Analysing Cryptocurrency Market

Nisha Selvarajan

- Challenge
- Data Description
- Data Analysis
- Prepare Data for Regression
- Linear Regression To Predict Market Price
- Residual Analysis in Linear Regression
- Linear Regression Prediction & Accuracy.
- Polynominal Regression
- Polynomial Regression Prediction & Accuracy
- Spline Regression
- Spline Regression Prediction & Accuracy
- Generalized Linear Model
- GAM Regression Prediction & Accuracy
- Scatter Plot 3D Visualization
- Penalized Cubic Regression Spline
- Cubic Regression Spline Prediction & Accuracy
- Extreme Gradient Boosting
- Extreme Gradient Boosting Prediction & Accuracy
- Compare Regression Models
- Conclusion

Challenge

To forecast cryptocurrency prices using all the trading features like price, volume, open, high, low values present in the dataset.

Probably one of the biggest things in recent years is Bitcoin. Bitcoin grew by around 800% last year, held a market cap of around 250 billion dollars, and sparked worldwide interest in cryptocurrencies. But what are cryptocurrencies? Basically they're digital currencies that use complex computer algorithms and encryption to generate more currency and to protect transactions. What's really cool about cryptocurrencies is that they utilize a network of thousands of computers that forward people's transactions to what's known as a blockchain (essentially a big record of transactions kept secure by the network of computers). Once a transaction is in the blockchain, it's never coming out again; this protects cryptocurrencies from double-spends. So it's pretty clear that cryptocurrencies are a cool new way to spend money — what if we could predict how its prices fluctuate?

By analyzing bit coin historical features, such as bitcoin tradevolume, bitcoin blockssize, bitcoin difficultyto find a new block,total value of coinbase block rewards , transaction fees paid to miners, we can predict the variation and can predict the ups and downsof bitcoin price .

Data Description

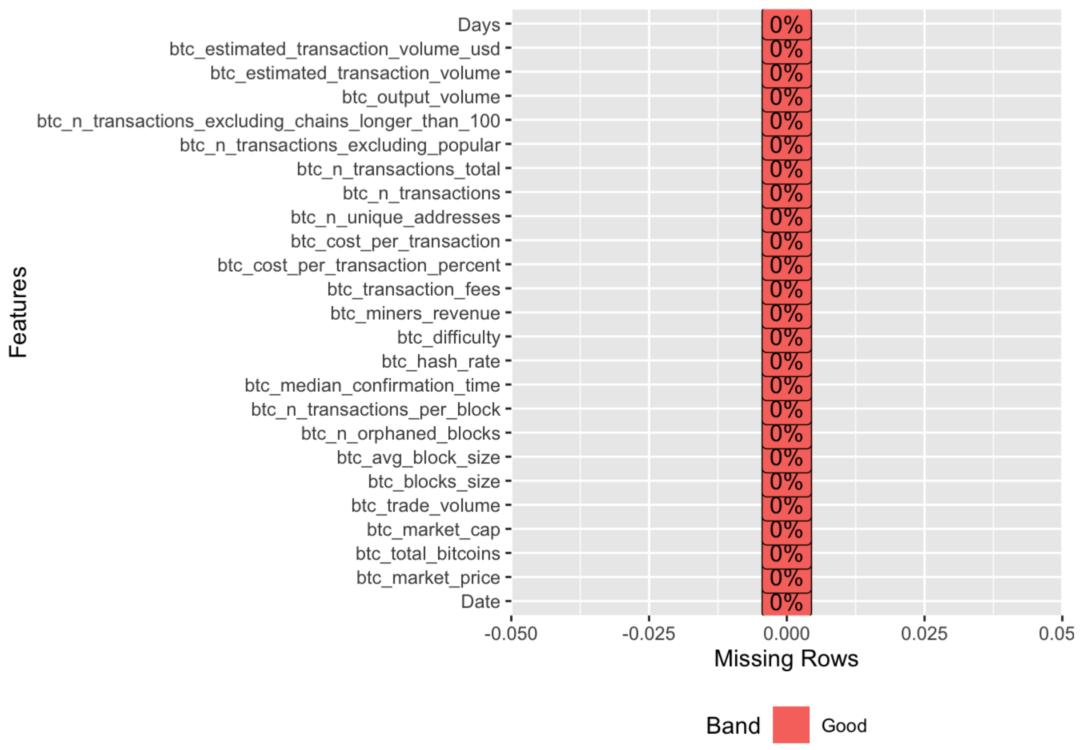
- The data used for this particular project is "Cryptocurrency Historical Prices by Sudalairajkumar". The data consists of information on the 17 currencies collected daily from various dates (Some January 2016, April 2013, August 2015, so on.) to October 2017. The original source of the data is coinmarketcap, Blockchain Info and Etherscan. (https://www.kaggle.com/sudalairajkumar/cryptocurrencypricehistory)
- We are using Bitcoin Dataset (bitcoin_dataset.csv) in the above dataset.The details of attributes are following.

Names	Description
Date	Date - Date of observation
btcmarketprice	Numerical - Average USD market price across major bitcoin exchanges
btctotalbitcoins	Numerical - Total number of bitcoins that have already been mined
btcmarketcap	Numerical - Total USD value of bitcoin supply in circulation
btctradevolume	Numerical - Total USD value of trading volume on major bitcoin exchanges
btcblockssize	Numerical - Total size of all block headers and transactions

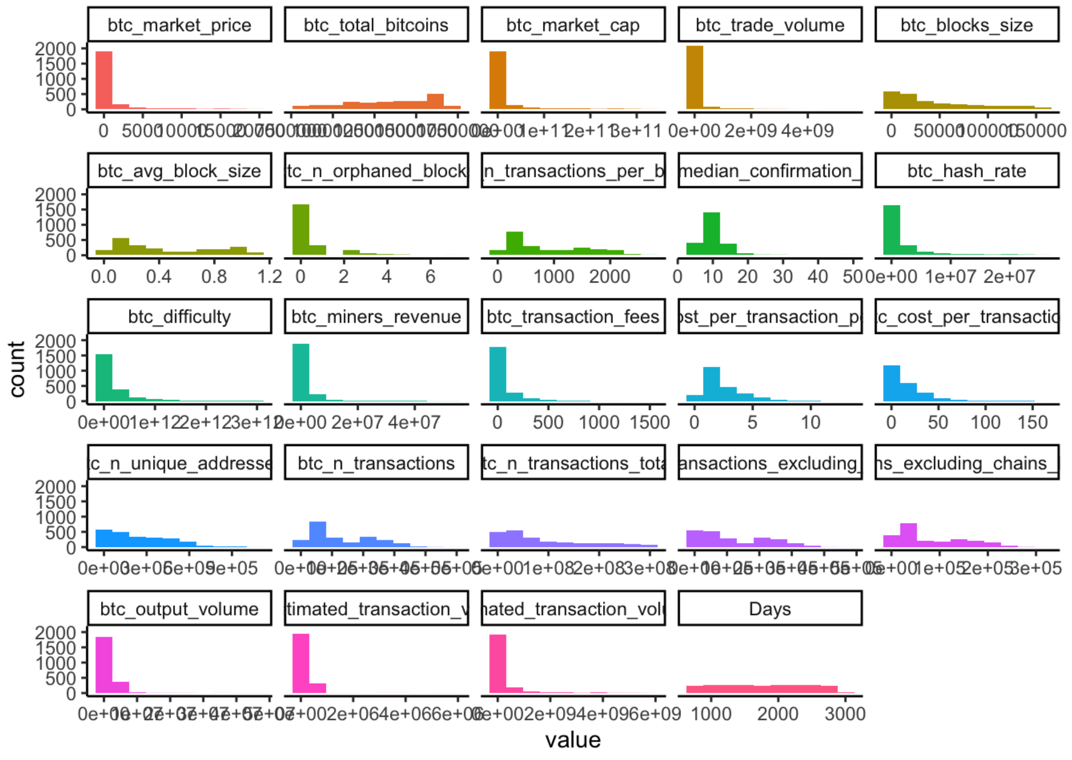
Names	Description
btcavgblock_size	Numerical - Average block size in MB
btcnorphaned_blocks	Numerical - Total number of blocks mined but ultimately not attached to blockchain
btcntransactionsperblock	Numerical - Average number of transactions per block
btcmedianconfirmation_time	Numerical - Median time for a transaction to be accepted into a mined block
btchashrate	Numerical - Estimated number of tera hashes per second the Bitcoin network is performing
btc_difficulty	Numerical - Relative measure of how difficult it is to find a new block
btcminersrevenue	Numerical - Total value of coinbase block rewards and transaction fees paid to miners
btctransactionfees	Numerical - Total value of all transaction fees paid to miners.
btccostpertransactionpercent	Numerical - Miners revenue as percentage of the transaction volume.
btccostper_transaction	Numerical - Miners revenue divided by the number of transactions
btcnunique_addresses	Numerical - Total number of unique addresses used on the Bitcoin blockchain.
btcntransactions	Numerical - Number of daily confirmed Bitcoin transactions
btcntransactions_total	Numerical- Total number of transactions
btcntransactionsexcludingpopular	Numerical- Total number of Bitcoin transactions, excluding the 100 most popular addresses
btcntransactionsexcludingchainslongerthan_100	Numerical- Total number of Bitcoin transactions per day excluding long transaction chains
btcoutputvolume	Numerical- Total value of all transaction outputs per day
btcestimatedtransaction_volume	Numerical- Total estimated value of transactions on the Bitcoin blockchain
btcestimatedtransactionvolumeusd	Numerical- Estimated transaction value in USD value

