Project 1 & 2

Mason Chase

Ethereum and ERC20 Tokens

Ethereum is a distributed platform that eliminates the need to have big centralized servers. It replaces the servers with many volunteers that the scripts run on. So instead of a third-party service determining what can be contracted over network, Ethereum allows for a decentralized system that virtually eliminates censorship. The ERC20 token is type of digital currency that is built on the Ethereum platform. Instead of US money, where the federal reserve is in charge of the money in a centralized fashion, the Ethereum network allows ERC20 to be decentralized which resists government bias and also allows for a more secure network without one major point of failure.

BAT Token

The BAT or Basic Attention Token is a token aimed toward online advertising. There is a big problem with advertisements on various social media platforms and web pages. Given the annoyance of ads and popularity of ad blockers on the web the idea was to create a browser which blocked adds and rewarded websites for user attention. This called for a decentralized ad exchange. The idea is that users would be anonymized and that advertisers would be rewarded with basic attention tokens when users spend attentive time looking at their ad and users would be rewarded when they choose to receive ads. Then in turn the advertisers would reward the publishers who displayed their ad to the user. This allows for accountability, removes the middle men and allows for a freer market place that returns the power back to the users, publishers, and advertisers.

Project Goal

There is a twofold goal to this project. The first goal is to find the best fit distribution to the number of buys and sells that a user makes. The first thing to do is remove outliers that had transactions of impossible amounts and then find which distribution fit the data best. In the second part of the project the goal is to create layers of transactions of increasing amounts and then find the correlation between these layers and different features in the data. Finally we will create a linear regression model to see if we can model price return based off of block chain features and purchase patterns.

Project 1 - Part 1

My UT-D User ID mod 20 is 12, so I picked the 12th token ordered by file size which was BAT.

Preprocessing

```
## Read in the BAT token
bat <- read_delim('networkbatTX.txt', delim = " ", col_names = F)
```

```
## Parsed with column specification:
## cols(
##   X1 = col_integer(),
##   X2 = col_integer(),
##   X3 = col_integer(),
##   X4 = col_double()
## )
```

```
names(bat) <- c('fromID', 'toID', 'unixTime', 'tokenAmount')

## filter out tokenAmounts > the total circulating token amount
decimals <- 10^18
```

```r
supply <- 1.5 * 10^9
batFiltered <- bat %>% filter(tokenAmount < decimals * supply)

## Find number of transactions with > 10^18 tokens
## This is negligable, so move on.
bat %>% filter(tokenAmount >= decimals * supply) %>% nrow()
```

```
## [1] 6
```

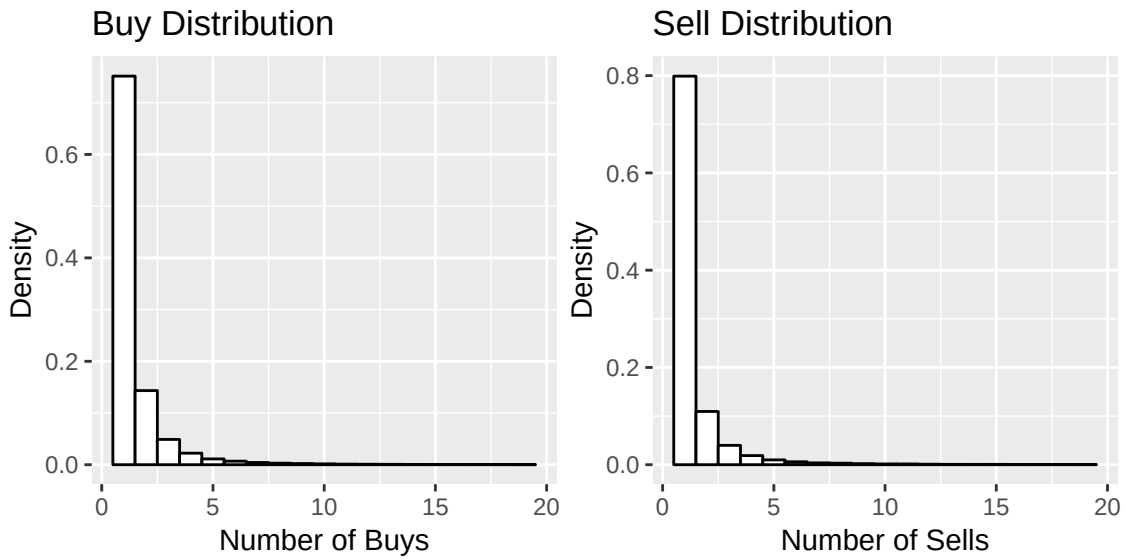
```r
## number of buys by user id
buys.distribution <- batFiltered %>% group_by(toID) %>% summarise(n = n()) %>% ungroup

## show highest 10 buyers and their number of buys
buys.distribution %>% arrange(-n) %>% head(10)
```

```
## # A tibble: 10 x 2
##      toID     n
##     <int> <int>
##  1 351141 41614
##  2      5 13780
##  3 351248 12905
##  4     49  3344
##  5 296381  2338
##  6  75994  1506
##  7 352681  1177
##  8 310645   958
##  9 142341   890
## 10 352820   880
```

```r
## number of sells by user id
sells.distribution <- batFiltered %>% group_by(fromID) %>% summarise(n = n()) %>% ungroup

## show highest 10 buyers and their number of buys
buys.distribution %>% arrange(-n) %>% head(10)
```

```
## # A tibble: 10 x 2
##      toID     n
##     <int> <int>
##  1 351141 41614
##  2      5 13780
##  3 351248 12905
##  4     49  3344
##  5 296381  2338
##  6  75994  1506
##  7 352681  1177
##  8 310645   958
##  9 142341   890
## 10 352820   880
```

