CS 6313.002 Statistical Methods of Data Science - Final Project

Shweta Rakesh Sahalot (sxs180145), Siddhi Chechani (sxc170042)

December 3, 2018

## Abstract

The aim of the project is to implement different statistical methods on the TENXPAY Token transaction data.The results of the methods are analyzed to derive inferences about the token data.In 4.1, we found how many times a user buys or sells the TENXPAY token, then fit a distribution and estimate the best distribution.In 4.2, we find the correlation between the TENXPAY Token price data and layer features of the TENXPAY Token.In 4.3, we find the most active buyer and seller of our TENXPAY token and then track them in all other tokens, plot how many unique tokens have they invested in the provided time frame then fit and estimate distributions as part 1. In 4.4, we created a multiple linear regression model to explain price return on day t.

**source code**

## Introduction

### 1.1 Key Concepts:

Let us first understand a few concepts, before we get into the detailing of the project

#### 1.1.1 Blockchain:

It is a public ledger formed of multiple blocks.Each block contains crytographic hash of the previous block, transaction data, timestamp and other metadata.The blockchain is a chain of these blocks linked together cryptographically and stored on a distributed peer-to-peer network.

#### 1.1.2 Ethereum

Ethereum is adecentralized platform that runs smart contracts: applications that run exactly as programmed without any possibility of downtime, censorship, fraud or third-party interference.

These apps run on a custom builtblockchain, an enormously powerful shared global infrastructure that can move value around and represent the ownership of property.

Smart Contract is a set of instructions of the format if-this-then-that, it is formed when someone needs to performing particular task involving one or more than one entities of the blockchain. The contract is a code which executes itself on occurence of a triggering event such as expiration date.The smart contracts can be written with different languages such as solidity

The EVM is a runtime environment for smart contracts in Ethereum. Every Ethereum node in the network runs an EVM implementation and executes the same instructions.

Ether is the digital currency(cryptocurrency) of Ethereum. Every individual transaction or step in a contract requires some computation. To perform any computation user has to pay a cost calulated in terms of ‘Gas’ and paid in terms of ‘Ether’. The Gas consist of two parts:

Gas limit: It is the amount of units of gas Gas price: It is the amount paid per unit of gas

#### 1.1.3 Ethereum ERC-20 Tokens:

If a user needs some service provided by the DAPPS, then he has to pay for that service in terms of ‘token’ associated with the DAPPS. These Ethereum tokens can be bought using Ether or other cryptocurrencies and can serve the following two purposes:

1. Usage Token: These tokens are used to pay for the services of the Dapp
2. Work Token: These tokens identify you as a shareholder in the DAPP

ERC-20 is a technical standard used for smart contracts on the Ethereum Blockchain for implementing tokens. It is a common list of rules for Ethereum token regarding interactions between tokens,transferring tokens between addresses and how data within th e token is accessed

### 1.2.1 Project Primary Token: Tenxpay

When we as the co-founders of TenX got together to start this company, it was our vision to have assets on the blockchain be not only available to industry insiders, but rather something that can be used by any individual user in the “real world”.

Additionally, with the emergence of more and more different tokens, a growing number of users and businesses truly struggle to leverage on the existing infrastructure to make this interconnectedness of physical and virtual platforms become a reality.

At TenX, we strive to offer the user access to as large as possible a range of blockchain assets at a maximum degree of convenience, while adhering to the highest security standards in the ecosystem

To the end-user, we offer the TenX Card, a debit (and, in time to come, credit) card, with an accompanying TenX Wallet, a mobile wallet that can be funded not only with Bitcoin (BTC),Ether (ETH), and Dash (DASH) as currently possible, but also with virtually any blockchain asset in time to come. TenX payment facilities which include the physical and virtual debit card can be used in almost 200 countries at over 36 million points of acceptance today.

This is possible as we have card issuance partnerships with major credit card companies.

Moreover, users and businesses can exchange their blockchain assets seamlessly from one user to another in a decentralized manner, removing any risk that is usually associated with current centralized solutions.

#### 1.2.1.1 Product Advantages:

1. Multi-asset (any blockchain asset compatible with and accepted by the TenX Wallet)
2. Assets stay in cryptocurrency
3. Best available foreign exchange and transaction fees (with no other charges)
4. Decentralized and trustless storage

### 1.2.2 Tenxpay Token:

TenX connects your blockchain assets for everyday use. TenX’s debit card and banking licence will allow us to be a hub for the blockchain ecosystem to connect for real-world use cases.

Details: 12/03/2018

1 PAY = $0.67 USD Market Cap = $29,302,619.00 USD  
Circulating Supply = 109,347,861.00 PAY  
Total Supply = ($54,993,597.93) 205,218,255.948577763364408207 PAY Subunits =

## 2. Data Description

The Data used for the project is divided in two parts:

1. Token network edge files:

There are 40 Token network edge files.Token edge files have 4 columns: from\_node, to\_node, unix-time, total-amount.For each row it implies that from\_node sold total-amount of the token to to\_nodeid at time unix-time.

1. from\_node : Id which sells the token in the transaction
2. to\_node : Id which buys the token in the transaction
3. unixtime : Unix time of the transaction
4. totalamount : Total amount of the tenxpay token involved in the transaction For Part 1,2 and 4 of the project we will only use the tenxpay token network edge file. For part 3 we will use the token network edge files of all 40 tokens.
5. Token price files: Price dataset for ten token it contains 334 rows and 7 columns as follows:
6. Date
7. Open : Opening price of the token on that day
8. High : Max price of the token on that day
9. Low : Min price of the token on that day
10. Close : Closing price of the token on that day
11. Volume : Volume of the token on that day
12. Market Cap: Market Cap of that token on that day We use price data in part 2 and 4.

## 3. Preprocessing

There could be some records in the transactions where total amount is very large. Some of this records could be due to some bug or glitch(integer overflow problem). These values can be separated from data using a threshold value.

Calculating this value for the tenxpay token:The value of transaction amount can’t be greater than the max value where, max value = total supply of tokens \* subunits.subsituting the values from above.

##Load Token   
data <- read.delim("networktenxpayTX.txt",header=FALSE,sep=" ")  
tokenFrame<-as.data.frame(data)  
colnames(tokenFrame)<- c(" fromNodeID", "toNodeID", "Time", "Amount")  
  
## fromNodeID toNodeID Time Amount  
## 560 1452 1524611450 1.728672e+20  
## 2011173 2011174 1524611865 4.556238e+20  
## 75989 1822217 1524612292 5.795000e+20  
## 40002 6382858 1524612655 4.481100e+20  
## 17 2029263 1524612851 4.998000e+21  
## 222770 4848204 1524612957 3.283584e+20  
  
#Claculating Outlier Threshold  
x <- 205218255.948577763364408207  
y <- 10^18  
threshold <- x\*y  
  
## Finding Outliers  
Outlierdata <-tokenFrame[which(tokenFrame$Amount> threshold),]  
  
## Total number of Outliers  
message("Total number of outliers: ", length(Outlierdata$tokenAmount))

## Total number of outliers: 0

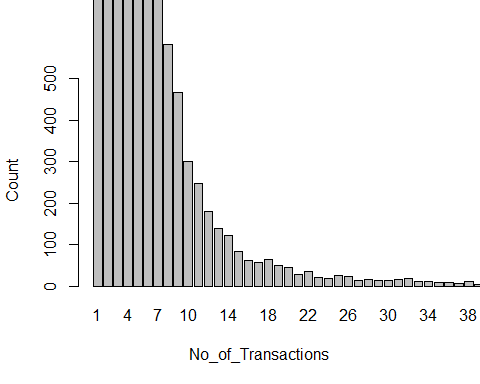
## Data Without Outliers  
data1 <-tokenFrame[which(tokenFrame$Amount< threshold),]  
View(data1)  
  
## Distribution   
## fromNodeID toNodeID Time Amount  
## 560 1452 1524611450 1.728672e+20  
## 2011173 2011174 1524611865 4.556238e+20  
## 75989 1822217 1524612292 5.795000e+20  
## 40002 6382858 1524612655 4.481100e+20  
## 17 2029263 1524612851 4.998000e+21  
## 222770 4848204 1524612957 3.283584e+20

## 4.Token Data Analysis

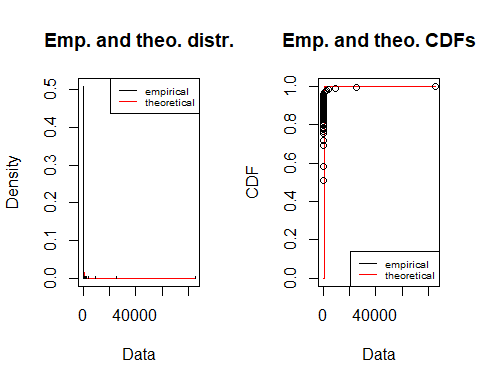
### 4.1 Finad and Fit Distribution

**The package we use to fit distribution:**  fitdistrplus, provide the function fitdist() we can use to fit distribution of our data. **The function we use to fit distribution:**  fitdist(). Fit of univariate distributions to different type of data with different estimate method we can choose: maximum likelihood estimation (mle), moment matching estimation (mme), quantile matching estimation (qme), maximizing goodness-of-fit estimation (mge), the default and we mostly used one is MLE. Output of this function is S3 object, we can use several methods like plot(), print(), summary() to visualize it or get more detailed information.We use plot in this project.