Data Analysis

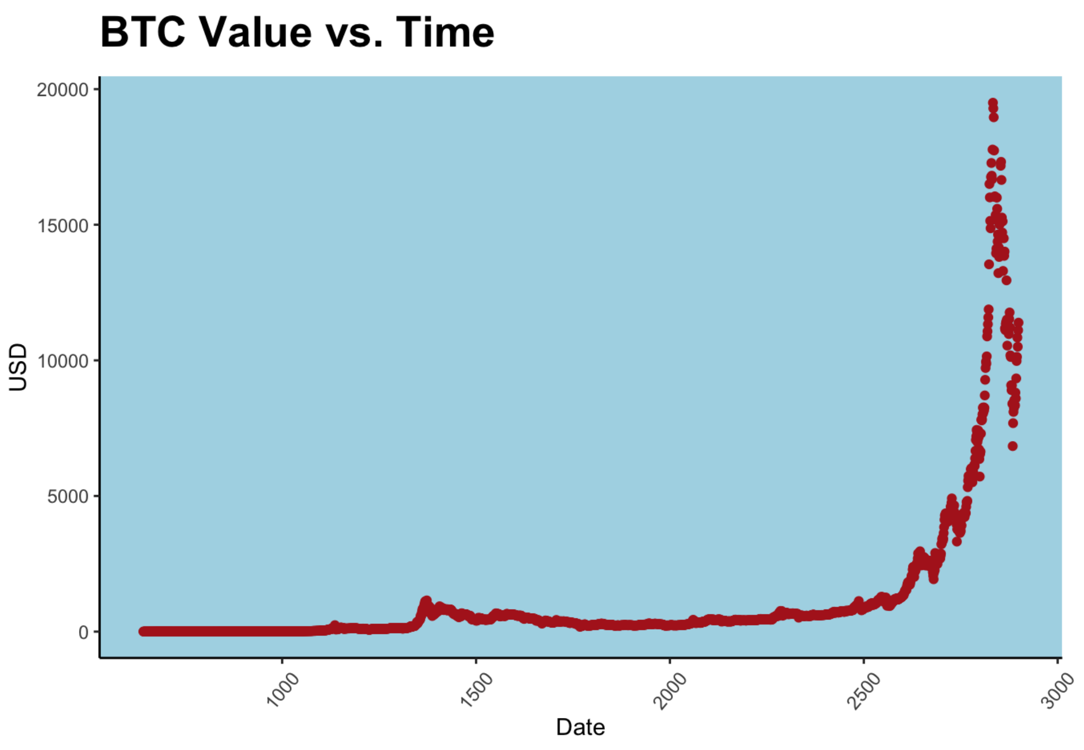
- Plot missing values of all the features in the dataset.



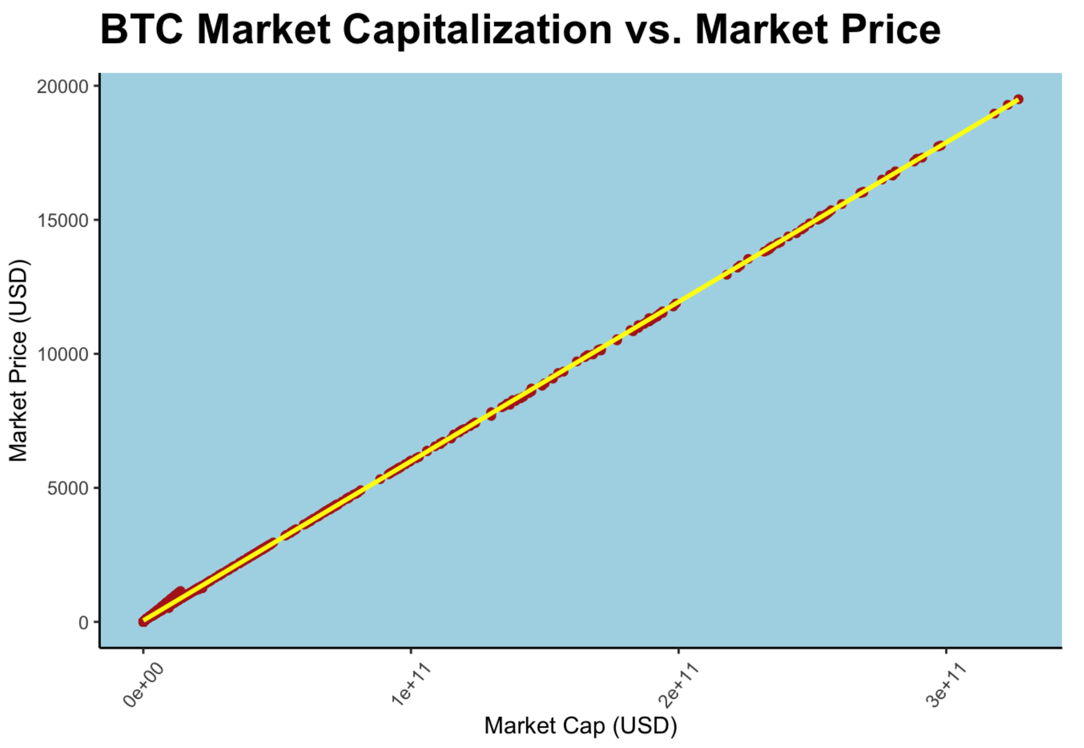
- Ploting histograms for numerical variables.



- Plot of daily bitcoin priceshows that this plot has a general upward trend with no obvious seasonality or intervention point. There is obvious changing variance. Successive observations suggest autoregressive behavior.

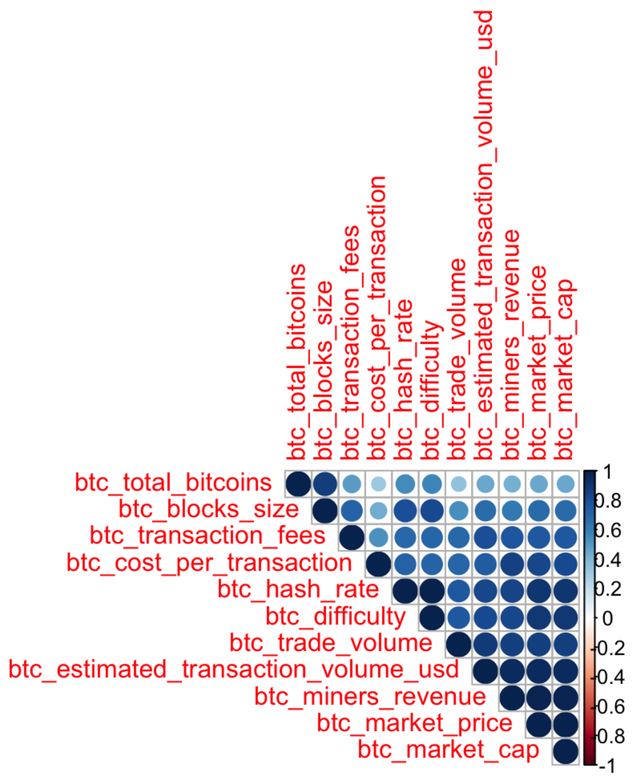


- Plot of market price with market capitalization.There is a linear trend for market cap with respect to market price.



- Correlation Plot to show the dependence between multiple variables in bitcoin dataset.Highlighting the most correlated variables in a data table. In this visual, correlation coefficients are also displayed.

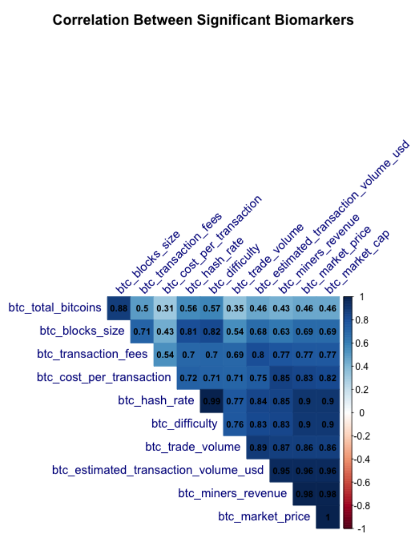
```
'data.frame':	2258 obs. of  11 variables:
 $ btc_market_price                  : num  3.14 3.13 2.99 2.93 3.05 ...
 $ btc_total_bitcoins                : num  7787350 7794850 7801700 7809700 7817650 ...
 $ btc_market_cap                    : num  24436704 24397803 23327083 22882421 23843832 ...
 $ btc_trade_volume                  : num  181505 363126 263375 90500 170165 ...
 $ btc_blocks_size                   : num  572 574 576 578 580 583 585 587 589 591 ...
 $ btc_hash_rate                     : num  8.51 8.13 7.43 8.68 8.62 ...
 $ btc_difficulty                    : num  1090716 1090716 1090716 1090716 1090716 ...
 $ btc_miners_revenue                : num  24646 23487 20490 23449 24257 ...
 $ btc_transaction_fees              : num  4.3 4.09 3.51 3.62 3.5 ...
 $ btc_cost_per_transaction          : num  3.9 4.24 3.87 3.69 3.45 ...
 $ btc_estimated_transaction_volume_usd: num  10383421 11525011 9581607 5518235 17766452 ...
```



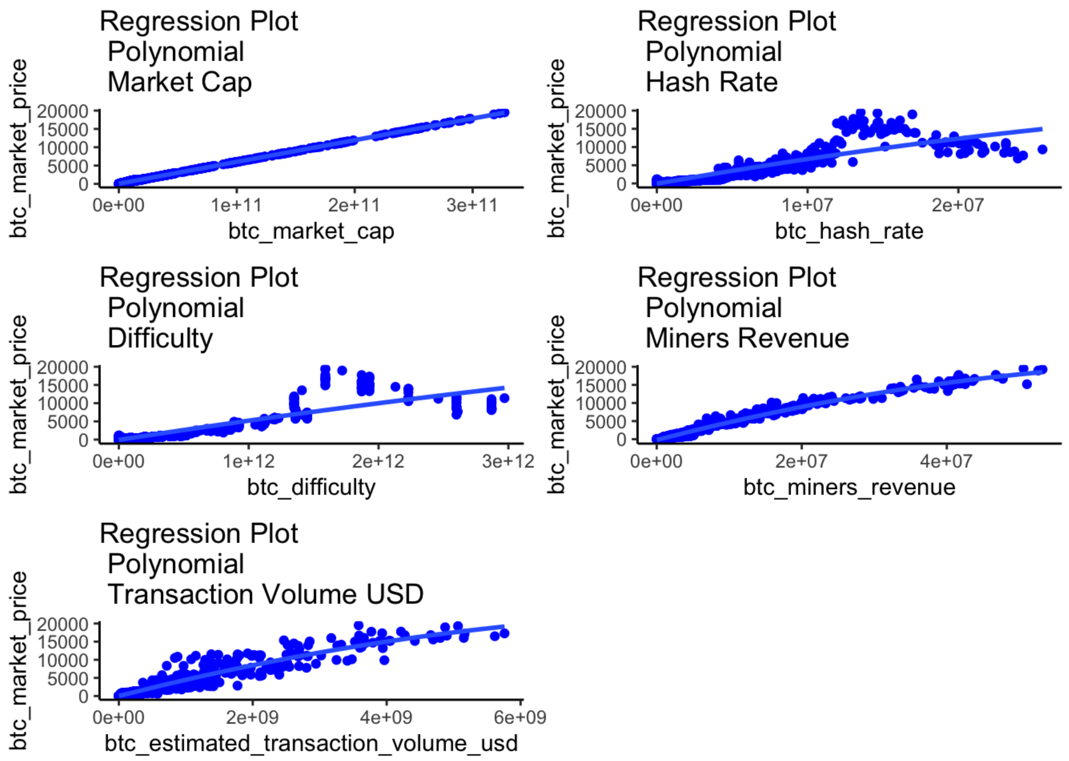
1

0

1



- Sometimes it’s nice to quickly visualise the data that went into a simple linear regression, especially when you are performing lots of tests at once. Here is a quick and dirty solution with ggplot2 to create the following regression plot with respect to btc_market_price:



Prepare Data for Regression

- Select variables relevant to market price:Based on the variable importance, we will use btc_market_cap, btc_hash_rate,btc_difficulty, btc_miners_revenue, btc_estimated_transaction_volume_usd feature for further analysis.

```
'data.frame':	2258 obs. of  6 variables:
 $ Market Price                 : num  3.14 3.13 2.99 2.93 3.05 ...
 $ Market Cap                   : num  24436704 24397803 23327083 22882421 23843832 ...
 $ Hash Rate                    : num  23 14 6 29 26 12 17 15 23 10 ...
 $ Difficulty                   : num  1090716 1090716 1090716 1090716 1090716 ...
 $ Miners Revenue               : num  24646 23487 20490 23449 24257 ...
 $ Estimated Transaction Volume USD: num  10383421 11525011 9581607 5518235 17766452 ...
```

- Load the cleaned dataset: - Omit rows which has N/A.
- Data slicing:
    - Dataset is split into 80 percent of training data, 20 % of test set.

