using weave

```
##
#load packages
using Plots
using StatsPlots
using Indicators
using MarketData
using IterableTables
using DataFrames
using StatsBase
using Statistics
using TimeSeries
using Impute: Impute
using FreqTables
using HypothesisTests
using StatsModels
using Lathe
using GLM


##
```

```
function process_assets(symbol,period) #gotta figure out how to fix date
    data = yahoo(symbol, YahooOpt(period1 =(now() - period),period2 =now()-Day(4)))
    returns = percentchange(data.Close)
    merged_data = merge(data,returns.*100)
    merged_data= TimeSeries.rename(merged_data::TimeArray, :Close_1 => :Return)
    merged_df = DataFrame(merged_data)

    return merged_data, merged_df
end

eth, eth_df = process_assets(string("ETH-USD"),Month(2))
lnk, lnk_df = process_assets(string("LINK-USD"),Month(2))
snp, snp_df = process_assets(string("^GSPC"),Month(2))
```

Note to self

if missing data split into training and test set before imputation, normalization, etc

```
##-- candlestick and MAs function
function crypto_MAs(ts,df,n1=7,n2=2)
    plot(ts, seriestype = :candlestick,title="Candlestick")
    movingaverage=sma(sort(df, :timestamp).Close, n=n1)
    short_MA = sma(sort(df, :timestamp).Close, n=n2)
    plot!(movingaverage, linewidth=2, color=:black)
    plot!(short_MA, linewidth=2, color=:blue,label=n2)

    end
##--
```

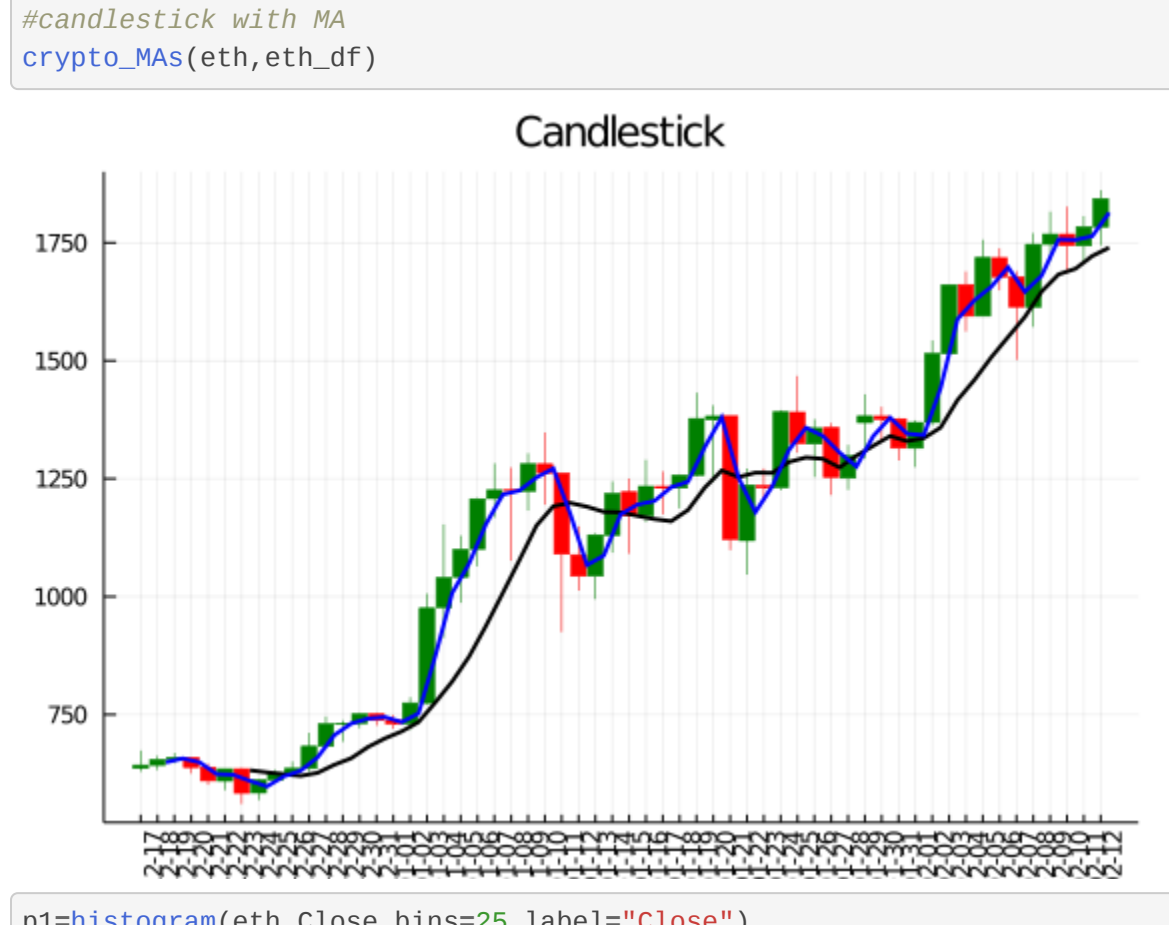
```
crypto_MAs (generic function with 3 methods)
```

Let's look at our Ethereum dataset.