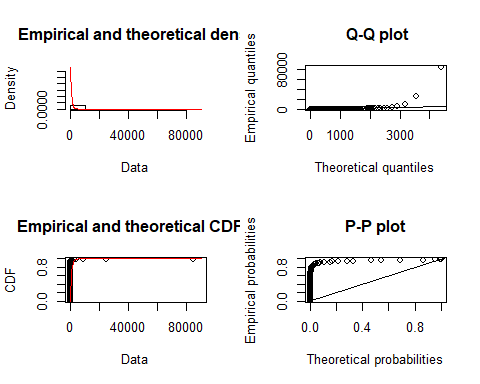
## Table 1: BUyer Transaction and its frequency   
BuyerData <- (table(data1[2]))  
DataFrequency <- as.data.frame(BuyerData)  
colnames(DataFrequency) <- c("UserID", "frequency")  
View(DataFrequency)  
  
## UserID frequency  
## 5 20  
## 8 244  
## 13 2  
## 15 1  
## 17 14  
## 22 7  
## 26 130  
  
  
## Table 2: Frequency of No\_of\_Transactions  
FreqBuyers= table(DataFrequency$frequency)  
frenumbuy=as.data.frame(FreqBuyers)  
colnames(frenumbuy)<- c("No\_of\_Trasactions", "Count")  
View(frenumbuy)  
  
## No\_of\_Transctions Count  
## 1 84884  
## 2 24985  
## 3 8990  
## 4 4375  
## 5 2376  
## 6 1482  
## 7 87  
  
  
## Bar Plot  
barplot(frenumbuy$Count, names.arg = frenumbuy$No\_of\_Trasactions, ylab=" Count", xlab="No\_of\_Transactions", xlim=c(0,40),ylim=c(0,500))



##Fit Distribution  
library(MASS)  
library(lsei)  
library(npsurv)  
library(survival)  
library(fitdistrplus)  
fit<-fitdist(frenumbuy$Count, "pois", method='mle')  
plot(fit)

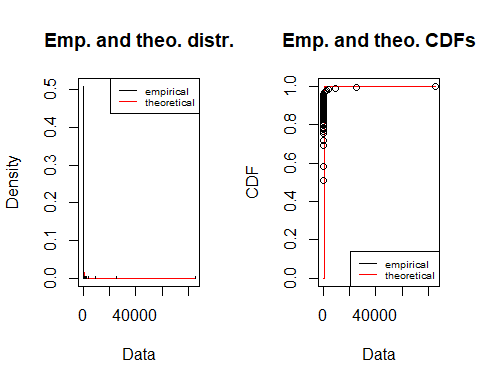


# fit distribution  
# Poisson Distribution  
library(fitdistrplus)  
fit <- fitdist(frenumbuy$Count, "pois", method="mle")  
# Exponential Distribution  
fit1 <- fitdist(frenumbuy$Count, "exp",method = "mme")  
# Normal Distribution  
fit2 <- fitdist(frenumbuy$Count, "norm")  
plot(fit1)



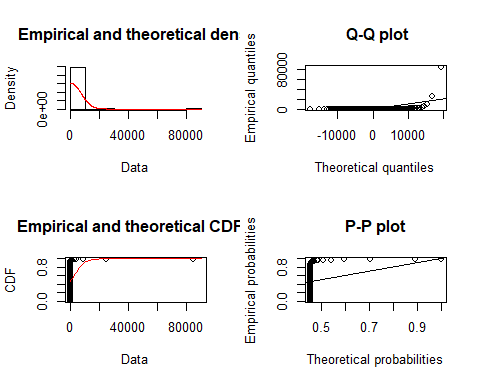
Fitting with Weibull Distribution-Buys

plot(fit)



Fitting with Weibull Distribution-Buys

plot(fit2)

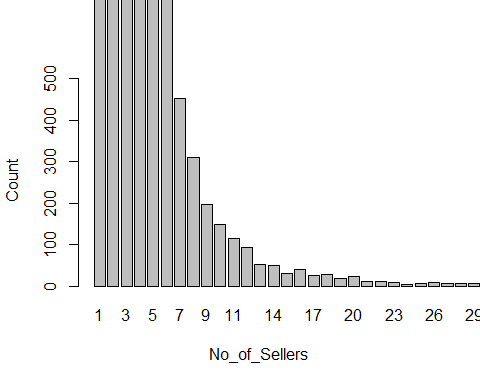
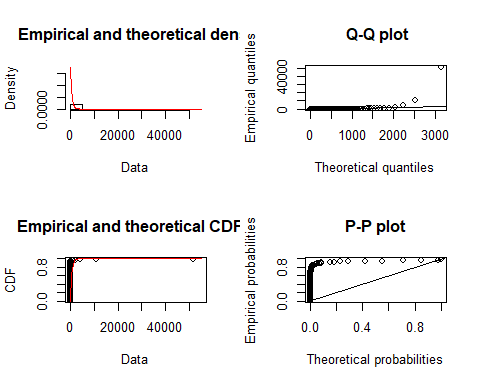


Fitting with Weibull Distribution-Buys

## Fitting of the distribution ' pois ' by maximum likelihood   
## Parameters:  
  
## p1 = 0.033  
## p2 = 0.364  
## p3 = 0.603

The sells part is similiar to buys part, so we didn’t show the code in this report, the follow is the plot of sells:

# Fit Distribution -- Sells  
##Table 3: Seller Transaction and its frequency  
SellerData <- (table(data1[1]))  
Data2frequency <- as.data.frame(SellerData)  
colnames(Data2frequency) <- c("SellerID", "frequency")  
View(Data2frequency)  
  
## SellerID frequency  
## 5 1  
## 8 413  
## 13 83  
## 15 1  
## 17 92081  
## 22 5  
## 26 290  
  
## Table 4: Frequency of No\_of\_Transactions  
FreqSellers= table(Data2frequency$frequency)  
frenumsell = as.data.frame(FreqSellers)  
colnames(frenumsell) <- c("No\_of\_Sells", "Count")  
View(frenumsell)  
  
## No\_of\_Sells Count  
## 1 51681  
## 2 11039  
## 3 4213  
## 4 2091  
## 5 1084  
## 6 701  
## 7 453  
  
##Bar Plot  
barplot(frenumsell$Count, names.arg = frenumsell$No\_of\_Sells, ylab=" Count", xlab="No\_of\_Sellers", xlim=c(0,30),ylim=c(0,500))

  ### 4.2 Calulate Correlations of Token Data and Price Data

Loading the token data:

tokenData <- read.delim("networktenxpayTX.txt", header = FALSE, sep = " ")  
tokenFrame<- as.data.frame(tokenData)  
colnames(tokenFrame)<-c('fromNode','toNode','unixTime','totalAmount')  
head(tokenFrame)

## fromNode toNode unixTime totalAmount  
## 1 560 1452 1524611450 1.728672e+20  
## 2 2011173 2011174 1524611865 4.556238e+20  
## 3 75989 1822217 1524612292 5.795000e+20  
## 4 40002 6382858 1524612655 4.481100e+20  
## 5 17 2029263 1524612851 4.998000e+21  
## 6 222770 4848204 1524612957 3.283584e+20

Removing Outliers from the token data:

TotalSupply <-205218255.9  
Decimals <- 10^18  
OutlierValue <- TotalSupply\*Decimals  
WithoutOutlierdata <- tokenFrame[ which(tokenFrame$totalAmount < OutlierValue),]  
head(WithoutOutlierdata)