Linear Regression To Predict Market Price

- TrainingParameters :
  - train() method is passed with repeated cross-validation resampling method for 10 number of resampling iterations repeated for 3 times.
- RMSE or Root Mean Squared Error is the default metrics used to evaluate algorithms on regression datasets in caret.RMSE or Root Mean Squared Error is the average deviation of the predictions from the observations.

```
# setup cross validation and control parameters
metric <- "RMSE"
tuneLength <- 10

# Training process
# Fit / train a Linear Regression model to  dataset
linearModelReg <- caret::train(btc_market_price~
                                 btc_market_cap+btc_hash_rate+
                                 btc_difficulty+btc_miners_revenue+
                                 btc_estimated_transaction_volume_usd
                      ,data=subTrain1, method="lm", metric=metric,
                      preProc=c("center", "scale"), trControl=control, tuneLength = tuneLength)
```

```
summary(linearModelReg)
```

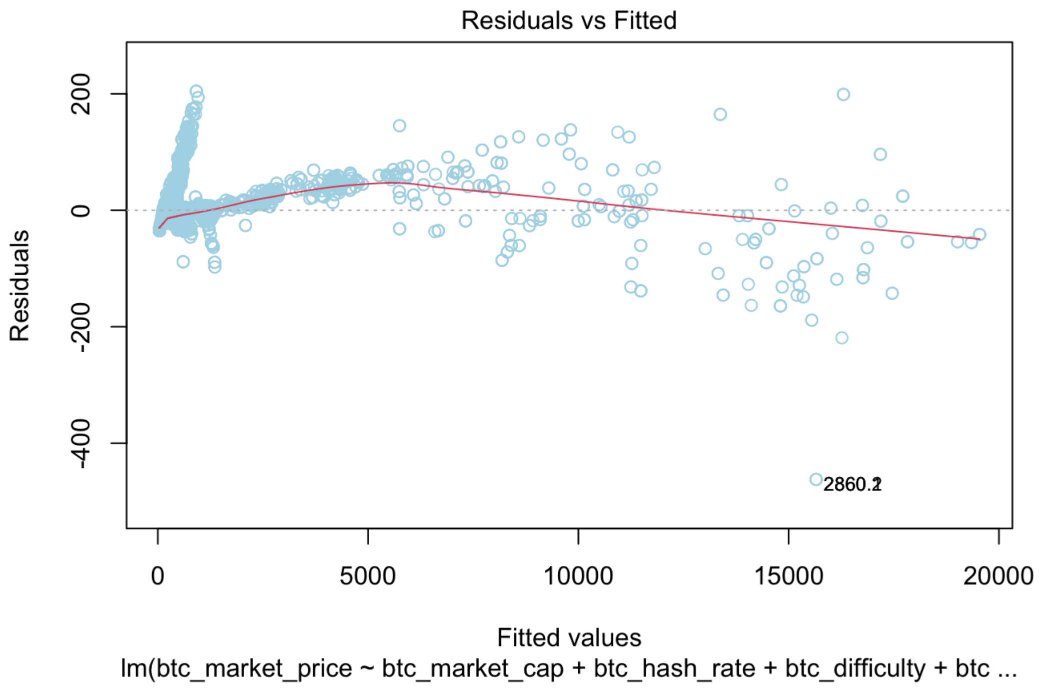
```

Call:
lm(formula = .outcome ~ ., data = dat)

Residuals:
    Min      1Q  Median      3Q     Max 
-462.15  -28.95  -10.49   14.14  204.46 

Coefficients:
                                      Estimate Std. Error  t value Pr(>|t|)    
(Intercept)                          1138.6559     0.5804 1961.911  < 2e-16 ***
btc_market_cap                       2422.4909     6.4585  375.086  < 2e-16 ***
btc_hash_rate                        -126.0507     5.0542  -24.940  < 2e-16 ***
btc_difficulty                        145.3881     5.7482   25.293  < 2e-16 ***
btc_miners_revenue                    201.3575     4.9855   40.389  < 2e-16 ***
btc_estimated_transaction_volume_usd   -9.2744     2.1326   -4.349 1.39e-05 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 42.73 on 5415 degrees of freedom
Multiple R-squared:  0.9997, Adjusted R-squared:  0.9997 
F-statistic: 4.101e+06 on 5 and 5415 DF,  p-value: < 2.2e-16
```

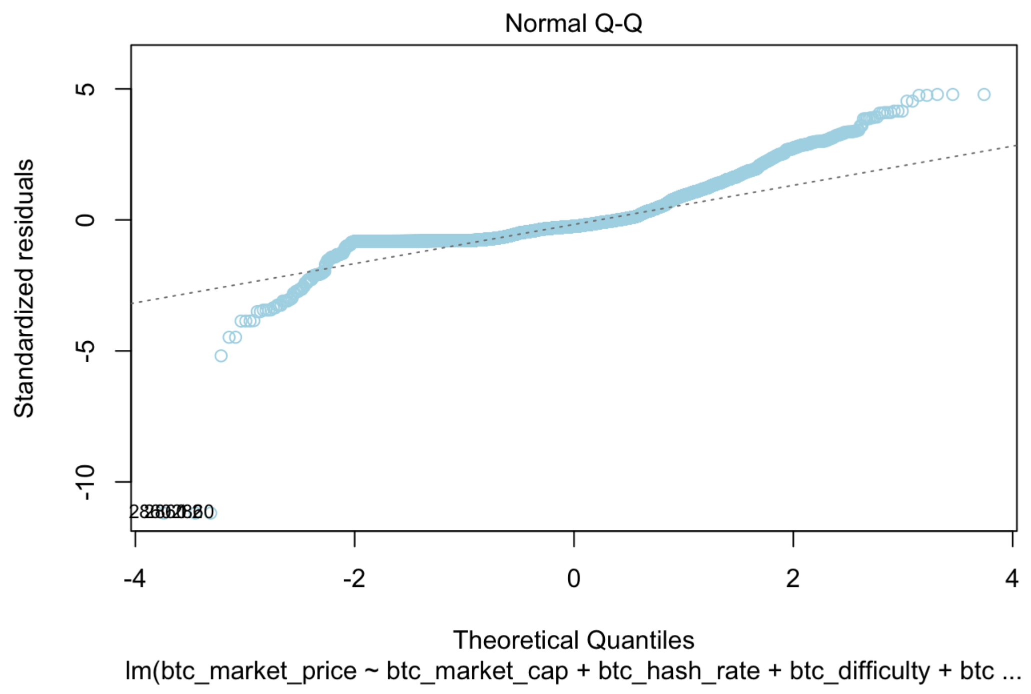
Residual Analysis in Linear Regression

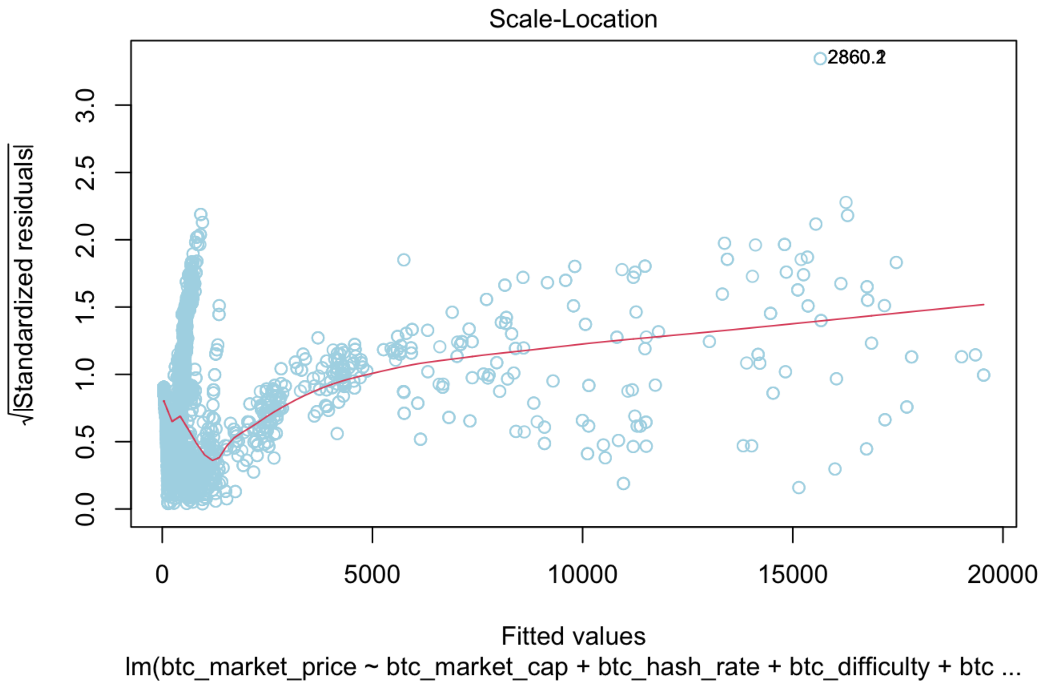
- Linear regression is a statistical method for for modelling the linear relationship between a dependent variable y (i.e. the one we want to predict) and one or more explanatory or independent variables(X).
- This residual plot will explain how residual plots generated by the regression function can be used to validate that some of the assumptions that are made about the dataset indicating it is suitable for a linear regression are met.There are a number of assupmtions we made about the data and these must be met for a linear model to work successfully and the standard residual plots can help validate some of these. These are:
  - The dataset must have some linear relationship
  - Multivariate normality - the dataset variables must be statistically Normally Distributed (i.e. resembling a Bell Curve)
  - It must have no or little multicollinearity - this means the independent variables must not be too highly correlated with each other. This can be tested with a Correlation matrix and other tests
  - No auto-correlation - Autocorrelation occurs when the residuals are not independent from each other. For instance, this typically occurs in stock prices, where the price is not independent from the previous price.