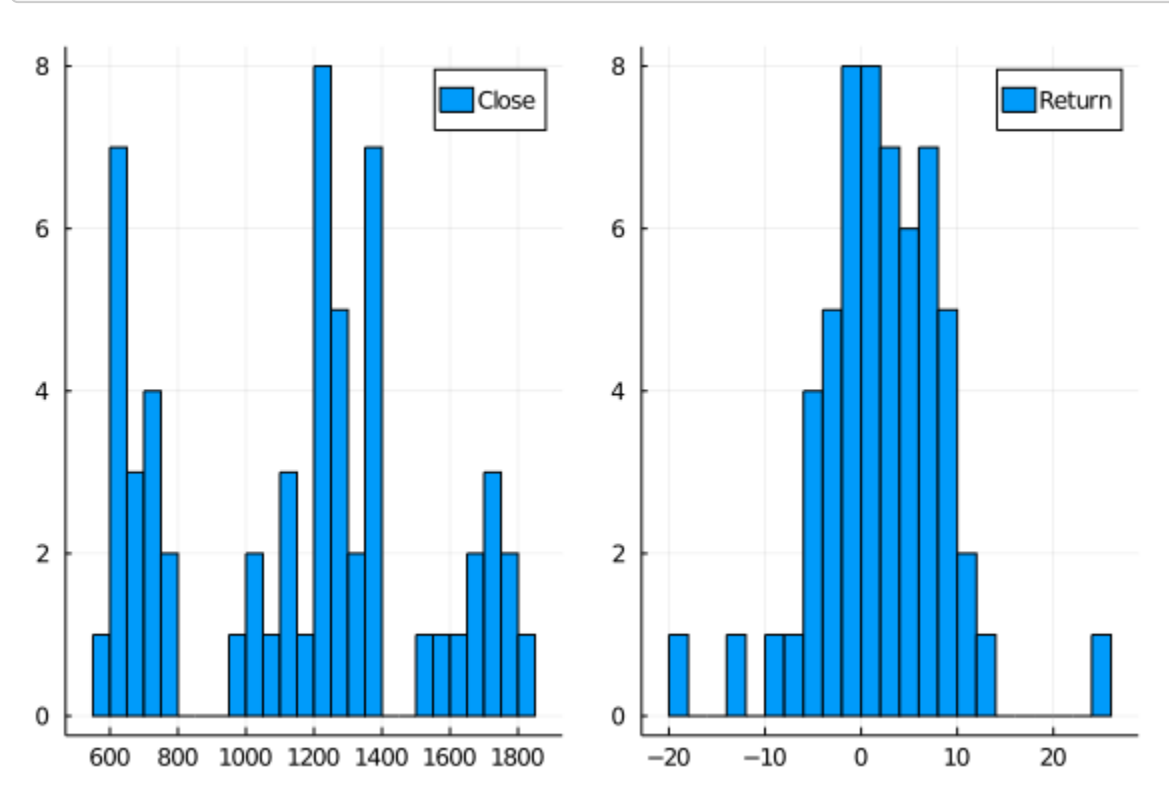
```
print(describe(eth_df),"\n")
sigma = std(eth_df.Close)
print("sigma: ",sigma)
```

```
8×7 DataFrame
 Row │ variable   mean        min         median      max         nmissing
 eltype
     │ Symbol     Union…      Any         Union…      Any         Int64
 DataType
─────┼──────────────────────────────────────────────────────────────────────
──────────
   1 │ timestamp              2020-12-17              2021-02-12         0
 Date
   2 │ Open       1139.16     584.136     1224.85     1783.49            0
 Float64
   3 │ High       1202.07     613.815     1271.92     1861.36            0
 Float64
   4 │ Low        1082.93     560.364     1128.05     1744.17            0
 Float64
   5 │ Close      1158.82     583.715     1227.93     1843.53            0
 Float64
   6 │ AdjClose   1158.82     583.715     1227.93     1843.53            0
 Float64
   7 │ Volume     3.32817e10  1.28309e10  3.62198e10  6.07336e10         0
 Float64
   8 │ Return     2.07201     -18.8605    2.03065     25.9475            0
 Float64
sigma: 372.4089807971295
```

```
#candlestick with MA
crypto_MAs(eth,eth_df)
```