## fromNode toNode unixTime totalAmount  
## 1 560 1452 1524611450 1.728672e+20  
## 2 2011173 2011174 1524611865 4.556238e+20  
## 3 75989 1822217 1524612292 5.795000e+20  
## 4 40002 6382858 1524612655 4.481100e+20  
## 5 17 2029263 1524612851 4.998000e+21  
## 6 222770 4848204 1524612957 3.283584e+20

Summary of the token data without outlier:

summary(WithoutOutlierdata$totalAmount)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 1.000e+00 5.445e+19 2.656e+20 1.844e+21 1.005e+21 7.108e+24

Max amount of the token data without outlier:

max(WithoutOutlierdata$totalAmount)

## [1] 7.108079e+24

First step: Ensure that the tenxpay token data is within the time frame of the Price Data

Following is the date interval of the price data:

Start Date: 06/27/2017 End Date: 07/14/2018

Lets check the date range for the token data

First convert the Unix time of the token data to Date format

newwithoutoutlierdata <- WithoutOutlierdata  
newwithoutoutlierdata$unixTime <- as.Date(as.POSIXct(WithoutOutlierdata$unixTime,origin="1970-01-01",tz="GMT"))  
class(newwithoutoutlierdata$unixTime)

## [1] "Date"

loading the price data :

priceData <- read.delim("tenxpay", header = TRUE, sep = "\t")  
head(priceData)

## Date Open High Low Close Volume Market.Cap  
## 1 07/15/2018 0.746418 0.747655 0.722171 0.730223 4,856,290 81,509,100  
## 2 07/14/2018 0.698361 0.801071 0.696400 0.746002 7,753,930 76,261,300  
## 3 07/13/2018 0.600009 0.844887 0.600009 0.701817 12,405,200 65,521,200  
## 4 07/12/2018 0.598843 0.616721 0.568886 0.588453 4,906,510 65,393,900  
## 5 07/11/2018 0.592847 0.656002 0.567525 0.599807 5,622,960 64,739,100  
## 6 07/10/2018 0.629064 0.665708 0.591947 0.592098 4,754,660 68,694,000

priceData$Date <- as.Date(priceData$Date, "%m/%d/%Y")  
nrow(priceData)

## [1] 384

add the price return to the dataframe:

priceData$Open[1]

## [1] 0.746418

(priceData$Open[2]-priceData$Open[1])/priceData$Open[1]

## [1] -0.0643835

create the price return column:

return <- numeric(nrow(priceData))  
return[379] <- priceData$Open[379]  
for (i in 378:1) {  
 return[i] <- (priceData$Open[i]-priceData$Open[i+1])/priceData$Open[i+1]  
}  
return[2]

## [1] 0.1639175

return

## [1] 0.068813980 0.163917541 0.001947088 0.010113908 -0.057572838  
## [6] -0.022667458 -0.017043008 0.037745029 -0.078915831 -0.023267054  
## [11] -0.038467971 -0.066616848 0.061265590 -0.043285639 0.041549616  
## [16] 0.405611444 0.045854751 0.033958197 -0.065093445 0.001849437  
## [21] -0.004377022 -0.012588872 -0.121068156 -0.022758107 -0.028354658  
## [26] 0.008215009 0.018451945 0.010484659 -0.009340419 -0.033928047  
## [31] 0.057882651 -0.084831760 -0.138437066 0.049033261 -0.134656730  
## [36] -0.053444088 -0.008566712 0.001699272 -0.014321205 0.016852473  
## [41] -0.068688407 0.021114237 0.029949275 -0.019852877 0.028216621  
## [46] -0.014066779 0.050288032 -0.086291661 -0.026592122 0.005148938  
## [51] -0.011660375 0.023407089 -0.105367593 -0.092436975 -0.055555556  
## [56] 0.000000000 -0.038167939 0.023437500 -0.104895105 0.091603053  
## [61] 0.007692308 0.040000000 0.077586207 0.017543860 -0.102362205  
## [66] -0.059259259 -0.014598540 -0.028368794 -0.007042254 -0.089743590  
## [71] -0.042944785 0.012422360 -0.018292683 0.018633540 0.025477707  
## [76] 0.046666667 0.006711409 0.034722222 -0.076923077 0.098591549  
## [81] -0.154761905 0.043478261 0.032051282 0.006451613 -0.006410256  
## [86] -0.018867925 0.204545455 0.100000000 0.000000000 -0.062500000  
## [91] 0.032258065 0.050847458 0.035087719 0.106796117 0.111651947  
## [96] 0.028711353 -0.009399110 0.025451465 0.034162295 -0.031317471  
## [101] -0.014015048 -0.102322000 0.114486768 0.051646197 -0.010843279  
## [106] -0.019205141 -0.037248845 -0.130020952 -0.009433962 -0.138211382  
## [111] -0.075187970 0.064000000 0.024590164 0.051724138 -0.025210084  
## [116] -0.008333333 0.052631579 0.000000000 0.027027027 -0.051282051  
## [121] 0.008620690 -0.008547009 -0.152173913 0.022222222 -0.021739130  
## [126] 0.069767442 -0.051470588 0.000000000 -0.087248322 -0.128654971  
## [131] -0.085561497 -0.055555556 0.112359551 0.119496855 -0.042168675  
## [136] 0.121621622 -0.086419753 0.006211180 0.066225166 -0.013071895  
## [141] -0.055555556 0.038461538 -0.065868263 -0.097297297 -0.106280193  
## [146] 0.050761421 -0.079439252 0.091836735 -0.010101010 0.020618557  
## [151] 0.071823204 -0.076530612 0.113636364 -0.038251366 0.051724138  
## [156] 0.108280255 0.082758621 -0.088050314 0.169117647 -0.200000000  
## [161] -0.154228856 0.057894737 -0.020618557 -0.163793103 -0.008547009  
## [166] -0.075098814 -0.083333333 0.095238095 0.054393305 -0.036290323  
## [171] -0.035019455 0.070833333 0.052631579 -0.065573770 -0.155709343  
## [176] 0.107279693 0.035714286 -0.023255814 0.053061224 -0.283625731  
## [181] -0.102362205 0.029729730 0.072463768 0.109324759 -0.131284916  
## [186] -0.070129870 -0.012820513 -0.012658228 0.061827957 -0.055837563  
## [191] -0.150862069 0.013100437 0.085308057 0.052369077 -0.009876543  
## [196] 0.125000000 -0.138755981 0.012106538 0.004866180 -0.065909091  
## [201] -0.028697572 -0.052301255 0.231958763 0.315254237 -0.189560440  
## [206] 0.019607843 0.034782609 -0.141791045 0.066312997 0.295532646  
## [211] 0.187755102 -0.042968750 0.094017094 0.054054054 0.042253521  
## [216] 0.103626943 -0.106481481 -0.076923077 0.193877551 -0.121076233  
## [221] -0.155303030 -0.022222222 0.129707113 0.017021277 -0.020833333  
## [226] 0.121495327 0.033816425 -0.224719101 0.126582278 0.017167382  
## [231] 0.159203980 0.063492063 0.005319149 -0.005291005 0.016129032  
## [236] -0.060606061 0.185628743 -0.023391813 0.042682927 0.000000000  
## [241] -0.012048193 -0.017751479 0.030487805 0.064935065 -0.083333333  
## [246] -0.028901734 -0.130653266 0.047368421 0.124260355 0.056250000  
## [251] -0.018404908 -0.006097561 -0.046511628 0.116883117 0.019867550  
## [256] -0.151685393 0.005649718 0.035087719 -0.011560694 -0.044198895  
## [261] 0.005555556 -0.005524862 -0.037234043 0.105882353 -0.034090909  
## [266] -0.016759777 -0.005555556 -0.072164948 -0.020202020 0.064516129  
## [271] -0.036269430 0.000000000 0.010471204 -0.025510204 -0.053140097  
## [276] -0.054794521 0.078817734 0.085561497 0.000000000 -0.117924528  
## [281] -0.027522936 0.023474178 -0.027397260 -0.075949367 -0.020661157  
## [286] -0.113553114 0.011111111 0.080000000 -0.090909091 -0.107142857  
## [291] 0.232000000 0.077586207 0.120772947 -0.009569378 0.039800995  
## [296] 0.248447205 -0.139037433 -0.055555556 -0.095890411 0.063106796  
## [301] -0.046296296 -0.027027027 0.283236994 -0.260683761 -0.067729084  
## [306] -0.019531250 -0.015384615 -0.029850746 -0.028985507 -0.132075472  
## [311] 0.074324324 0.010238908 0.061594203 -0.218130312 -0.024861878  
## [316] -0.095000000 -0.065420561 -0.013824885 0.030878860 0.113756614  
## [321] 0.044198895 -0.018970190 0.048295455 -0.030303030 0.000000000  
## [326] 0.022535211 -0.024725275 -0.099009901 -0.031175060 0.142465753  
## [331] -0.147196262 -0.089361702 0.132530120 -0.091903720 -0.103921569  
## [336] 0.073684211 0.053215078 0.571428571 0.341121495 -0.027272727  
## [341] 0.067961165 0.113513514 -0.135514019 0.712000000 0.237623762  
## [346] 0.091962722 -0.019394975 0.064223434 -0.005896347 -0.040324768  
## [351] 0.035347716 -0.077441098 -0.002089932 0.146049858 -0.205174766  
## [356] 0.184364616 0.101814124 0.222846107 -0.047915874 0.124502635  
## [361] -0.116032237 -0.000933483 0.247376133 -0.050363882 -0.113521878  
## [366] -0.061262899 -0.100750350 0.207522308 -0.053344192 -0.224498434  
## [371] -0.186960360 0.000000000 -0.984807008 0.151820905 0.185164425  
## [376] 0.217747440 0.562388909 -0.131790123 32.400000000 0.000000000  
## [381] 0.000000000 0.000000000 0.000000000 0.000000000