Plot of distribution of Buys and Sells



Fitting Best Distribution - Buys

We will try numerous different distributions to find the best fit.

```
fit.exp.buy <- fitdist(buys.distribution$n, 'exp')
fit.gamma.buy <- fitdist(buys.distribution$n, 'gamma',
                         lower = c(0, 0), start = list(scale = 1, shape = 1))
fit.geometric.buy <- fitdist(buys.distribution$n, 'geom')
fit.log.buy <- fitdist(buys.distribution$n, 'logis')
fit.lnorm.buy <- fitdist(buys.distribution$n, 'lnorm')
fit.nbinom.buy <- fitdist(buys.distribution$n, 'nbinom')
fit.norm.buy <- fitdist(buys.distribution$n, 'norm')
fit.pois.buy <- fitdist(buys.distribution$n, 'pois')
fit.unif.buy <- fitdist(buys.distribution$n, 'unif')
fit.weibull.buy <- fitdist(buys.distribution$n, 'weibull')

## find the best fit distribution
gofstat(list(fit.weibull.buy, fit.gamma.buy, fit.lnorm.buy,
             fit.exp.buy, fit.log.buy))
```

```
## Goodness-of-fit statistics
##                               1-mle-weibull  2-mle-gamma  3-mle-lnorm
## Kolmogorov-Smirnov statistic     0.4699829    0.3982284    0.4366277
## Cramer-von Mises statistic    6482.4578347 6556.1684561 5431.7968081
## Anderson-Darling statistic             Inf          Inf          Inf
##                                 4-mle-exp  5-mle-logis
## Kolmogorov-Smirnov statistic    0.3983325    0.4190231
## Cramer-von Mises statistic   6556.5789931 6019.7818411
## Anderson-Darling statistic            Inf          Inf
##
## Goodness-of-fit criteria
##                                1-mle-weibull 2-mle-gamma 3-mle-lnorm
## Akaike's Information Criterion      478922.1    520944.3    299214.8
## Bayesian Information Criterion      478941.8    520964.0    299234.5
##                                4-mle-exp 5-mle-logis
## Akaike's Information Criterion  520942.4    684645.5
## Bayesian Information Criterion  520952.2    684665.2
```

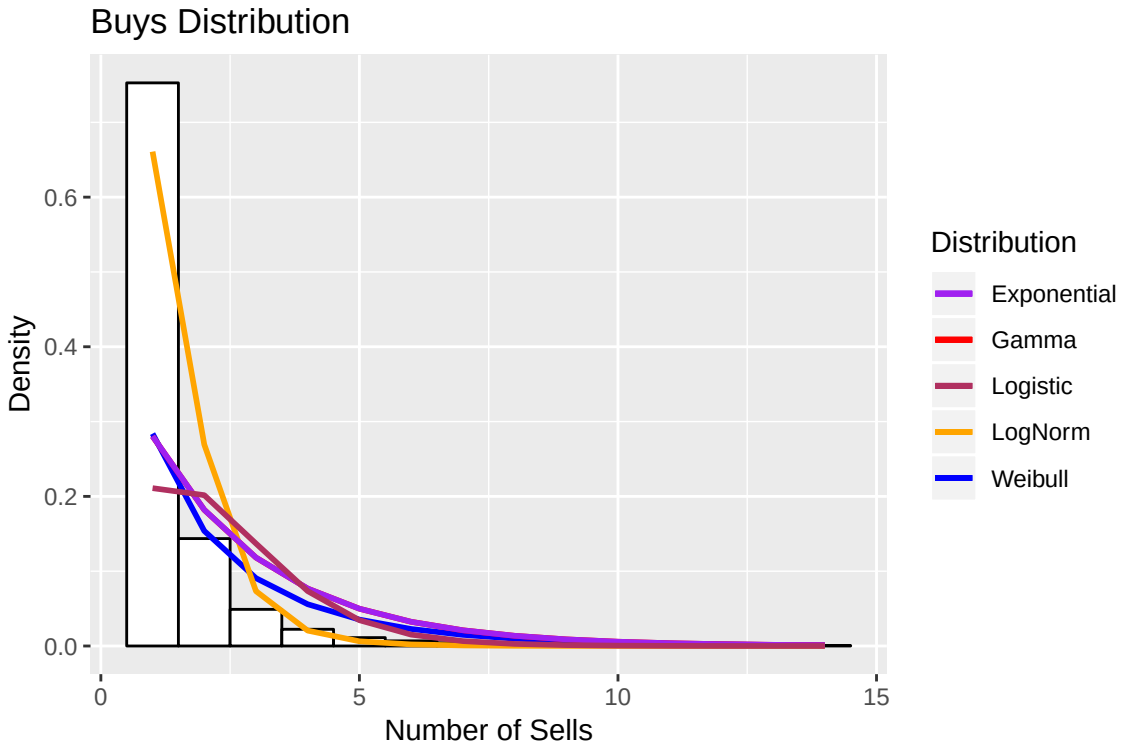
Fitting Best Distribution - Buys

We will try numerous different distributions to find the best fit for sells as well.