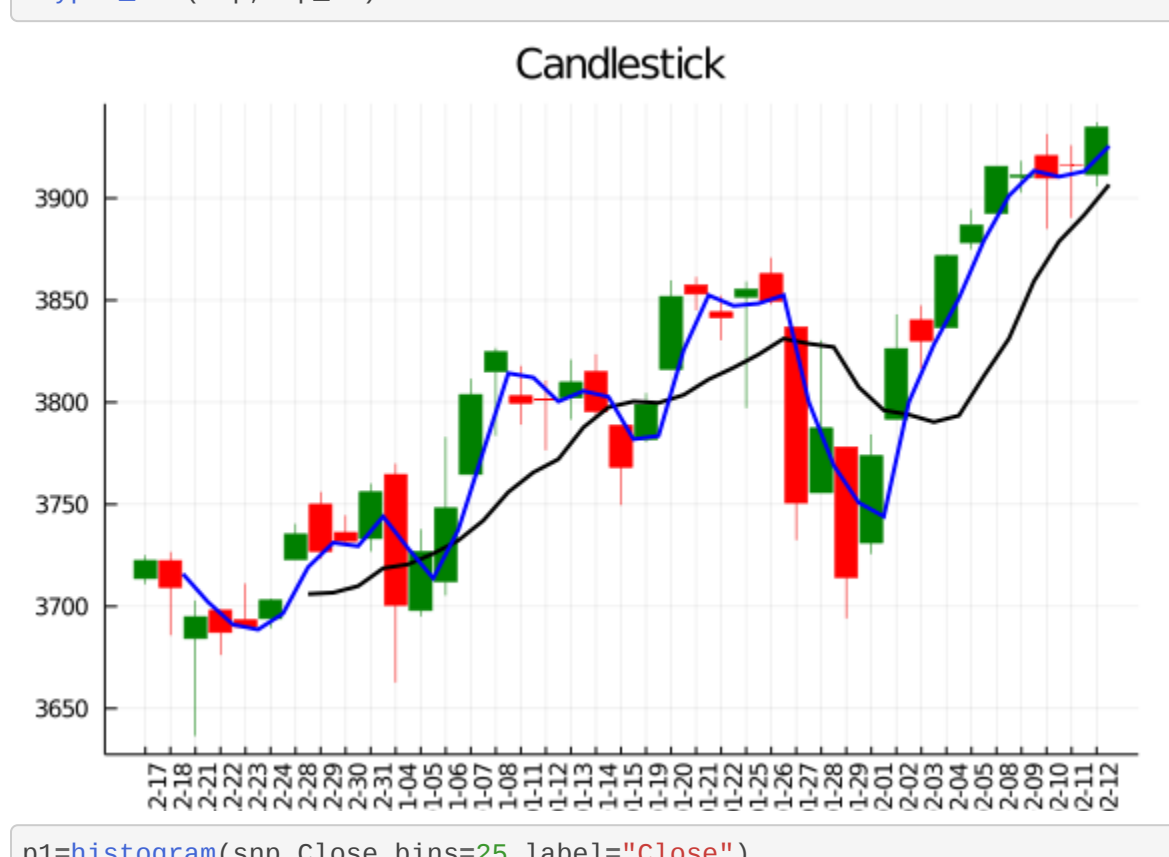



```
p1=histogram(eth.Close,bins=25,label="Close")
p2=histogram(eth.Return,bins=25,label="Return")
plot(p1, p2, layout = (1, 2))
```