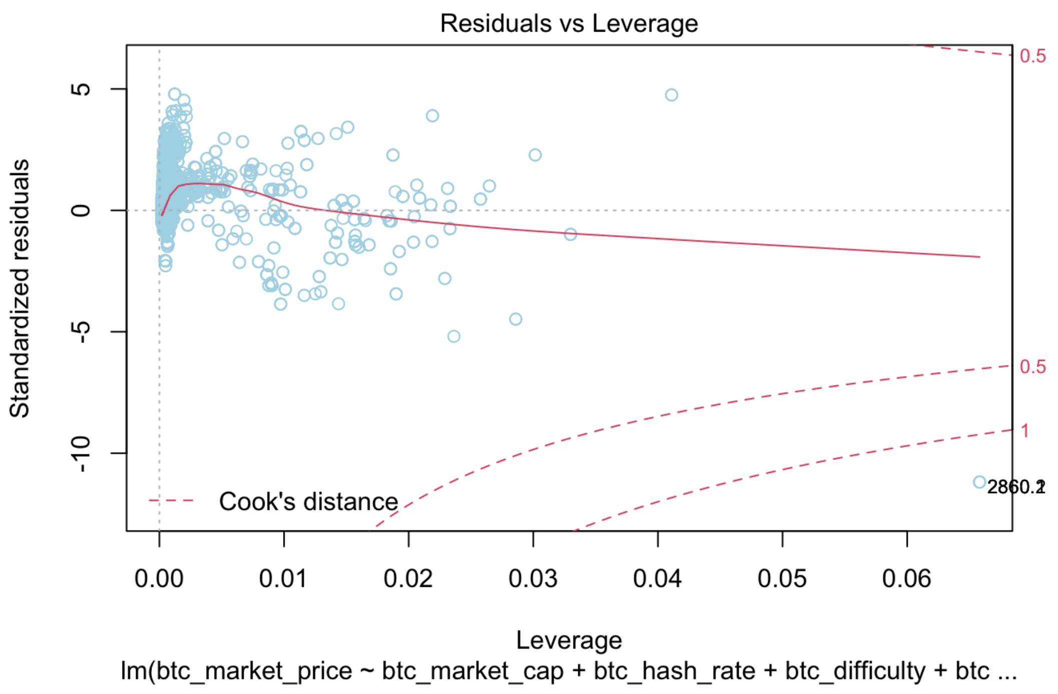
- Homoscedasticity - meaning that the residuals are equally distributed across the regression line i.e. above and below the regression line and the variance of the residuals should be the same for all predicted scores along the regression line.
- Four standard plots can be accessed using the plot() function with the fit variable once the model is generated. These can be used to show if there are problems with the dataset and the model produced that need to be considered in looking at the validity of the model. These are:
- Residuals vs Fitted Plot
- Normal Q–Q (quantile-quantile) Plot
- Scale-Location
- Residuals vs Leverage

- Looking at the summary, it has p-value of 2.2e-16, which indicates that there is a highly statistically significant relationship between the two variables. So, why do we need to look at other things like residuals?
- P-values by themselves can potentially be misleading without analyis of the residuals to ensure the model does not have any problems.
- 1.Residuals vs Fitted Plot: Residuals represent variation left unexplained by the model.Residual plots are used to look for underlying patterns in the residuals that may mean that the model has a problem.For a correct linear regression, the data needs to be linear so this will test if that condition is met.
- 2.Normal Q–Q (quantile-quantile) Plot Residuals should be normally distributed and the Q-Q Plot will show this. If residuals follow close to a straight line on this plot, it is a good indication they are normally distributed.For our model, the Q-Q plot shows pretty good alignment to the the line with a few points at the top slightly offset. Probably not significant and a reasonable alignment.
- 3.Scale-Location This plot test the linear regression assumption of equal variance (homoscedasticity) i.e. that the residuals have equal variance along the regression line. It is also called the Spread-Location plot. The residuals in our case have equal variance(occupy equal space) above and below the line and along the length of the line.
- 4.Residuals vs Leverage This plot can be used to find influential cases in the dataset. An influential case is one that, if removed, will affect the model so its inclusion or exclusion should be considered.An influential case may or may not be an outlier and the purpose of this chart is to identify cases that have high influence in the model. Outliers will tend to exert leverage and therefore influence on the model.An influential case will appear in the top right or bottom left of the chart inside a red line which marks Cook's Distance. An example is shown below. The case on the right shows 4 item, in the red dashed line. Removing this from the dataset would have a significant affect on the model, which may or may not be desirable.

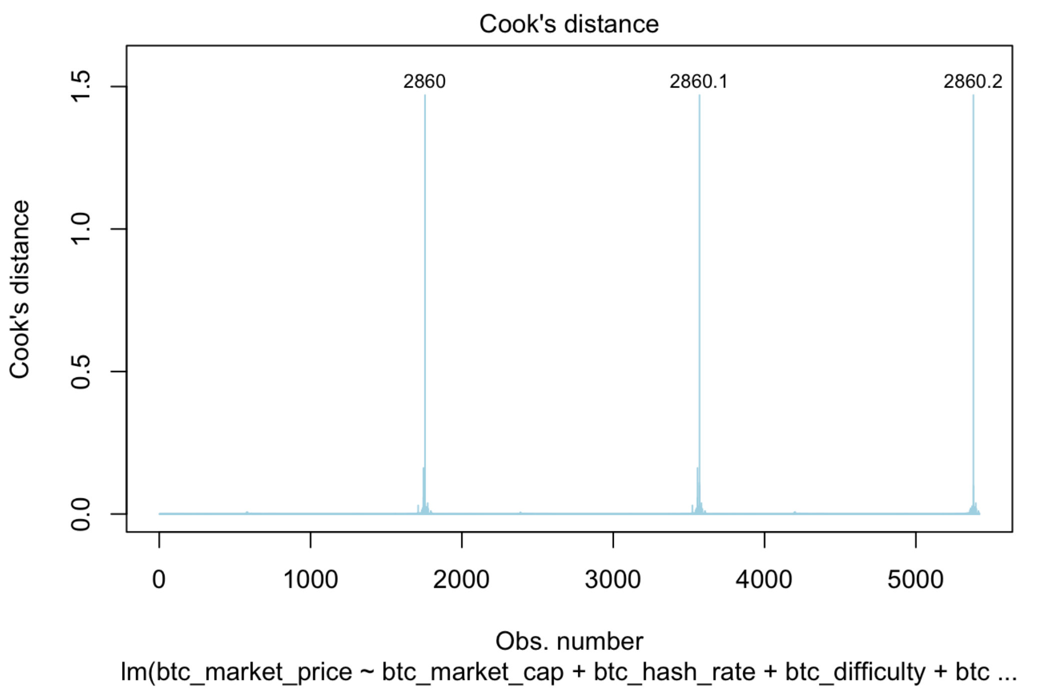



Normal Q-Q
Standardized residuals
5
0
-5
-10
2860
-4
-2
0
2
4
Theoretical Quantiles
lm(btc_market_price ~ btc_market_cap + btc_hash_rate + btc_difficulty + btc ...

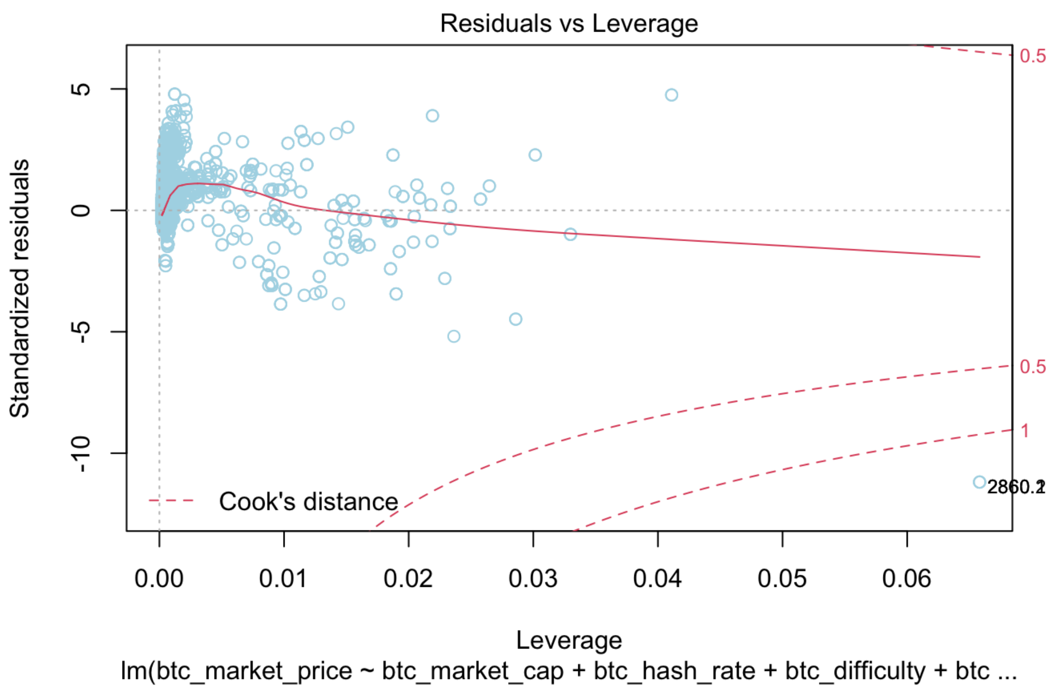

Scale-Location
√|Standardized residuals|
0.0
0.5
1.0
1.5
2.0
2.5
3.0
2860
0
5000
10000
15000
20000
Fitted values
lm(btc_market_price ~ btc_market_cap + btc_hash_rate + btc_difficulty + btc ...