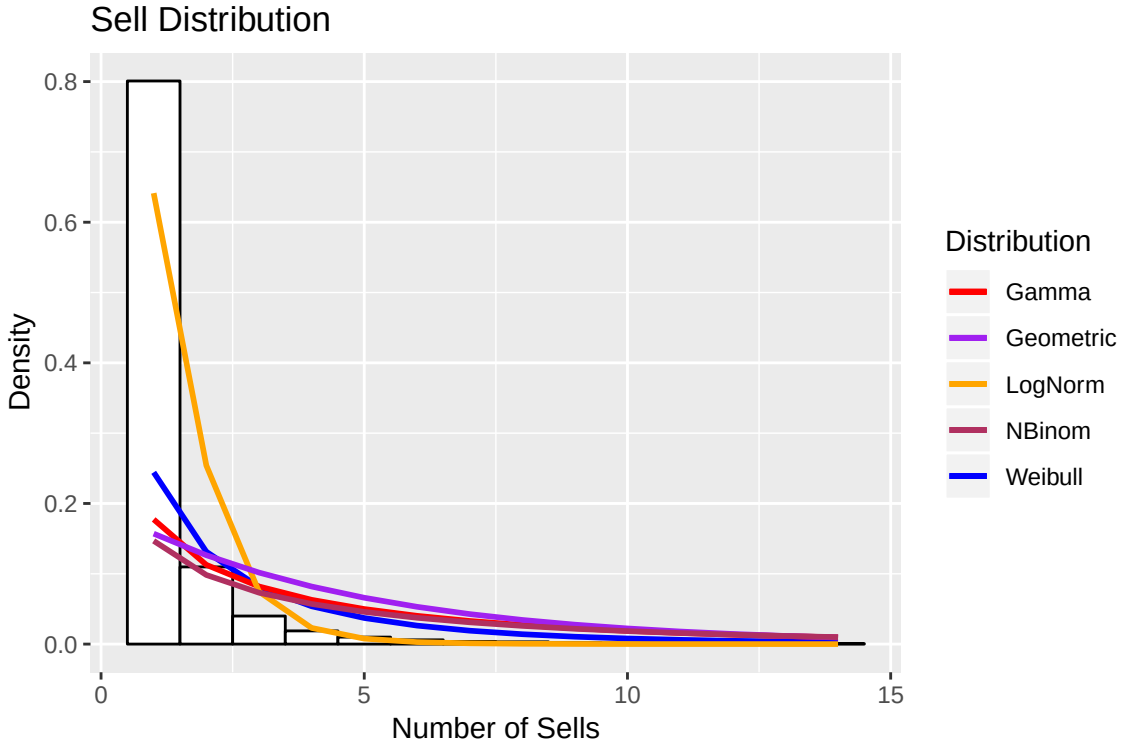
```
fit.exp.sell <- fitdist(sells.distribution$n, 'exp')
fit.gamma.sell <- fitdist(sells.distribution$n, 'gamma',
                          lower = c(0, 0), start = list(scale = 1, shape = 1))
fit.geometric.sell <- fitdist(sells.distribution$n, 'geom')
fit.log.sell <- fitdist(sells.distribution$n, 'logis')
fit.lnorm.sell <- fitdist(sells.distribution$n, 'lnorm')
fit.nbinom.sell <- fitdist(sells.distribution$n, 'nbinom')
fit.norm.sell <- fitdist(sells.distribution$n, 'norm')
fit.pois.sell <- fitdist(sells.distribution$n, 'pois')
fit.unif.sell <- fitdist(sells.distribution$n, 'unif')
fit.weibull.sell <- fitdist(sells.distribution$n, 'weibull')

gofstat(list(fit.weibull.sell, fit.gamma.sell, fit.lnorm.sell,
             fit.geometric.sell, fit.nbinom.sell))
```

```
## Goodness-of-fit statistics
##                                1-mle-weibull  2-mle-gamma  3-mle-lnorm
## Kolmogorov-Smirnov statistic     0.4825589    0.4309947    0.4543434
## Cramer-von Mises statistic    4186.6137751 4948.4589616 3557.3393306
## Anderson-Darling statistic             Inf          Inf          Inf
##                                 4-mle-geom 5-mle-nbinom
## Kolmogorov-Smirnov statistic    0.4409136    0.4744926
## Cramer-von Mises statistic   5177.1666324 4748.7628105
## Anderson-Darling statistic            Inf          Inf
##
## Goodness-of-fit criteria
##                                  1-mle-weibull 2-mle-gamma 3-mle-lnorm
## Akaike's Information Criterion      286861.7    356821.9    174059.5
## Bayesian Information Criterion      286880.2    356840.4    174078.1
##                                  4-mle-geom 5-mle-nbinom
## Akaike's Information Criterion   402104.8     378436.6
## Bayesian Information Criterion   402114.1     378455.2
```