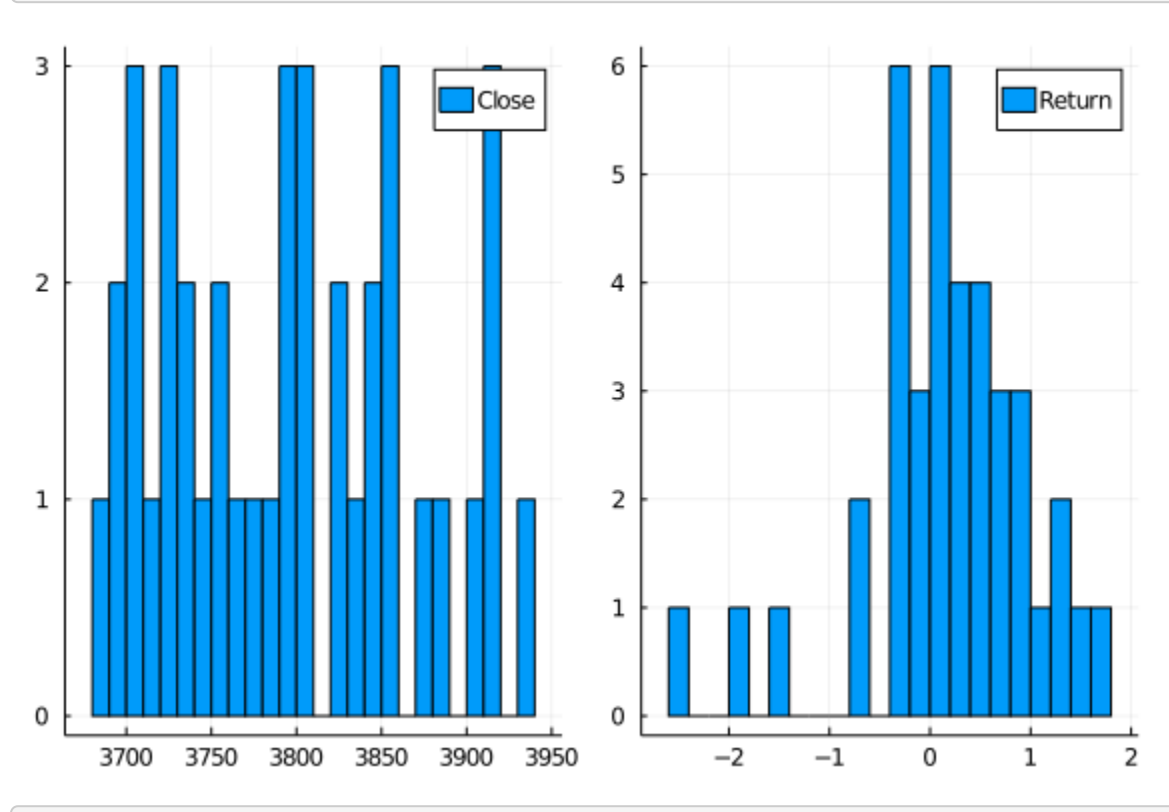


Now let's look at the S&P index

```
crypto_MAs(snp,snp_df)
```




```
p1=histogram(snp.Close,bins=25,label="Close")
p2=histogram(snp.Return,bins=25,label="Return")
plot(p1, p2, layout = (1, 2))
```



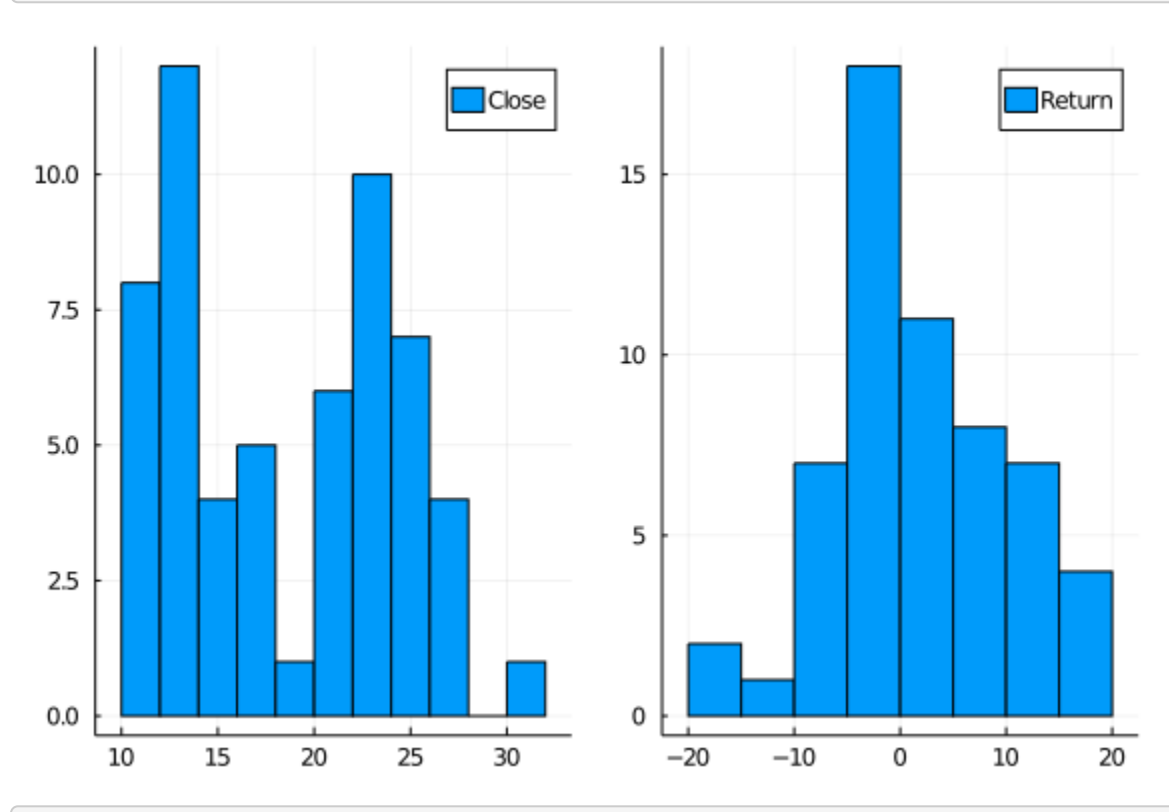

```
print(describe(snp_df),"\n","sigma ",std(snp_df.Close))
```

```
8×7 DataFrame
 Row │ variable   mean       min        median     max        nmissing  e
ltype
     │ Symbol     Union…     Any        Union…     Any        Int64     D
ataType
─────┼──────────────────────────────────────────────────────────────────────
────────
   1 │ timestamp             2020-12-17            2021-02-12         0  D
ate
   2 │ Open       3793.05    3684.28    3791.84    3920.78            0  F
loat64
   3 │ High       3812.43    3698.26    3817.86    3937.23            0  F
loat64
   4 │ Low        3772.4     3636.48    3776.51    3905.78            0  F
loat64
   5 │ Close      3795.27    3687.26    3798.91    3934.83            0  F
loat64
   6 │ AdjClose   3795.27    3687.26    3798.91    3934.83            0  F
loat64
   7 │ Volume     4.93776e9  1.88509e9  4.81538e9  9.87804e9          0  F
loat64
   8 │ Return     0.160757   -2.56779   0.166246   1.60518            0  F
loat64
sigma 72.81800872063174
```

