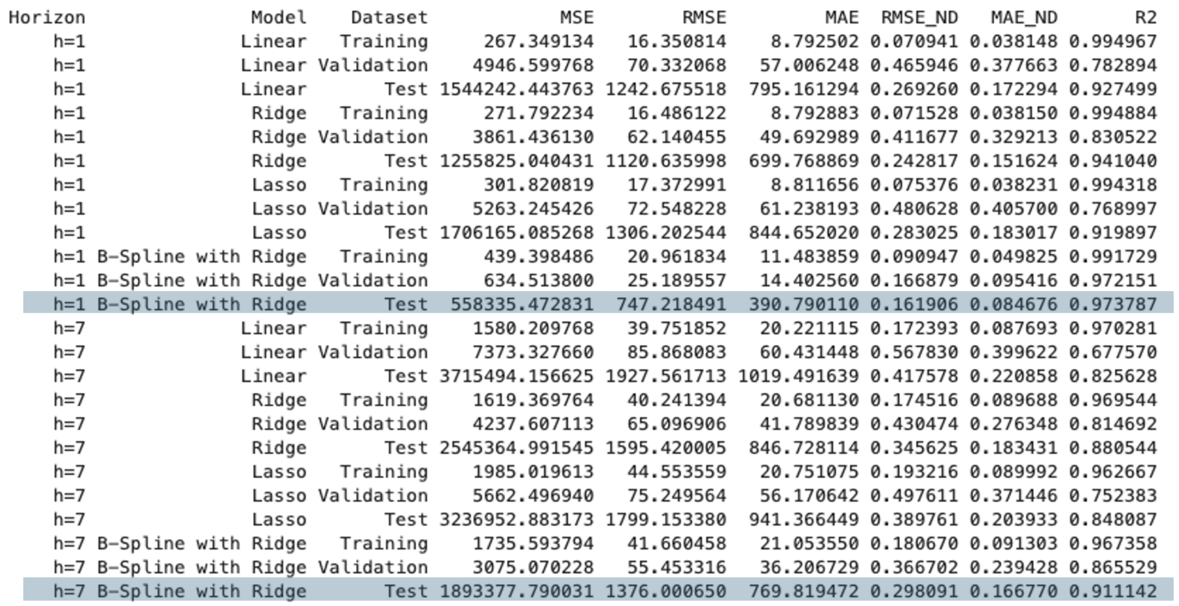
1. Linear regression involving all of the 58 features without any regularization,
2. Lasso regression,
3. Ridge regression,
4. Cubic B-Splines on four features combined together with ridge regression.

These methods allow us to compare a naive approach (straightforward linear regression) with more sophisticated approaches that involve regularization and/or different basis functions (cubic polynomials in the case of the splines). For all the models, we determine the regression coefficients by fitting to the training data, and we select the best models for $h = 1$ and $h = 7$ based on their performance on the corresponding validation set. We then analyze the performance of the chosen best models on the test set, focusing on mean-squared error (MSE) and R^2 scores, as well as on model interpretation, as illustrated further in Section 4.

Our code is organized into a notebook, **`regression_models.py`**, that interacts with users on the front end and calls a utility file **`Utilities.py`** and a second script **`BitcoinModeler.py`** that does much of the work on the back end by implementing the model initialization, fitting, evaluations, etc. in a series of classes (e.g.,

DataManager, **ModelWrapper**, **ExperimentRunner**). Data is read and manipulated in the form of **Pandas DataFrames** or **NumPy** arrays. We extensively utilize the **sklearn** package. In particular, we use the **RidgeCV** and **LassoCV** models in **sklearn**, with the regularization parameter λ selected through cross-validation over 500 logarithmically spaced values from 10^{-6} to 10^{6}. We implement our spline model as a custom class that is based on the **SplineTransformer** function in **sklearn**. The splines augment a linear ridge regressor with a small set of cubic B-spline basis functions to capture smooth, nonlinear effects while keeping the model regularized. From a hand-curated pool of potentially smooth signals (e.g., price, EMA/macd, RSI, rolling stats), we select at most 4 of these features to create B-spline basis functions that augment the ridge regressor. This feature selection is done using **mutual_info_regression** (MI), as defined in **sklearn**, that we compute on the training set only to avoid leakage. MI is preferred over Pearson correlation because it detects nonlinear and nonmonotonic dependencies between each candidate feature and the target, and it is invariant to monotone rescalings; this makes it a better filter for signals whose effect on the target is not strictly linear. For the chosen drivers, we expand them with **SplineTransformer(degree=3, knots="quantile", extrapolation="linear")**, concatenate the raw passthrough features, and fit a ridge penalty to control the increased degrees of freedom. Hyperparameters, like the number of knots and regularization parameter λ, are tuned with a grid search. Limiting the number of spline features and using quantile-spaced knots keeps the basis compact, and it focuses capacity where the data are dense to reduce overfitting risk.