Plot Distributions Best Fit - Buys



Plot Distributions Best Fit - Sells



Conclusion - Part 1 - Buys

Considering both the goodness of fit statistics and the line on the plot above, the Log-Normal distribution seems to fit the Buy distribution the best. It makes sense that the Log-Normal distribution fits best as most of the buyers

will make relatively few transactions with relatively few heavy hitters. The Log-Normal distribution tends to have a higher density towards 0 which leads us to believe our finding are reasonable. The distribution parameters are as follows:

```
fit.lnorm.buy
```

```
## Fitting of the distribution ' lnorm ' by maximum likelihood
## Parameters:
##          estimate  Std. Error
## meanlog 0.2619459 0.001423540
## sdlog   0.5356539 0.001006579
```

Conclusion - Part 1 - Sells

Considering both the goodness of fit statistics and the line on the plot above, the Log-Normal distribution seems to fit the Sell distribution the best as well. Similar to the distribution for buys, this makes sense as well. Most people who do not buy a lot of a transient product are not going to sell a lot either. So for both the buy and sell distribution, Log-Normal is the best fit distribution of the distributions tested. The distribution parameters are as follows:

```
fit.lnorm.sell
```

```
## Fitting of the distribution ' lnorm ' by maximum likelihood
## Parameters:
##          estimate  Std. Error
## meanlog 0.2372370 0.002023510
## sdlog   0.5704862 0.001430818
```

Project 1 Part 2

For the second part of the project, the goal is to create layers based on increasing transaction amounts and find the correlation with these layers and different features in the data.

First, read in the price file for BAT. Since the token purchase have a high positive skew, the initial idea was to normalize the token amount so that the levels would be more normally distributed.

```
## Read in and merge the price and trade data

## get the prices
batprices <- read_delim('bat.txt', delim = "\t", col_names = T)
```

```
## Parsed with column specification:
## cols(
##   Date = col_character(),
##   Open = col_double(),
##   High = col_double(),
##   Low = col_double(),
##   Close = col_double(),
##   Volume = col_number(),
##   `Market Cap` = col_character()
## )
```

```
names(batprices) <- make.names(names(batprices))
batprices <- batprices %>% mutate(date = as.Date(Date, format = '%m/%d/%Y'))

## get the trades
bat <- read_delim('networkbatTX.txt', delim = " ", col_names = F)
```

```
## Parsed with column specification:
## cols(
##   X1 = col_integer(),
```

```
##   X2 = col_integer(),
##   X3 = col_integer(),
##   X4 = col_double()
## )
```

```
names(bat) <- c('fromID', 'toID', 'unixTime', 'tokenAmount')
decimals <- 10^18
supply <- 1.5 * 10^9
batFiltered <- bat %>%
  filter(tokenAmount < decimals * supply)

## convert the timestamp to a date
bat.time <- batFiltered %>%
  mutate(date = anydate(unixTime))

## merge the prices
bat.time.prices <- merge(bat.time, batprices %>% dplyr::select(Open:Volume, date))

## find the skewness
skewness.1 <- skewness((bat.time.prices$tokenAmount))
cat('The skewness of the token amounts is:', skewness.1)
```

```
## The skewness of the token amounts is: 385.1852
```

```
## make the data normal
skewness.norm <- skewness((bat.time.prices$tokenAmount) ^ (0.0784))
cat('The skewness of the token amounts after normalization is:', skewness.norm)
```