Lastly, ChainLink

```
#Plotting link
crypto_MAs(lnk,lnk_df)
```


```
p1=histogram(lnk.Close,bins=15,label="Close")
p2=histogram(lnk.Return,bins=15,label="Return")
plot(p1, p2, layout = (1, 2))
```

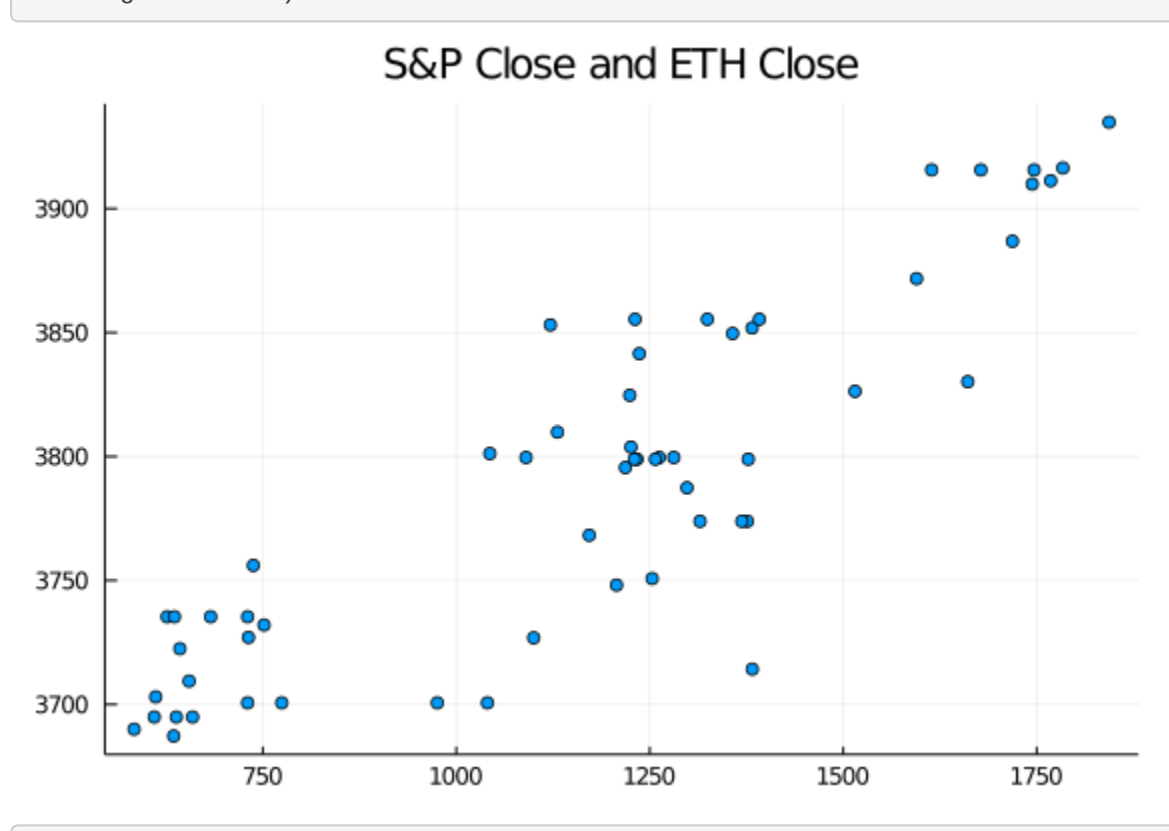



```
print(describe(lnk_df),"\n","sigma ",std(lnk_df.Close))
```

```
8×7 DataFrame
 Row │ variable   mean       min        median     max        nmissing  e
ltype
     │ Symbol     Union…     Any        Union…     Any        Int64     D
ataType
─────┼──────────────────────────────────────────────────────────────────────
────────
   1 │ timestamp             2020-12-17            2021-02-12         0  D
ate
   2 │ Open       18.358     10.8706    17.7325    27.8579            0  F
loat64
   3 │ High       19.5572    11.6637    19.8161    31.381             0  F
loat64
   4 │ Low        17.289     8.50915    16.4999    26.8283            0  F
loat64
   5 │ Close      18.6609    10.8313    18.1253    30.6407            0  F
loat64
   6 │ AdjClose   18.6609    10.8313    18.1253    30.6407            0  F
loat64
   7 │ Volume     2.85827e9  1.14201e9  2.81049e9  6.83169e9          0  F
loat64
   8 │ Return     1.74282    -15.6659   0.135013   17.9877            0  F
loat64
sigma 5.646157031358134
```

```
#join ethereum tabl to snp table to get paired values
eth_snp_df= leftjoin(eth_df,snp_df, on = :timestamp,makeunique=true)
closing_df = eth_snp_df[:,["Close","Close_1"]]

#account for missing values with next observation carried back method
eth_snp_df = Impute.nocb(eth_snp_df)
closing_df = Impute.nocb(closing_df)

eth_lnk_df = leftjoin(eth_df,lnk_df, on = :timestamp,makeunique=true)
eth_lnk_df_close = eth_lnk_df[:,["Close","Close_1"]]
```