adding the price return column to the pricedata dataframe

priceData$return <- return  
priceData$return[378]

## [1] -0.1317901

head(newwithoutoutlierdata$Date)

## NULL

colnames(newwithoutoutlierdata)<-c('fromNode','toNode','Date','totalAmount')  
newwithoutoutlierdata$Date[4]

## [1] "2018-04-24"

class(priceData$Date)

## [1] "Date"

merge both the dataframes according to dates

newdataset <- merge(newwithoutoutlierdata,priceData,by = "Date")  
head(newdataset$Date)

## [1] "2017-06-27" "2017-06-27" "2017-06-27" "2017-06-27" "2017-06-27"  
## [6] "2017-06-27"

Creating a feature of no of transactions from the data for doing simple linear regression.

frequencytrans <- table(newdataset$Date)  
freqtrans<- as.data.frame(frequencytrans)  
summary(freqtrans)

## Var1 Freq   
## 2017-06-27: 1 Min. : 9.0   
## 2017-06-28: 1 1st Qu.: 357.2   
## 2017-06-29: 1 Median : 790.0   
## 2017-06-30: 1 Mean :1049.8   
## 2017-07-01: 1 3rd Qu.:1272.2   
## 2017-07-02: 1 Max. :7088.0   
## (Other) :308

colnames(freqtrans) <- c("Date","no\_of\_trans")  
freqtrans$Date <- as.Date(freqtrans$Date)

mergeing freq trans with newdataset

newsimpletable <- merge(priceData,freqtrans,by = "Date")  
head(newsimpletable$return)

## [1] 0.0 0.0 0.0 0.0 0.0 32.4

newsimpletable<- newsimpletable[,c(-2:-7)]  
head(newsimpletable$no\_of\_trans)

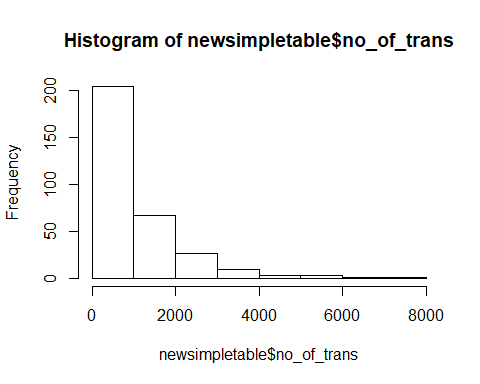
## [1] 25 13 15 10 12 15

Data Preparation:

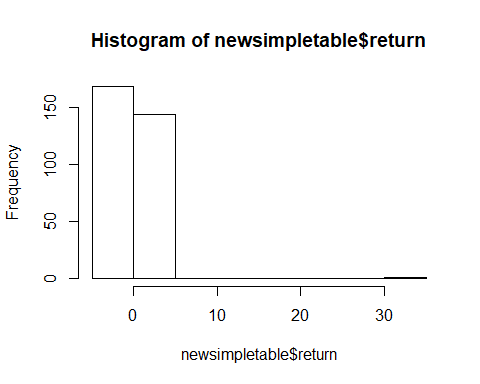
summary(newsimpletable)

## Date return no\_of\_trans   
## Min. :2017-06-27 Min. :-0.98481 Min. : 9.0   
## 1st Qu.:2017-09-13 1st Qu.:-0.05329 1st Qu.: 357.2   
## Median :2017-11-30 Median : 0.00000 Median : 790.0   
## Mean :2017-11-30 Mean : 0.10975 Mean :1049.8   
## 3rd Qu.:2018-02-16 3rd Qu.: 0.06177 3rd Qu.:1272.2   
## Max. :2018-05-06 Max. :32.40000 Max. :7088.0

hist(newsimpletable$no\_of\_trans)

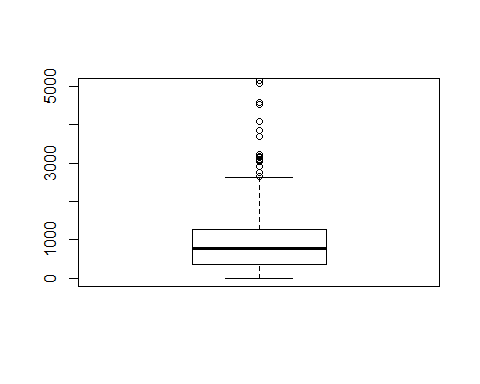


hist(newsimpletable$return)



Analysis:

par(mfrow = c(1,1))  
bx = boxplot(newsimpletable$no\_of\_trans,ylim = c(0,5000))

 checking the distribution of no of transactions:

quantile(newsimpletable$no\_of\_trans, seq(0,1,0.02))

## 0% 2% 4% 6% 8% 10% 12% 14% 16%   
## 9.00 15.00 139.12 168.80 201.00 219.30 225.36 239.82 253.24   
## 18% 20% 22% 24% 26% 28% 30% 32% 34%   
## 268.02 288.40 300.16 318.16 385.04 403.84 429.70 470.92 543.56   
## 36% 38% 40% 42% 44% 46% 48% 50% 52%   
## 586.76 629.00 674.00 702.00 736.16 749.98 764.24 790.00 805.28   
## 54% 56% 58% 60% 62% 64% 66% 68% 70%   
## 823.02 846.68 868.32 904.00 940.66 968.64 1023.16 1080.84 1134.20   
## 72% 74% 76% 78% 80% 82% 84% 86% 88%   
## 1166.40 1250.10 1323.80 1430.48 1549.20 1647.18 1812.72 1938.72 2073.76   
## 90% 92% 94% 96% 98% 100%   
## 2204.00 2461.08 2787.52 3164.68 4416.56 7088.00

boxplot stats:

bx$stats

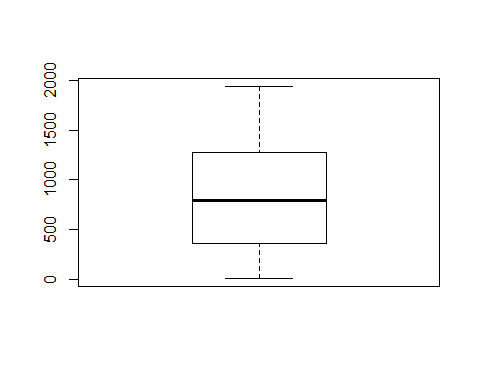
## [,1]  
## [1,] 9  
## [2,] 356  
## [3,] 790  
## [4,] 1275  
## [5,] 2628  
## attr(,"class")  
##   
## "integer"

summary(newsimpletable$no\_of\_trans)