Residuals vs Leverage
Standardized residuals
5
0
-5
-10
0.00
0.01
0.02
0.03
0.04
0.05
0.06
Leverage
lm(btc_market_price ~ btc_market_cap + btc_hash_rate + btc_difficulty + btc ...
Cook's distance
0.5
0.5
1
2860.2
2860.1



Cook's distance
Cook's distance
1.5
1.0
0.5
0.0
0
1000
2000
3000
4000
5000
Obs. number
lm(btc_market_price ~ btc_market_cap + btc_hash_rate + btc_difficulty + btc ...
2860
2860.1
2860.2



Linear Regression Prediction & Accuracy.

```
[1] "RMSE 33.8595903089837"
```

```
[1] "Error rate 0.0544967670074829"
```

```
[1] "R2 0.999093423880329"
```

Polynominal Regression

- Polynomial regression is a special case of linear regression where we fit a polynomial equation on the data with a curvilinear relationship between the target variable and the independent variables.
- In a curvilinear relationship, the value of the target variable changes in a non-uniform manner with respect to the predictor (s).
- poly function returns or evaluates orthogonal polynomials of degree 1 to degree over the specified set of points x: these are all orthogonal to the constant polynomial of degree 0.

```
poly_reg<-lm( btc_market_price~
                poly( btc_market_cap,2)+ poly( btc_hash_rate,2)+
             poly( btc_difficulty,2)+ poly( btc_miners_revenue,2)+
             poly( btc_estimated_transaction_volume_usd,2), data = subTrain1)
summary(poly_reg)
```

```
Call:
lm(formula = btc_market_price ~ poly(btc_market_cap, 2) + poly(btc_hash_rate, 
    2) + poly(btc_difficulty, 2) + poly(btc_miners_revenue, 2) + 
    poly(btc_estimated_transaction_volume_usd, 2), data = subTrain1)

Residuals:
    Min      1Q  Median      3Q     Max 
-148.28  -11.54   -4.62   14.30  265.60 

Coefficients:
                                                 Estimate Std. Error  t value
(Intercept)                                     1.139e+03  3.568e-01 3191.369
poly(btc_market_cap, 2)1                        1.795e+05  3.158e+02  568.435
poly(btc_market_cap, 2)2                        1.135e+03  1.172e+02    9.681
poly(btc_hash_rate, 2)1                        -1.127e+04  2.833e+02  -39.798
poly(btc_hash_rate, 2)2                         8.981e+02  1.111e+02    8.080
poly(btc_difficulty, 2)1                        9.082e+03  3.307e+02   27.467
poly(btc_difficulty, 2)2                       -1.172e+03  1.142e+02  -10.258
poly(btc_miners_revenue, 2)1                    1.788e+04  2.301e+02   77.726
poly(btc_miners_revenue, 2)2                   -4.560e+03  8.601e+01  -53.019
poly(btc_estimated_transaction_volume_usd, 2)1 -2.062e+03  1.208e+02  -17.063
poly(btc_estimated_transaction_volume_usd, 2)2  5.619e+02  5.648e+01    9.950
                                               Pr(>|t|)    
(Intercept)                                     < 2e-16 ***
poly(btc_market_cap, 2)1                        < 2e-16 ***
poly(btc_market_cap, 2)2                        < 2e-16 ***
poly(btc_hash_rate, 2)1                         < 2e-16 ***
poly(btc_hash_rate, 2)2                        7.89e-16 ***
poly(btc_difficulty, 2)1                        < 2e-16 ***
poly(btc_difficulty, 2)2                        < 2e-16 ***
poly(btc_miners_revenue, 2)1                    < 2e-16 ***
poly(btc_miners_revenue, 2)2                    < 2e-16 ***
poly(btc_estimated_transaction_volume_usd, 2)1  < 2e-16 ***
poly(btc_estimated_transaction_volume_usd, 2)2  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 26.27 on 5410 degrees of freedom
Multiple R-squared:  0.9999, Adjusted R-squared:  0.9999 
F-statistic: 5.426e+06 on 10 and 5410 DF,  p-value: < 2.2e-16
```

Polynomial Regression Prediction & Accuracy

```
[1] "RMSE 33.8595903089837"
```

```
[1] "Error rate 0.0544967670074829"
```

```
[1] "R Square 0.999682461536609"
```

Spline Regression

- Spline is a special function defined piece-wise by polynomials. The term “spline” is used to refer to a wide class of functions that are used in applications requiring data interpolation and/or smoothing. The data may be either one-dimensional or multi-dimensional.
- Spline Regression is one of the non-parametric regression technique. In this technique the dataset is divided into bins at intervals or points which we called as knots. Also this bin has its separate fit.
- The disadvantages of the polynomial regression can be overcome by using Spline Regression. Polynomial regression only captures a certain amount of curvature in a nonlinear relationship. An alternative, and often superior, approach to modeling nonlinear relationships is to use splines.
- Splines provide a way to smoothly interpolate between fixed points, called knots. Polynomial regression is computed between

knots. In other words, splines are series of polynomial segments strung together, joining at knots.
- The generic function quantile produces sample quantiles corresponding to the given probabilities.
- bs {splines} Generate the B-spline basis matrix for a polynomial spline.bs uses knots which is the internal breakpoints that define the spline.Typical values are the mean or median for one knot, quantiles for more knots.

```
knots <- quantile( subTrain1$btc_market_price, p = c( 0.25, 0.5, 0.75))

splinemodel<-lm( btc_market_price~
                bs( btc_market_cap, knots = knots)+ bs( btc_hash_rate, knots = knots)+
                bs( btc_difficulty, knots = knots)+ bs( btc_miners_revenue, knots = knots)+
                bs( btc_estimated_transaction_volume_usd, knots = knots), data = subTrain1)
```

Spline Regression Prediction & Accuracy

```
[1] "RMSE 13.4968546288147"
```

```
[1] "Error rate 0.0217230904251439"
```

```
[1] "R Square 0.999810613324564"
```

Generalized Linear Model

- Why Use GAM?Relationships between the individual predictors and the dependent variable follow smooth patterns that can be linear or nonlinear.
- Mathematically speaking, GAM is an additive modeling technique where the impact of the predictive variables is captured through smooth functions which—depending on the underlying patterns in the data—can be nonlinear.
- GAM can capture common nonlinear patterns that a classic linear model would miss.
- GAM framework allows us to control smoothness of the predictor functions to prevent overfitting. By controlling the wiggliness of the predictor functions, we can directly tackle the bias/variance tradeoff.

```
lmfit6 <- gam(btc_market_price ~ btc_estimated_transaction_volume_usd + btc_miners_revenue,
data=bitcoin_dataset)

summary(lmfit6)
```

```

Family: gaussian
Link function: identity

Formula:
btc_market_price ~ btc_estimated_transaction_volume_usd + btc_miners_revenue

Parametric coefficients:
                                      Estimate Std. Error t value Pr(>|t|)
(Intercept)                          2.834e-11  4.107e-13    69.0   <2e-16 ***
btc_estimated_transaction_volume_usd 8.448e-07  4.667e-08    18.1   <2e-16 ***
btc_miners_revenue                   3.234e-04  4.686e-06    69.0   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1


Rank: 2/3
R-sq.(adj) =  0.974   Deviance explained = 97.4%
GCV = 1.8968e+05  Scale est. = 1.8951e+05  n = 2258
```

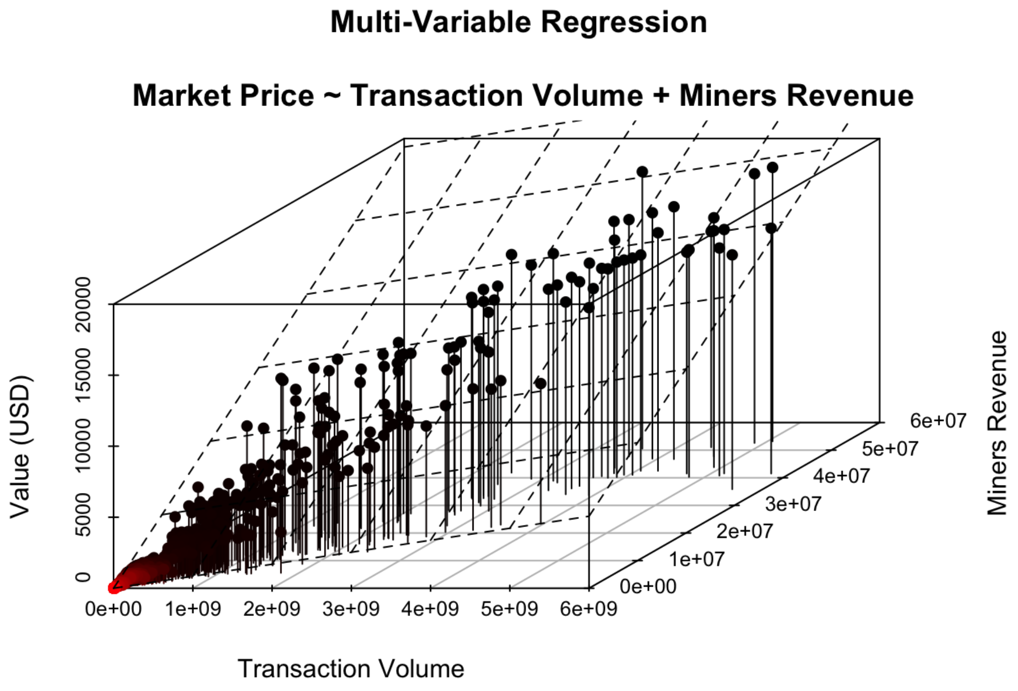
GAM Regression Prediction & Accuracy

```
[1] "RMSE 241.540564283585"
```