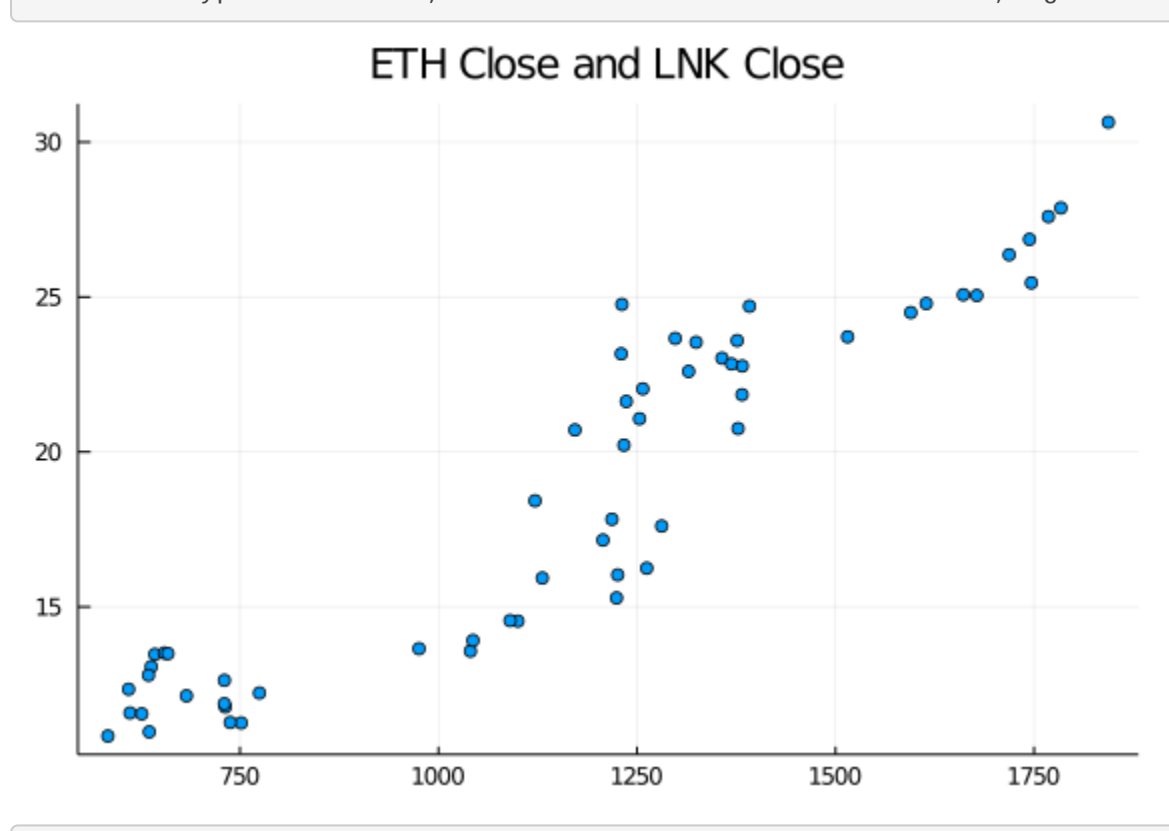
```
gr()
plot(closing_df.Close,closing_df.Close_1,
    seriestype = :scatter,
    title = "S&P Close and ETH Close",
    legend=false)
```




```
#no correlation between returns for snp and eth
@df eth_snp_df print("Return Correlation :",cor(:Return,:Return_1),"\n")
#moderate correlation in closing prices
@df eth_snp_df print("Closing Price Correlation :",cor(:Close,:Close_1))
```

```
Return Correlation :0.004829710905922176
Closing Price Correlation :0.8732967219733851
```

```
#ETH vs LNK
plot(eth_lnk_df_close.Close,eth_lnk_df_close.Close_1,
    seriestype = :scatter, title = "ETH Close and LNK Close",legend=false)
```




```
#no correlation between returns for snp and eth
@df eth_lnk_df print("Return Correlation :",cor(:Return,:Return_1),"\n")
#good correlation in closing prices
@df eth_lnk_df print("Closing Price Correlation :",cor(:Close,:Close_1))
```

```
Return Correlation :0.5666621302109638
Closing Price Correlation :0.9323325661709652
```