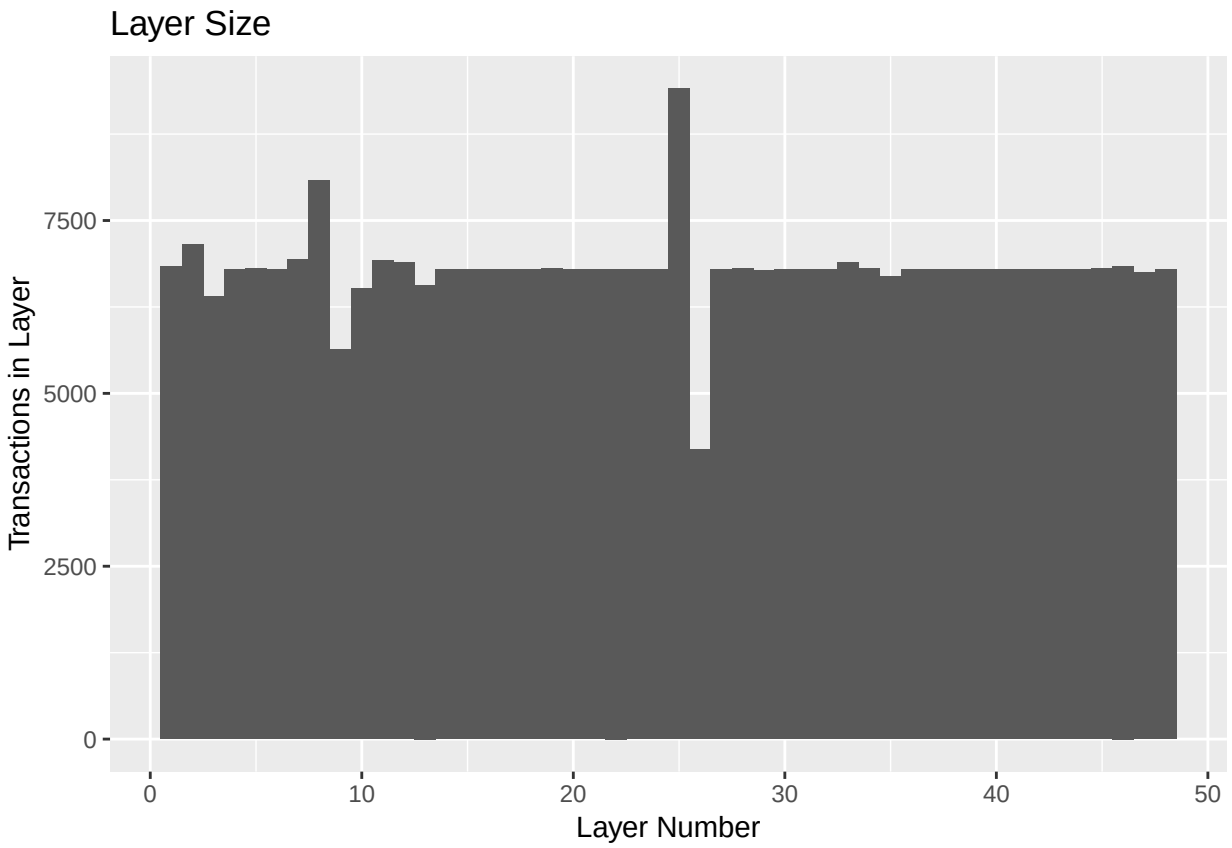
```
## The skewness of the token amounts after normalization is: -5.778597e-05
```

Create Layers

Now that we have a normalized token amount, we decided that we can create our layers based on the normalized amounts. The goal was to have somewhere around 20 transactions per layer per day, so we ended up having 48 total layers.

```
G <- ceiling((nrow(bat.time.prices) / length(unique(bat.time.prices$date))) / 20)
BAT.data <- bat.time.prices %>%
  mutate(normTokenAmt = tokenAmount ^ 0.0784,
         layer = as.numeric(cut2(normTokenAmt, g = G)))
```

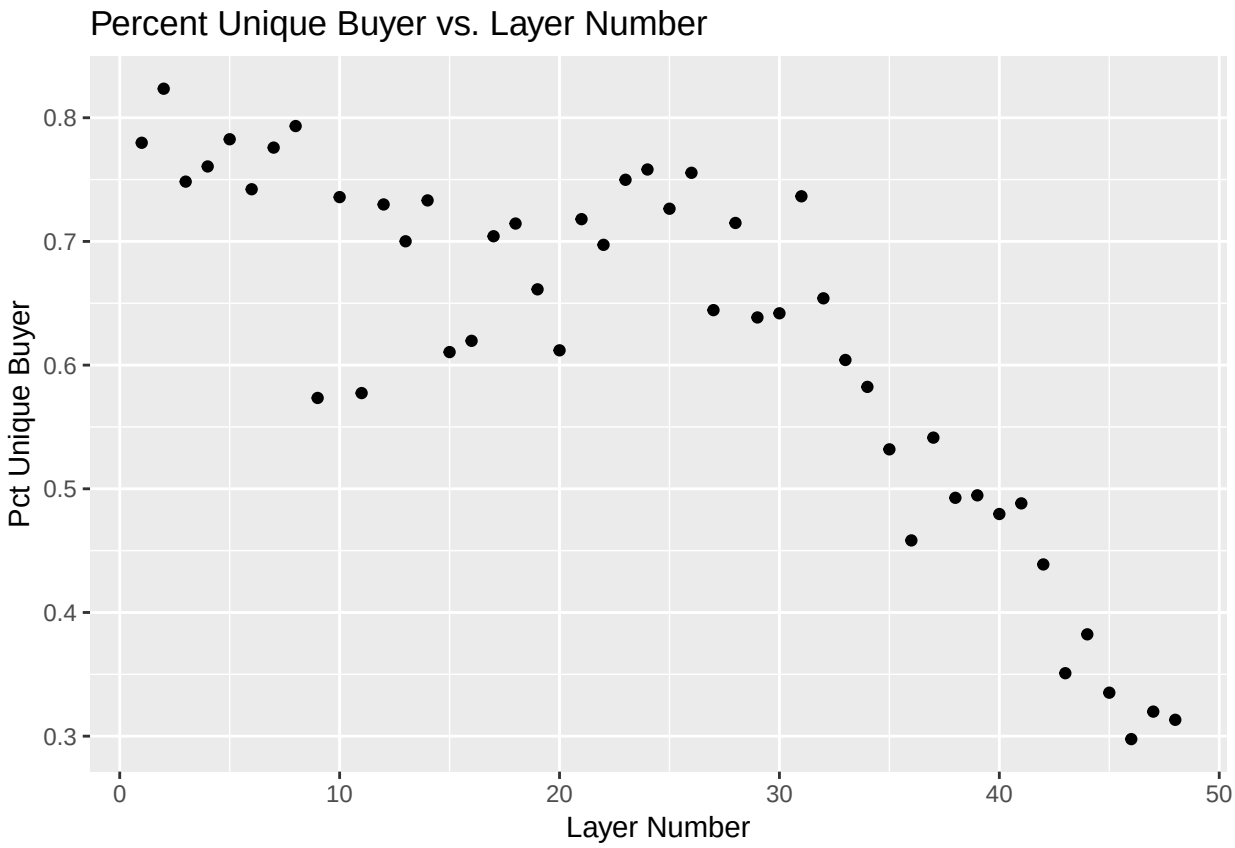
We came to the conclusion that normalizing the token amounts was not all that advantageous in this scenario because we ended up taking equal (more or less) width layers. A histogram of the layer distribution is shown below:



Correlation between Layer and Unique Buyers

Next, we wanted to see how our layers correlate with the number of unique buyers. Since the transaction amount increases as the layer increases, intuitively we thing that the ratio of unique buyers to total buyers will tend to drop as the layer number increases. The idea is that there are more people willing to trade at lower transaction amounts than at higher transaction amounts. (i.e. there are only a few heavy hitters)

```
cor.unique.buyers <- BAT.data %>%
  group_by(layer) %>%
  summarise(num_dist_buyers = n_distinct(toID),
            num_buyers = n(),
            pct_dist_buyers = num_dist_buyers / num_buyers)
```



Our intuition was correct. In fact, the correlation between Percentage of Unique Buyers and Layer Number is over 0.8 (in absolute value).

```
cor(cor.unique.buyers$pct_dist_buyers, cor.unique.buyers$layer)
```

```
## [1] -0.8361999
```

Correlation Between Number of Transactions in a Given Layer and Closing Price

Now we wish to find the correlation between the number of transactions in a given layer and the Closing price of BAT. We will see what gives a higher correlation: the number of transactions, or the number of unique buyers.

```
cor.npurch.by.layer <- BAT.data %>%
  group_by(date, layer) %>%
  summarise(n = n(),
            ndist = n_distinct(toID),
            High = mean(High),
            Open = mean(Open),
            Close = mean(Close),
            Low = mean(Low)) %>%
  as.data.frame()

c_mean_n <- c()
c_mean_dist_n <- c()

for (i in unique(cor.npurch.by.layer$layer)) {
  temp <- cor.npurch.by.layer %>% filter(layer == i)

  cor.i.n <- cor(temp$n, temp$Close)
  cor.i.nd <- cor(temp$ndist, temp$Close)

  c_mean_n <- c(cor.i.n, c_mean_n)
```

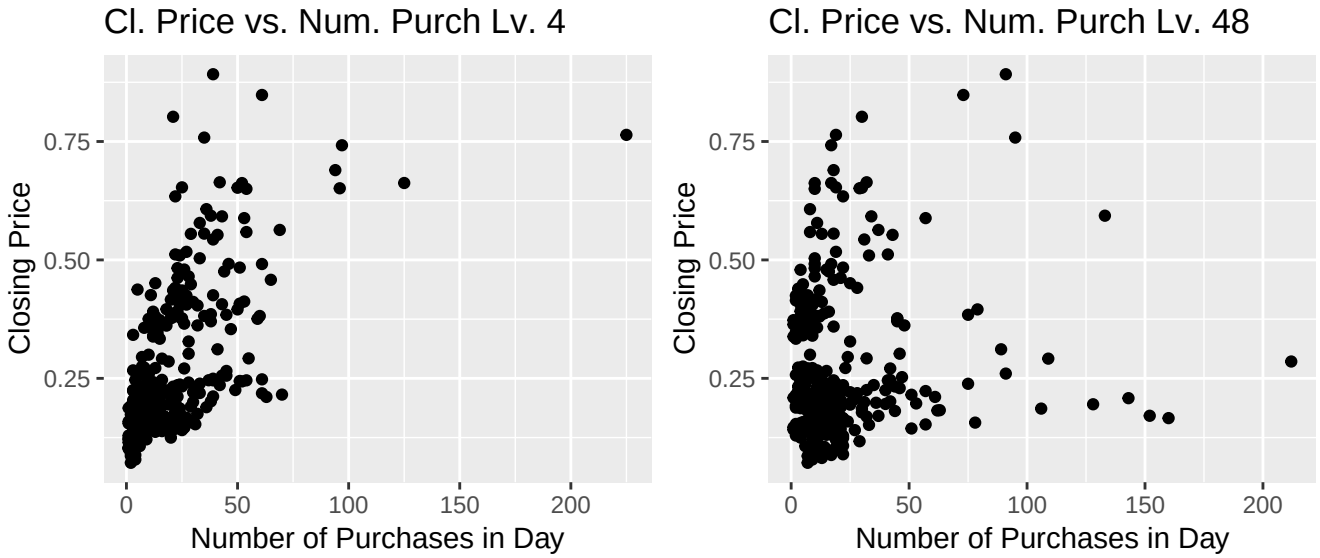
```
  c_mean_dist_n <- c(cor.i.nd, c_mean_dist_n)
}

mean(c_mean_n)
```

```
## [1] 0.3440177
```

```
mean(c_mean_dist_n)
```

```
## [1] 0.3257981
```

We find that the number of transactions gives a higher average correlation with the Closing price than the number of unique buyers. Some of the layers give a rather high corelation between the number of transactions on a given day and the closing price. Others however are not as strongly correlated, but all of the layers show a positive correlation of some sort except for layer 1. The following is a plot of two different layers, one of which has yields a higher correlation between the number of transactions and the closing price and one of which yeilds a lower correlation.