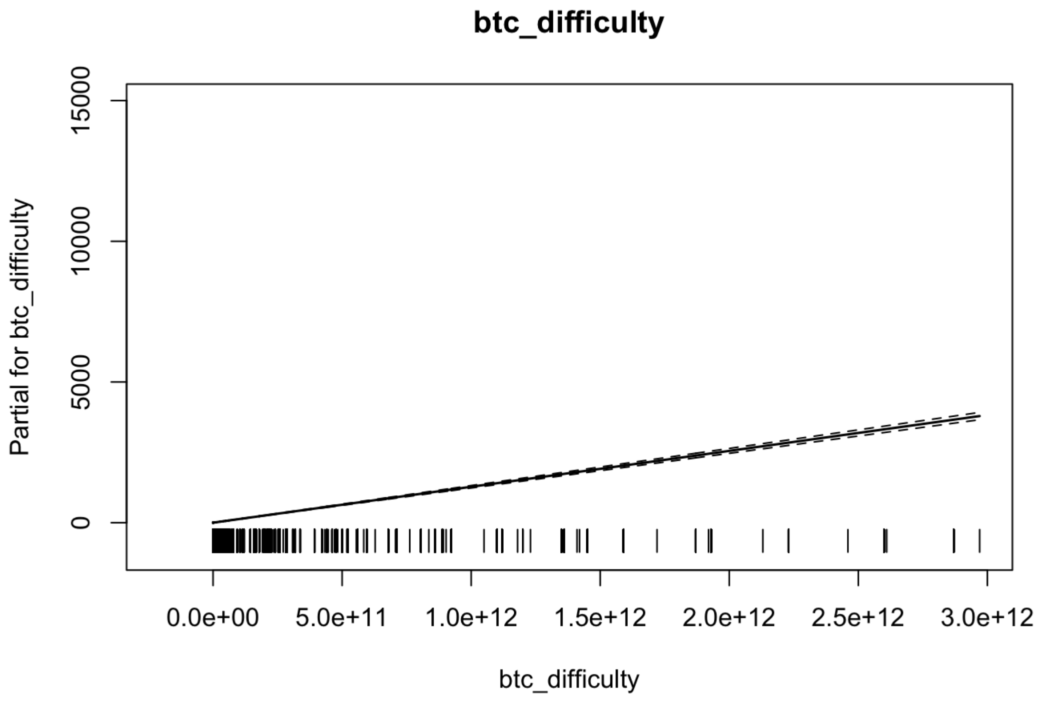
```
[1] "Error rate 0.388757800507882"
```

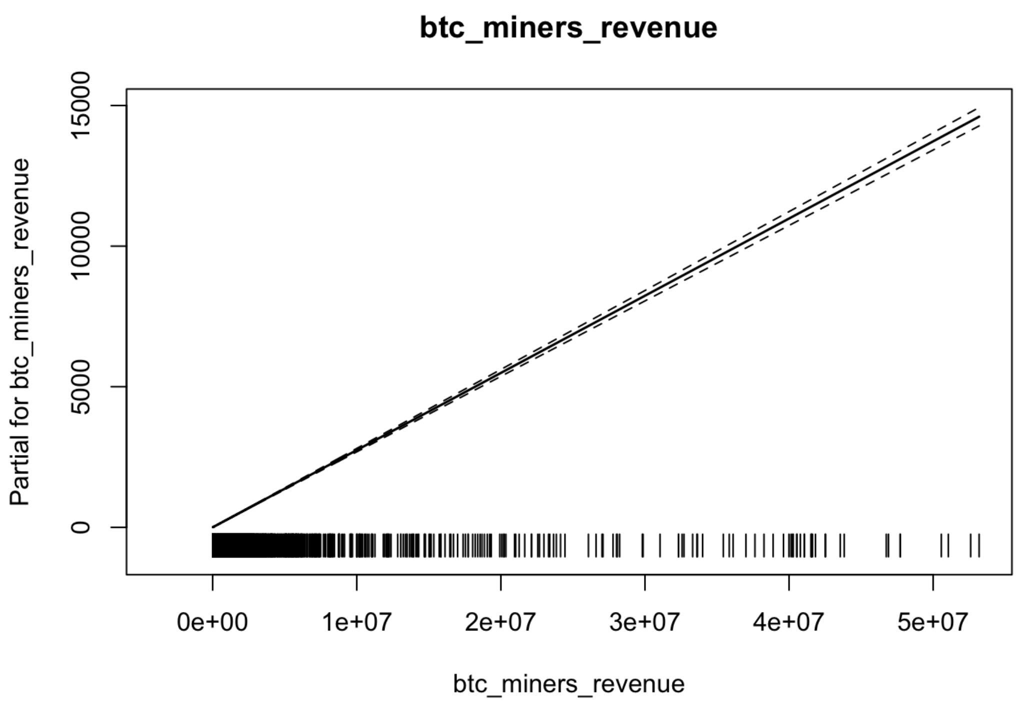
```
[1] "R Square 0.958059200834039"
```

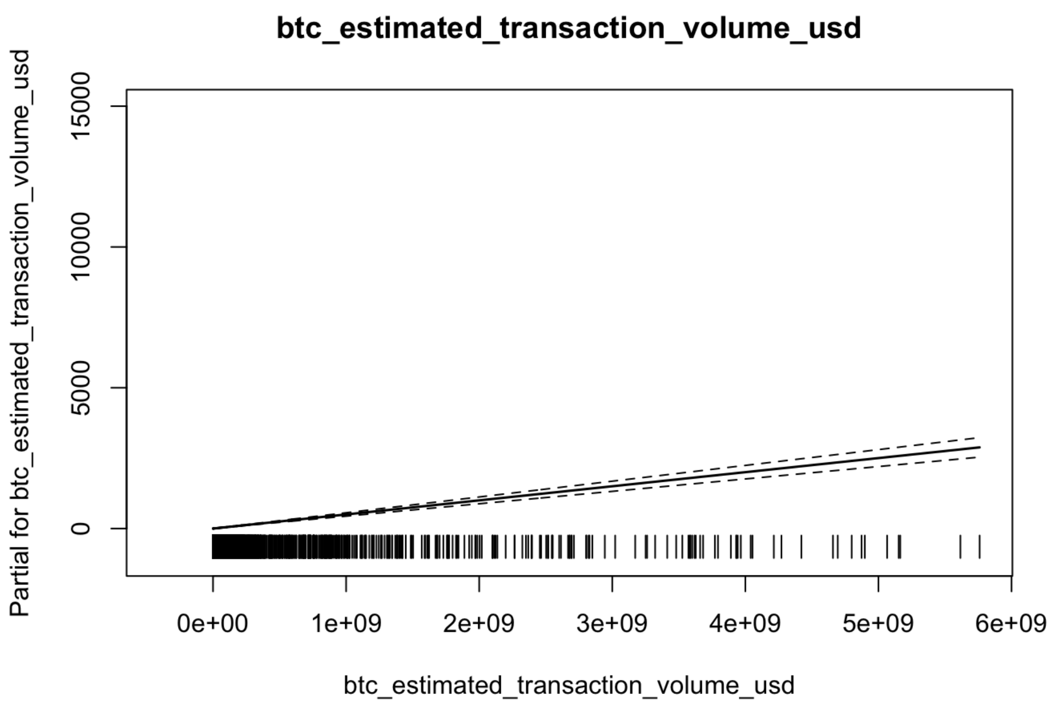
Scatter Plot 3D Visualization



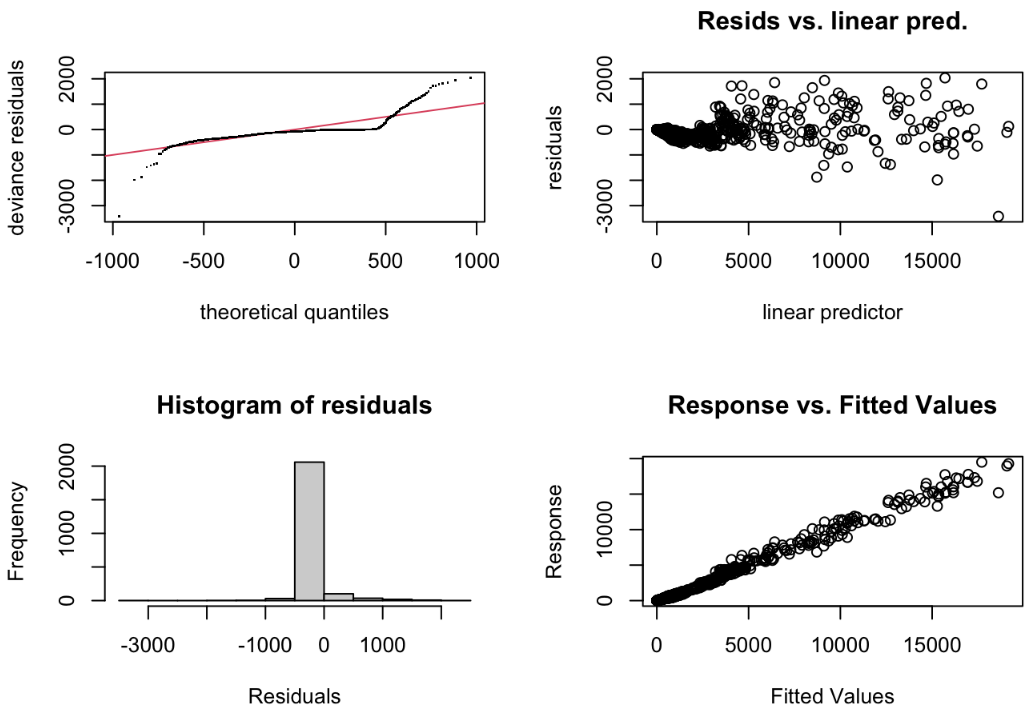
- Visualization part of gam model







- Diagnosing GAM model issues



```
Method: GCV   Optimizer: magic
Model required no smoothing parameter selectionModel rank =  3 / 4
```

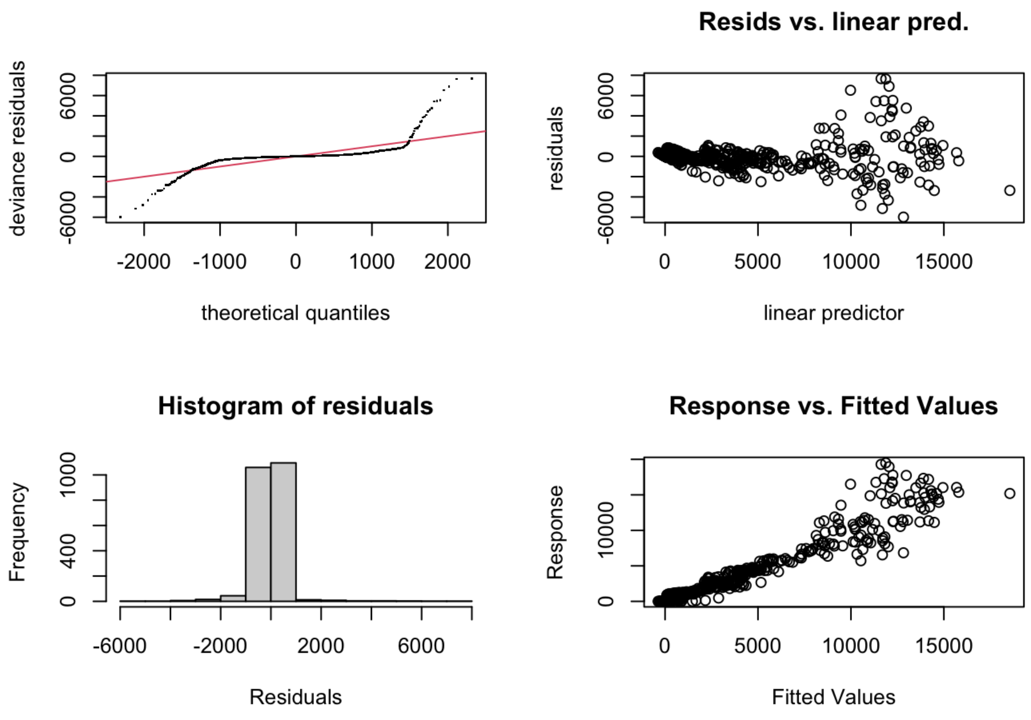
Penalized Cubic Regression Spline

- Cubic Regression Spline is specified by bs="cr". These have a cubic spline basis defined by a modest sized set of knots spread evenly through the covariate values. They are penalized by the conventional intergrated square second derivative cubic spline penalty.