4 Results



```
Horizon                Model    Dataset             MSE         RMSE          MAE  RMSE_ND   MAE_ND       R2
    h=1               Linear   Training      267.349134    16.350814     8.792502 0.070941 0.038148 0.994967
    h=1               Linear Validation     4946.599768    70.332068    57.006248 0.465946 0.377663 0.782894
    h=1               Linear       Test 1544242.443763 1242.675518   795.161294 0.269260 0.172294 0.927499
    h=1                Ridge   Training      271.792234    16.486122     8.792883 0.071528 0.038150 0.994884
    h=1                Ridge Validation     3861.436130    62.140455    49.692989 0.411677 0.329213 0.830522
    h=1                Ridge       Test 1255825.040431 1120.635998   699.768869 0.242817 0.151624 0.941040
    h=1                Lasso   Training      301.820819    17.372991     8.811656 0.075376 0.038231 0.994318
    h=1                Lasso Validation     5263.245426    72.548228    61.238193 0.480628 0.405700 0.768997
    h=1                Lasso       Test 1706165.085268 1306.202544   844.652020 0.283025 0.183017 0.919897
    h=1 B-Spline with Ridge   Training      439.398486    20.961834    11.483859 0.090947 0.049825 0.991729
    h=1 B-Spline with Ridge Validation      634.513800    25.189557    14.402560 0.166879 0.095416 0.972151
    h=1 B-Spline with Ridge       Test  558335.472831   747.218491   390.790110 0.161906 0.084676 0.973787
    h=7               Linear   Training     1580.209768    39.751852    20.221115 0.172393 0.087693 0.970281
    h=7               Linear Validation     7373.327660    85.868083    60.431448 0.567830 0.399622 0.677570
    h=7               Linear       Test 3715494.156625 1927.561713  1019.491639 0.417578 0.220858 0.825628
    h=7                Ridge   Training     1619.369764    40.241394    20.681130 0.174516 0.089688 0.969544
    h=7                Ridge Validation     4237.607113    65.096906    41.789839 0.430474 0.276348 0.814692
    h=7                Ridge       Test 2545364.991545 1595.420005   846.728114 0.345625 0.183431 0.880544
    h=7                Lasso   Training     1985.019613    44.553559    20.751075 0.193216 0.089992 0.962667
    h=7                Lasso Validation     5662.496940    75.249564    56.170642 0.497611 0.371446 0.752383
    h=7                Lasso       Test 3236952.883173 1799.153380   941.366449 0.389761 0.203933 0.848087
    h=7 B-Spline with Ridge   Training     1735.593794    41.660458    21.053550 0.180670 0.091303 0.967358
    h=7 B-Spline with Ridge Validation     3075.070228    55.453316    36.206729 0.366702 0.239428 0.865529
    h=7 B-Spline with Ridge       Test 1893377.790031 1376.000650   769.819472 0.298091 0.166770 0.911142
```

Figure 1: Results for the four regression models on the two horizons. For both horizons, the model that performs best on the validation set and test set is the B-Spline with Ridge model, highlighted in blue. RMSE_ND and MAE_ND denote nondimensionalized versions of the root-mean-squared error (RMSE) and mean absolute error (MAE), respectively, which align well with trends in R^2, as explained in the text.