From the above plots we see that the correlation is evident in both plots, but the plot on the right has a lower correlation. Below is the correlation number for the plots above.

```
layer4 <- cor.npurch.by.layer %>% filter(layer == 4)
layer48 <- cor.npurch.by.layer %>% filter(layer == 4)

cor.layer4 <- cor(layer4$n, layer4$Close)
cor.layer48 <- cor(layer48$ndist, layer48$Close)

cat('The correlation in layer 4 between the number of transactions and Closing price is', cor.layer4)
```

```
## The correlation in layer 4 between the number of transactions and Closing price is 0.6037849
```

```
cat('The correlation in layer 48 between the number of transactions and Closing price is', cor.layer48)
```

```
## The correlation in layer 48 between the number of transactions and Closing price is 0.6244748
```

Correlation Between Layer and Closing Price

The correlation between the layer number and the closing price is virtually non-existent becasue the layer is just a proxy for the size of the transactions. So while the inclusion of layers helps with correlating the number of transactions of a given size, or the correlation between layer and unique buyers, it is hard to create layers in which closing price is directly correlated to layer number. One reason for this is that a relatively high closing price in one month may be a relatively low closing price just 6 months layer. This would lead to buyers being relatively unexcited about BAT only a few months after they were jumping all over it, even at the same price. Even though all of this is true, the correlation between layer and price is shown below.

```
cor.layer.close <- cor(BAT.data$layer, BAT.data$Close)
cat('The correlation between layer number and closing price is', cor.layer.close)
```

```
## The correlation between layer number and closing price is -0.02121437
```

Conclusion - Part 2

Creating layers allowed for a correlation to be made both with the number of unique buyers and the number of transactions in a given layer and the days closing price. On this front, the findings were quite interesting and shows the powers of segmenting data. In the future, it might be helpful to fit regression and other machine learning models to the data, engineer more features, and apply principle component analysis to see what influences the number and size of transactions that happen in a given day. Overall, this was a satisfying project in order to learn more about block chain networks, the Ethereum network in particular, and statistical models.

Project 2

For the final part of this project we are going to see if we can fit a linear regression model to the price return on day t from information gathered about the token on day $t-1$.

The first thing that we must do is create the price return, which is given as:

$$\frac{P_t - P_{t-1}}{P_{t-1}}$$

We found that the easiest way to incorporate this was to put the price return for day t in the same line of the data frame with line $t-1$. The following code does just that.

```
## get price return
bat.prices.return <- batprices %>%
  mutate(Pt = lag(Open, 1),
         Pt_1 = Open,
         Price.Return = (Pt - Pt_1) / Pt_1)
```

The next thing that we want to do is engineer features in order to create our linear regression model. The following code adds the number of transactions, the number of unique buyers on a given day, and the mean number of tokens purchased per transaction. Then we will merge these engineered features and other features that we already had to the data frame that contains the price returns.

```
## add num transactions, num unique buyers, and mean token amount
bat.eng.features <- bat.time %>%
  group_by(date) %>%
  summarise(nTransactions = n(),
            nUniqueBuyers = n_distinct(toID),
            mTokenAmt = mean(tokenAmount)) %>%
  ungroup()

## merge the data frames
bat.merged <- merge(bat.prices.return, bat.eng.features)
```

Next, we will clean up the data and select the columns out of our merged data frame that we wish to create our linear regression model off of. Then we will find the correlation between these columns and the Price Return.

```
bat.merged$MC <- as.numeric(gsub(',', '', bat.merged$Market.Cap))
```

```
## Warning: NAs introduced by coercion
```

```
selected <- c('Price.Return', 'Open', 'High', 'Low', 'Close', 'Volume', 'MC', 'nTransactions', 'nUniqueBuy
bat.merged.2 <- bat.merged[complete.cases(bat.merged),]
print(cor(bat.merged.2[, selected])[, 'Price.Return'])
```

```
##  Price.Return          Open          High           Low         Close
##   1.000000000  -0.082296437   0.043866274   0.007700074   0.098686974
##        Volume            MC nTransactions nUniqueBuyers     mTokenAmt
##   0.220243703  -0.082296455   0.231257753   0.232251541   0.017706035
```