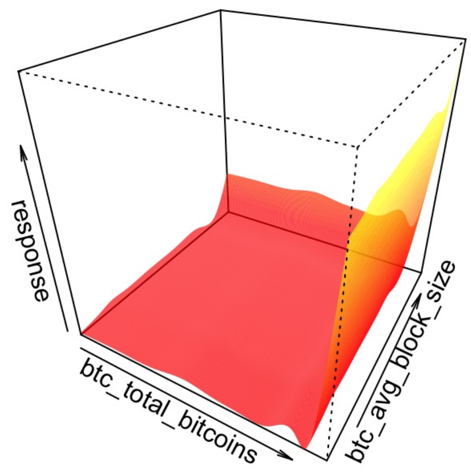
```
mod_lm4 <- gam(btc_market_price ~ s(btc_total_bitcoins, bs="cr")+s(btc_avg_block_size, bs="cr")+
                 s(btc_transaction_fees, bs="cr"),
               data=bitcoin_dataset)
summary(mod_lm4)
```

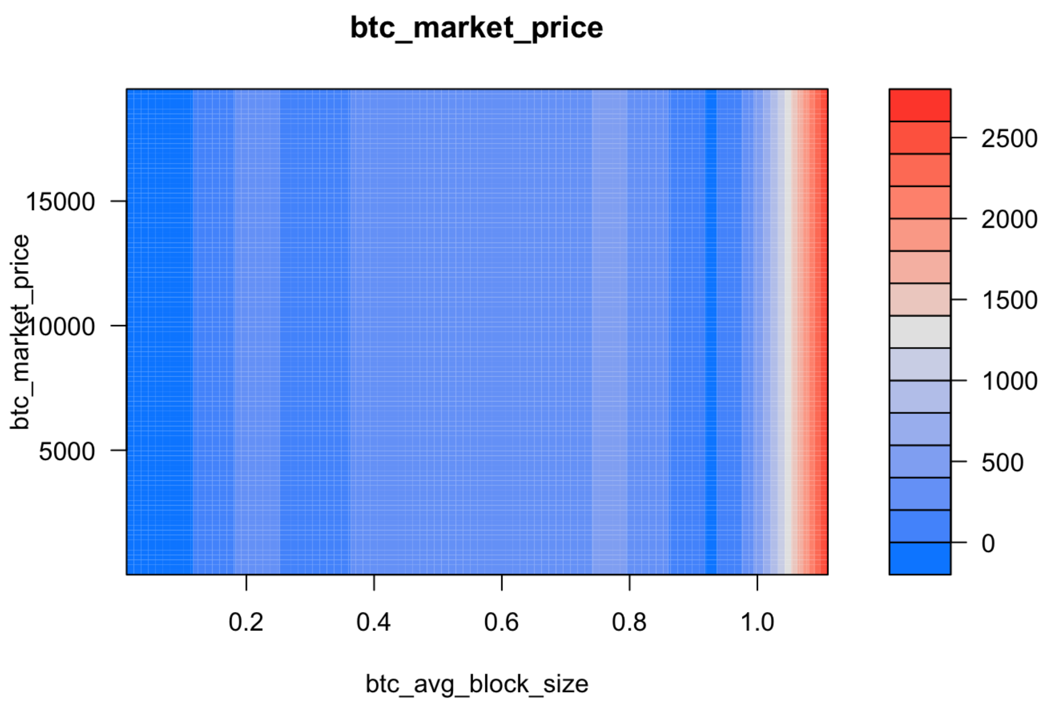
```
Family: gaussian
Link function: identity

Formula:
btc_market_price ~ s(btc_total_bitcoins, bs = "cr") + s(btc_avg_block_size,
    bs = "cr") + s(btc_transaction_fees, bs = "cr")

Parametric coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  1156.94      13.87    83.4   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Approximate significance of smooth terms:
                          edf Ref.df      F p-value
s(btc_total_bitcoins)   8.994  9.000 454.74  <2e-16 ***
s(btc_avg_block_size)   8.120  8.686  13.66  <2e-16 ***
s(btc_transaction_fees) 4.930  5.552 215.07  <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

R-sq.(adj) =  0.939   Deviance explained =   94%
GCV = 4.3899e+05  Scale est. = 4.3451e+05  n = 2258
```

- Diagnosing Penalized Cubic Regression Spline GAM model issues



```
Method: GCV   Optimizer: magic
Smoothing parameter selection converged after 17 iterations.
The RMS GCV score gradient at convergence was 0.5643143 .
The Hessian was positive definite.
Model rank =  28 / 28

Basis dimension (k) checking results. Low p-value (k-index<1) may
indicate that k is too low, especially if edf is close to k'.

                          k'  edf k-index p-value
s(btc_total_bitcoins)   9.00 8.99    0.14  <2e-16 ***
s(btc_avg_block_size)   9.00 8.12    0.91  <2e-16 ***
s(btc_transaction_fees) 9.00 4.93    1.00    0.45
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```





Cubic Regression Spline Prediction & Accuracy

```
[1] "RMSE 164.983422387511"
```

```
[1] "Error rate 0.265539631398429"
```

```
[1] "R Square 0.979740412173328"
```

Extreme Gradient Boosting

- XGBoost is an implementation of the Gradient Boosted Decision Trees algorithm.
- XGBoost supports various objective functions, including regression, classification and ranking.
- XGBoost gives among the best performances in many machine learning applications. It is optimized gradient-boosting machine learning library. The core algorithm is parallelizable and hence it can use all the processing power of your machine and the machines in your cluster. In R, according to the package documentation, since the package can automatically do parallel computation on a single machine, it could be more than 10 times faster than existing gradient boosting packages.
- XGBoost shines when we have lots of training data where the features are numeric or mixture of numeric and categorical fields. It is also important to note that xgboost is not the best algorithm out there when all the features are categorical or when the number of rows is less than the number of fields (columns).
- The data argument in the xgboost R function is for the input features dataset. It accepts a matrix, dgCMatrix, or local data file. The nrounds argument refers to the max number of iterations (i.e. the number of trees added to the model).
- There are different hyperparameters that we can tune and the parametres are different from baselearner to baselearner. In tree based learners, which are the most common ones in xgboost applications, the following are the most commonly tuned hyperparameters:
- learning rate: learning rate/eta- governs how quickly the model fits the residual error using additional base learners. If it is a smaller learning rate, it will need more boosting rounds, hence more time, to achieve the same reduction in residual error as one with larger learning rate. Typically, it lies between 0.01 - 0.3
- The three hyperparameters below are regularization hyperparameters.
- gamma: min loss reduction to create new tree split. default = 0 means no regularization.
- lambda: L2 reg on leaf weights. Equivalent to Ridge regression.
- alpha: L1 reg on leaf weights. Equivalent to Lasso regression.

- max_depth: max depth per tree. This controls how deep our tree can grow. The Larger the depth, more complex the model will be and higher chances of overfitting. Larger data sets require deep trees to learn the rules from data. Default = 6.
- subsample: % samples used per tree. This is the fraction of the total training set that can be used in any boosting round. Low value may lead to underfitting issues. A very high value can cause over-fitting problems.
- colsample_bytree: % features used per tree. This is the fraction of the number of columns that we can use in any boosting round. A smaller value is an additional regularization and a larger value may be cause overfitting issues.
- n_estimators: number of estimators (base learners). This is the number of boosting rounds.

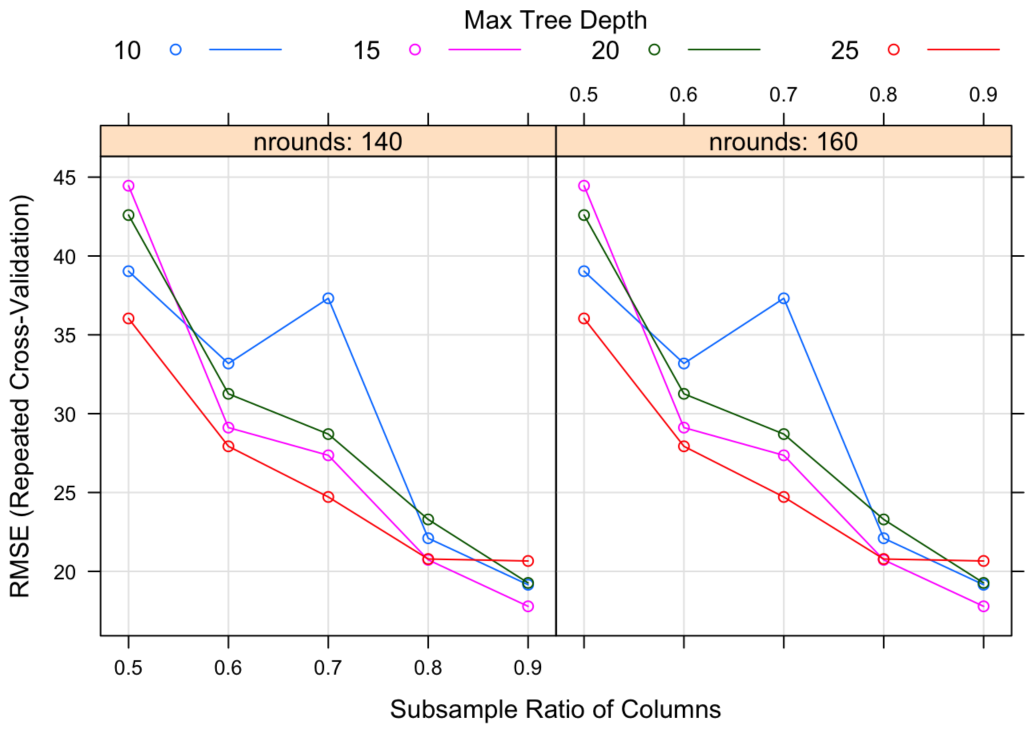
```
xgbGrid <- expand.grid(nrounds = c(140,160),  # this is n_estimators in the python code above
                       max_depth = c(10, 15, 20, 25),
                       colsample_bytree = seq(0.5, 0.9, length.out = 5),
                        The values below are default values in the sklearn-api.
                       eta = 0.3,
                       gamma=0,
                       min_child_weight = 1,
                       subsample = 1
)

model_xgb <- train(btc_market_price ~ .,
                   data = subTrain,
                   method = "xgbTree",
                   preProcess = c("scale", "center"),
                   trControl = trainControl(method = "repeatedcv",
                                            number = 5,
     s                                       repeats = 3,
                                            verboseIter = FALSE),
                   tuneGrid = xgbGrid,
                   verbose = 0)
```