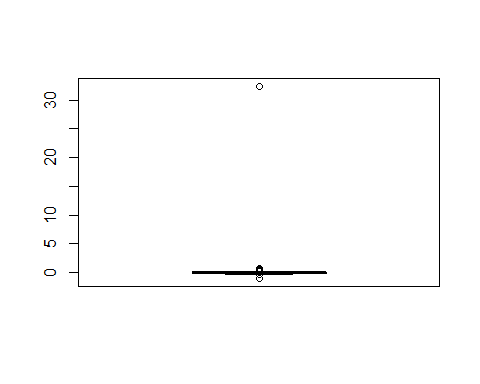
## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 9.0 357.2 790.0 1049.8 1272.2 7088.0

capping at 96%

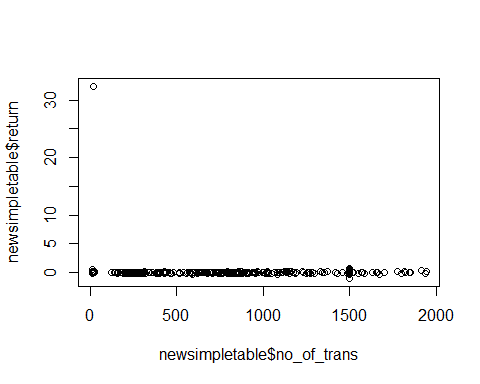
newsimpletable$no\_of\_trans<-ifelse(newsimpletable$no\_of\_trans>2000,1500,newsimpletable$no\_of\_trans)  
boxplot(newsimpletable$no\_of\_trans)

 checking the response variable:

boxplot(newsimpletable$return)

 bivariate analysis:

plot(newsimpletable$no\_of\_trans,newsimpletable$return)



lets check the correlation:

cor(newsimpletable[,c(2,3)])

## return no\_of\_trans  
## return 1.00000000 -0.08815697  
## no\_of\_trans -0.08815697 1.00000000

fitting the linear model:

library(fmsb)  
final\_data <- lm(return~no\_of\_trans,data=newsimpletable)  
VIF(final\_data)

## [1] 1.007833

summary(final\_data)

##   
## Call:  
## lm(formula = return ~ no\_of\_trans, data = newsimpletable)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -0.882 -0.260 -0.128 0.028 32.031   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 0.3735991 0.1978206 1.889 0.0599 .  
## no\_of\_trans -0.0003173 0.0002030 -1.563 0.1190   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 1.828 on 312 degrees of freedom  
## Multiple R-squared: 0.007772, Adjusted R-squared: 0.004591   
## F-statistic: 2.444 on 1 and 312 DF, p-value: 0.119

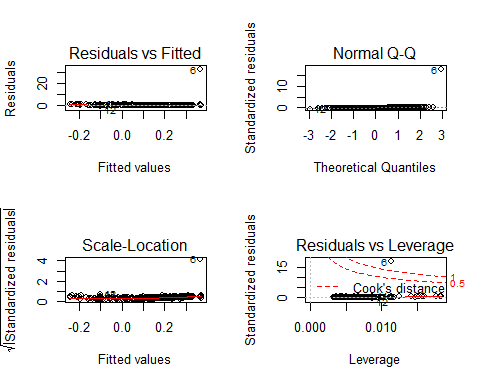
library(lmtest)

## Loading required package: zoo

##   
## Attaching package: 'zoo'

## The following objects are masked from 'package:base':  
##   
## as.Date, as.Date.numeric

par(mfrow = c(2,2))  
plot(final\_data)

 now trying by taking the sqaureroot of the predictor variable:

newsimpletable$no\_of\_trans <- (newsimpletable$no\_of\_trans)^(0.5)  
final\_data <- lm(return~no\_of\_trans,data=newsimpletable)  
VIF(final\_data)

## [1] 1.018227

summary(final\_data)

##   
## Call:  
## lm(formula = return ~ no\_of\_trans, data = newsimpletable)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -0.835 -0.319 -0.101 0.090 31.707   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 0.79052 0.30337 2.606 0.00961 \*\*  
## no\_of\_trans -0.02509 0.01052 -2.385 0.01769 \*   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 1.819 on 312 degrees of freedom  
## Multiple R-squared: 0.0179, Adjusted R-squared: 0.01475   
## F-statistic: 5.687 on 1 and 312 DF, p-value: 0.01769

creating 2 new features no of tokens and no of buys:

so first dividing the total amount of the dataset with 10^8 to find the no of tokens and then grouping it and adding it acording to the date

newdataset$totalAmount <- (newdataset$totalAmount/(10^8))  
tokentable <- aggregate(newdataset$totalAmount, by = list(Category = newdataset$Date), FUN = sum)  
head(tokentable)

## Category x  
## 1 2017-06-27 1.409702e+14  
## 2 2017-06-28 3.119692e+15  
## 3 2017-06-29 2.003250e+14  
## 4 2017-06-30 3.125740e+14  
## 5 2017-07-01 1.575796e+14  
## 6 2017-07-02 4.625641e+14

colnames(tokentable)<- c("Date","total\_token")  
newsimpletable <- merge(newsimpletable,tokentable,by = "Date")  
class(newdataset$Date)

## [1] "Date"

buyerstable <- aggregate(data=newdataset, toNode ~ Date, function(x) length(unique(x)))  
head(buyerstable)

## Date toNode  
## 1 2017-06-27 21  
## 2 2017-06-28 11  
## 3 2017-06-29 10  
## 4 2017-06-30 7  
## 5 2017-07-01 11  
## 6 2017-07-02 10

summary(buyerstable)

## Date toNode   
## Min. :2017-06-27 Min. : 7.0   
## 1st Qu.:2017-09-13 1st Qu.: 234.0   
## Median :2017-11-30 Median : 532.0   
## Mean :2017-11-30 Mean : 658.4   
## 3rd Qu.:2018-02-16 3rd Qu.: 810.0   
## Max. :2018-05-06 Max. :4708.0

colnames(buyerstable) <- c("Date","unique\_buyers")  
newsimpletable <- merge(newsimpletable,buyerstable,by = "Date")

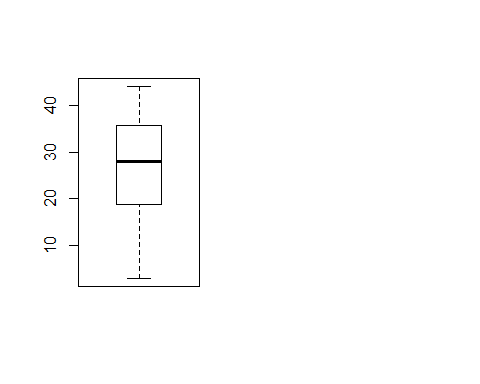
time for data preparation:

summary(newsimpletable)

## Date return no\_of\_trans   
## Min. :2017-06-27 Min. :-0.98481 Min. : 3.00   
## 1st Qu.:2017-09-13 1st Qu.:-0.05329 1st Qu.:18.90   
## Median :2017-11-30 Median : 0.00000 Median :28.11   
## Mean :2017-11-30 Mean : 0.10975 Mean :27.14   
## 3rd Qu.:2018-02-16 3rd Qu.: 0.06177 3rd Qu.:35.67   
## Max. :2018-05-06 Max. :32.40000 Max. :44.07   
## total\_token unique\_buyers   
## Min. :1.410e+14 Min. : 7.0   
## 1st Qu.:4.955e+15 1st Qu.: 234.0   
## Median :7.916e+15 Median : 532.0   
## Mean :1.935e+16 Mean : 658.4   
## 3rd Qu.:1.854e+16 3rd Qu.: 810.0   
## Max. :5.225e+17 Max. :4708.0

then lets do univariate analysis on all independent variables

par(mfrow=c(1,2))  
bx = boxplot(newsimpletable$no\_of\_trans)

 capping the outliers:

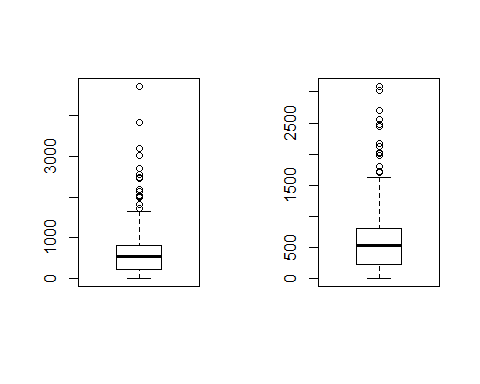
# quantile(newsimpletable$no\_of\_trans, seq(0,1,0.02))  
# newsimpletable$no\_of\_trans<-ifelse(newsimpletable$no\_of\_trans>60,59,newsimpletable$no\_of\_trans)  
# boxplot(newsimpletable$no\_of\_trans)

checking unique buyers:

par(mfrow=c(1,2))  
bx = boxplot(newsimpletable$unique\_buyers)  
quantile(newsimpletable$unique\_buyers, seq(0,1,0.02))