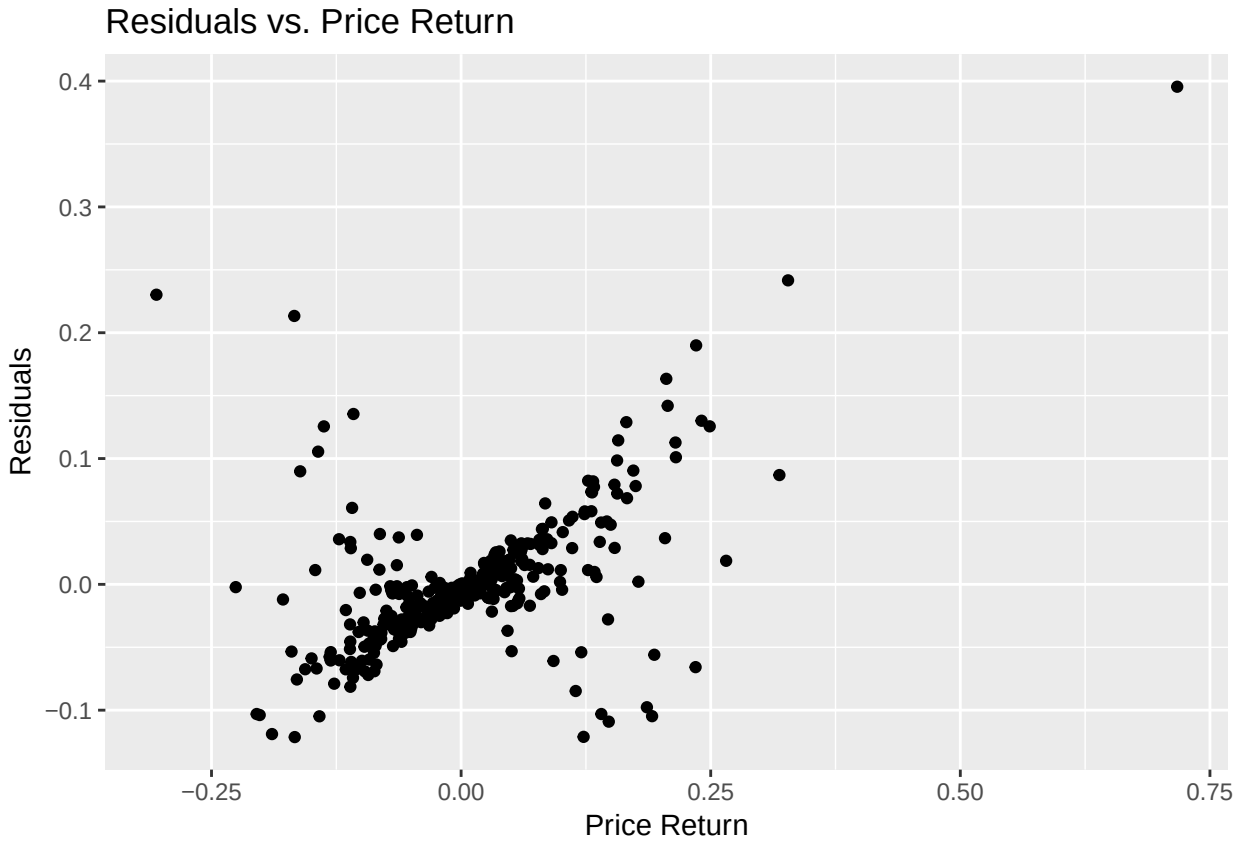
Using the variables with a higher correlation to the Price Return, we fit a linear regression model.

```
fit <- lm(Price.Return ~ Close + Volume + nTransactions + MC + nUniqueBuyers, data = bat.merged)
print(summary(fit))
```

```
##
## Call:
## lm(formula = Price.Return ~ Close + Volume + nTransactions +
##     MC + nUniqueBuyers, data = bat.merged)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.12141 -0.02708 -0.00590  0.01652  0.39554
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)   -2.952e-03  8.436e-03  -0.350    0.727
## Close          2.645e+00  1.075e-01  24.602   <2e-16 ***
## Volume        -2.149e-10  4.737e-10  -0.454    0.650
## nTransactions -6.490e-06  7.988e-06  -0.812    0.417
## MC            -2.637e-09  9.725e-11 -27.118   <2e-16 ***
## nUniqueBuyers  2.509e-05  1.596e-05   1.572    0.117
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.05522 on 333 degrees of freedom
##   (1 observation deleted due to missingness)
## Multiple R-squared:  0.7138, Adjusted R-squared:  0.7095
## F-statistic: 166.1 on 5 and 333 DF,  p-value: < 2.2e-16
```

```
bat.merged.2$fitted <- predict(fit)
bat.merged.2$resid <- fit$residuals
```

We see that the R^2 value is 0.7138, which is decent for a linear regressor with so few variables to learn off of. Finally, we will look at the resituals. The standard error of the residuals is 0.05522 and has an average mean error of 0.0352. The plot of the residuals vs price return is shown below:



We see that the model slightly underfits that data because as the price return increases, on average the residuals gain in magnitude. Finally we will look at a plot of the actual price return vs the fitted values.

This leads us to the same conclusion as the previous graph. The model fits quite well when price return is close to zero, but the model doesnt quite fit as well when the magnitude of the price return increases.

Conclusion - Project 2

We were able to create a linear regression model off of block chain features as well as engineer our own features. We got nice results and with more time and features, a better model could possible be made. Learning about block chain was a very interesting process. With more advanced machine learning methods one could possible make a good profit in the industry.