```
model_xgb$results
```

```
   eta max_depth gamma colsample_bytree min_child_weight subsample nrounds
1  0.3        10     0              0.5                1         1     140
3  0.3        10     0              0.6                1         1     140
5  0.3        10     0              0.7                1         1     140
7  0.3        10     0              0.8                1         1     140
9  0.3        10     0              0.9                1         1     140
11 0.3        15     0              0.5                1         1     140
13 0.3        15     0              0.6                1         1     140
15 0.3        15     0              0.7                1         1     140
17 0.3        15     0              0.8                1         1     140
19 0.3        15     0              0.9                1         1     140
21 0.3        20     0              0.5                1         1     140
23 0.3        20     0              0.6                1         1     140
25 0.3        20     0              0.7                1         1     140
27 0.3        20     0              0.8                1         1     140
29 0.3        20     0              0.9                1         1     140
31 0.3        25     0              0.5                1         1     140
33 0.3        25     0              0.6                1         1     140
35 0.3        25     0              0.7                1         1     140
37 0.3        25     0              0.8                1         1     140
39 0.3        25     0              0.9                1         1     140
2  0.3        10     0              0.5                1         1     160
4  0.3        10     0              0.6                1         1     160
6  0.3        10     0              0.7                1         1     160
8  0.3        10     0              0.8                1         1     160
10 0.3        10     0              0.9                1         1     160
12 0.3        15     0              0.5                1         1     160
14 0.3        15     0              0.6                1         1     160
16 0.3        15     0              0.7                1         1     160
18 0.3        15     0              0.8                1         1     160
20 0.3        15     0              0.9                1         1     160
```

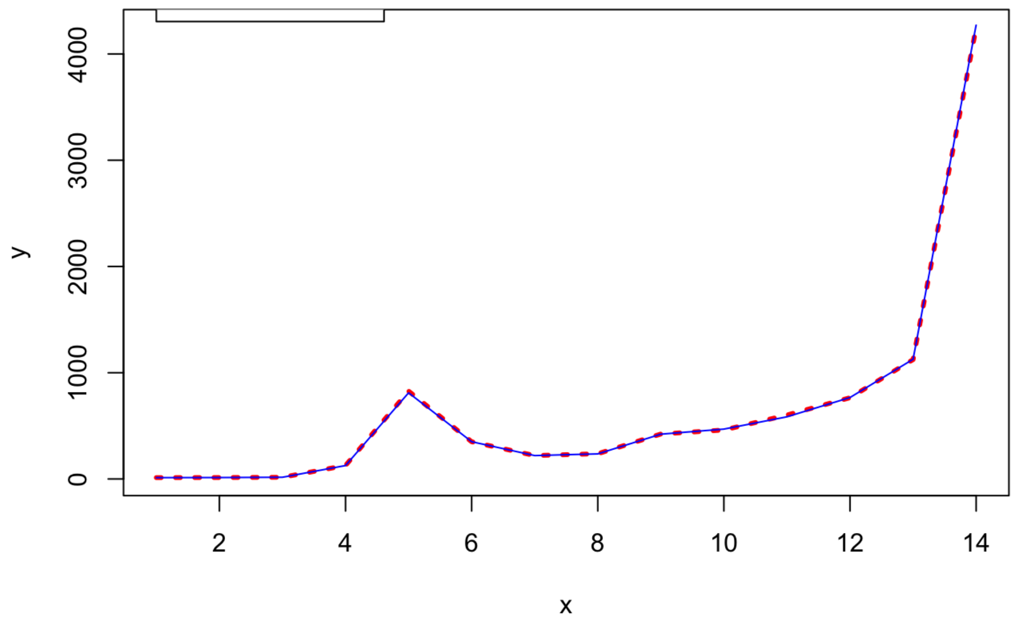
```
22 0.3       20     0           0.5                1         1     160
24 0.3       20     0           0.6                1         1     160
26 0.3       20     0           0.7                1         1     160
28 0.3       20     0           0.8                1         1     160
30 0.3       20     0           0.9                1         1     160
32 0.3       25     0           0.5                1         1     160
34 0.3       25     0           0.6                1         1     160
36 0.3       25     0           0.7                1         1     160
38 0.3       25     0           0.8                1         1     160
40 0.3       25     0           0.9                1         1     160
       RMSE  Rsquared      MAE    RMSESD   RsquaredSD     MAESD
1  39.03400 0.9997143 3.867351 25.023371 3.546794e-04 1.7577150
3  33.18108 0.9998120 3.425709 17.872421 1.829663e-04 1.3737375
5  37.31503 0.9997661 3.674326 19.179381 2.239330e-04 1.5042837
7  22.10033 0.9999162 2.489683 10.446471 8.835360e-05 0.9782250
9  19.15121 0.9999365 2.228535  9.277475 6.452859e-05 0.7787798
11 44.45382 0.9996732 4.240647 22.769227 3.019777e-04 1.5515580
13 29.11588 0.9998498 3.121364 16.054773 1.656704e-04 1.3285063
15 27.35755 0.9998652 2.875670 16.348729 1.340044e-04 1.4208245
17 20.72887 0.9999220 2.417198 12.214128 9.424477e-05 1.1041989
19 17.78226 0.9999508 2.127081  6.524760 3.807719e-05 0.6610867
21 42.59284 0.9996913 3.991861 22.418154 2.996915e-04 1.3100600
23 31.25821 0.9998335 3.172345 15.587641 1.466992e-04 1.2304602
25 28.70454 0.9998739 2.949386 10.537141 9.008602e-05 0.7973135
27 23.29094 0.9999056 2.660002 12.429602 9.178515e-05 1.1248811
29 19.26550 0.9999407 2.247684  7.132943 4.805357e-05 0.7821813
31 36.03812 0.9998017 3.798457 13.336798 1.265363e-04 0.9967262
33 27.92913 0.9998598 3.068617 15.623109 1.469525e-04 1.6367994
35 24.71652 0.9998876 2.767246 14.545188 1.309887e-04 1.2894564
37 20.78230 0.9999266 2.390776  9.009320 6.746089e-05 0.8252055
39 20.65820 0.9999333 2.252980  7.886065 5.210519e-05 0.7411809
2  39.03396 0.9997143 3.840191 25.023252 3.546726e-04 1.7560321
4  33.18124 0.9998120 3.401858 17.872200 1.829661e-04 1.3735135
6  37.31472 0.9997661 3.652922 19.179483 2.239312e-04 1.5032044
8  22.10016 0.9999162 2.469937 10.446723 8.835524e-05 0.9790382
10 19.15092 0.9999365 2.211328  9.277738 6.453012e-05 0.7808012
12 44.45381 0.9996732 4.240394 22.769204 3.019772e-04 1.5515495
14 29.11586 0.9998498 3.121129 16.054796 1.656708e-04 1.3285211
16 27.35753 0.9998652 2.875456 16.348726 1.340042e-04 1.4208145
18 20.72887 0.9999220 2.417110 12.214123 9.424471e-05 1.1041939
20 17.78226 0.9999508 2.126974  6.524756 3.807719e-05 0.6611015
22 42.59281 0.9996913 3.991856 22.418093 2.996899e-04 1.3100574
24 31.25817 0.9998335 3.172340 15.587609 1.466987e-04 1.2304575
26 28.70448 0.9998739 2.949379 10.537133 9.008597e-05 0.7973170
28 23.29092 0.9999056 2.659998 12.429597 9.178509e-05 1.1248812
30 19.26545 0.9999407 2.247677  7.132948 4.805362e-05 0.7821824
32 36.03809 0.9998017 3.798454 13.336753 1.265357e-04 0.9967265
34 27.92913 0.9998598 3.068615 15.623119 1.469526e-04 1.6368023
36 24.71647 0.9998876 2.767241 14.545150 1.309883e-04 1.2894575
38 20.78225 0.9999266 2.390771  9.009340 6.746094e-05 0.8252071
40 20.65812 0.9999333 2.252975  7.886036 5.210475e-05 0.7411819
```

```
plot(model_xgb)
```



Extreme Gradient Boosting Prediction & Accuracy

```
MSE:  80.92633 MAE:  5.168907  RMSE:  8.995907
```



```
[1] "RMSE 8.99590656624651"
```

```
[1] "Error rate 0.0133612673135272"
```

```
[1] "R Square 0.999954333588047"
```

Compare Regression Models

```
'data.frame':	1 obs. of  4 variables:
 $ Algorithm: chr "Linear Regression"
 $ RMSE     : num 33.9
 $ R2       : num 0.999
 $ Error    : num 0.0545
```

```
'data.frame':	1 obs. of  4 variables:
 $ Algorithm: chr "Polynomial Regression"
 $ RMSE     : num 33.9
 $ R2       : num 1
 $ Error    : num 0.0545
```

```
'data.frame':	1 obs. of  4 variables:
 $ Algorithm: chr "Spline Regression"
 $ RMSE     : num 13.5
 $ R2       : num 1
 $ Error    : num 0.0217
```

```
'data.frame':	1 obs. of  4 variables:
 $ Algorithm: chr "GAM Regression"
 $ RMSE     : num 242
 $ R2       : num 0.958
 $ Error    : num 0.389
```

```
'data.frame':	1 obs. of  4 variables:
 $ Algorithm: chr "Cubic Regression Spline"
 $ RMSE     : num 165
 $ R2       : num 0.98
 $ Error    : num 0.266
```

```
'data.frame':	1 obs. of  4 variables:
 $ Algorithm: chr "Extreme Gradient Boosting"
 $ RMSE     : num 9
 $ R2       : num 1
 $ Error    : num 0.0134
```

Conclusion

As such, we provide evidence suggesting that technical analysis is useful in a market like bitcoin whose value is mainly driven by by fundamental factors. Extreme Gradient Boosting outperforms other model with lesser error and with R2=1 the model completely fit.