## 0% 2% 4% 6% 8% 10% 12% 14% 16%   
## 7.00 11.26 115.52 135.34 155.00 164.50 174.00 180.28 190.16   
## 18% 20% 22% 24% 26% 28% 30% 32% 34%   
## 199.34 211.60 218.72 230.00 239.00 253.64 275.60 302.96 322.26   
## 36% 38% 40% 42% 44% 46% 48% 50% 52%   
## 353.04 388.62 415.20 444.84 475.88 489.94 517.44 532.00 559.32   
## 54% 56% 58% 60% 62% 64% 66% 68% 70%   
## 579.04 589.12 605.08 624.00 652.78 677.64 686.90 704.00 734.50   
## 72% 74% 76% 78% 80% 82% 84% 86% 88%   
## 768.00 787.62 856.44 886.50 955.60 1030.30 1108.56 1174.08 1214.44   
## 90% 92% 94% 96% 98% 100%   
## 1331.50 1401.96 1551.12 1889.12 2467.54 4708.00

newsimpletable$unique\_buyers<-ifelse(newsimpletable$unique\_buyers>3087,3088,newsimpletable$unique\_buyers)  
boxplot(newsimpletable$unique\_buyers)

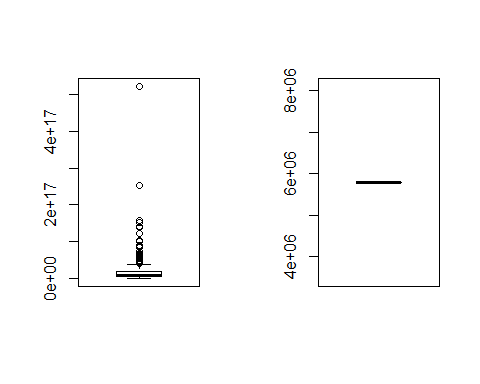


checking for no of tokens:

par(mfrow=c(1,2))  
bx = boxplot(newsimpletable$total\_token)  
quantile(newsimpletable$total\_token, seq(0,1,0.02))

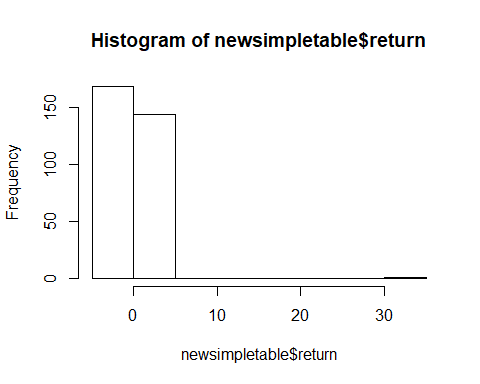
## 0% 2% 4% 6% 8%   
## 1.409702e+14 4.536653e+14 1.471612e+15 2.110669e+15 2.551931e+15   
## 10% 12% 14% 16% 18%   
## 3.116653e+15 3.456227e+15 3.748938e+15 4.016769e+15 4.224678e+15   
## 20% 22% 24% 26% 28%   
## 4.374631e+15 4.646226e+15 4.904279e+15 5.058124e+15 5.232375e+15   
## 30% 32% 34% 36% 38%   
## 5.460118e+15 5.621495e+15 5.875627e+15 6.158514e+15 6.320170e+15   
## 40% 42% 44% 46% 48%   
## 6.533721e+15 6.761005e+15 7.071005e+15 7.222123e+15 7.506538e+15   
## 50% 52% 54% 56% 58%   
## 7.915912e+15 8.456711e+15 8.757019e+15 9.169928e+15 9.785257e+15   
## 60% 62% 64% 66% 68%   
## 1.001398e+16 1.030422e+16 1.077617e+16 1.149179e+16 1.283490e+16   
## 70% 72% 74% 76% 78%   
## 1.360846e+16 1.493365e+16 1.811259e+16 1.944754e+16 2.147736e+16   
## 80% 82% 84% 86% 88%   
## 2.259460e+16 2.812904e+16 3.055752e+16 3.345500e+16 3.943318e+16   
## 90% 92% 94% 96% 98%   
## 4.429327e+16 4.860446e+16 5.921017e+16 7.865907e+16 1.176165e+17   
## 100%   
## 5.225067e+17

newsimpletable$total\_token<-ifelse(newsimpletable$total\_token>5791001,5791001,newsimpletable$total\_token)  
boxplot(newsimpletable$total\_token)

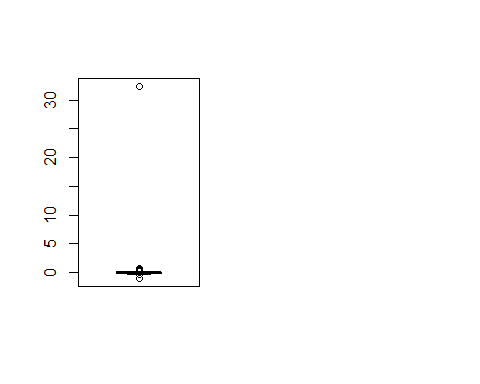


lets check the response variable:

hist(newsimpletable$return)

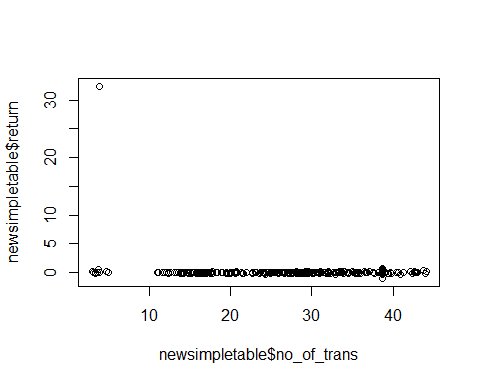


par(mfrow=c(1,2))  
bx = boxplot(newsimpletable$return)

 bivariate analysis:

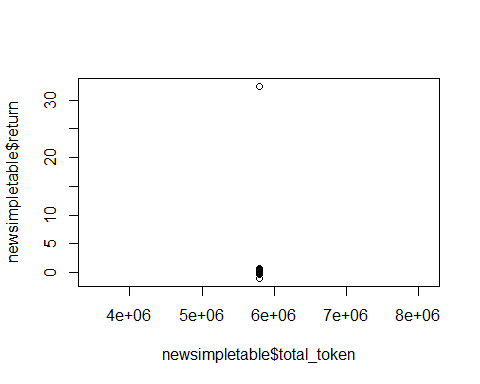
square root number of transactions vs return:

plot(newsimpletable$no\_of\_trans,newsimpletable$return)



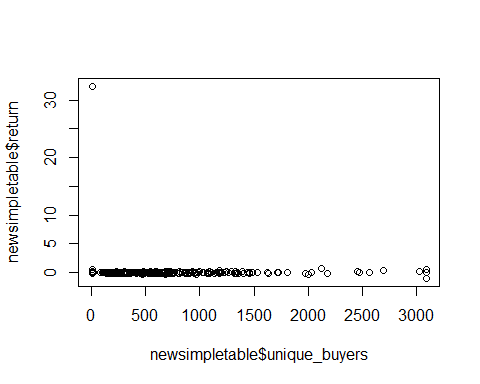
lets check for total token vs return:

plot(newsimpletable$total\_token,newsimpletable$return)



lets check unique buyers vs return:

plot(newsimpletable$unique\_buyers,newsimpletable$return)

 lets check the correlation:

cor(newsimpletable[,c(2:5)])

## Warning in cor(newsimpletable[, c(2:5)]): the standard deviation is zero

## return no\_of\_trans total\_token unique\_buyers  
## return 1.00000000 -0.1337920 NA -0.06212329  
## no\_of\_trans -0.13379200 1.0000000 NA 0.75555525  
## total\_token NA NA 1 NA  
## unique\_buyers -0.06212329 0.7555553 NA 1.00000000

lets extract one more feature

no of investors who have bought more than ten token:

library(dplyr)

##   
## Attaching package: 'dplyr'

## The following object is masked from 'package:MASS':  
##   
## select

## The following objects are masked from 'package:stats':  
##   
## filter, lag

## The following objects are masked from 'package:base':  
##   
## intersect, setdiff, setequal, union

summary(newdataset)