The table in Figure 1 summarizes our results for all four regression models on the two horizons $h = 1$ and $h = 7$. For both horizons, the B-spline with Ridge model performs best on all metrics for the validation set, meaning it has the lowest MSE, lowest mean absolute error (MAE), and highest R^2 on the validation set. As a result, we designate it as our best model, and from the table, we see that it also has the best performance on the test set as well. Interestingly, the other models tend to perform better on the training set, but worse

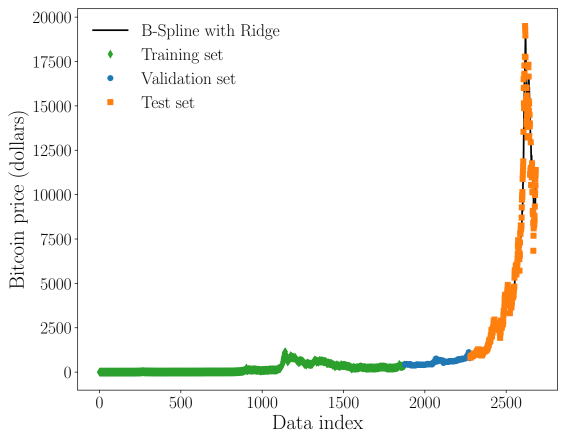


Figure 2: Time series of Bitcoin prices. The x-axis denotes the data index (row index), which is a surrogate for the date because the files are sorted chronologically. Results here are for $h = 1$ (plot for $h = 7$ is similar).

on the validation and training sets, which is a manifestation of the bias–variance tradeoff. This suggests that the B-spline with Ridge model is relatively robust to variances that stem from the validation and test sets being taken from later dates (see Table 1), variances that reflect increases in the Bitcoin price over time. In particular, the test set exhibits a dramatic increase in the price, as seen in Figure 2. The MSE and MAE are orders of magnitude higher in the test set than in the validation set, despite the models generally having a higher R^2 score on the test set, simply because the average price and variance in the test set is much higher. To account for this, we have produced nondimensionalized versions of RMSE and MAE where we divide these quantities by the standard deviations of the true target values. The table in Figure 1 shows that the nondimensionalized RMSE and MAE follow the same trends as R^2 on the test set vs. the validation set.

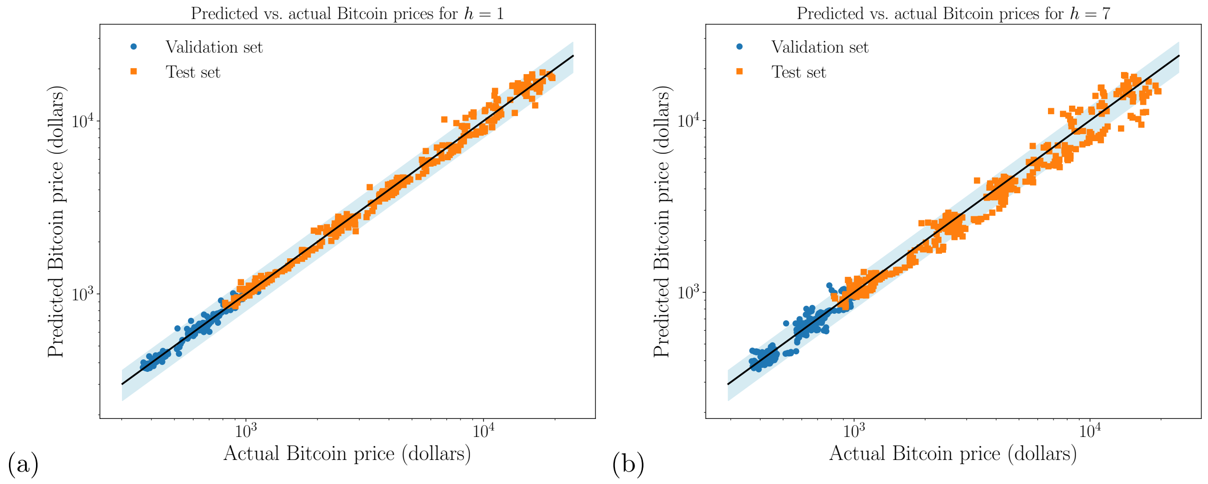


Figure 3: Predicted vs. actual Bitcoin prices for the B-spline with Ridge model for: (a) the $h = 1$ horizon and (b) the $h = 7$ horizon. The black line shows the line $y = x$, indicating a perfect prediction, and the blue band denotes $\pm 20\%$ error bars around that line.