Regression Model

Since the correlation between the closing price of ethereum and the closing price of ChainLink is so high, let's run a regression model and see what happens.

First split data

```
combined_data=DataFrame(eth=eth_df.Close,lnk=lnk_df.Close)
train, test = Lathe.preprocess.TrainTestSplit(combined_data,.70)
```

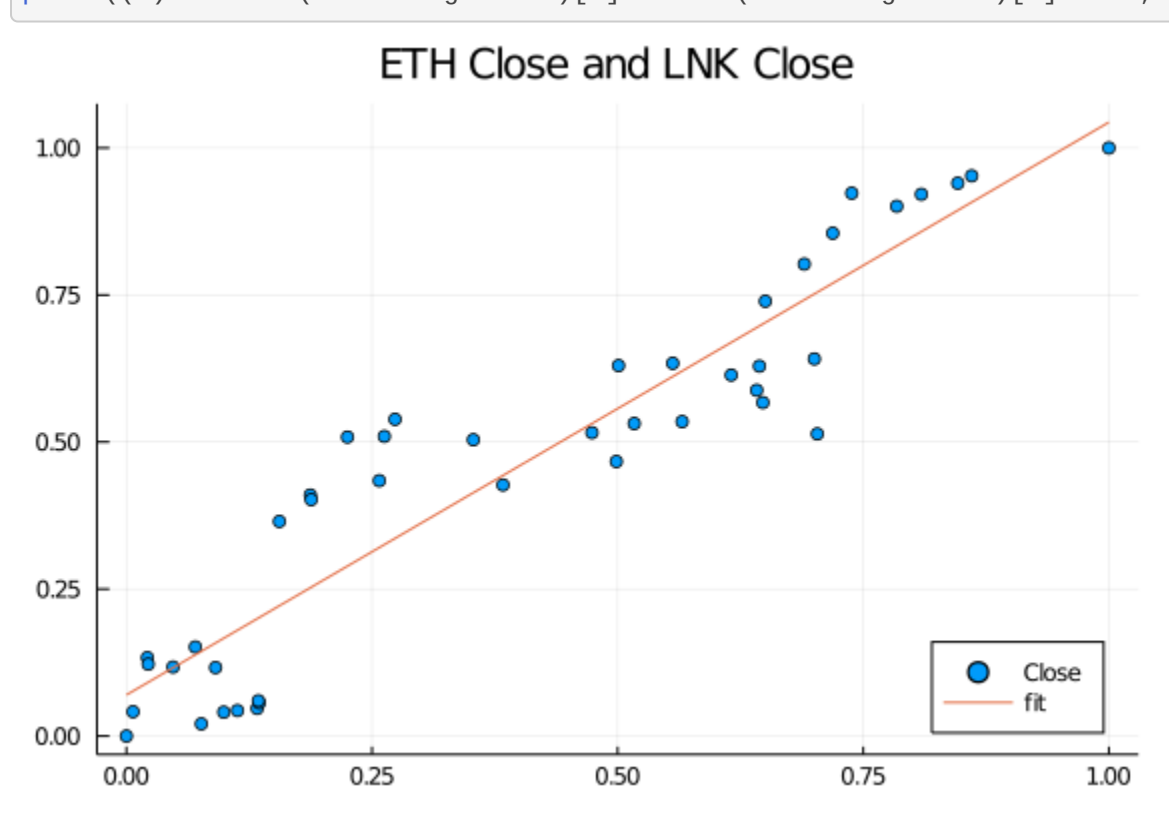
```
(43×2 DataFrame
 Row │ eth       lnk
     │ Float64   Float64
─────┼───────────────────
   1 │  642.869  13.4671
   2 │  654.812  13.5112
   3 │  659.298  13.4922
   4 │  638.291  13.0701
   5 │  609.818  12.342
   6 │  634.854  12.7939
   7 │  583.715  10.8313
   8 │  635.836  10.9649
  ⋮  │    ⋮         ⋮
  37 │ 1594.76   24.499
  38 │ 1718.65   26.3633
  39 │ 1746.62   25.4561
  40 │ 1768.04   27.5942
  41 │ 1744.24   26.8596
  42 │ 1783.8    27.8742
  43 │ 1843.53   30.6407
          28 rows omitted, 15×2 DataFrame
 Row │ eth       lnk
     │ Float64   Float64
─────┼───────────────────
   1 │  611.607  11.5755
   2 │  626.411  11.5467
   3 │  682.642  12.1297
   4 │  730.368  11.8726
   5 │  975.508  13.6502
   6 │ 1040.23   13.5711
   7 │ 1207.11   17.1553
   8 │ 1281.08   17.6075
   9 │ 1230.17   23.1668
  10 │ 1236.51   21.6295
  11 │ 1382.52   22.7734
  12 │ 1314.99   22.5984
  13 │ 1369.04   22.8396
  14 │ 1677.85   25.0493
  15 │ 1614.23   24.7901)
```

```
#Next step is to normalize
#function to normalize a pair of columns ie eth and snp returns
function normalize_column(arr)
    dt=StatsBase.fit(UnitRangeTransform, arr; dims=1, unit=true)
    dt_norm = StatsBase.transform(dt,arr)
    return dt_norm
end

eth_norm_close = normalize_column(train.eth)
lnk_norm_close = normalize_column(train.lnk)

eth_lnk_norm_close = DataFrame(eth=eth_norm_close,lnk=lnk_norm_close)
```

```
#Run linear regression on lnk and eth


fm = @formula(eth ~ lnk)
linearRegressor = lm(fm, eth_lnk_norm_close)

print(linearRegressor,"\n",r2(linearRegressor))
```

```
StatsModels.TableRegressionModel{LinearModel{GLM.LmResp{Array{Float64,1}},G
LM.DensePredChol{Float64,LinearAlgebra.Cholesky{Float64,Array{Float64,2}}}}
,Array{Float64,2}}

eth ~ 1 + lnk

Coefficients:
──────────────────────────────────────────────────────────────────────────
                 Coef.  Std. Error      t  Pr(>|t|)  Lower 95%  Upper 95%
──────────────────────────────────────────────────────────────────────────
(Intercept)  0.0699258   0.0292742   2.39    0.0216  0.0108054   0.129046
lnk          0.973822    0.0588425  16.55    <1e-19  0.854987    1.09266
──────────────────────────────────────────────────────────────────────────
0.869796218490742
```

```
gr()
plot(eth_lnk_norm_close.lnk,eth_lnk_norm_close.eth,
    seriestype = :scatter, title = "ETH Close and LNK Close",label="Close",legend=:bottomright)
plot!((x) -> coef(linearRegressor)[1] + coef(linearRegressor)[2]  * x, 0, 1, label="fit")
```