## Date fromNode toNode   
## Min. :2017-06-27 Min. : 5 Min. : 5   
## 1st Qu.:2017-08-15 1st Qu.: 17 1st Qu.: 431538   
## Median :2017-09-17 Median : 297031 Median :2062611   
## Mean :2017-10-16 Mean :1644441 Mean :3179799   
## 3rd Qu.:2017-12-15 3rd Qu.:1943406 3rd Qu.:6393477   
## Max. :2018-05-06 Max. :6438429 Max. :6438432   
##   
## totalAmount Open High Low   
## Min. :0.000e+00 Min. : 0.5685 Min. : 0.6111 Min. : 0.4787   
## 1st Qu.:5.445e+11 1st Qu.: 1.7300 1st Qu.: 1.9500 1st Qu.: 1.5800   
## Median :2.658e+12 Median : 2.3200 Median : 2.4900 Median : 2.0300   
## Mean :1.844e+13 Mean : 3.6169 Mean : 3.9031 Mean : 2.2656   
## 3rd Qu.:1.005e+13 3rd Qu.: 3.6000 3rd Qu.: 3.8600 3rd Qu.: 3.1600   
## Max. :7.108e+16 Max. :73.0600 Max. :86.2600 Max. :49.0500   
##   
## Close Volume Market.Cap   
## Min. : 0.570 144,769,000: 7088 - : 9809   
## 1st Qu.: 1.740 11,828,700 : 6226 471,783,000: 7088   
## Median : 2.220 20,158,400 : 5945 333,047,000: 6226   
## Mean : 2.545 72,243,200 : 5161 131,122,000: 5945   
## 3rd Qu.: 3.590 17,115,200 : 5073 300,417,000: 5161   
## Max. :63.830 9,172,650 : 4588 224,227,000: 4532   
## (Other) :295556 (Other) :290876   
## return   
## Min. :-0.98481   
## 1st Qu.:-0.06126   
## Median : 0.00000   
## Mean : 0.02160   
## 3rd Qu.: 0.07759   
## Max. :32.40000   
##

we will first need to find the total number of investors per day and then find the total tokens bought by each investor per day. then we have to find the number pf investors who bought more than 10 token that day.

investortable <- newdataset %>% group\_by(Date)  
totalinvestor\_table<- summarize(investortable , total\_investors = n\_distinct(toNode))  
head(totalinvestor\_table)

## # A tibble: 6 x 2  
## Date total\_investors  
## <date> <int>  
## 1 2017-06-27 21  
## 2 2017-06-28 11  
## 3 2017-06-29 10  
## 4 2017-06-30 7  
## 5 2017-07-01 11  
## 6 2017-07-02 10

newinvestortable <- newdataset %>% group\_by(Date,toNode)  
newinvestortable\_totaltoken <- summarise(newinvestortable,total\_token = sum(totalAmount))  
summary(newinvestortable\_totaltoken)

## Date toNode total\_token   
## Min. :2017-06-27 Min. : 5 Min. :0.000e+00   
## 1st Qu.:2017-08-21 1st Qu.: 460418 1st Qu.:6.031e+11   
## Median :2017-09-26 Median :3571979 Median :2.500e+12   
## Mean :2017-10-18 Mean :3617180 Mean :2.940e+13   
## 3rd Qu.:2017-12-15 3rd Qu.:6400372 3rd Qu.:1.000e+13   
## Max. :2018-05-06 Max. :6438432 Max. :7.108e+16

subsetnewinvestortable\_totaltoken <- newinvestortable\_totaltoken[ which(newinvestortable\_totaltoken$total\_token>100), ]  
finalinvestortable <- summarise(subsetnewinvestortable\_totaltoken, final\_investors = n\_distinct(toNode))  
newfinalinvestortable <- merge(totalinvestor\_table,finalinvestortable,by = "Date")  
newfinalinvestortable$percentage\_investors <- (newfinalinvestortable$final\_investors/newfinalinvestortable$total\_investors)\*100  
newsimpletable <- merge(newsimpletable,newfinalinvestortable,by = "Date")

writiting the previous day fix :

head(newsimpletable$Date)

## [1] "2017-06-27" "2017-06-28" "2017-06-29" "2017-06-30" "2017-07-01"  
## [6] "2017-07-02"

tail(newsimpletable$return)