Another trend that we observe from the summary table is that all the models perform better for the $h = 1$ horizon than the $h = 7$ horizon. Figure 3 illustrates this for the particular example of the B-spline with Ridge model. For $h = 1$, the model is able to predict nearly all of the data points to within an error of $\pm 20\%$, whereas for $h = 7$, there are noticeably more points that fall outside of that error band. This agrees

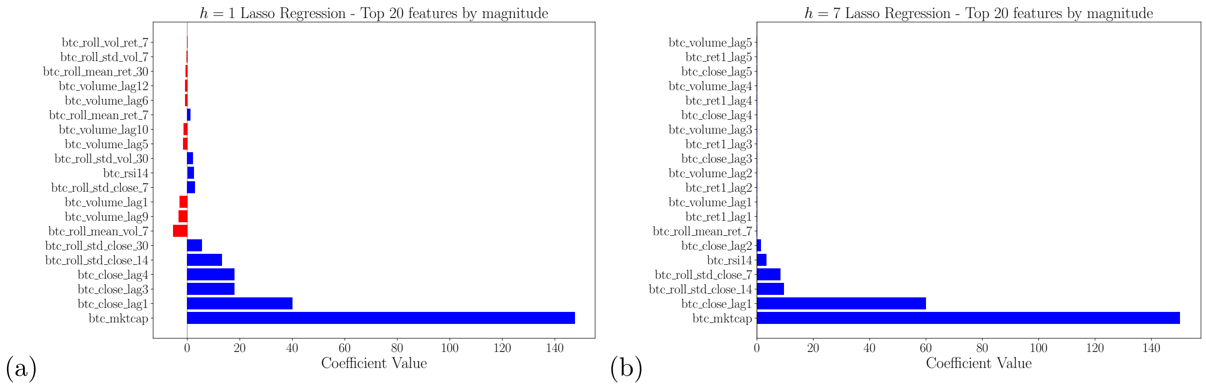


Figure 4: Top 20 regression coefficients by magnitude for the Lasso model for a) $h = 1$ and (b) $h = 7$.

with the intuitive notion that the task of predicting the price in the next day ($h = 1$) would be easier than predicting a more distant price in the future, such as next week ($h = 7$). Inspection of the model coefficients reveals more insight into the differences between the two time horizons. Out of the 58 features, Lasso shrinks all but 20 of them to zero for $h = 1$, and it goes even further for $h = 7$, where only 7 features have nonzero coefficients (Figure 4). This suggests that for predicting prices over longer time horizons, most of the features become irrelevant and only a small portion of them explain the observed variability. Interestingly, although there are fewer nonzero coefficients, the 2-norm of the coefficients vector is larger in $h = 7$, so there is a relatively heavy concentration in a smaller set of coefficients with elevated importance. Similar behavior (not shown in Figure 4 for brevity) is seen for the Ridge and B-Spline with Ridge models as well, although these models do not shrink any of the coefficients to zero. The Linear model unsurprisingly has the largest 2-norm, and it performs the worst on the validation and test sets for the $h = 7$ horizon precisely because it lacks the regularization needed to help concentrate the values in a smaller, but important subset of the coefficients.

5 Conclusions

In summary, we have produced data files and scripts to analyze and make predictions on Bitcoin prices over two time horizons: $h = 1$ (next day) and $h = 7$ (next week). To prepare the data, we have added dozens of features, many of which are inspired from finance/banking, performed cleaning to remove invalid (`NA`) or missing values, and subsequently removed some of the features that have a very high VIF to improve the model interpretability. The data for each horizon is split chronologically into training (first 70%), validation (next 15%), and test (last 15%) sets, with each file containing 58 features that are standardized with data from the training set. We have applied four models to predict Bitcoin prices: 1) linear regression without any regularization, 2) lasso regression, 3) ridge regression, and 4) B-splines augmenting a ridge regressor. For both horizons, we find that the B-Spline with Ridge model performs best on the validation and test sets in terms of metrics like R^2 and MSE, even if other models perform better on the training set, reflecting the bias–variance tradeoff. The B-Spline with Ridge model is relatively robust to variances in the data, including being able to handle the sharp rise and fall in Bitcoin prices that are exhibited in the test set but not present in the training set. It has an R^2 score of about 0.97 and 0.91 on the $h = 1$ and $h = 7$ test sets, respectively. In fact, all the models perform better on $h = 1$ than on $h = 7$, agreeing with the expectation that next-day prices should be easier to predict than prices in the next week. Inspection of the model coefficients reveals that the variability in the $h = 7$ horizon is explained by a smaller subset of the features than in $h = 1$, which suggests that more of the features become irrelevant as we attempt to predict prices further out in the future. This also underlines the importance of regularization (particularly for $h = 7$), since it helps to temper the less important features. To further build on this report, one may explore other time horizons (perhaps 1 month) and other models introduced later in the class (e.g., random forests, support vector machines).