Predict on Test Data

```
eth_norm_close_test = normalize_column(test.eth)
lnk_norm_close_test = normalize_column(test.lnk)

test = DataFrame(eth=eth_norm_close_test,lnk=lnk_norm_close_test)

ypredicted_test = predict(linearRegressor, test)


#predicted vs actual
pred_df = DataFrame(eth_pred=ypredicted_test,eth_actual=test.eth)
pred_df.err = pred_df.eth_actual-pred_df.eth_pred
rmse = sqrt(mean(pred_df.err.*pred_df.err))
print("RMSE:", rmse)
```

```
RMSE:0.15059727156265065
```

Time for Bayes

```
#Bayesian Inference section
#Build function to calculate BI probability

#function------------------------
function bayes_prob(df1,df2,increase=true)

    df = leftjoin(df1,df2, on = :timestamp,makeunique=true)
    if sum(describe(df).nmissing) > 0
        df = Impute.nocb(df) #in case missing values exist

    end

    return_df = DataFrame(A=df.Return.>0,B=df.Return_1.>0)
    my_table = freqtable(return_df.B,return_df.A)

    if increase == true
        prob_a = sum(my_table[:,Name(true)])/sum(my_table)
        prob_b = sum(my_table[Name(true),:])/sum(my_table)
        prob_ba = sum(my_table[Name(true),Name(true)])/sum(my_table[:,Name(true)])

        prob_ab = prob_ba*prob_a/prob_b
    else
        prob_a = sum(my_table[:,Name(true)])/sum(my_table)
        prob_b = sum(my_table[Name(false),:])/sum(my_table)
        prob_ba = sum(my_table[Name(false),Name(true)])/sum(my_table[:,Name(true)])

        prob_ab = prob_ba*prob_a/prob_b
    end


    return prob_ab

end

#end function
```

```
print("Probability ETH returns increase given that ChainLink returns increase: ")
print(round(bayes_prob(eth_df,lnk_df),digits=3), "\n")

print("Probability ETH returns increase given that ChainLink returns decrease: \n")
print(round(bayes_prob(eth_df,lnk_df,false),digits=3))
```

```
Probability ETH returns increase given that ChainLink returns increase: 0.8
Probability ETH returns increase given that ChainLink returns decrease:
0.464
```


Published from cic_analytics.jmd using Weave.jl v0.10.6 on 2021-02-16.