## [1] 0.04666667 0.02547771 0.01863354 -0.01829268 0.01242236 -0.04294479

shift <- function(x, n){  
 c(x[-(seq(n))], rep(NA, n))  
}  
newsimpletable$return <- shift(newsimpletable$return, 1)  
newsimpletable <- newsimpletable[1:(nrow(newsimpletable)-1),]

```

### 4.4 Create Linear Regression Model

In this part we created a multiple linear regression model on the price data and our primary token data.Our response variable(y) is simple price return given by where, is the token price in dollar for day. The regressor variables(x1…xn) are as follows: 1)No of token transactions per day 2)No of total token bought or sold per day 3)No of unique buyers per day 4)Percentage of investors who bought more than token per day.

Calculate response variable from the price data using open price. The first few records of the response variable as follows:

return <- numeric(nrow(priceData))  
return[379] <- priceData$Open[379]  
for (i in 378:1) {  
 return[i] <- (priceData$Open[i]-priceData$Open[i+1])/priceData$Open[i+1]  
}  
priceData$return <- return  
head(priceData$return)

## [1] 0.068813980 0.163917541 0.001947088 0.010113908 -0.057572838  
## [6] -0.022667458

Merge both the datasets according to date which would help us in calculating the regressor variables.

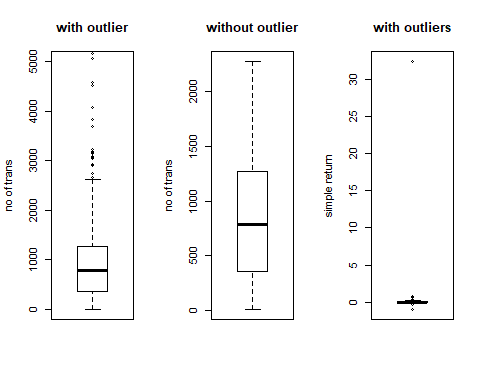
colnames(newwithoutoutlierdata)<-c('fromNode','toNode','Date','totalAmount')  
newdataset <- merge(newwithoutoutlierdata,priceData,by = "Date")

#### 4.4.1 Model 1:Simple Linear Regression

## [1] 25 13 15 10 12 15

Data preparation: univariate and bivariate analysis. **Univariate Analysis**:

Check for outliers and then remove them by capping our data to the max value.To remove majority of the outlier we are capping the data 92% quantile. Then we will alsocheck the response variable.



Boxplots of variables for Univariate analysis in simple regression

But for simple return response varaible case we won’t be removing any outliers as we don’t want to have any impact on the response variable.

**Bivariate Analysis**: Draw the scatterplot between the regressor and the response variable.Before we do that it is important to shift our response variable up by one row because we will be using yesterday to model today’s response.

shift <- function(x, n){  
 c(x[-(seq(n))], rep(NA, n))  
}  
newsimpletable$return <- shift(newsimpletable$return, 1)  
newsimpletable <- newsimpletable[1:(nrow(newsimpletable)-1),]  
head(newsimpletable$return)

## [1] 0.0000000 0.0000000 0.0000000 0.0000000 32.4000000 -0.1317901

#plot(newsimpletable$no\_of\_trans,newsimpletable$return)

We can see that there isn’t much relation between no of transactions each day and the simple return variable. We can confirm this by finding the correlation between the two

cor(newsimpletable[,c(6,7)])

## return no\_of\_trans  
## return 1.00000000 -0.07630676  
## no\_of\_trans -0.07630676 1.00000000

After completing the analysis we move ahead to fit the linear model.

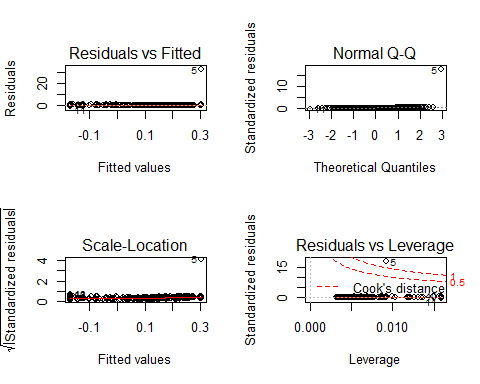
final\_data <- lm(return~no\_of\_trans,data=newsimpletable)  
summary(final\_data)

##   
## Call:  
## lm(formula = return ~ no\_of\_trans, data = newsimpletable)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -0.816 -0.231 -0.141 -0.014 32.099   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 0.3039009 0.1770723 1.716 0.0871 .  
## no\_of\_trans -0.0002076 0.0001538 -1.350 0.1781   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 1.833 on 311 degrees of freedom  
## Multiple R-squared: 0.005823, Adjusted R-squared: 0.002626   
## F-statistic: 1.821 on 1 and 311 DF, p-value: 0.1781

From the summary of the linear model we can clearly see that it is not a good model, the p-values of the intercept and the regressor are very high, also the r-square conveys only 2% of the response variable.

**Testing**

We can test this linear model using the plot function:



Residual Plot of the linear model

1. **Residuals vs fitted**:

Even though we see a horizontal line in this plot we can see that the residuals are not evenly spread across the line.It may not suggest a pattern but we can see most of the points in the left speculating a non-linear behaviour

1. **Normal Q-Q**:

The plot leads us to believe that the residuals follow normality as most of the points are seen to lie on the line or near it although few observations such as 42,2 and 156 are significantly away from the line.

1. **Scale-Location**:

The horizontal line in this plot suggests that homoscedasticity is observed in the residuals.

1. **Residuals vs leverage**:

This plots finds the observations which have a high influence on the regression model according to the cooks distance. We can see that observation 42 is out of the region thus is an outlier which could have an impact on the model if we do remove or decide to keep it in the model.

#### 4.4.2 Model 2:Multiple Linear Regression

Create three new features for our multiple regression model:

1. Total number of tokens involved per day:

First divide the total amount of each transaction with number of sub units of the tenxpay token(10^8) to get the number of tokens, then we will group by and sum each amount according to the date

First few records are as follows

## [1] 1.409702e+14 3.119692e+15 2.003250e+14 3.125740e+14 1.575796e+14  
## [6] 4.625641e+14

1. Number of Unique buyers per day:

Group the dataset according to the date and then count the number of unique buyers for each day.

First few records are as follows:

## [1] 21 11 10 7 11 10

1. Percentage of investors who have bought more than 10 tokens:

Find the total number of investors per day and then find the total tokens bought by each investor per day. Then find the number of investors who bought more than 10 token that day.

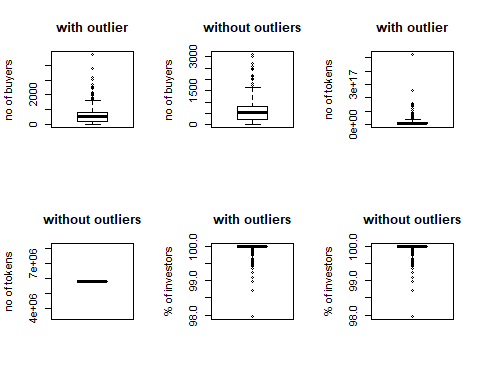
First few records are as follows:

## [1] 100 100 100 100 100 100

Now that we have all the required features lets do the analysis as we did for the simple linear regression:

**Univariate analysis**:

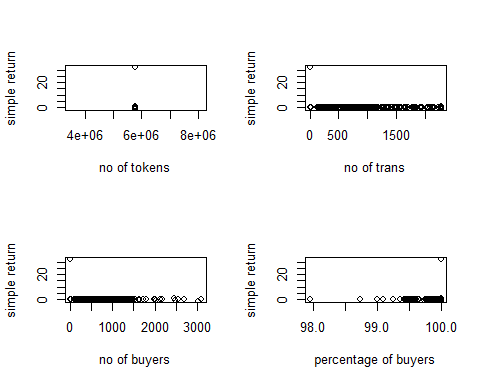
Create a boxplot of unique buys regressor variable and then we check for the number of tokens(and capping at 94%) and percentage of investors variable:



Boxplot if variables for univariate analysis in multiple regression

After the univariate analysis lets do the **bivariate analysis**:

plot number of transactions vs return:  
plot total token vs return:  
plot unique buyers vs return:  
plot percentage investors vs return:



Scatter plot of variables for Bivariate analysis in multiple regression

After analysing lets create an initial multiple regression linear model, Use the car library to calculate variable inflation factor and find the multicollinearity.

final\_data3 <- lm(return~no\_of\_trans+total\_token+unique\_buyers+percentage\_investors,data = newsimpletable)  
library(car)

Since the VIF for all the regressor variables is not much higher than 5 we will keep all of them in the initial model

summary(final\_data3)

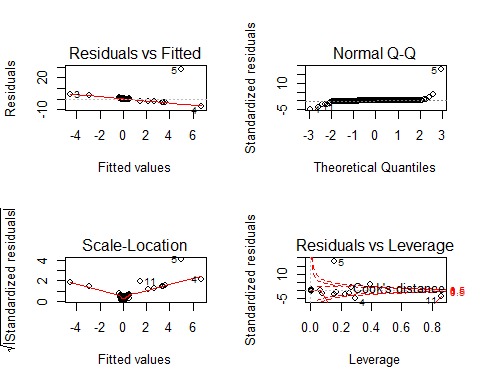
##   
## Call:  
## lm(formula = return ~ no\_of\_trans + total\_token + unique\_buyers +   
## percentage\_investors, data = newsimpletable)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -0.830 -0.233 -0.138 -0.002 32.091   
##   
## Coefficients: (1 not defined because of singularities)  
## Estimate Std. Error t value Pr(>|t|)  
## (Intercept) -1.449e+01 5.307e+01 -0.273 0.785  
## no\_of\_trans -2.048e-04 3.104e-04 -0.660 0.510  
## total\_token NA NA NA NA  
## unique\_buyers -2.541e-07 3.763e-04 -0.001 0.999  
## percentage\_investors 1.480e-01 5.310e-01 0.279 0.781  
##   
## Residual standard error: 1.839 on 309 degrees of freedom  
## Multiple R-squared: 0.006073, Adjusted R-squared: -0.003577   
## F-statistic: 0.6294 on 3 and 309 DF, p-value: 0.5965

From the summary, p-value for the variables is very high suggesting that there is no linear relation between the regressor variable and the response variables.But before giving up on this we haven’t yet used any features from the price data lets use the inherent features such as open, low, high etc to create a new linear model and see how we fare:

First find the multicollinearity and then use the step function to find the right combination of features:

Finally, Check the summary and plot of our final data model:

##   
## Call:  
## lm(formula = return ~ total\_token + Open + High + Low, data = newsimpletable)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -6.6956 -0.1706 -0.0263 0.1039 27.4404   
##   
## Coefficients: (1 not defined because of singularities)  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) -0.32508 0.11549 -2.815 0.00519 \*\*   
## total\_token NA NA NA NA   
## Open 0.30760 0.05222 5.891 1.00e-08 \*\*\*  
## High -0.28557 0.05075 -5.627 4.11e-08 \*\*\*  
## Low 0.18978 0.04375 4.338 1.95e-05 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 1.703 on 309 degrees of freedom  
## Multiple R-squared: 0.1476, Adjusted R-squared: 0.1393   
## F-statistic: 17.84 on 3 and 309 DF, p-value: 1.057e-10



Residual plot of the linear model for multiple regression

## 5.Conclusion

### 5.1 Q1

We totally tried 5 distributions while fiting distributions with our data: Poisson Distribution, Weibull Distribution, Exponential Distribution, Geometric Distribution and Normal Distribution. After compare with parameters and graphs of different distributions, we can find: The distribution of buyers and sellers are similiar. The estimate parameters of Normal Distribution are very close to the parameters of our token data, but the graphs don’t fit well. The graphs of Poisson Distribution and Exponential Distribution both fit our token data well, but for the parameters,we should not use Poisson Distribution.

Hence, we finally conclude that **Exponential Distribution** fit our distribution best.

### 5.2 Q2

The number of layers is 15. We tried several features of token data : (1) number of transactions(best correlation is 0.193) (2)Number of Unique buyers per day(best correlation is 0.14943) (3)Percentage of investors who have bought more than 10 tokens(best correlation is 0.1347)

Hence, the best correlation is 0.4943, when feature of token data is number of unique buyers per day.

### 5.4 Q4

When we use the price data columns and remove the features which have high p-value, we get value of **71%**. We have analysed how we can use simple and multiple linear regression to create linear models by doing feature extraction on the tenxpay token and price data. The model can be redeveloped and tested with different features to improve the R-squared value however the method of creation and analysis is mostly similar.